ISSN 0254-2757

QI QUAESTIONES INFORMATICAE

Volume 4 Number 3 October 1986

An official publication of the Computer Society of South Africa and of the South African Institute of Computer Scientists

'n Amptelike tydskrif van die Rekenaarvereniging van Suid-Africa en van die Suid-Afrikaanse Instituut van Rekenaarwetenskaplikes

QUÆSTIONES INFORMATICÆ

An official publication of the Computer Society of South Africa and of the South African Institute of Computer Scientists

'n Amptelike tydskrif van die Rekenaarvereniging van Suid-Africa en van die Suid-Afrikaanse Instituut van Rekenaarwetenskaplikes

Editor

Professor G. Wiechers
INFOPLAN
Private Bag 3002
Monument Park 0105

Editorial Advisory Board

Subscriptions

Annual subscriptions are as follows:

	SA	US	UK
Individuals	R10	\$ 7	£ 5
Institutions	R15	\$14	£10

Circulation and Production

Mr C.S.M. Mueller
Department of Computer Science
University of the Witwatersrand
1 Jan Smuts Avenue
Johannesburg, 2001

Quæstiones Informaticæ is prepared by the Computer Science Department of the University of the Witwatersrand and printed by Printed Matter, for the Computer Society of South Africa and the South African Institute of Computer Scientists.

LAWS AND TECHNIQUES OF INFORMATION SYSTEMS

J. Mende
Department of Accounting,
University of the Witwatersrand,
Johannesburg, 2000

Research has a utilitarian role of providing the knowledge needed by the designers of mankind's artefacts. In this context four distinct categories of knowledge are identifiable.

1. *Laws of Nature* describe inherent properties of natural entities found in artefacts.
2. *Laws of the Artificial* describe contrived properties of artificial entities.
3. *Natural Techniques* help combine natural entities to form artificial entities.
4. *Artificial Techniques* help designers combine artificial entities to form artefacts.

This classification can be used to identify deficiencies and special features of research on information systems.

1. META-RESEARCH

"Information Systems" is one of the youngest subjects of academic enquiry. Researchers have not had much time to consider what kinds of knowledge are necessary, let alone to find that knowledge. Accordingly, our subject matter has not reached the level of precision and consistency of more mature fields such as Physics and Electrical Engineering.

So in parallel with research on actual information systems, there is a need for "meta research" on the research process itself, to identify desirable directions of exploration.

A previous article [8] in this journal has suggested some potential inputs and outputs of research projects. The present article focuses on the output question: What kinds of knowledge should research produce? In an attempt to answer that question, it analyses the development of human artefacts in the context of increasing economic productivity.

2. INCREASING PRODUCTIVITY

Economic History [3] shows that centuries ago man toiled strenuously to secure the basic necessities of life - consumables such as food and drink, clothing and shelter. But over the ages people have created machines such as tractors, railways, computers etc, and they have established procedural systems such as mass production and information systems. These "artefacts" [10] have increased human productivity, allowing more consumables to be produced with less labour. As a result, Man today enjoys a much higher standard of living than his ancestors.

But Man is not satisfied with his present standard of living. He continually strives to replace existing machines and systems with new artefacts that produce even more consumables with still less labour.

However, the designs of these artefacts are constrained by Man's knowledge. For instance, the earliest civilisations had comparatively little knowledge of Science or Engineering. So designers could only develop relatively labour intensive artefacts such as bellows-driven smelters and oared galleys.

Today, Man's knowledge is both more extensive and more precise. Electrical Science and Engineering provide a basis for the invention of arc furnaces- which eliminate bellows from smeltering. Mechanical Science and Engineering have made it possible to invent steamships - which transport goods with a fraction of a galley's labour input. Chemistry and Chemical Engineering have allowed us to devise oil refineries and leach plants of unprecedented efficiency. Therefore better knowledge now enables us to design artefacts which produce consumables with less labour.

In order to achieve even more productive designs in the future, we need to improve the knowledge base for design. Firstly, we must extend it by gaining new insights on phenomena which are still shrouded in mystery. Secondly, as our understanding is rarely perfect, we also require more accurate knowledge of familiar phenomena [7]. Accordingly Man's urge to develop more productive artefacts calls for ongoing discovery of new knowledge.

That means the design process depends on the research process - which suggests that research output requirements can be identified by *needs analysis* of design activities.

3. THE DESIGN PROCESS

Watching an electrical designer, a chemical designer or a program designer at work, one observes that he performs a series of physical actions : he writes, he draws, he calculates. But these actions do not happen at random. Each step must be activated by a prior decision. For example, before an electrical engineer draws a block on a circuit diagram, he determines whether it should be a resistance, a capacitor, a diode, etc. Similarly when a programmer draws a flowchart, he has to select the appropriate symbol before drawing it. So the physical steps in the design are accompanied by a series of corresponding design decisions.

Designing the thousands of artefacts required to enhance human productivity involves millions of such decisions. However, experience shows that many of them are similar. That means the same kinds of design decisions are being made over and over again - which presents an opportunity for standardisation. Suppose formal "techniques" of making every recurrent design decision were published in professional journals. Then any designer can simply apply the published technique whenever he faces such a decision. Therefore standard techniques diminish the effort of design.

For example, "Kirchoff's Rules" [6] constitute a general technique for determining unknown currents (I), resistances (R) and electromotive forces (E) in an electrical circuit. The technique reduces circuit calculations to three easy steps:

1. For every function, set $\Sigma I = 0$
2. For every loop, set $\Sigma IR = E$
3. Solve the resulting equations.

Similarly Jackson's technique of structure charting [4] reduces program design to the following simple steps:

1. Hierarchically decompose the input and output data into sequences, selections and iterations.
2. Draw a procedure structure chart with sequences, selections and iterations that correspond to the superposition of the data hierarchies.

In the interests of efficiency, the knowledge base of design should therefore include *techniques*.

However techniques alone are not enough. Every artefact consists of constituents with characteristic properties of their own. Unless these properties are recognised in the design process, the artefact will not perform as intended. Therefore design decisions should be based on "Laws" - statements which describe the properties of design components.

That implies techniques should be derived from laws. For example, Kirchoff's technique mentioned earlier depends on two laws of electricity.

Step 1 ($\Sigma I = 0$) is a direct consequence of the law of Charge Conservation:
"The total charge in an isolated circuit never changes".
Step 2 ($\Sigma IR = E$) follows from Ohm's law:
"The current in a conductor is proportional to the voltage".

Similarly, Jackson's technique of structure charting derives from Böhm and Jacopini's law of flowcharting [1] plus three (unstated) laws of the computer. These computer laws can be expressed in terms of arbitrary procedures P and Q which process arbitrary data sets D and E:

1. The sequence P-Q processes the sequence D-E
2. The selection P or Q processes the selection of D or E
3. The iteration of P processes the iteration of D.

Law-based techniques such as the above implicitly ensure that recurrent decisions are consistent with underlying laws. But for non-recurrent decisions the designer also needs explicit statements of the properties of design components. For example, many electric circuits will malfunction unless the designer recognises the internal resistance of the energy source - Thevenin's Law. And some computer programs will fail unless the designer knows that division by zero in impossible.

Therefore, in addition to techniques, the knowledge base for design must include *laws*.

These two types of knowledge are different in character. Laws specify the properties of the design components; techniques help the designer combine the components into an artefact. Laws describe the *operands* of design; techniques prescribe the *operations* that can be performed on the operands.

Next, different types of laws and techniques will be identified by analysing the components of artefacts.

4. DESIGN COMPONENTS

In the beginning, Man's artefacts were uncomplicated. Their components were obtained form natural sources: for example, primitive smelters produced iron from surface ores and charcoal. Their designs were based on observations of a few basic patterns of Nature, such as carbon's ability to reduce oxides.

However, today's artefacts are much more complex. Natural components are first combined into functional components, and those are then assembled into useful artefacts. For example, at the first level of segmentation, an electric power drill consists of manufactured components: a motor, gears, capacitors etc. At the next level, those functional components consist of manufactured components, such as wire, steel plate etc. Finally at the next level those components in turn consist of natural materials such as iron and copper.

The natural components at the lower levels of the "systems hierarchy" [9] are characterised by *"Laws of Nature"*. Higher-order manufactured components have properties not found in nature, so they are described by a different type of law. For example, Ohm's law merely refers to the low-level copper wires in an electrical motor. The device as a whole is described by the formulæ of alternating current motors - *"Laws of the Artificial"*.

Therefore the knowledge base for design should contain two kinds of laws: natural and artificial.

These two types of law describe entities of quite different origin. Motors, capacitors and gears would not exist if Man had not specifically constructed them. Thus artificial entities originate with Man. On the other hand, natural entities such as copper and iron have existed well before Man. So their origin is non-human.

The dichotomy of origin affects the properties of natural and artificial entities. Natural entities are imbued with inherent properties which stem from their non-human origin. But whenever Man creates an artificial entity from natural entities, he gives it additional attributes which suit his own utilitarian purposes. So within broad limits imposed by the Laws of Nature, Man *contrives* the properties of artificial entities. Thus Laws of Nature describe inherent properties, but Laws of the Artificial specify contrived properties of an artefact's components.

A similar distinction also applies to techniques. When a design involves natural components, one employs "natural" techniques based on Laws of Nature. For example, Kirchoff's Rules are used to combine natural conductors into circuits. And these are derived from the Laws of Nature which describe the inherent properties of electric conductors.

On the other hand, when a design involves artificial components, one needs "artificial" techniques based on Laws of the Artificial. For example, Jackson's technique is used in combining artificial hardware functions to form information systems. And the technique depends on contrived properties of flowcharts and computers which are described by artificial laws.

Therefore the knowledge base for design should also include two types of techniques -

natural and artificial.

Again, the dichotomy of origin gives rise to differences between the two types. Natural techniques are consequent upon inherent properties of natural design components. Artificial techniques are consequent upon contrived properties. Man cannot influence inherent properties, whereas he can and does determine contrived properties. Consequently, artificial techniques depend on Man's utilitarian purposes, but natural techniques do not.

5. KNOWLEDGE NETWORK

The foregoing shows that research in general should produce four distinct types of knowledge:

- Laws of Nature (such as Ohm's Law), which describe inherent properties of natural entities;
- Natural Techniques (such as Kirchoff's Rules), which prescribe how to combine natural entities;
- Laws of the Artificial (such as the Böhm-Jacopini Law) which describe contrived properties of artificial entities;
- Artificial Techniques (such as Jackson's structure charting), which prescribe how to combine artificial entities.

These are interrelated (fig. 1). Natural techniques are based upon Laws of Nature. Artificial techniques are based upon Laws of the Artificial. And Laws of the Artificial are constrained by Laws of Nature.

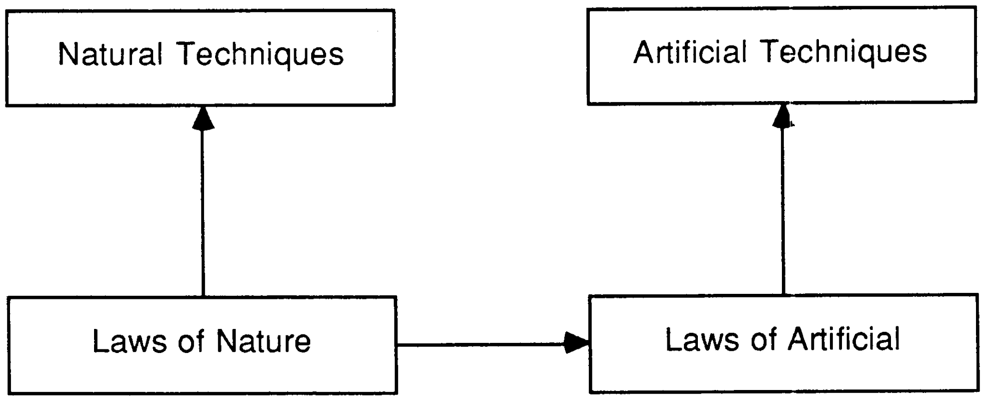


figure 1

Knowledge for Design

All four categories are represented in the subject Information Systems. The systems we study involve natural entities such as people, as well as artificial entities such as computers and information. The natural entities are subject to natural laws, and design decisions affecting them require natural techniques. The artificial entities are described by artificial laws, and are incorporated into systems with the aid of artificial techniques.

6. IMPLICATIONS

Researchers in the field of Information Systems can derive many implications from this hierarchy. For instance, the *classification scheme* reveals significant contrasts between our work and that of colleagues in other fields.

At present, much of our research is concerned with artificial systems components - computers and information. On the other hand, scientists are primarily concerned with natural entities. Therefore we mainly produce *artificial* laws and techniques, while they establish *natural* laws and techniques.

These differences have two implications. Firstly, our artificial laws and techniques depend on Man's utilitarian purposes. But those purposes can and do change. So our research results are subject to rapid *obsolescence,* whereas our scientific colleagues' results are more permanent.

Secondly, in order to ensure a longer effective life for our artificial laws and techniques, we need to consider Man's *future* purposes. This suggests that some research should perhaps be based on forecasts, not simply on current technology and information needs.

Thirdly, research on the human aspects of information systems is likely to be discomforting. Humans are natural rather than artificial components. Therefore a researcher accustomed to the mechanistic artificial components of information systems may have to re-orientate himself drastically in humanistic research.

Further, the *logical relationships* between the four categories point out some significant deficiencies in current publications on information systems.

Firstly, techniques should be based on laws. This shows up serious weaknesses in many of our published techniques. Some textbooks omit explicit statements of the underlying artificial laws, e.g. Jackson's Principles of Program Design [4]. Others fail to show the logical derivation, e.g. Gane and Sarson's Structured Systems Analysis [2]. Both omissions cast an unfortunate shadow of doubt on otherwise magnificent work.

Secondly, the logical relationships affect our research methods. Popular textbooks of Research Methodology (e.g. 5) urge the researcher to adopt "empirical" methods. That is sound advice for natural laws, but not a sole requirement for artificial laws and techniques. Why verify an artificial law or a technique by exhaustive empirical testing when it can be derived logically from underlying laws? So empirical methods should be the exceptions rather than the norm in a subject which is largely concerned with the artificial.

7. IN CONCLUSION

Researchers of different backgrounds might find the above taxonomy illuminating. It may renew a scientist's hopes of finding laws in a field presently dominated by techniques. The engineer may look forward to establishing techniques that are solidly based on laws. And the business specialist may be influenced to join the search for laws and law-based techniques.

It is also recommended for consideration by authors and lecturers. In writing a textbook it may be advantageous to emphasise underlying laws before presenting the consequent techniques. Similarly in lecturing on a technique, one may find it helpful to discuss laws and establish the logical necessity of the technique before proceeding to its use.

Finally, the taxonomy can affect attitudes. Intellectuals tend to regard Laws of Nature with considerable more respect than Laws of Artificial [10]. Indeed the latter are not usually even admitted to the status of "Laws": for instance Ohm's *Law* on the one hand but motor *formulæ* on the other. However the realm of the artificial is becoming increasingly important. People live in artificial dwellings, wear artificial clothes, and work with artificial devices in artificial organisations. So in terms of human progress, artificial laws and techniques may actually become more significant than their natural counterparts. Then the subject Information Systems could become one of the most prestigious in academe.

REFERENCES

1. Böhm, C. and Jacopini, G. (1966). Flow Diagrams, Turing Machines and Languages with Only Two Formation Rules, *Comm. ACM*, 9, 366 - 371.
2. Gane, C. and Sarson, T. (1979). *Structured Systems Analysis*, Prentice-Hall, Englewood Cliffs, New Jersey.
3. Hohenburg, P. (1968). *A Primer on the Economic History of Europe*, Random House, New York.
4. Jackson, M.A. (1975). *Principles of Program Design*, Academic Press, New York, 15 - 91
5. Kerlinger, F.N. (1973). *Foundations of Behavioral Research*, Holt, Rinehart and Winston, New York.
6. Kip, A. (1962). *Fundamentals of Electricity and Magnetism*, McGraw-Hill,

New York, 153 - 157.
7. Kuhn, T. (1970). *The Structure of Scientific Revolutions*, University of Chicago Press, Chicago, p7.
8. Mende, J. (1986). Research Directions in Information Systems, *Quæstiones Informaticæ*, **4**, 1 - 4.
9. Short, R. (1985). Levels of Abstraction and Systems Hierarchies, *in Proceedings of the 15th annual UCSLA Conference.*
10. Simon, H.A. (1969). *The Sciences of the Artificial*, Massachusetts Institute of Technology Press, Cambridge, Mass.

BOOK REVIEW

Macintosh Graphics in Modula-2, Russell L. Schnapp, Prentice-Hall, Englewood Cliffs, New Jersey, 1986. ISBN 0–13–542309–0. 190 pages.

Reviewed by: Philip Machanick, *University of the Witwatersrand, Johannesburg 2001*

At first sight, this seems to be a book addressing a very narrow niche market—a specific class of application in a specific language on a specific computer. Despite this apparent limitation, it is valuable in several respects. Of particular interest is the way it—without any pretensions at doing so—suggests the viability of Modula-2 as a first teaching language, as well as the utility of features supposedly supporting programming-in-the-large in relatively small programs.

The former point is illustrated by the author's approach to defining useful modules, then following them with examples using them. An interesting approach to teaching beginners to program would be to present them with such a predefined module as a starting point for exploratory programming in the style of Logo. The way Modula-2's abstraction and information hiding features are developed through examples (the full text of most is presented—a disk with the Modula-2 source is available at a modest price) and exercises suggests students could be gently broken in to such an approach.

An illustration of the way the programming-in-the-large features of Modula-2 are put to good use is the way events (how asynchronous actions like mouse movements and keystrokes are passed to Macintosh programs) are hidden inside a module in one of the examples; unfortunately, this approach is not carried through to the following example.

An unusual (but positive) feature is that the author doesn't hesitate to point out problems with the language as they are encountered.

Aside from a few minor errors, the book is useful as extra documentation for the Macintosh's Toolbox and QuickDraw routines. The author's occasional references to *Inside Macintosh*[1] and the MacModula-2 manual are a bit of a joke—his book would not be filling such an obvious need if that documentation were comprehensible. From this point of view, it is a pity he did not explain some of the details a bit more thoroughly—such as his use of CODE in procedures accessing built-in routines. This is a relatively minor issue, since this technique only applies to the 128K versions of the modules (at least with the current compiler). Of more significance is the fact that later releases have major enhancements which would have been worth covering—such as access to Resources (crucial for good use of the user interface). Nonetheless, his mainly well-written and documented examples are a valuable extension to existing documentation. Minor quibbles include excessive use of INTEGER where a subrange would have been enough, and the occasional over-use of globals (though this a problem in most module-based languages—variables private to the module cannot be passed as parameters to publically-known procedures).

The book would have been of more value if it had explained design as well as presenting complete examples. But then it would have lost one of its chief virtues: brevity. Perhaps a second volume about design—drawing on the examples of this book—is called for.

REFERENCE

[1] *Inside Macintosh* (three volumes), Addison-Wesley, Reading, Massachusetts, 1985. ISBN 0–201–17737–4.

A HIGH-LEVEL INTERFACE TO A RELATIONAL DATABASE SYSTEM

S. Berman and L. Walker
University of Cape Town
Rondebosch 7700

A high-level language for accessing a relational database is being developed at UCT. The system, known as HAL (High-level Access Language), provides complete data independence as users view the database in terms of objects and their properties. Constraints on data usage can be incorporated in the data definition and enforced by HAL. In this way a simpler, more controlled interface to a relational database is obtained.

KEY WORDS: relational database, access language, metadata, data independence, subschema

1. INTRODUCTION

Relational databases are easier to use than others primarily because their non-procedurality reduces the navigation problems of the earlier systems. However they still require a thorough knowledge of the relational structure of the data and the correct use of non-trivial operations such as join, which is a form of navigation in that it enables one to pass from one relation to another. The HAL project was initiated to present the programmer with a natural view of data as objects and their properties, where no knowledge of the underlying relations is required. Additional advantages of HAL are enforcing more integrity constraints on data and enabling the metadata (data definition) to be accessed as well as data.

This paper describes the HAL subschema and data manipulation language. Implementation considerations are then outlined and the system evaluated by comparison with conventional relational languages.

2. THE HAL SUBSCHEMA

An example of a simple subschema can be seen in the appendix. A HAL subschema comprises a list of objects optionally followed by a list of tasks. An object is described by several properties each associated with an attribute or relation in the database. The tasks are database manipulating routines which can be called from within programs. Every property is designated as mapping to 1 or M (many) values and has a valuetype which is STRING, INTEGER, REAL, or an object name. The latter case implies that the property references other objects in the database. The application programmer sees only a list of object and property names, and a list of task names.

Constraints can stipulate that certain properties are "compulsory" and cannot be null, or are "id-properties" and cannot have duplicate values. Further, where ever a property of one object references another object (called a "foreign key"), the update and deletion of the foreign object can be specified as nullifying or cascading through to the property, or as being restricted by the property.

"Tasks" [8] are useful for standard operations: it is preferable to store the handling of data once with the data definition, than many times over in programs that use it, possibly inconsistently. A task can have parameters and local variables and can return a result. Tests can be interspersed with executable statements to verify conditions at that particular point. An exception-handling task specified with a test will be invoked whenever the test fails.

3. THE DATA MANIPULATION LANGUAGE

The HAL commands are GET, CREATE, DELETE, UPDATE and NULLIFY. The latter is a special form of UPDATE which allows individual values of a multivalued property to be nullified, leaving other values of that property intact. The user requires no knowledge whatsoever of how the data is stored; he need only know what properties of objects exist. Commands reference data by property names and deal with sets of objects rather than working on an object-by-object basis. So that programmers need not know the ordering of properties within an object, the property, not the object, is the unit of access. Any combination of properties in any order, can be referred to in one statement, and GET can retrieve properties of different kinds of object at the same time. Particular object occurrences to be affected by a command are selected by a predicate expressed in terms of any properties of the object, not only its "key" properties.

The option of file output is provided for extracting large numbers of objects from the database. Arithmetic operations and aggregate functions (average, minimum, maximum, sum, and count) can be applied to database values, and any desired sequencing of objects in a result can be obtained. The concise notation enables multivalued and compound properties, and properties of properties to be referenced easily. Consider the following statement for the HAL subschema in the appendix:

```
CREATE project(name="IBM"; jobhistory.worker.name, jobhistory.hours="Doe",
                    36, "Tod", 45, "Fig", 27)
```

Jobhistory is a compound , multivalued property of project which is here being given three values. Each of these comprises a pair of worker-name and hours values, respectively. A "mapping" [6] permits properties of properties to be designated simply: example "jobhistory.worker.name". Special forms of GET are provided to enable metadata to be retrieved. Examples:

```
GET(obj = OBJECT, del = DELETES, edit = UPDATES) ;
```

retrieves all object names with their deletion and update constraints;

```
GET(prop=PROPERTY, of=OBJECTOF, t=TYPE, c=COMPULSORY, u=UNIQUE,
     m=MULTIPLICITY);
```

obtains all property names and descriptions.

The commands are designed to be embedded within a C [7] host language program so their syntax is compatible with that of C. The major difference lies in the fact that C is record-oriented while HAL is set-oriented. This problem is overcome by treating a retrieval statement as a loop control. The (compound) statement following a GET is iterated once for every object retrieved. HAL commands could not be implemented as library routines because of problems with parameter passing for complex predicates. Instead, a preprocessor replaces HAL statements within a C program by appropriate commands manipulating the underlying database. It should be noted that the overhead of such a preprocessor pass through a program exists in most relational database systems.

4. IMPLEMENTATION

The metadata in the HAL subschema is stored in the following relations in the database:

RELATIONS -	this stores the relation and attributes in the underlying database as well as their data type
OBJECTS -	here there are tuples for each object type and every relation in which that object is referenced. If the object is a foreign key the restrict/cascade/nullify information is recorded along with the name of the foreign key attribute
PROPERTIES -	this gives the attribute or relation representing each property. If a relation, the property is termed "complex" (eg. jobhistory and workdone in the appendix.) The objecttype and id columns in PROPERTIES indicate foreign keys. In the appendix "worker" and "operation" are two such properties mapped to work.eno (specifying workers via their employee number) and work.jno

(designating a project number), respectively.

JOINS - this gives the attributes to match when joining one relation to another for a complex property.

The appendix illustrates the metadata created for a simple subschema.

The HAL preprocessor parses commands, checks them against the stored metadata and translates them into equivalent operations on the relational database. When a task is invoked the preprocessor extracts any condition associated with the call in the program, translates it into an equivalent predicate on the underlying database and appends it to all database manipulation commands in the task. The system is being tested on an Ingres database and hence HAL statements are translated to EQUEL [9]. A HAL command generally requires several EQUEL operations, since what appears to the programmer as a single object is usually represented by several relations. The preprocessor deduces that a join is required when it encounters a foreign or complex property.

Consider firstly the handling of a foreign key such as worker. Its PROPERTIES entry equates it with work.eno, and its "objecttype" and "id" indicate that it references employee objects by number. Employee number is found in PROPERTIES to be represented by emp.eno. Thus the join clause "WHERE work.eno = emp.eno" is used. For a complex property, PROPERTIES shows that it is represented by an entire relation. The JOINS tuple for that property is thus examined to obtain the join clause.

Handling CREATE, NULLIFY and UPDATE commands requires that properties be re-ordered before equivalent EQUEL statements are devised. In this way properties in one relation are treated, then those in the next relation, and so on. When the PROPERTIES entry indicates that a complex property is being inserted, the metadata for the corresponding relation is examined to determine its compulsory properties. Unfortunately, the foreign key value may be given under some other property name, eg:

```
CREATE project(number=102; name="NCR"; jobhistory.hours,
               jobhistory.worker.name=12, "Doe", 24, "Fig", 9 "Poe");
```

The complex property jobhistory is represented by the work relation. Its attribute work.jno represents "operation", a compulsory foreign key. The statement did indeed indicate a particular operation giving the project number 102, but has understandably not repeated it in the form jobhistory.operation.number. To deal with this, the missing property is determined from "id" in PROPERTIES (here, project number). This property value is therefore sought in the statement if the expected foreign key is not explicitly given.

Another difficulty arises from indirect identification of the foreign key, eg:

```
CREATE activity(worker.name, hours, operation.number="Doe", 12, 24, "fig",
                27, 18, "Poe", 33, 3);
```

Activity is represented by the work relation. Its worker property is a foreign key represented as employee numbers. However, the command above has identified employees by their name. The system detects that name and not number has been supplied, and first retrieves the employee number corresponding to the given name. The EQUEL statements to handle insertion of the first activity are thus:

```
RANGE OF empx IS emp
RETRIEVE (nvar[0] = empx.eno, flag = ANY(empx.eno where empx.ename = "Doe"))
      WHERE empx.ename = "Doe"
If (flag == 1) APPEND TO work (eno = nvar[0], hrs = 12, jno = 24)
```

The HAL system has to impose all subschema constraints. To enforce restricted updates and deletions, a RETRIEVE ANY is used to detect tuples preventing the desired operation. Additional EQUEL statements are created to propagate changes and deletions through the database. Finally, enforcing uniqueness of id-properties involves using ANY to detect duplicates.

5. EVALUATION

The major advantages of HAL over existing relational languages are that several underlying relations can be referenced in one statement as if the information were all stored together, and that objects can be uniquely distinguished via any identifier. There is no need to distinguish "attributes" from "associations" even though they may be implemented differently and the join operation of their relational algebra is replaced by the simple concept of a mapping. In addition HAL can restrict data removal and alteration where appropriate, detect duplicates or null values where they are not allowed, and propagate deletions and updates through the database. The metadata relations created by the system are accessible to programmers, enabling them to query the data definition. Figure 1 illustrates the relative simplicity of HAL compared with the conventional database languages: relational algebra[5], relational calculus[4], EQUEL[9] and SQL[1]. Query: Give the names of all employees who have worked on the "ICL" project (for the database in the appendix).
Phrasing the same query in HAL:

```
GET (result=jobhistory.worker.name) WHERE project.name == "ICL";
```

Phrasing the sample query in the relational algebra:

```
JOIN emp AND work OVER eno GIVING templ
JOIN templ AND job OVER jno GIVING temp2
SELECT temp2 WHERE jname = "ICL" GIVING temp3
PROJECT temp3 OVER ename GIVING result
```

Phrasing the sample query in the relational calculus:

```
RANGE wx work
RANGE jx job
GET result(emp.ename): ∃wx ∃jx(emp.eno = wx.eno & wx.jno = jx.jno &
                                jx.jname = "ICL")
```

Phrasing the sample query in EQUEL:

```
RANGE OF ex IS emp
RANGE OF jx IS job
RANGE OF wx IS work
RETRIEVE (result=ex.ename) WHERE ex.eno=wx.eno and wx.jno=jx.jno and
                                jx.jname = "ICL"
```

Phrasing the sample query in SQL:

```
SELECT ename
FROM emp
WHERE eno IN (SELECT eno
             FROM work
             WHERE jno IN (SELECT jno
                          FROM job
                          WHERE jname = "ICL"))
```

figure 1
Comparison of Five Relational Languages

HAL has demonstrated that a high-level interface to a relational database can provide complete data independence, thus enhancing ease of use and making programs immune to changes in the schema. The major problem remaining is that the data manipulation commands are embedded in a foreign host programming language. A persistent programming language based on HAL is accordingly being developed [2].

REFERENCES

1. Astrahan M.M. and Chamberlin D.D. [1975], Implementation of a Standard English Query Language, *Communications of the ACM*, **18**.
2. Berman S. [1986], PPL - A Preliminary Report, *Quæstiones Informaticæ*, 4, 25-32.
3. Codd E.F. [1970], A Relational System of Data for Large Shared Data Banks, *Communications of the ACM*,**13**.
4. Codd E.F. [1971], A Data Base Sublanguage Founded on Relational Calculus, *Proc . ACM SIGFIDET Workshop on Data Description, Access and Control.*
5. Codd E.F. [1972], Relational Completeness of Data Base Sublanguages, *Data Base Systems*, Courant Computer Science Symposia Series (6), Prentice-Hall.
6. Hammer M. and McLeod D. [1981], Database Description with SDM: A Semantic Database Model, *ACM Transactions on Database Systems*, **6**, 351-386.
7. Kernighan B.W. and Ritchie D.M. [1978], *The C Programming Language*, Prentice-Hall.
8. Mylopoulos J., Bernstein P.A. and Wong H.K.T. [1980], A Language Facility for Designing Database-Intensive Applications, *ACM Transactions on Database Systems*, **5**, 185-207.
9. NCR Corp. [1983], *TOWER Database Manager*, Relational Technology Inc.

APPENDIX - SIMPLE EXAMPLE

Employee-Work Database Relations

```
EMP(ename,eno,wage)
JOB(jname,jno,charge)
WORK(jno,eno,hrs)
```

Subschema for Employee-Work Database (The TRANSLATION section is transparent to application programmers)

```
OBJECT Employee
 ID-PROPERTY
 Name                            1 STRING compulsory
 Number                          1 INTEGER unique compulsory
PROPERTIES
 Wage                            1 REAL
 Workdone                        M Activity update cascades
OBJECT Project
 ID-PROPERTY
 Name                            1 STRING unique compulsory
 Number                          1 INTEGER unique
PROPERTIES
 Charge                          1 REAL
 Jobhistory                      M Activity update cascades
OBJECT Activity
 ID-PROPERTY
 Operation                       1 Project compulsory delete restricted update cascades
 Worker                          1 Employee delete nullifies update cascades
PROPERTIES
 Hours                           1 REAL

TRANSLATION
 OBJECT Employee
 Name                            EMP.ename
 Number                          EMP.eno
 Wage                            EMP.wage
 Workdone                        WORK
 OBJECT Project
 Name                            JOB.jname
 Number                          JOB.jno
 Charge                          JOB.charge
 Jobhistory                      WORK
```

```
OBJECT Activity
Operation                     WORK.jno = Project.Number
Worker                        WORK.eno = Employee.Number
Hours                         WORK.hrs
```

Metadata for above subschema

RELATIONS

relation	column	format
EMP	eno	int
EMP	ename	char
EMP	wage	int
JOB	jno	int
JOB	jname	char
JOB	charge	int
WORK	eno	int
WORK	jno	int
WORK	hrs	int

OBJECTS

objectname	relation	column	deletes	updates	prop
employee	emp				
employee	work	eno	nullify	cascade	number
project	job				
project	work	jno	restrict	cascade	number
activity	work				

PROPERTIES

propname	object	relation	column	objecttype	id	c	u	m
name	employee	emp	ename			Y	N	1
number	employee	emp	eno			Y	Y	1
wage	employee	emp	wage			N	N	1
workdone	employee	work		activity		N	N	M
name	project	job	jname			Y	Y	1
number	project	job	jno			N	Y	1
charge	project	job	charge			N	N	1
jobhistory	project	work		activity		N	N	M
worker	activity	work	eno	employee	number	N	N	1
ııııııııı	activity	work	jno	project	number	Y	N	1
hours	activity	work	hrs			N	N	1

(in the above relation the names "compulsory", "multiplicity" and "unique" have been abbreviated to "c", "m" and "u" respectively).

JOINS

property	relation1	column1	relation2	column2
jobhistory	job	jno	work	jno
workdone	emp	eno	work	eno

PROTECTION OF COMPUTERISED PRIVATE INFORMATION: A COMPARATIVE ANALYSIS

K.G. van der Poel
Department of Medical Informatics
Groote Schuur Hospital
Observatory 7925

I.R. Bryson
Graduate School of Business
University of Cape Town
Rondebosch 7700

The principles of protection of private information have gradually been defined during the last decades. The US Privacy Act of 1974 regulates practices of the federal government. Two committees of the UK government have prepared the Data Protection Act of 1984. Codes of practice play an important role in the Netherlands. In South Africa, data privacy is protected by the law of delict. However this provides a weak and reactive opportunity for redress. While the law commission is preparing to study the subject in depth, professional organizations should establish codes of practice.

1. INTRODUCTION

George Orwell foresaw a rather unpleasant society in 1984. In that society every aspect of human life was centrally controlled by a power that had access to and could manipulate information about all citizens at all time. Individual privacy and the right of a person to restrict and control information about himself were totally denied.

The year 1984 has passed. Orwell's predictions have not materialized, or at least not quite. The possibility that personal privacy could be eroded by unbridled use of technology has, however, been a concern to many people. Steps have therefore been taken to redress the balance and legislation has been passed in several countries. In the following, an overview is given of some of these measures and the principles on which they are based and conclusions are drawn for South Africa.

2. PRIVACY

Privacy is easy to understand, but hard to define precisely. McQuoid-Mason[1] quotes the following descriptive definition by Westin:

> "The essence of privacy is no more and certainly no less, than the freedom of the individual to pick and choose for himself the time and circumstances under which, and most importantly, the extent to which, his attitudes, beliefs, behaviour and opinions are to be shared with or withheld from others. The right to privacy is therefore a positive claim to a status of personnel dignity - a claim for freedom, if you will, but freedom of a very special kind."

The general concept of privacy as a civil right is well anchored in Western civilization. Safeguards against the invasion of privacy are found in Roman Law, Jewish Law and English Law. Concepts such as dignity, honour and human rights are closely related to it. John Locke wrote: "Everyman has a property in his own person. This nobody has a right to but himself." Judge Cooley defined it as "the right to be left alone". John Stuart Mills said: " There is a limit to the legitimate interference of a collective opinion with individual independence and to find that limit ... is indispensable to a good condition of human affairs".

Invasions or violations of privacy under modern legislation are divided into 4 categories: intrusions of the private sphere, public disclosure of private facts, placing a person in false light, and appropriation of another's name or likeness.

The second and third of these invasions of privacy have very much to do with practices of data collection, storage and dissemination. The defence against them is called data privacy. As technological development influences the need and the capacity for data privacy, it is to be expected that legislation must be continuously adjusted to make sure the balance of right and wrong is not upset. It may be worth emphasizing that concern for data privacy does not imply the intention to protect people who have something to hide such as crimes or trespasses or tax evasion. Its intention is to ensure that a proper balance is struck between the interests of the individual and those of the community or, for that matter, of the dominant groups in the community.

3. COMPUTERS AND PRIVATE INFORMATION

Private information is loosely defined as all information concerning an individual. It includes information which is generally considered to be in public domain, such as name and address. It also includes information which is more sensitive or confidential, such as data about diseases, religion, possessions and debts, tastes and preferences etc. The advent of computers has had a significant effect on the ability to collect, store and disseminate private information in such a way that data privacy may be affected.

Computers have had several negative effects on data privacy: firstly private information is no longer confined to the place and format in which it was collected. It can now be be transmitted, collected and retrieved with very little effort. Secondly private information has become much more accessible and access to the data may be totally unobserved. Thirdly, data may be corrupted without any indication that this has happened.

The use of computers can also have positive effects on data privacy. Proper controls make it possible to limit access to private information effectively. Advanced processing techniques make inspection and correction feasible. Appropriate security measures make corruption virtually impossible.

The conclusion is that computer technology is capable of either undermining or supporting data privacy in a big way. In the following, some examples will be considered of how different countries have dealt with this problem.

4. MOVES TOWARDS LEGISLATION

The desire to formally protect data privacy can be traced to an article by Warren and Brandeis in 1890.[2] For the first time they presented arguments to establish an explicit legal defence against eavesdropping, illegal search and seizure or publication of private facts. This was followed by the City of New York enacting the first modern statute to protect data privacy in 1902.

With the advent of computers, a number of countries reacted by drafting specific legislation to protect data privacy. A list of the nations that took such action before 1984 is shown in Table 1. It is interesting to note that this list closely resembles the list of the more developed nations.

The first specific legal protection of computerised data was through the enactment of the Land of Hessen Data Protection Act of 1970. This act applied to private information held by local and public bodies within the jurisdiction of this German province (Land). The purpose of the act was to:

> "lay down penalties for the examination, alteration, extraction and destruction of data by unauthorized persons, and provide for the correction of inaccurate data and for the appointment of a data protection commissioner to oversee the handling of information provided by individuals and the consideration of complaints."[3]

The first national act was the Swedish Data Act of 1973. This is hardly surprising as Sweden has a long history of legislature which gives citizens access to any information assembled by the state.

The Swedish Data Act was followed by the American Privacy Act of 1974 and subsequently by many other national data protection acts. These regulations vary greatly in extent and nature. For example Canada regulates only the government and the public activities, while most other nations regulate both the public and private sector data processing societies. Denmark even has separate acts for private and public data holders. Common to all, though , are regulations governing the right of access by an individual to his records and certain controls on the establishment and operation of databases for private information. The regulations generally apply to all personal information with specific exemptions in respect of national security, the administration of justice and other critical activities e.g. health care. In some cases insignificant little data bases are exempted. In other cases certain types of sensitive data (e.g. religion, political affiliation) are subject to extra controls.

In 1984 the Data Privacy Act was promulgated in the United Kingdom. This was the result of extensive deliberations of 2 committees and will be reviewed below.

Legislation has been under consideration in the Netherlands for the last 10 years. The situation in this country will all be reviewed below.

Australia	P	L	R
Austria		L	
Belgium	P		R
Brazil	P		
Canada		L	R
Columbia	P		
Denmark		L	
Finland	P		R
France		L	
Germany	P	L	R
Greece			R
Hungary		L	
Iceland		L	
Ireland			R
Israel		L	
Italy	P		R
Japan			R
Luxembourg		L	
Netherlands	P		
New Zealand		L	
Norway		L	R
Portugal	P		
Spain	P		
Sweden		L	R
Switzerland	P	L	R
Turkey			R
U K		L	R
U S A	P	L	R
Yugosalvia			R

Key: L = law adopted P = legislation proposed R = government report

Source: Transnational Data and Communication Report January 1986

table 1
Status of data protection legislation - 1986

5. THE UNITED STATES PRIVACY ACT OF 1974

There has been a groundswell of support for limitations to the secrecy of government information in the United States. Increasing use of secretive operations and classification of papers after the second world war drew sharp criticism. "If government is to be truly of, by and for the people, the people must know in detail the activities of government."[4]

This movement led to the promulgation of the Freedom of Information Act in 1966. This gave all persons the judicially enforcable right to see the records of federal government agencies, except to the extent that the records may be covered by an exemption. However, it was soon found that this act was far from ideal as government agencies were only obliged to show final documents, not the supporting details and could charge exorbitant fees for the search effort.

Against this background, the Privacy Act of 1974 was enacted. The purpose of the act was clearly stated in its preamble: "To safeguard individual privacy from the misuse of the federal records and to provide that individuals be granted access to records concerning them which are maintained by federal agencies."[5]

The act was deliberately restricted to federal agencies because of the complexity of the statute that would have been required if private agencies were similarly regulated.

Some of the salient provisions of the law follow.

- The retention of personal data is limited: "each agency shall maintain in its records only such information about an individual as is relevant and necessary to accomplish the purpose of the agency".
- Procedures for the collection of data prescribed: "agencies should collect information to the greatest extent practicable directly from the individual".
- The right of notification required the agency to publish in the Federal Register the existence and character of the system of records - including name and location of the system and nature of the files maintained.
- The right of access of the subject is arranged: "Each agency shall upon request by any individual ... permit him to review the record and have a copy made of all or any portion thereof in a form comprehensible to him."
- Concern for the accuracy of data is required: "an agency shall maintain all records about an individual with such accuracy, relevance, timeliness and completeness as is reasonably necessary to assure fairness to the individual".

Regulations governing the dissemination to other agencies and non-government users are comprehensively specified in the act. In most cases the subject must give written permission before the record can be disclosed to other users, while agencies must keep accurate accounts of the nature and purpose of a disclosure.

The act specifies that non-observance may lead to both civil and criminal charges. It does not go as far as establishing a special data protection authority.

Discussion

The act has been in force for over 10 years. Its main benefit appears to be that it has served as a standard of good data processing practice. A commission charged with recommending to congress any changes to the act reported in 1977. Its recommendations included the establishment of a data privacy board and many technical corrections. They did not recommend extension of act's coverage to the private sector.[6] In fact, the act has not been altered significantly since these recommendations.

6. THE UK DATA PROTECTION ACT OF 1984

The Data Protection Act is a result of over twelve years of parliamentary effort, based on the work of two major committees of enquiry. These were the Committee on Privacy (Younger Committee) reporting in 1972 and the Data Protection Committee (Lindop Committee) reporting in 1978.

The Younger Committee examined the entire subject of personal privacy. It considered "the computer problem as it affects privacy in Great Britain to be one of apprehensions and fears and not so far one of fact and figures". As a result, the committee wanted to keep the dangers of privacy arising from computers in perspective and did not propose any specific legislation. However, the committee did formulate some important principles for the proper handling of private information. These are known as the Younger Committee principles. They are shown in table 2.

1. Information should be regarded as held for a specific purpose and should not be used, without appropriate authorization, for others.
2. Access to information should be confined to those authorized to have it for the purpose for which it was supplied.
3. The amount of information collected and held should be the minimum necessary for the achievement of the specified purpose.
4. In computerised systems handling information for statistical purposes, adequate provision should be made in their design and programs for separating the identities from the rest of the data.
5. There should be arrangements whereby the subject could be told about the information concerning him.
6. The level of security to be achieved by a system should be specified in advance by the user and should include precautions against the deliberate abuse or misuse of information.
7. A monitoring system should be provided to facilitate the detection of any violation of the security system
8. In the design of information systems, periods should be specified beyond which information should not be retained.
9. Data held should be accurate. There should be machinery for the correction of inaccuracy and he updating of information.
10. Care should be taken in coding value judgements.

table 2
The Younger Committee principles for handling personal information by computer [7]

The Lindop Committee was established in 1975, as a result of a recommendation of the Younger Committee, to advise the government about legislation to protect personal data handled in computerised systems. The committee recommended seven principles as the basis for legislation. (see table 3). It further recommended the institution of a Data Protection Authority, establishment of codes of practice and enactment of a law control computerised private information. These recommendations led to the Data Protection Act of 1984.

1. Data subjects should know what personal data is held, why and how it will be used, by whom and for what purpose, and for how long.
2. Personal data should only be used for purposes made known when it is collected.
3. Personal data should be accurate, complete, relevant and timely.
4. The minimum amount of data should be used.
5. Data subjects should be able to verify compliance.
6. Users of data should be able to do so for their lawful interests without undue costs or use of their resources.
7. The community at large should enjoy any benefits and be protected from any predjudice, which may flow from the handling of personal data.

table 3
The Lindop Committee principles for data protection legislation

The purpose of the Data Protecton Act is: "to regulate the use of automatically processed information relating to individuals and the provision of services in respect of such information".[8]

The office of a Data Protection Registrar is created, whose duty it is to promote the observance of the data protection principles by data users and persons carrying on computer bureaux.

To be legal, all personal data files must be registered with the office of the Registrar, giving particulars about purpose and content of the file.

The Act stipulates that personal data shall be obtained fairly and lawfully and that the purpose for which data is collected should be clear to the data subject.

Codes of practice, although recommended by the Lindop committee, are not prescribed in the law, but it is left to the registrar to encourage such arrangements.

The right of access of data subjects is guaranteed, in that individuals are entitled: "to be informed by any data user whether the data held by him include personal data of which that individual is the subject, and to be supplied by any data user with a copy of the information constituting any such personal data held by him". The data user may request a fee for such access, up to a prescribed maximum.

Although the subject can challenge the accuracy of the record, he is limited in his ability to force the data user to amend the record or to claim damages for the inaccuracy.

The Act contains virtually no regulations concerning the dissemination of data. As a consequence, unhindered transfer of data between registered users may occur, provided both parties are registered and legitimate users of such data.

Discussion

In accordance with the title of the Act, great emphasis has been placed on the control of information (protection) rather than the type of information (privacy). A positive aspect of the Act is certainly the fact that both the government and private systems are covered. A major problem is that it applies to all computerised files, regardless of the number of records or their content. This is likely to lead to considerable expense for the large number of small users. An aspect which is often queried is the omission of codes of practice. These could have been used to regulate certain types of files more precisely.

7. PROPOSED LEGISLATION IN THE NETHERLANDS

The move towards specific legislation to protect data privacy started in the Netherlands in 1972. One of the reasons was the popular resistance to the census of 1971. The move resulted in an elaborate proposal of law, 23 pages long, introduced in 1981. After long deliberation this proposal was withdrawn and replaced by a simpler proposal, consisting of 12 pages.[9] The present proposal is still far reaching in that it covers computerised as well as manual, and governmental as well as private files containing personal information. It requires registration of all such files and contains the normal safeguards of protection against unauthorized use and access by the data subject.

An interesting feature of the proposal is that special provision is made for codes of practice and privacy regulations to be promulgated by specific sectors. Voluntary regulations have, in fact already been in force in several sectors such as municipalities, health care, etc. The understanding is that the new law will mainly serve as a framework for these regulations.[10]

8. INTERNATIONAL REGULATIONS

Several international agencies have involved themselves with aspects of the protection of information. The purpose was to promote the rights of the individual in the member states and to avoid unfair competition. Such unfair competition could occur if member states adopted widely different practices, but also if the rights of the member state citizens could be circumvented by transferring data abroad to "countries of convenience".

In 1980 the Organization for Economic Cooperation and Development (OECD) issued "guidelines governing the protection of privacy and transborder flows of personal data" thus standardizing to some extent the principles of personal information handling in its 24 member states. An important stipulation is the provision that the transfer of personal data to countries not observing adequate privacy protection may be limited. The guidelines of the OECD are, however, not binding to member states.

The Council of Europe accepted a convention of rules concerning automated processing of private information in 1981. These rules follow the stipulations of the OECD very closely. They are expected to become binding on the member states in 1986.[11]

9. LEGAL PROTECTION IN SOUTH AFRICA

There is presently no specific law in South Africa concerning private information. For the protection of data privacy, the individual must therefore rely on the general principles of Roman Dutch law. The basis for any action lies in the law of delict. The law of delict provides for compensation for unlawfully inflicted injury to a person.[12] Invasions of data privacy can conceivably be actioned under the Actio Injuriarum or the Law of Defamation.

The Actio Injuriarum redresses wrongs to interests of personality generally. Its scope has been described: "to embrace all willful invasions of rights of another, which every man has as a matter of natural right in respect of his person, his dignity and his reputation".[13] However, it is generally considered that the stringent requirements of this Actio are unlikely to be met by a data privacy case.

The Law of Defamation protects the reputation. The delict of defamation is described as: " the unlawful publication animo injuriandi (with intent to injure) of a statement concerning another person which has the effect of injuring that person in his reputation".[15] There are various defences against this action, notably those of justification, qualified privilege and willing consent. These defences make it unlikely that this action will be applicable except in the most blatant situations.

It would therefore appear that the currently available protection of data privacy in South Africa is very limited. An action would only be successful after gross abuse. It is also a reactive form of redress. The specific legislation adopted in other countries has the merit that it is largely pro-active i.e. enforces and controls certain standards of behaviour.

10. THE ROAD AHEAD

There appears to be a disparity between the protection of data privacy under existing South African law and under the new legislation and rules adopted in most developed countries. There is no obvious reason for this disparity to persist: the sophistication of private information handling is at least at a par and the people are no less keen on their civil liberties. Already, the SA law commission has recognised the discrepancy and has identified the area of data privacy as requiring investigation as part of project 44: "A comprehensive and comparative enquiry into the protection of all rights of personality". However, other pressing needs for legislation may keep this project from being undertaken for some time.

The remaining option is that interested groups and parties get together in the short term to consider voluntary codes and regulations. Such regulations could set standards for responsible data processing behaviour. If these regulations would be adopted by professional societies such as the Computer Society, the Medical and Dental Association and the Institute of Chartered Accountants, then meaningful protection would be provided in the vast majority of situations. General acceptance of such regulations might even forestall the need for legislation. At present a draft code of conduct is being studied by the Computer Society of South Africa. Adoption, promulgation and further discussion of this code will set an example, likely to be followed by others.

REFERENCES

1. McQuoid-Mason, DJ (1978). *The law of privacy in South Africa..,* Juta & Co. Cape Town.
2. Warren, S and Brandeis, L (1890). The right to privacy. *Harward Law Review* , 4 p 193.
3. Home Secretary (1975). *Computers and Privacy.* HMSO. London p 313.
4. Michigan Law Review (1975). Government information and the rights of citizens. *Michigan Law Review,* May-Jun 1975.
5. U S Congress (1974). Privacy Act 1974. Washington. *United States Government code 5 Section 552 (a).*
6. American Enterprise Institute (1979). *Privacy Protection Proposals.* Washington.
7. Younger, K (1972). *Report of the committee on privacy.* HMSO. London.

8. Data Protection Act (1984). HMSO. London.
9. Minister van Justitie (1982). *Wet persoonsregistraties.* Staatsdrukkerij Den Haag.
10. Holvast J (1986). Wet persoons registraties. *Informatie* **28** , May 1986.
11. Kuitenbrouwer, F (1984). *Privacy en persoonsregistratie: een overzicht..* Kluwer. Deventer.
12. Boberg PQR (1984). *Law of delict.* Juta & Co. Cape Town.
13. De Villiers JA (1922). Case: Matthews v Young AD 492.
14. Price E (1986). No place to hide. *Financial Mail* 28 March 1986 p 93.
15. Kinghorn, C (1979) *Defamation law of South Africa* , 7. Butterworth & Co. Cape Town.
16. McQuoid-Mason, DJ (1978). *The law of privacy in South Africa.* Juta & Co. Cape Town p198.

BOOK REVIEW

An Introduction to LISP, A. Narayanan and N. E. Sharkey, Ellis Horwood, 1985. ISBN 0–85312–968–1. 227 pages.

Reviewed by: Philip Machanick, *University of the Witwatersrand, Johannesburg 2001*

The growth of Artificial Intelligence and the proliferation of low-cost implementations of AI languages has created a demand for books offering a less formal treatment than that of established text books. Such books need to present carefully chosen subsets of a complex language—and a complex application area. Furthermore, they need to explain difficult concepts to the uninitiated in a comprehensible way.

Narayanan and Sharkey have succeeded in some respects in meeting these needs. Their choice of examples—based largely on describing parts of robots—has more appeal than the all-too-common approach of introducing arbitrary names for variables and functions—such as FOO and BAR. Furthermore, they have not shied away from introducing recursion before other forms of iteration, nor from using an almost purely functional style in the opening chapters. The sections on semantic nets and the blocks world are ambitious for a beginner's text and well executed, if a bit brief for the intended readership.

However, the book is marred by flaws in the authors' understanding of several key concepts. Examples include the stack mechanism, lexical scoping and reader macros. While most of these problem areas will not impact the beginner's conceptual grasp of LISP programming, they are a disappointment. An anomoly for a non-technical treatment is the amount of space devoted to describing the cons-cell representation of lists. Of more concern is the lack of consistency in the choice of dialects for examples. DEFUN is mostly used to define functions, while DF and DM are introduced to define FEXPRS and MACROS. A substantial example towards the end of the book is clearly not in the same dialect as the other examples.

Perhaps if Common LISP does become a widely accepted standard (even on "small machines"—some of which at least have enough memory), the difficulties relating to dialects will be overcome. At least appendices giving details of several dialects are provided.

Despite some well-executed diagrams, the casual reader is more likely to be put off by mediocre typesetting than by innaccuracies of detail. A further negative point is the over-use of "cute" names (Ann Droid the robot) and subheadings (The Pros of CONDs). But perhaps the sentence on page 141 makes it all worthwhile:

Artificial Intelligence is the study of metal faculties ...

MODELS TO EVALUATE THE STATE OF COMPUTER FACILITIES AT SOUTH AFRICAN UNIVERSITIES

P.J.S. Bruwer
Post-graduate School for Management
PU for CHE, Potchefstroom

J.M. Hattingh
Operations and Information Systems
PU for CHE, Potchefstroom

Computer technology has grown exponentially in the course of the past decade in terms of sophistication - not only with regard to hardware and networks, but also with regard to mainframe and micro-computer systems. A university environment has unique needs in terms of computer facilities. Here it is not only a case of transaction processing systems with the concomitant higher levels of making available information for purposes of management, but primary activities at a university which have to be supported are research and teaching. In this research project an attempt was made to determine where the biggest problems lay with regard to adequate computer facilities at South African universities, and to propose possible solutions.

1. INTRODUCTION

The variety of activities which have to be supported by the computer services department at universities makes it very difficult for some universities to provide for these needs with their available capital. Training in computer science is, for example, a component where provision should be made for a minimum level of facilities.

Research at any university is a matter of high priority and special equipment and program material are essential in many cases. Administratively a specific level of availabilty and turn-around time are expected of the computer, so that it becomes a complex management task to determine priorities and to maintain an acceptable level of service.

With this research project it was decided to make a survey of the following by means of questionnaires:

- the status of computer support for research, teaching and administration at the universities;
- the level of satisfaction with computer facilities and the factors determining this (among others the availabilty of resources), with a view to make some recommendations.

The purpose of the project therefore was to develop a model which will bring the total success rate of computer services at a university in relation with the factors influencing it. It was felt that apart from the important role played by the availabilty of resources, there are other internal factors which can influence the success. An example is to be found in an orderly POLICY centring on the development of computer facilities, and other such factors may exist.

As it cannot be expected of the State to finance internal incompetence by the universities, it was decided to try and determine how the maximum level of success can be attained for any given level of availability of resources.

2. COLLECTION OF DATA

A questionnaire of reasonably wide scope in which 70 aspects are covered in three sections was developed. In Section A in the questionnaire we concentrated on questions around computer support of teaching and research. In the second place the questions dealt with administrative support. All the questions in this section had to be evaluate to on a seven-point scale. This type of question has been used with great success in past research projects.[1,2,3,4] The questions in section B dealt with general information related to computer services at the universities. These were factual questions involving the number of personnel at a university, the number of students, etc. The final section contained composite evaluations which three different individuals had to complete questionaires independently of each other. Typically the three persons could be the Director of Computer Services, the Chairman of the Computer Board of Control, and the Deputy Principal to whom the department of Computer Services reports. The questionnaire was sent to all South African Universities.

3. PROCESSING OF DATA

It took a fair amount of time to get responses to the questionnaires. Eleven of the eighteen responded and much later two more were returned so that a total of 13 out of 18 were received. The 13 questionnaires were not always completed fully. The data that was provided was keyed in on the IBM 4381 of the PU for CHE and by means of BMDP programming[5] the average response to each question was determined. The standard deviation for each response was also determined.

Table 1 provides a few of the interesting findings with regard to the first section. The average provided in the table represents an evaluation on a 7-point scale.

Aspect	Average	Standard Deviation
Availability of computer strength for academic teaching	4,4	1,6
Availability of disk space	4,8	1,7
Satisfaction with application programs such as mathematical and statistical programming	4,9	1,4
Satisfaction with personnel involved with regard to quality and number	3,5	1,9
Availability of terminal equipment such as terminals and printers for academic teaching	3,6	1,7
Facilities for computer-assisted teaching	2,4	2,0
Availabilty of data base facilities for academic teaching	3,5	1,8
Availabilty of high level inquiry languages	4,2	1,7
Availabilty of information centre facilities and training facilities for academic teaching	3,4	1,4
Computer-assisted design and manufacture (CAD/CAM)	1,9	1,0
Image processing facilities	1,5	0,9
Office automation for senior academic personnel	2,8	1,4
Availability of word-processing and publication facilities for academic support	4,0	1,9
Computer strength for administrative support	5,0	1,4
Disc space for administrative support	5,3	1,2
Management systems programming for administrative support	5,7	1,0
Utility programming (translators,sorting programs, ets.)	5,8	0,8
Measure of integration of administrative computer systems	4,3	1,4
Financial systems	4,6	1,3
Student record systems	5,4	1,4
Personnel systems	4,3	1,9
Systems of service departments	3,8	1,5
Information systems for Deans and Heads of departments	3,5	1,9
Systems for management decision-making	3,5	1,6
Systems which should render SANSO information	5,0	1,3
Systems for general office automation	2,4	1,9
Use of fourth-generation languages	4,5	1,8

table 1
Computer Support of Teaching, Research and Administration

General Information on Computer Services

From the responses to Section A of the questionnaire it emerged that an average of 14 300 students had been enrolled per university. This number was somewhat inflated by UNISA's high numbers. (Without UNISA's students taken into account this average drops to approximately 9000). More meaningful is perhaps the average number of personnel (academic 581 and administrative 521)

There is, furthermore, an average of 87 separate micros per university while an average of 140 terminals had been linked to the mainframe computers. Intelligent work stations linked to the mainframes came to an average of 31.

The average expected increase in computer work by consumers for the next five years is 5,8, that is, 5,8 times more than at present. The leasing and management expenses have increased by 22% between 1984 and 1985, while personnel expenditures increased by 20%. Less capital, R374 000 in 1985 as against R484 000, in 1984 was spent. The total subsidy of the state only increased by 11,4% from 1984 to 1985.

4. DEVELOPMENT OF THE MODEL

The last few questions in each questionnaire were questions aimed at a composite evaluation by each university. In this part of the questionnaire three copies were provided so that three persons at each university could provide views of evaluation. The purpose of this was dual. The first was to base the evaluation not simply on the view of one person. The second was to obtain more data so that the model could be rendered more reliable.

It was decided to develop a model through the use of restricted regression methods. In Bruwer and Hattingh (1985) the methods and the philosophy are outlined in detail. For the sake of clarity the steps involved are outlined below.

STEP 1. Identification of Variables

It was decided to study the total success of Computer Services at a University (dependent variable).

The variable was evaluated in a question in the questionnaire and we call this SUCCESS (for the sake of brevity). Apart from this it was accepted that (for the purposes of this study) the following variables (independent variables) can have a significant influence on SUCCESS:

- The satisfaction with the way in which budget perspectives are provided (indicated as SATISFIED).
- The availability of resources at a University (indicated as RESOURCES).
- The ability of a university to recruit and to keep expert personnel (indicated as EXPERT).
- An orderly policy centring on deployment of computer facilities (indicated as POLICY).
  The extent to which there is success in obtaining the support of key persons at a University (indicated as SUPPORT).
- The extent to which a university succeeds in transmitting computer technology to the community by providing, amongst others, consumer training (indicated as CONSUMER TRAINING).

By mean of regression methods it was discovered that a linear regression equation can be found which offers a good application by linking SUCCESS with the following variables:

- SATISFIED
- RESOURCES
- POLICY
- SUPPORT

The data (averages where possible) are indicated in Table 2.

CASE	SUCCESS	SATISFIED	RESOURCES	POLICY	SUPPORT
1	5,0	3,0	2,7	5,3	4,7
2	4,3	5,0	3,7	4,7	5,0
3	7,0	1,0	6,5	6,0	7,0
4	6,0	5,0	6,7	6,3	5,3
5	6,0	6,0	5,5	6,5	5,5
6	5,5	4,0	4,0	4,5	6,0
7	4,3	4,0	3,3	5,7	5,0
8	5,3	1,0	4,0	3,3	5,0
9	4,0	4,0	3,7	4,3	4,7
10	4,5	4,0	3,0	3,0	4,0
11	4,0	3,0	5,0	2,3	4,0
12	5,0	3,0	2,0	4,0	4,0
13	5,0	4,0	5,0	3,7	4,7

table 2
Average Value of Variables

The obtained regression equation is

```
SUCCESS = 2,19 - 0,19 (SATISFIED) + 0,19 (RESOURCES) + 0,26 (POLICY) + 0,13 (SUPPORT)
```

The inclusion of SATISFIED in the regression model was not really significant, but was included for the purposes of this analysis.

STEP 2. Restricted Regression Model

At the various levels of the variable RESOURCES, minima and maxima of SUCCESS were determined and outlined by means of linear programming techniques[2] as shown in figure 1.

The interpretation of this figure for this specific case is the following: At each given level of RESOURCES indicated on the horizontal axis both the minimum and the maximum values of SUCCESS can be read off, as well as the levels of the other variables which give the mentioned maxima and minima. These minima and maxima are those which were realised in the information.

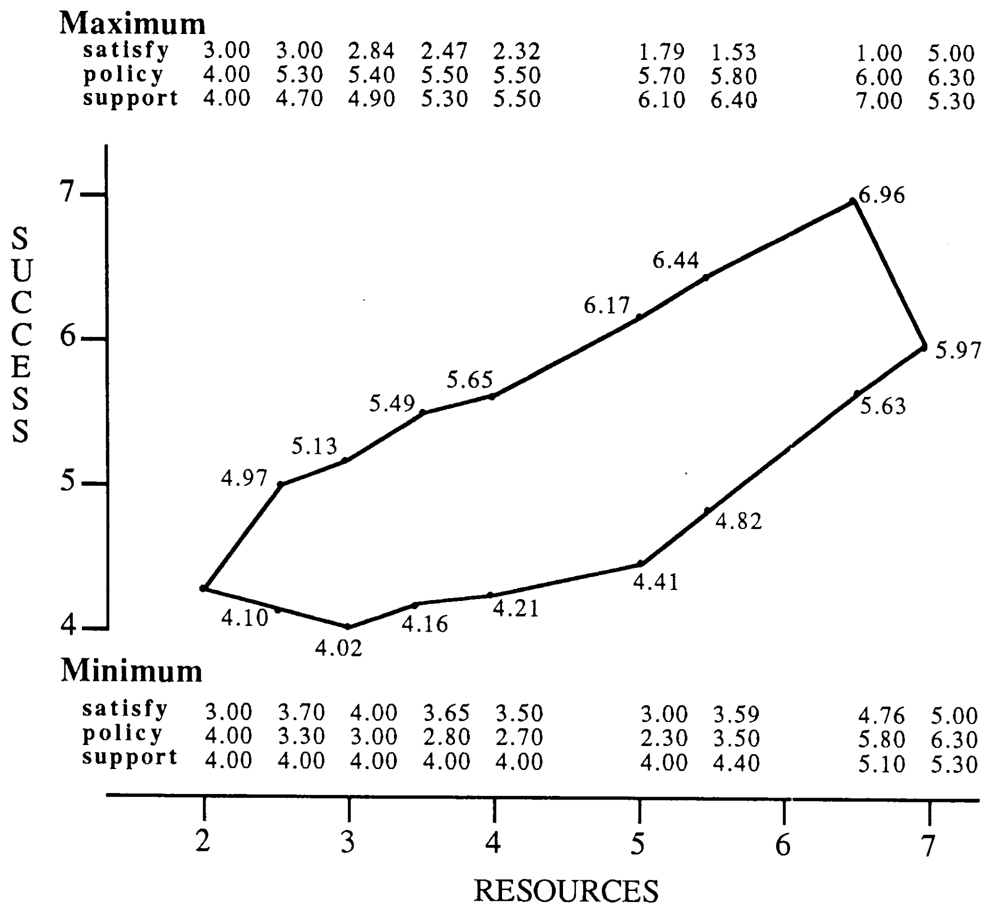
Example:

Suppose the level of resources at an university is found to be 4 (on the 7-point scale). Suppose furthermore the level of success is 4.21 (fig. 1). This level can be raised to 5.65 by changing the levels of the decision variable accordingly i.e. SATISFY, POLICY and SUPPORT from the minimum levels (3.50, 2.70 and 4.00) to the maximum levels (2.32, 5.50 and 5.50).

5. DISCUSSION OF THE RESULTS

It is clear from the figure that:

- In general more SUCCESS can be attained for higer levels of RESOURCES.
- The universities (those involved in the analysis) generally attain average or above-average success (4 or higher on the 7-point scale).
- Some universities do significantly better than others (for a given level of RESOURCES) because they have a healthy POLICY and enjoy good SUPPORT).



Maximum

satisfy	3.00	3.00	2.84	2.47	2.32	1.79	1.53	1.00	5.00
policy	4.00	5.30	5.40	5.50	5.50	5.70	5.80	6.00	6.30
support	4.00	4.70	4.90	5.30	5.50	6.10	6.40	7.00	5.30

Minimum

satisfy	3.00	3.70	4.00	3.65	3.50	3.00	3.59	4.76	5.00
policy	4.00	3.30	3.00	2.80	2.70	2.30	3.50	5.80	6.30
support	4.00	4.00	4.00	4.00	4.00	4.00	4.40	5.10	5.30

figure 1

Maximum and Minimum Values for Dependent Variable Success and Decision Variables Satisfy, Policy and Support

Some universities have indicated how much they intend to spend on computer facilities (1985) and when this is expressed in terms of Rand/student it amounts to R212,00. The Rand/student expenditure which the State contributed in terms of subsidy was R166,00 for this group. It is clear that this group of universities (if their own contribution of about 20% is taken into account) spends practically exactly what is provided for in the norms.

From Figure 1 it then emerges clearly that (as far as their own perception of the matter is concerned) in general they achieve a fair amount of SUCCESS in the cases where internal POLICY and SUPPORT are sound.

The dramatic decrease in the value of our currency with regard to other monetary units will dramatically change the perception about the availability of resources at universities for computer facilities, because our dependence on computer technology overseas countries.

6. RECOMMENDATIONS AND CONCLUSIONS

With regard to the above findings the following recommendations can be made:

6.1 That with the buying power of the Rand applicable at the end of 1984 reasonable levels of service could be maintained, especially by the universities where an orderly policy with regard to computer deployment is in operation and where Computer Services receive the support of key persons.

6.2 That the Committee of University Principals should make representations to the state to increase the average provision of funds for Computer Services by at least 50% in the subsidy formula in order to effect reasonable levels of service at universities in view of our weak currency.

6.3 That individual universities will use the results of the model above as an internal management instrument to ensure maximal utilisation of resources.

Many further analyses are possible of the data obtained. Somė questions, however, were not responded to fully enough and not all universities co-operated.

It is also clear that most universities are still in their infancy when it comes to the most important aspects of computerization. It is curious to note that this has not, in many cases, had a negative influence on the perception of the success obtained. This indicates that universities often resign themselves to mediocre situations as a result of conditioning. It would be possible for an appointed group to look individually at each situation and to evaluate its SUCCESS more objectively.

REFERENCES

1. BRUWER, P.J.S. 1983. Evaluating the performance of Computer_based Information Systems using a Restricted Linear Regression Model., *Quaestiones Informatica*, 2(3), 1-6, September.
2. BRUWER, P.J.S. 1983. Faktore wat die werkverrigting van rekenaargebaseerde inligtingstelsels beïnvloed. *S.-Afr. Tydskr. Bedryfsl.*, 14(1), 6-10.
3. BRUWER, P.J.S. 1984. A descriptive Model of Success for Computer-based Information Systems. *Information and Management*. 7(2), 63-67, April.
4. BRUWER, P.J.S & HATTINGH, J.M. 1985. Constrainted Regression Models for Optimization and Forecasting. *Orion*, 1(1), 2-15.
5. DIXON, W.T. & BROWN, M.B. 1981. BMDP-81 Biochemical Computer Programs, p-series, Berkley, Los Angeles: University of California Press.

LOW-COST ARTIFICIAL INTELLIGENCE RESEARCH TOOLS

Philip Machanick
Computer Science Department
University of the Witwatersrand, Johannesburg

There has been a proliferation of low-cost AI tools. Consideration is given as to how significant research may be conducted using these tools, in the light of experience of developing AI tools on large, expensive machines. As a case study, research in Knowledge Engineering-Based Learning (KEBL) is detailed, with mention of the tools used in the project. Results of the KEBL research are not yet available, but the style of the research is an indication of the potential of the tools described.

CR categories: I.2.5 Artificial Intelligence Programming Languages and Software, J.3 Computer Applications: Life and Medical Sciences

1. INTRODUCTION

As interest in Artificial Intelligence (AI) has spread, versions of languages and other AI tools have become available on relatively cheap computers. Such tools raise the possibility for AI research related to needs of relatively poor parts of the world.

The background to the development of cheap AI tools is considered, in with consideration of the lessons of traditional expensive AI tools. Brief consideration is given to how the experience of Turbo Pascal relates to the process by which AI tools can be expected to develop.

With the present state of available tools, consideration is given to whether significant research is feasible. The case for tools for practical applications is not an issue here; the number of potential tools for that market is also growing at an impressive rate [18]. Although there are many flaws in lower-cost tools, potential candidates for interesting developments are noted.

Research in Knowledge Engineering-Based Learning (KEBL) is used as an example of how relatively cheap tools can be a basis for AI-style program development. So far, the tools described — despite some major flaws — have proved adequate to the task. The problems detailed (and the occasional spectacular operating system crash) are indications that the technology is not as mature as one would like it to be, however.

In conclusion, experience with the tools used for KEBL — ExperLISP and ExperOPS5 — is summed up, with some consideration of how XLISP addresses some of the deficiencies of more conventional LISP implementations.

2. BACKGROUND

Most LISP books aimed at the low-cost computer market [10, 21] deal with primitive versions of the language, on the assumption that "big-machine" versions are not accessible to their readers.

Dialects of Common LISP [22] such as GC LISP (IBM PC — not too cheap) [7], ExperLISP (Apple Macintosh — moderately cheap) and XLISP (IBM PC and Macintosh — public-domain) [4] are challenging this assumption. Current exchange rates make the required computers expensive in South Africa, but US prices are declining.

A recent announcement in the personal computer field by Atari has a megabyte of RAM at under $1000 (albeit excluding a monitor) and is sure to put pressure on other manufacturers to drop prices, even if it does not turn out to be a major success. The significance of this development can be put into context by considering that the Interlisp environment was possible to develop because a large amount of memory was available by the standards of its time: 256K of 36 bit words of *virtual* storage [20]. If such a set of tools of acknowledged superiority could be developed with what now seems meagre resources, what will transpire now that so much computing power is so freely available?

It may be argued that Interlisp was put together by top-ranking researchers, and mass-market software will not achieve a similar level of sophistication. However, the lessons of the Interlisp experience are available for future developers of tools. Another positive factor is that a language such as LISP lends itself to incremental refinement [19]. Powerful tools — even in Interlisp — developed from humble beginings, with refinements developing as the need for them became obvious. Such an approach to development, while carrying the risk of uncontrolled proliferation by the undisciplined, is well-suited to mass-market tools, as the initial investment does not need to be high. An example of this is the development of tools around Turbo Pascal. Although Pascal is not seen as an AI language, Turbo has a significant feature in common with AI tools: an evironment which allows rapid recompilation after minor incremental changes.

The major difference between Turbo Pascal and AI tools comes in where large projects are being attempted: a good AI environment allows incremental recompilation, and has powerful tools to reduce the need for the programmer to keep track of bookkeeping details [20].

The extent to which the experience of Turbo Pascal can be applied to the development of cheap AI tools will be interesting to observe.

As an example of what can be achieved by such tools, a learning environment for Knowledge Engineering-Based Learning was implemented in ExperOPS5 — a version of OPS5 [6] implemented in ExperLISP. Despite the early stage of development of these tools, the implementation of OPS5 was viable for constructing a prototype of the required environment [15]. The final version was implemented directly in LISP to improve performance.

3. FEASIBILITY OF SIGNIFICANT RESEARCH

Most of the AI tools available in the past for small computers were primitive versions of languages like LISP [5] — or "expert system shells" with relatively few of the features of the systems used in major research projects [18].

Apple have announced a 1Mbyte version of the Macintosh — the MacPlus, which should be capable of supporting sophisticated tools. Many of the Macintosh's features — overlapping windows, mouse, powerful graphics — mimic features of sophisticated AI workstations [1, 20]. However, earlier versions were handicapped by having too little memory and disk drives which were too small and too slow. Since the Macintosh can be expanded to 4Mbyte without a major redesign, it has considerable potential for future development.

In the IBM range, the potential for expansion beyond the limitations of MS-DOS opened up by the PC AT are also being exploited: at least one full version of Smalltalk-80 [9] is available for this machine. Though a PC AT with all the required options is not cheap, it costs far less than a traditional AI machine — such as a Symbolics 3600 or LISP Machine.

Taking these points into account, there is not a lack of suitable hardware for the development of low-cost AI environments; suitable software is the missing ingredient. At one end of the spectrum, software is available, but expensive. The first version of Smalltalk-80 implemented for the PC-AT, at $1000, is not as expensive as many less exciting tools, but is not particularly cheap. Nor is Nexpert [8], one of the more interesting-looking tools for the Macintosh, with a launch price of $5 000. These prices should be seen in contrast with prices like $60 000 for KEE (Knowledge Engineering Environment) [13], which runs on a LISP Machine, and ExperLISP for the Macintosh at $495.

Although ExperLISP has many worthwhile features, it is at an early stage of development. It lacks sophisticated tools for supporting programming-in-the large, such as those found in Interlisp. However, it does support incremental compilation, and has a multi-window user interface, which facilitates development of complex programs in a modular fashion — functions which are closely related can be grouped together in a single file, and many files can be use to build up a large program. Use of the mouse to selectively compile pieces of text allows simple testing: by compiling small pieces out of a function, it is possible to see exactly what happens at each stage of execution. It also has a sophisticated virtual memory system, which makes good use of the Macintosh's resources. On the other hand, it lacks one of the most elementary debugging tools: a break loop [4]. In most LISP interpreters, it is possible to examine values of variables in the context where a run-time error (or break caused by a breakpoint) occured. This is particularly valuable in a modern version of the language, using lexical scoping.

On the IBM PC, GC LISP is an implementation of a "subset" of Common LISP. Although it has enough of a the flavour of the full language to be useful as a teaching aid, it is not as cheap as ExperLISP ($495 for an interpreted version; a similar amount extra for a compiler), and lacks some key features — such as lexical scoping. Furthermore, while ExperLISP runs on a 512K Macintosh, GC LISP needs at least 640K on an IBM PC to perform realistically. Despite these limitations, it has some useful features: GMACS (a version of the well-known EMACS editor), more sophisticated debugging facilities than ExperLISP's (although with some flaws) and a tutorial program (the San Marco Explorer) [7].

Turbo PROLOG, with a launch price of $99, has the potential to open up a completely new market. If it has the same impact on the AI community as Turbo Pascal had on programmers (effectively displacing BASIC as the PC "standard"), interesting possibilities are opened up. Although some features of PROLOG which are difficult to compile have been left out , and typing has been introduced, the negative effects will depend on a specific programmer's style. The viability of this environment for serious research is worth investigating.

4. CASE STUDY: *KEBL*

Knowledge Engineering-Based Learning grew out of several concerns: the need for education to move away from rote learning where acquisition of sophisticated skills is concerned, the need to develop technologies suitable for a third-world context and a desire to exploit the potential offered by the steady improvement in performance of computers in relation to price.

Medical education was identified as an area in which the first concern was taken seriously by many educationalists [3]. Specifically, diagnostic skills are not easy to teach, and many students only move from memorization of book knowledge to knowing how to apply this knowledge in a clinical situation two to three years after graduating [12]. Given that a shortage of medical skills is widespread in underdeveloped parts of the world, medical education is an obvious area to tackle in terms of the second concern. Use of computers as diagnostic aids was rejected as a first step because the number of computers needed would be much higher. The points mentioned above in relation to the development of AI tools for cheap computers was a consideration in exploring an AI-related approach; the experience of expert system builders in honing their skills during knoweldge acquisition was another.

ExperLISP and ExperOPS5 on the Macintosh were chosen as the best compromise between cost and performance available at the time the research was initiated. These tools offer an approximation to those found on large systems, at a fraction of the cost. The relationship of ExperLISP to other AI tools has been detailed above. ExperOPS5 was found to be an adequate approximation to the language as implemented elsewhere [6], and was a useful prototyping tool, but was not fast enough for the response time required, so the final implementation was in LISP.

Most studies of how diagnosis takes place focus on the development of hypotheses, leading to the acceptance of a specific hypothesis as the diagnosis. The approach is very similar to that of scientific method — pieces of evidence are gathered in a way directed by the currently active hypotheses. Various different processes are concurrently occurring, including hypothesis activation or deactivation, confirmation of active hypotheses and denial of alternative hypotheses [11]. An approach to teaching this style of learning — called problem-based learning — has been advocated [3], but with disappointing results [2].

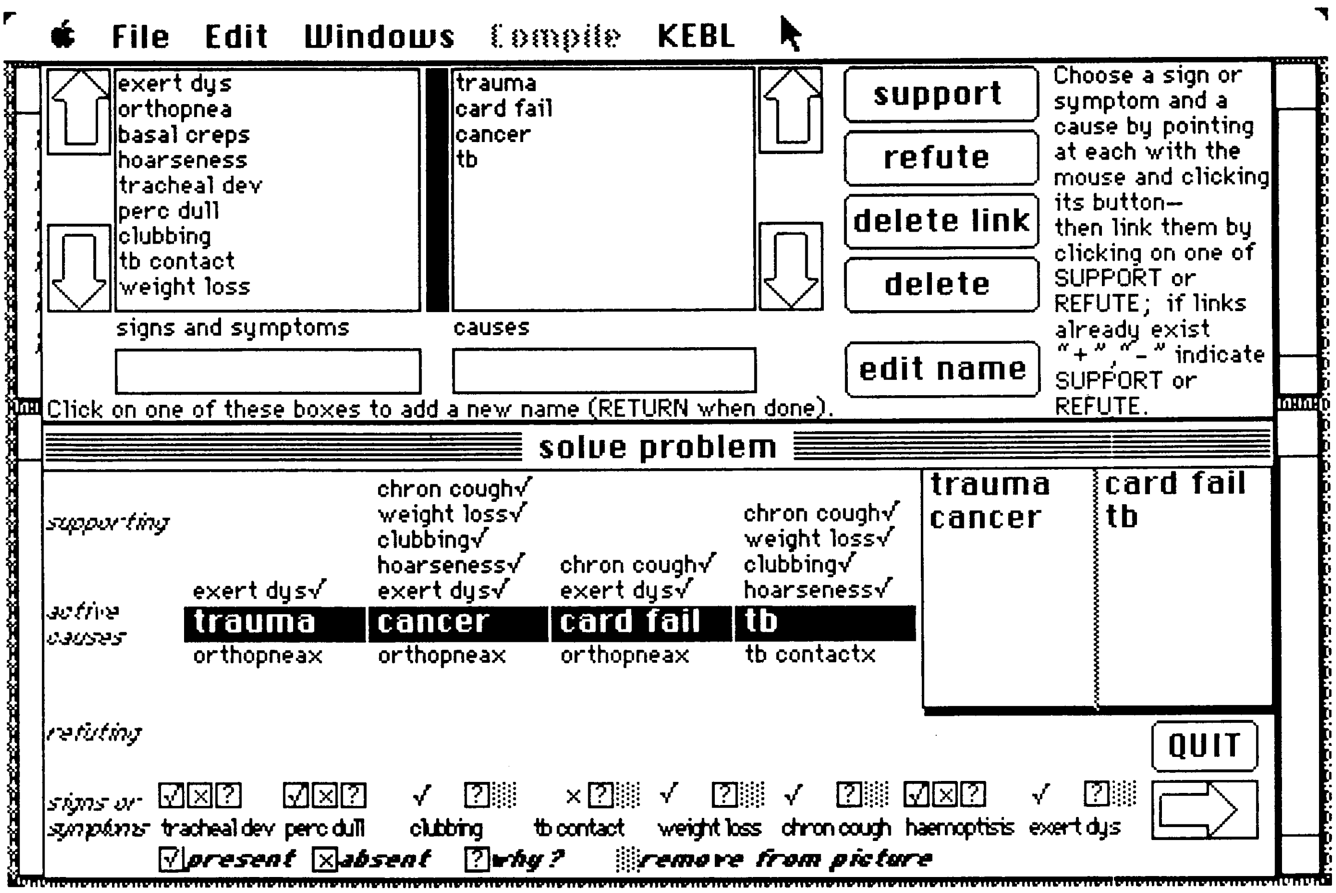
A major problem is that different individuals have different approaches to problem solving. For instance, given the same problem, one person could start by considering objective measurable medical facts, whereas another could consider the patient's social background — and both ought to arrive at the same diagnosis [17].

Doubts are being cast on the validity of labelling the problem-solving process as consisiting of evidence supporting or refuting hypotheses [16]. In order to avoid this debate, the more neutral terms "signs and symptoms" and "causes" were used instead.

Taking the experience of expert system builders into account, the construction of a simple expert system by medical students is a potential approach to developing their problem-solving ability. However, the construction of a full expert system requires significant skills which would take too much overhead to teach to non-computer scientists. The approach used in KEBL is to concentrate on discussion of a problem-solving approach in the context of a simplified approach to constructing and interpreting rules.

The rules take the form of `<name of sign or symptom> supports <name of cause>`, or `<name of sign or symptom> against <name of cause>`. Problem-solving is split into rule acquisition and tackling a specific problem. During knowledge acquisition, names of signs and symptoms and names of causes are added using the top half of the screen. A mouse pointing device is used to select the box in which a new name is to be typed. Once added to the system, names can be linked by selecting them and activating soft buttons on the screen to set up supporting or refuting links.

The specific problem phase, using the lower half of the screen, takes the form of activating or deactivating hypotheses, and activating (stating whether present or absent) or deactivating pieces of evidence. As an additional aid, it is possible to highlight all the hypotheses which relate to a specific piece of evidence. As with knowledge acquisition, the user interface relies heavily on using the mouse to activate "soft" buttons.



exert dys
orthopnea
basal creps
hoarseness
tracheal dev
perc dull
clubbing
tb contact
weight loss

trauma
card fail
cancer
tb

Choose a sign or symptom and a cause by pointing at each with the mouse and clicking its button– then link them by clicking on one of SUPPORT or REFUTE; if links already exist "+", "–" indicate SUPPORT or REFUTE.

Click on one of these boxes to add a new name (RETURN when done).

	trauma	cancer	card fail	tb
supporting	exert dys√	chron cough√ weight loss√ clubbing√ hoarseness√ exert dys√	chron cough√ exert dys√	chron cough√ weight loss√ clubbing√ hoarseness√
refuting	orthopneax	orthopneax	orthopneax	tb contactx

trauma cancer card fail tb

tracheal dev, perc dull, clubbing, tb contact, weight loss, chron cough, haemoptisis, exert dys

figure 1
The KEBL user interface

This situation gave rise to the following interaction between two simulated students (actually Computer Science lecturers) and a doctor:

doctor	what can you tell me?
first "student"	I think it's cancer
second "student"	I would say she has both TB and cancer
doctor	in fact, you couldn't tell the difference with this information; you would need to take an X-ray

The causes given in the rules appear in a table, and are activated or deactivated by pointing at them with the mouse and clicking the mouse button. Pieces of evidence are listed with buttons to add them ("√" for "present", "x" for "absent"), remove them or highlight the hypotheses relating to them. Once an hypothesis is activated, its name appears in the active area, where up to four hypotheses may appear. Supporting evidence appears above the name of the hypotheses; evidence against appears below (each tagged with a "√" or "x" to indicate its presence or absence).

An example of a KEBL screen appears in figure 1. In this instance, a practitioner entered the rules and simulated the patient while a Computer Science lecturer with no clinical knowledge attempted to gather information to form a diagnosis. Another Computer Science lecturer gave a second opinion. A more knowledgeable student would be expected to build up the rules as well.

By comparison with a conventional expert system, KEBL does relatively little. It does not weight signs and symptoms (evidence), form patterns explicitly, ask or answer questions or arrive at conclusions. The idea is that the user (and a tutor) would perform these functions. The student, by articulating the problem-solving process in a visible form, is in a position to discuss matters with a tutor (see figure 1). The formalization necessary to make the program run is also valuable as a focus for sorting out details. In many respects, the thought processes required are similar to those of the expert and knowledge engineer combined; the extra volume of work required to make the system function for another user in a general situation is missing.

In order to constrain the size of the domain, a well-known differential diagnosis is used as a starting point. More general diagnosis can be attempted at a later stage, but the approach is likely to fall down if the amount of knowledge acquisition needed before a specific problem can be dealt with becomes too high.

A preliminary study was carried out with first-year medical students (mainly because they were available at the time) [15]. Although the students had inadequate medical knowledge to carry out a thorough evaluation of the approach, some interesting points arose. For instance, a largely text-based user interface is a problem — it takes too much learning and the average student is a slow typist. Also, if the user interface constrains the approach too much, it inhibits exploration of alternative approaches. As a result, the user interface of figure 1 was designed. Not only does it require very little use of the keyboard, but all options are simultaneously available. The user is only constrained by the information currently in the system. In contrast to a conventional expert system, switching between knowledge acquisition and problem-solving is expected to happen at any time. It is possible, for instance, to modify rules in the middle of solving a problem.

An additional feature of KEBL is a record which is kept of the entire session. Statistical analysis of approaches to problem-solving, as well as playing a session back for evaluation and feedback, are possibilities which will be explored.

5. EXPERIENCE WITH TOOLS

ExperOPS5 may potentially be implemented more efficiently, but in the form used for the KEBL prototype, is not a practical proposition for a large project. It is, however, an excellent basis for learning about production systems, with a good range of debugging tools (single-step, display the conflict set, undo etc.), although better use of the Macintosh user interface could have been made.

ExperLISP, despite the positive points noted, is not without problems as well. It is very slow to load (at least a minute), and has some unexpected bugs, especially in relatively esoteric areas like menu manipulation and graphics programming. Furthermore, the current release does not support the Macintosh approach to programming its user interface through "resources", which means much which ought to be available through the operating system has to be explicitly programmed. Another problem is the operating system can easily be crashed by accessing invalid pointers. Clearly, from the point of view of providing safe access to such features as menu manipulation, a full class system (promised for the next release) is essential. Object-orientedness would also tie in naturally with the Macintosh interface, and would provide much-needed support for programming-in-the-large.

Even taking these negative points into account, the system is usable for fast prototyping and developing sophisticated programs. The key — as with any LISP system — is to develop suitable tools [14]. The current release of ExperLISP (version 1.5) is adequate for developing programs of up to 1 000 lines, or about 1 500 lines with a hard disk. Larger programs start exposing bugs in the compiler; in any case, developing programs much bigger than this would need tools for programming in the large (in the Interlisp style).

For an introduction to object-orientedness, XLISP is a worthwhile acquisition (especially as it is in the public domain). Although the syntax is sometimes clumsy, and large programs are likely to be unacceptably slow (it is not compiled), it is easy to generate some interesting examples, which make the lack of object-orientedness in other LISPs seem a major ommission. Furthermore, it is a good approximation to the spirit of Common LISP (it supports lexical scoping, unlike GC LISP), even if it is a relatively small subset of the language. On the Macintosh, the lack of a built-in editor can be overcome by using Switcher to run XLISP and a word processor more-or-less concurrently. The IBM PC version of XLISP can similarly be run using a resident editor such as Sidekick.

Since XLISP is continually being improved, perhaps it will rival some of the commercial products in time; $500 is a good lower bound on the price of sophisticated, usable (if flawed) AI tools.

REFERENCES

1. Alexander, Tom. The Next Revolution in Computer Programming, *Fortune* 29 October 1984 (65–70).
2. Babbot, David and Halter, William D. Clinical Problem-Solving Skills of Internists Trained in the Problem-Oriented System, *Journal of Medical Education* **58** 1983 (947–953).
3. Barrows, Howard S. and Tamblyn, Robyn M. An Evaluation of Problem-Based Learning in Small Groups Using a Simulated Patient, *Journal of Medical Education* **51**(1) January 1976 (52–54).
4. Betz, David. An XLISP Tutorial, *Byte* **10**(3) March 1985 (221–236).
5. Bortz, Jordan and Diamant, John. LISP for the IBM Personal Computer, *Byte* **9**(7) July 1984 (281–291).
6. Brownston, Lee, Farrell, Robert, Kant, Elaine and Martin, Nancy. *Programming Expert Systems in OPS5*, Addison-Wesley, Reading, Massachusetts, 1985.
7. D'Ambrosio, Bruce. Golden Common LISP, *Byte* **10**(13) December 1985 (317–321).
8. Dunn, Robert J. "Expandable Expertise for Everyday Users", *Computing SA* **5**(40) 14 Oct 1985 (14,15,18).
9. Goldberg, Adele and Robson, David. *Smalltalk-80: The Language and its Implementation*, Addison-Wesley, Reading, Massachusetts, 1983.
10. Hasemer, Tony. *A Beginner's Guide to LISP*, Addison-Wesley, Wokingham, Berkshire, 1984.
11. Kassirer, Jerome P. and Gorry, G. Anthony. Clinical Problem Solving: A Behavioural Analysis, *Annals of Internal Medicine* **89**(8) August 1978 (245–255).
12. Leaper, D. J., Gill , P. W., Staniland, J. R., Horrocks, Jane C. and de Dombal, F. T. Clinical Diagnosis Process: An Analysis, *British Medical Journal* **3**(9) 15 September 1973 (569–574).
13. Linden, Eugene. Intellicorp: The Selling of Artificial Intelligence, *High Technology* **5**(3) March 1985 (22–25, 82).
14. Machanick, Philip. Tools for Creating Tools: Programming in Artificial Intelligence, *Quæstiones Informaticæ* **3**(3) August 1985 1985 (30–36).
15. Machanick, Philip. *A Study in Knowledge Engineering-Based Learning*, Computer Science Department Report CS–PM–86–008, University of the Witwatersrand, Johannesburg, 1986.
16. McGuire, Christine H. Medical Problem-Solving: A Critique of the Literature, *Journal of Medical Education* **60**(8) August 1985 (587–595).
17. Mitchell, Graham. Private Communication, 1986.
18. *PC Magazine*. **4**(8) 16 April 1985: special issue on expert systems.
19. Sandewall, Erik. Programming in an Interactive Environment: The 'LISP' Experience, *Computing Surveys* **10**(1) March 1978 (35–71).
20. Teitelman, Warren and Masinter, Larry. The Interlisp Programming Environment, Computer **14**(4) April 1981 (25–33).
21. Touretzky, David S. *LISP: A Gentle Introduction to Symbolic Computation*, Harper and Rowe, New York, 1984.
22. Winston, Patrick Henry and Horn, Berthold Klaus Paul. *LISP* (second edition), Addison-Wesley, Reading, Massachusetts, 1984.

WHAT'S WRONG WITH CP/M?

S.P. Byron-Moore
Department of Computing Science
University of Zimbabwe
Harare
Zimbabwe

Trends in operating systems are examined in the light of advances in computer hardware. Consideration is given to the desirability of a good, up-to-date single-user operating system. The specification and implementation of such a system is discussed.

1. INTRODUCTION

1.1 Software

At the present time, most discussions about operating systems include mention of UNIX, a multi-user operating system developed by Bell Laboratories in the early 1970's. The aim of its designers was to "create a computing environment ... where they themselves could comfortably and effectively pursue their own work - programming research" [1]. Due to the management policies of Bell Laboratories, UNIX has only recently become commercially available, althougth it has been widely used by Bell and some universities since 1971. This system is being proposed by many authors and salesmen as the answer to everyone's dreams - "the operating system of the future", for both commercial and research usage.

1.2 Hardware

In the 16 years since UNIX was first conceived, advances in computer hardware have changed the face of the computer industry. Microcomputers have become common and due to their low price, many businesses, as well as researchers and hobbyists, have invested in small and often single-user systems. The introduction of microcomputers opened up new sales markets and 'price wars' raged as manufacturers competed for this lucrative trade. Consequently prices for hardware such as random-access memory and fast access, high capacity storage devices such as Winchester disc drives have been lowered considerably. Increasing miniaturisation of components and improved technology have allowed computer firms to produce portable (or at least transportable) computers, which have rapidly gained an expanding market.

1.3 The Future

With the reduction in prices of hardware for microcomputers, many researchers and business users are now able to have dedicated microcomputers sitting on their desks, which satisfy most of their computing needs. This is a rapidly expanding market - "Future Computing", a US research group, predicts that portable computer shipments alone will exceed 1 million units in 1988 [2]. More and more microcomputers are being used as stand-alone machines, with the capability of communicating with larger computers via networks. Many users have become accustomed to having a microcomputer for their own personal use, without having to share its facilities with other users. Given the current and anticipated increase in the number of personal computers, many for single-user applications, it seems ironical that UNIX, a large multi-user operating system, born in the age of mainframe and minicomputer dominance, should be coming to prominence.

2. SINGLE-USER COMPUTER SYSTEMS

2.1 Advantages

The main advantages of single-user operating systems compared to multi-user operating systems are:

- small size
- rapid response time
- versatility

A single user operating system does not have to include the security and accounting features which are necessary in a multi-user system, consequently the operating system is of a smaller size. This means that operating system loading time is reduced, execution time is lessened and more memory space is left free for the user.

System versatility is an important advantage of a single-user system. The user of a dedicated microcomputer can tailor his system to his own requirements and preferences. Like the owner of a bicycle, who adjusts his machine for his personal use, the user of a single-user system can adapt this system for efficiency and ease of use. For example, the user may devote part of the machine's memory space for usage as a disc emulator [4]. He may rename operating system commands to provide the type of operating system interface he prefers [5]. He may set up a menu-driven interface to the operating system. Such personalisation would be banned or seriously frowned upon in a multi-user operating system. Here several users may have differing requirements, so a user cannot be permitted to adapt the system for his own needs. Furthermore , users of a multi-user system, very reasonably, expect consistency from the system they are using. Thus this type of system cannot be easily adapted.

When a computer is to be utilised in a single-user environment, a single-user operating system is highly desirable since it can be more efficient than a multi-user operating system and is far more easily adaptable to its user's tastes and needs.

2.2 Current Operating Systems

Most single-user operating systems currently available have their deficiencies. Perhaps the most well known, CP/M 2.2 [6] is almost as old as UNIX. Its speed of disc access is painfully slow and its directory structure does not accomodate high capacity discs. CP/M Plus [13] supports bank switching to exploit extra memory and facilitates multi-record reads to speed disc access, but it still has a single-level directory structure and is fairly difficult to implement. MS-DOS 2.0 has a hierarchical directory structure and multiple sector buffers to speed disc access but has a strange set of system calls, because it is a compromise between CP/M and XENIX. It provides only a print spooler and not full concurrency.

Most operating system development work seems to be focused on multi-user systems. UNIX, for example, has many nice features [9], which are desirable in an operating system. Although UNIX is now being put on to microcomputers, albeit in a reduced form, it is still a multi-user system, not appropriate to small single-user systems. Digital Research has recently released Concurrent PC-DOS, a multi-user, multi-tasking operating system for 16-bit microcomputers, but this occupies 156KB of memory and is fairly slow [7].

3. PROPOSED SYSTEM

3.1 General

Given that there is now, and will be in the future, an increasing number of single user computer systems, we must consider what facilities are required in a modern single-user operating system.

3.2 Structured Directory System

Owners of single-user machines have traditionally had limited disc storage available and have been content to use a single level directory structure like that of CP/M 2.2. However, with the price of mass storage devices dropping, users are tending to keep many more filenames on a particular device. Since no user wants to see a hundred or more files displayed on the screen when he requests a directory listing, the operating system needs to support a suitably structured directory system in order to provide for fast and efficient location and retrieval of files.

A popular method for structuring the directory allows the user to set up a hierarchical file system. This system has great advantages in a multi-user operating system since it facilitates easy separation of different user's files. However, a hierarchical system can be rather confusing to new or inexperienced users, who have some difficulty in navigating the tree structure.

An alternative method is to allow the user to divide a physical disc into a number of logical discs, each with its own directory. This is conceptually easier for a non-professional computerist to understand, but is not suitable for a user who wishes to store a number of long files. There is also a tendency for space to be wasted, due to the fragmentation of the disc. Should our proposed operating system include both of the above methods, and allow the user to set up his directories as he prefers or is there some other structure which is ideally suited to a single-user environment?

3.3 Faster Disc Access

Slow speed of disc access is one of the major disadvantages of current operating systems. Many systems read small blocks of information from disc in order to conserve memory space. CP/M 2.2, for example, reads/writes one 128 byte record at a time. Many users have increased their disc access speed under CP/M by adding code to their BIOS (the user configurable part of CP/M) to perform multi-record reads/writes. CP/M Plus [14] includes a new operating system function to do just this. As memory space is becoming less crucial, a modern operating system should be able to transfer large amounts of data, in one go. Many modern floppy disc controllers, for example, allow an entire track read/write. The operating system should also be able to cope with the transfer of this amount of information.

With the increasing availability of memory, cache buffering is also a useful feature. One of the good points of MS-DOS, is the provision of a user-specifiable number of buffers for disc cacheing [15].

A disc track is normally divided into a number of sectors. The disc controller can only read/write a complete sector. Due to the speed of rotation of the disc, after the controller carries out a sector read/write, several more sectors may pass the disc head before the controller is ready to carry out more I/O. For this reason, many operating systems provide for a "skew" factor for disc I/O. CP/M, for example, has a standard skew of 6 sectors - this means that after reading sector one, sector seven will be read, followed by sector 13, etc.. The size of the skew factor depends on the capabilities of the hardware and the operating system overhead. Normally, when a skew factor is used, adjacent sectors on disc have contiguous numbering. With CP/M for example, a file of length 3 sectors, may be stored in physical sectors 2,8 and 14. The operating system has to map the logical sectors making up the file to the physical sectors on disc. This transformation can be avoided by renumbering the sectors on the disc so that the controller reads sectors numbered sequentially (i.e 1,2,3,4,...), although sector n-1 and sector n will not be physically adjacent on the disc. By renumbering in this way, operating system overhead is reduced and the transfer of the discs to another computer system is made more straightforward, since there is no need to know with what skew factor the disc was recorded.

3.4 Improved User Interface

The majority of operating systems now in use were written at a time when most computer users were professionals and there was little emphasis on user-friendliness. Ritchie, for example, states "Both input and output of UNIX programs tend to be very terse. This can be very disconcerting, especially to the beginner" [10]. Menu systems are being written as add-ons to many operating systems, for example NCR's menu driven interface to UNIX [3]. In the department of computing science at the University of Zimbabwe, we have a menu-driven interface to CP/M, which has been in use for three years for research work and for eighteen months for teaching purposes. We have found it extremly useful as it removes the burden of remembering a command set, reduces the amount of typing required and facilitates rapid familiarisation with the system. All modern operating systems should provide support for a menu-system as well as providing a command entry mode for experts who wish to avoid the menu-system or carry out special tasks. There should also be an operating system utility to enable the user to tailor his menus, to take full advantage of the single-user system's versatility.

3.5 I/O Redirection

This is one of the strong points of UNIX - it allows the user to change easily the source of input and destination of output of a process. This means that program writing can be made more general, without concern for the final I/O devices. It is certainly a desirable feature in a modern single-user operating system and is not too much of a problem to implement.

3.6 Pipelines

Pipes are another UNIX feature . They allow a process to communicate with another process without having to set up and manage one or more temporary files. To implement pipes, the operating system should include a protected buffer for the transfer of data and a scanning routine to insert/remove data from this buffer. Pipes are invaluable if concurrency is allowed.

3.7 Concurrency

The frustration of waiting for one job to finish before getting on with the next job is common to most users of single-tasking systems. Concurrency is a desirable feature, but not if it noticeably degrades system performance. Awalt [7], for example, claims a 29 percent performance penalty when running two tasks concurrently (instead of sequentially), under Concurrent PC-DOS.

By forcing the user, not the operating system, to specify the priority and/or weighting of any processes that he runs concurrently, it seems likely that we can minimise the visibility of any performance degradation. Possibly we should also limit the number of concurrent processes to two, to avoid serious degradation.

Providing for concurrency inevitably imposes an added burden on the operating system, which must protect one process from another and perform complex memory management functions. A concurrent operating system will be much larger and more complicated than a single-tasking system. However, this should not prove to be major problem with the continuing reduction in the price of memory. We will probably see more and more users moving to a banked memory system, which would allow transient parts of the operating system to be rapidly moved to main memory.

Is it worth paying the penalty of supporting concurrency? A user may only want to run concurrent processes on an occasional basis. He may never want concurrency or he may require its use frequently. It seems that the best solution is to provide a separate operating system module to handle concurrency. The user must specifically invoke this module if it is required, otherwise the operating system acts as a single-tasking system.

3.8 Virtual Memory Handling

This is a very old idea, first used on mainframes in the early 1960's. The user sees no limitation on the memory space available to him and if a program is too large for memory the operating system is responsible for rolling-in and rolling-out code or data as it is required. From the programmer's point of view this removes unnecessary limits on program size. It reduces problems regarding the number of concurrent processes that memory can hold at one time. But it does have its cost - a larger operating system size. Is virtual memory a desirable feature of a single-user operating system for microcomputers now that more memory is becoming available or does its processing overhead outweigh its advantages?

3.9 Other Considerations

The above discussion lists only the major design considerations for a new single-user operating system. There are many other lesser features that are worth considering for inclusion. For example, is it worth while providing an on-line help facility to explain the usage of operating system commands; a type ahead buffer for the terminal input and a command line editor to facilitate rapid command entry. Split-screen viewing is useful when a user wishes to perform a subsidiary task. Discussion with a number of users would help to identify which additional features are required.

No program should crash, but this is particularly important for an operating system. There must be good error trapping and reporting facilities.

4. IMPLEMENTATION

4.1 Alternatives

It seems desirable that our proposed single-user operating system should be compatible with CP/M 2.2, a well established and widely used single-user system, because:

- a large number of varied utilities have been developed for use under CP/M 2.2, many of them in the public domain [11, 12].
- many other CP/M utilities are available for moderate costs.
- there are many users of CP/M compatible operating systems(e.g. ConIX [18], RP/M [16], MRS/OS [17], C/NIX [19]), as well as users of Digital Research CP/M. The source code of CP/M 2.2, or a CP/M look-alike such as RP/M, MRS/OS can be obtained.

To maintain compatibility with CP/M 2.2 our alternatives are

- to enhance CP/M 2.2
- to write a new operating system compatible with CP/M 2.2

4.2 Enhancements to CP/M 2.2

Many of the enhancements, discussed in section 3, can be made to CP/M 2.2 without a great amount of difficulty. However, this technique is rather like building a house and than adding numerous extensions - the result is never as good as building the whole house in one operation. It seems preferable to carefully design a new efficient operating system.

4.3 New operating system.

Since the source code of CP/M is available it is possible to write a new operating system, which retains compatibility with CP/M by using the same system calls.

This new system should be written in a modular form, to make it portable to computers having different processors and make it suitable for use on machines with either small or large addressable memory. Advantages of this modular form include

- the straight forward selection of a subset of the operating system's facilities, depending on the size and capabilities of the computer system and the needs and preferences of that system's user. For example, the type ahead buffer mentioned in section 3.8, can only be implemented in a machine with a hardware interrupt capability.

- the easy amendment of a module for a particular user's software requirements and hardware configuration.
- the operating system may easily be separated into transient and resident portions for a banked memory system.

Ideally, the new operating system should be written in a middle-level language. Assembly code is unsuitable since it is processor dependent and we require our system to be easily transportable. High-level language programs, although transportable, tend to produce unnecessarily large code files, so a high level language should not be used. The writers of UNIX attempted to avoid this problem by creating the language "C". Unfortunately UNIX is still not as easily transportable as would be liked [8]. We propose that the new operating system should be written in a macro language so that only individual small macros have to be rewritten to implement the system on a machine with a different processor.

5. CONCLUSION

There is an increasing need for a good single-user operating system. Although there are a number of sound multi-user operating systems available, most single-user systems are relatively old and outdated. A modern single-user system is needed to fill this gap in the market. In section 3, we discussed some of the main design considerations for this new system. It is important that the system be transportable to different processors, with small or large amounts of memory. Ideally it should be available in the public domain.

REFERENCES

1. M D McIlroy, E N Pinson, B A Tague, UNIX Time-Sharing System: Foreword, pp 1899-1904. *The Bell System Technical Journal.*, Vol 57 No 6 part 2 July-Aug 1978.
2. O Tucker, Joining a market on the move. pp6-8, *Computing The Magazine*, Dec 13 1984.
3. Tower System, Administration manual., *NCR*, Vols 1 and 2 Release 1.0 July 1983.
4. E Nisley, Spinning your own VDISK., pp 100-109, *PC Tech Journal* , Vol 3 No 3 Mar 1985.
5. D Fielder, the UNIX tutorial, pp 257-278, *Byte magazine*, Vol 8 No 9 Sept 1984.
6. CP/M 2.2 manual., Digital Research Inc 1979.
7. D Awalt, Concurrent PC-DOS, pp 45-54, *PC Tech Journal* , Vol 3 No 3 Mar 1985.
8. M Tilson, Moving UNIX to new machines, pp 266-276, *Byte magazine*, Vol 8 No 10 Oct 1983.
9. D M Ritchie K Thompson, The UNIX Time-Sharing System, pp 1905-1929, *The Bell System Technical Journal.*, Vol 57 No 6 Part 2 July-Aug 1978.
10. D M Ritchie, UNIX Time-Sharing System: A Retrospective, pp 1947-1969, *The Bell System Technical Journal.*, Vol 57 No 6 Part 2 July-Aug 1978.
11. SIG/M (Special Interest Group for Microcomputers, Amateur Computer Group of New Jersey), Inc. Box 97 Iselin NJ 08830. USA.
12. CPMUG (CP/M User's Group) 1651 3rd Avenue NY10028. USA.
13. B R Ratoff, Implementing the Advanced Features of CP/M Plus, pp 26-29, *Microsystems*, Feb 1983
14. B R Ratoff, Implementing the Advanced Features of CP/M Plus: Part2, pp 70-72, *Microsystems*, Apr 1983.
15. W G Wong, MS/DOS : An Overview Part 1, pp 47-52, *Microsystems* , Mar 1984.
16. RP/M. Micro Methods. Box G Warrenton OR 97146. USA.
17. MRS/OS. OCCO inc. 16 Bowman lane Westboro, MA 01581, USA.
18. ConIX. Computer Helper Industries Inc. P.O. Box 680 Parkchester Station NY 10462. USA.
    D Lunsford, Software review ConIX, pp 83-86, *Computer language* , Jun 1985.
20. C/NIX. The Software Toolworks 15233 Ventura Boulevard Suite 1118 Sherman Oaks California 91403. USA.

IN PRAISE OF SOLID STATE DISCS

P.F.Ridler,
Dept. of Computing Science,
University of Zimbabwe.

Various ways of using random access memory to emulate a magnetic disc storage system are described and the advantages and disadvantages are enumerated. An algorithm for making use of a solid state disc is given which avoids the user having to take any action when the wanted file is on a magnetic disc or when the solid state disc becomes full.

1. WHAT IS A SOLID STATE DISC?

A solid state disc (s.s.d) is simply a section of random access memory which is treated as though it were a disc. It is assumed to have tracks and sectors, to hold directories and data, and have the same layout as any other disc permitted by the operating system in use. It does not, of course, have gaps between its sectors as with a magnetic disc and it is read by a simple data transfer operation which requires no commands to a disc controller chip. Its tracks are purely logical sub-divisions of what might otherwise be thought of as a linear array of data elements. The number of elements in a track may be related to the length of the physical section of memory which can be made available at any particular time.

2. WHY USE A SOLID STATE DISC?

Why should a computer system use a solid state disc? Well, in the olden days, the larger mainframes used to and it was referred to as 'fast backing store' or something similar. It was usually a relatively slow magnetic core store and was used because it was much faster than a disc or drum but not as expensive as the fastest cores which were available.

The price of high capacity semi-conductor chips is falling rapidly. The price per kilobyte has fallen spectacularly over the last few years as may be observed in the following table which is given for small quantities :

month	year	US$	chip size
DEC	1977	20.00	4k
DEC	1980	2.50	16k
DEC	1982	1.00	64k
DEC	1983	.50	64k
DEC	1984	.25	64k
DEC	1984	.95	256k
JUNE	1985	.20	256k
JUNE	1986	.10	256k

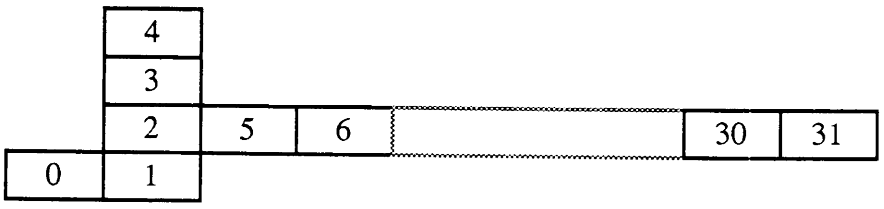
The overhead cost for a complete board holding thirty two of the chips referred to has remained roughly constant over the whole period.

This dramatic fall in the cost of memory has made it possible to use relaively high speed memory as backing store: a 1M-byte solid state disc now costs only about US$300.

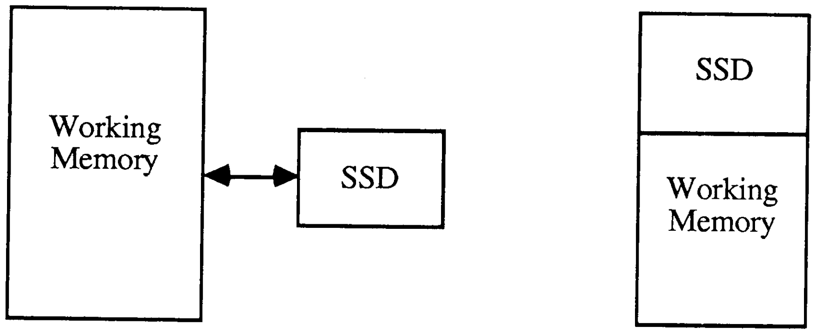
3. HOW IS A SOLID STATE DISC IMPLEMENTED?

There are four different methods of implementing a solid state disc:

(i) banked switched memory
(ii) port accessed memory
(iii) segmented memory
(iv) directly addressed memory.



a) Bank Switched S.S.D.



b) Port Accessed S.S.D. c) Memory Mapped S.S.D.

figure 1

3.1 Page switched memory

One way of implementing a s.s.d is to use a memory board which is divided into logical sections (pages), one or more of which can be made available at a time. One of the boards which the author has used is that referred to in [3]. This board is designed to work on the S100 bus, and consists of 256k of 4164 64k-byte chips. The memory controller chip is a 74S409 (equivalent to a DP8409) which contains the necessary circuitry to provide refreshing for the dynamic memory chips. This controller also has 18 address lines which enable it to access the whole 256k bytes on the board. However, as the Z80 processor with which it is used has only 16 address lines, the whole memory cannot be addressed directly.

To enable the Z80 to address the entire 256k, the two most significant bits of the Z80 address lines are not used to address the memory directly but are translated to a four bit address via registers which are written through I/O ports.

The result of all this is that any four of sixteen 16k memory pages are available at any one time, the particular pages enabled being detemined by the address translation registers.

The use of the board as a s.s.d is illustrated in fig. 1a. Memory pages 1,2,3 and 4 are used as a 64k main memory. Page 0 is used to hold the major part of the operating system so as to free space in the main memory and pages 5 through 16 are used as s.s.d.

The s.s.d is logically composed of tracks each having 128 128-byte data sectors. To access a particular data sector, the memory page containing the track and sector is enabled in place of one of pages 2 or 3 of the main memory bank taking care that the destination is still available. The 128-byte sector is then transferred to its destination. The transfer process takes about 5 μs per byte instead of 32 μs per byte from a flexible disc, and is about the same speed as for a Winchester drive. However if the actual disc drive head is not at the wanted track then it must be moved to the wanted track and this movement takes a time which is of the order of milli-seconds and is approximately the same for flexible or hard discs. Of course, the s.s.d is truly random access and no extra time is taken to move between logical tracks.

3.2 Port Accessed Memory

The second method of implementing a s.s.d which uses a separate memory array is that which accesses the memory through I/O ports [2]. Separate ports are used to set the track and sector and the data is read or written through the data port (fig 1b). This method handles data at the same rate as the previous one but is marginally easier to implement as far as software is concerned. The main disadvantage of the method is that a specialised memory board is required while in the paged system ordinary memory boards may be used.

3.3 Segmented Memory

A number of the available 16-bit microprocessor chips [4,5] have memory management circuitry on board. Addressing is done by adding an offset to a segment address. A segment register can thus be used to indicate the logical track of the s.s.d while a simple mapping of the sector number will give the required offset (fig 1c).

3.4 Directory Addressed Memory

The 68000 processor has a linear addressing system. All memory access is done by simply emitting a 32-bit address on the bus. It is relatively simple to devise a method of implementing a s.s.d using memory in an address range which will not be used by other programs. The s.s.d layout is relatively unimportant and any simple track and sector mapping can be used (fig 1c).

4. PRO'S AND CON'S OF SOLID STATE DISCS

There is little differences between the four methods of implementing a s.s.d which have been outlined above, and the decision as to which method to use will depend on other factors in the design of the machine.

4.1 Speed

The major advantage of a s.s.d is in its speed. It is many times faster than a flexible disc and considerably faster than a hard disc.

4.2 Volatility

All these methods of implementing a s.s.d are potentially volatile. If the power fails, or the operator turns the machine off before backing up data to magnetic discs, information will be lost. It is possible to use a battery which comes into play when the power supply feeding the memory falls below a certain voltage or an uninterruptable supply may be used for the whole machine. In a typical urban situation power supply failures are rare and the extra expense may be hard to justify.

A more subtle situation arises from the use by the electricity supply authorities of "ripple relay" switching. A voltage in the range 500-1500 Hz is superposed on the mains voltage to turn on or off water heating and other equipment at peak load times and this may interfere with a computer. The author has had several cases where data in a s.s.d has become corrupted at about 5 a.m. and surmises that this is due to the ripple tone creeping through the power supply to the bus; better power suppy design would probably eliminate the trouble.

4.3 Cost

The cost of a s.s.d may be as low as $300 per megabyte at the present time. The cost of a 5" flexible disc drive is around $500 including its controller. It is not possible to evaluate the cost/benefit ratio for a general case because the job mix affects it intimately, and estimates of the relative values of time and cash vary enormously.

The author has been using two different s.s.d's for a period of a year and would certainly not be without one in the future. Programming is now limited by "thinking time", as it should be, rather than by machine time.

4.4 Back-up

Periodically it is necessary to write files held in the s.s.d to some more permanent medium. Files such as editors and other utilities need never be saved, for they contain no new information, but program and data files may well have been modified and must be preserved. These files may be saved as they are created but this slows down the process to a speed which is limited by the magnetic disc drive. At the very slight risk of losing data through power failure, files which are to be saved should only be archived at intervals and on an incremental basis. At the command of the programmer, or at close down, those files which have been modified since the last archiving process are all written back to the discs from which they came [1]. This saves only the new data without slowing down the process excessively by unnecessary writing to slow backing store.

5. AUTOMATIC ROLL-IN OF FILES

When the system is switched on initially, the solid state disc will be empty. It must then have its directories initialised and those files which it is anticipated will be used brought into the memory space of the solid state disc. This process is necessarily slow, the time depending on the transfer rate and the track-to-track stepping time of the disc drive. Typically, to bring in a menu system, an editor and assembler and a few utility programs totalling 50k bytes will take 100 seconds.

As other files are required they must be transferred from the magnetic disc to the s.s.d; normally, this is done by means of a file transfer program.

Again, as the s.s.d becomes full it is necessary to transfer files from it back to a magnetic disc. Those files which are the least recently used will probably be those least needed and can either be erased if they have not changed or transferred to backing store if they hold new information.

These procedures may be carried out automatically, files being brought into the s.s.d on demand and transferred back to magnetic discs as space in needed in the s.s.d for new files. A flow diagram of the process is shown in fig.2.

6. OPERATING SYSTEM REQUIREMENTS

The extra requirements placed on the operating system by the use of a solid state disc are fairly small. The disc read and write routines are quite small and the backup function would normally be carried out by a utility program. Automatic file roll-in and roll-out is a major addition, but can be dispensed with if so desired.

6.1 Solid State Disc Handling Routines

The read and write routines for accessing data on s.s.d's are almost trivial; much simpler than those for mechanical disc drives. Obviously they must be made part of the operating system and must be made compatible with the disc formats which the operating system can handle.

Otherwise, s.s.d's pose no problems in this area.

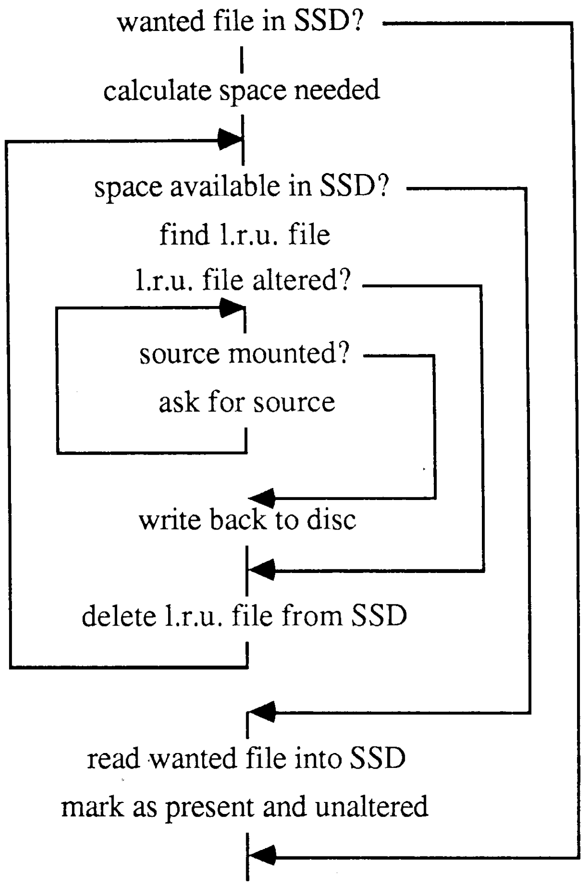


figure 2

6.2 File Roll-In/Roll-Out Routine

Automatic roll-in of wanted files and roll-out of redundant files in the s.s.d seems to be a highly desirable feature of an operating system. However, with a fair-sized (256k-1Mbyte) s.s.d such a system can be dispensed with in a typical system programming environment. The author presently uses a 512k byte s.s.d and is working on a compiler the source program of which occupies 120k and the source of the runtime package another 50k bytes.

6.3 Archiving Routines

As mentioned in 4.4, back-up of files must be done at intervals. Programs to do this are available, but it seems desirable that the operating system, in its directory area, keep track of those files which have been written to so that the archiving utility can keep track of random access files.

It also seems desirable that the operating system have the ability to write the file currently in use back to magnetic discs without using the archiving utility. This would enable more frequent back-up of a workfile without distracting attention from the job in hand, the use of the archive utility then being only necessary at power down.

REFERENCES

1 Fiedler.D., "QBAX": an incremental backup utility. (review), *MicroSystems*, **4** No 10, Oct 83, Amanuensis

Inc., RD1 Box 236 Grindstone, PA 15442, USA.
2 4931 512k S-100 Semidisk handbook., Semidisk Systems, Box GG, Beaverton, OR 97075, USA.
3 California Digital CT256-I handbook., California Digital, 4738 156th St., Lawndale, CA 90260, USA.
4 8086 family user's manual. Intel Corporation.
5 Z8000 manual., Zilog Corporation.
6 68000 manual., Motorola Corporation.

Second Announcement and Call for Papers

Interdisciplinary Conference on

MATHEMATICAL LOGIC AND RELATED SUBJECTS

Durban, 6-10 July 1987.

This conference is intended to bring together logicians and those who use logic in other disciplines. The organizers interpret "related subjects" in a wide sense in order for the conference to be of interest to mathematicians, computer scientists, linguists, philosophers and in general to all those doing research in areas overlapping logic.

The conference is organized by the Departments of Mathematics of the University of Cape Town and the Rand Afrikaans University; it is sponsored by the Department of Mathematics of the University of Natal (Durban), and will be presented as the annual Hanno Rund Colloquium of the Deaprtment. Accommodation will be available in a University residence as well as in a moderately-priced hotel. The registration fee for the conference is R25,00, but this will be waived for students. In special cases limited financial assistance may be available to intending participants.

The format envisaged for the conference is that survey lectures will be presented by invited speakers, with other participants contributing shorter research papers. The survey lectures will give different perspectives on logic and its applications, and should be accessible to all participants. Several speakers from abroad have already accepted invitations to present survey lectures. A subcommittee has been formed to compile a program for the subject are *Logic and Informatics,* of interest especially to computer scientists and sponsored by the South African Institute of Computer Scientists. Enquiries regarding this part of the conference should be directed to one of its organizers: Proff. S.W. Postma and N.C.K. Phillips, Department of Computer Science, University of Natal, Pietermaritzburg. The *Proceeding* of the conference will appear as a special edition of the *South African Journal of Philosophy.* All invited and contributed papers may be submitted for publication at the time of the conference. All submitted papers will be subject to the normal refereeing procedures.

Research papers of approximately 30 minutes duration on any aspect of logic are hereby solicited.

There is as yet no deadline for final commitment to take part in the conference, but those who are interested are asked to inform the organizers of their decision as soon as it is taken. In particular, early commitment to contribute a paper will be much appreciated, even if full details cannot be given at present.

The organizers are:

- Prof. C. Brink, Department of Mathematics, University of Cape Town, Rondebosch 7700.
  Telephone (021) 69-8531; Telex 57-2208 SA. (from 1 January 1987.)
- Prof. J. Heidema, Department of Mathematics, Rand Afrikaans University, PO Box 524, Johannesburg 2000,
  Telephone (011) 726-5000; Telex 42-4526 SA.

THE DEVELOPMENT OF AN RJE/X.25 PAD: A CASE STUDY

C.W. Carey
C. Hattingh
D.G. Kourie
R.J. van den Heever
R.F. Verkroost
Department of Computer Science
University of Pretoria
0002 Pretoria

This paper describes the practical aspects of a protocol system implementation. The software for a BSC/X.25 Packet Assembler/Disassembler (PAD) was designed, developed and tested. The PAD was designed to support the communication of IBM 2780/3780 and compatible Remote Job Entry (RJE) stations over an X.25 network.

In the paper the RPAD/X.25 network environment is introduced, the target system is described, and the software design is discussed. Some attention is given to the problems and techniques of protocol specification, and the methods employed in this particular project are discussed. The implementation and testing phases of the development are described, and finally, the importance of complete, accurate and structured documentation is stressed.

1. INTRODUCTION

The increasing use of packet switching technology for data communications has resulted in the development of a variety of Packet Assembler/Disassemblers (PADs). These PADs enable non-X.25 devices to communicate over an X.25 network - the PAD 'maps' the device native protocol to/from X.25.

In this paper a case study of a PAD development is described, including the design, development and testing of the software for a particular target system enabling Remote Job Entry (RJE) devices - in particular IBM 2780/3780 and related devices - to communicate over an X.25 network. The PAD is referred to as RPAD (for RJE PAD), and its associated software as RTX (for RJE/X.25).

The RPAD/X.25 environment, the target system, and the software design are discussed. The problems and techniques of protocol specification are addressed by descibing the methods used in this particular development. The implementation and testing phases of the project are described, and some attention is given to the all-too-frequently shunned area of documentation.

An RJE or X780 device (used here as a generic term for all RJE devices supported by RPAD) is typically a station comprising some or all of the following components:

- card reader
- card punch
- printer
- floppy storage
- BSC adapter

The stations are traditionally used for remote data entry to a mainframe.

The IBM 3780 protocol used by these devices is a subset of standard IBM BSC (Binary Synchronous Communication). It operates in point-to-point or multipoint mode, half duplex, and is symmetrical at both the transmitting and receiving ends of the communication channel. RPAD, however, supports point-to-point working only.

Although RJE stations are becoming somewhat outdated, the 3780 protocol is widely emulated by non-RJE devices and used for file transfer. Indeed, 3780 emulator cards are available for IBM PC's to transfer files to/from (for example) an IBM mainframe. Moreover, through RPAD, it is possible to transfer files to/from other 3780 emulators (or equipped PCs) worldwide through an X.25 network such as SAPONET.

2. OVERVIEW OF RPAD/X.25 ENVIRNONMENT

Figure 1 illustrates the context in which RPAD is used in the X.25 environment. In the classical environment, X780 devices communicate with a host machine or similar X780 device by direct or switched (dial-up) connection using the BSC protocol. RPAD enables X780 devices and Host machines to communicate with each other across an X.25 network, thereby harnessing the advantages of packet switching, as well as widening connectivity.

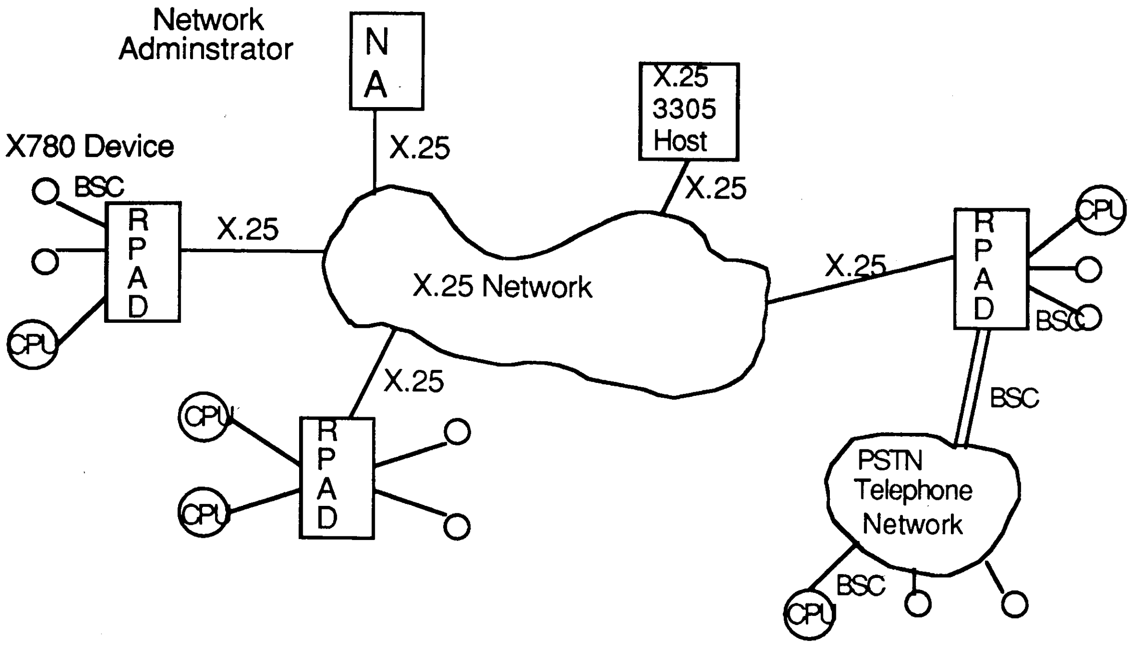


figure 1
RJE/X.25 Enviroment

The X780 device now communicates with RPAD using a direct (or switched) BSC connection (ie RPAD emulates an X780 device); RPAD maps the BSC procedures to X.25 and communicates with its remote peer RPAD using the 3305 [4] level 4 transport protocol; the remote RPAD maps the X.25 procedures back to BSC and communicates with the remote X780 device using a direct BSC connection (again, RPAD emulates an X780 device). Both X780 devices are unaware of the intervening network connection, thus obviating the need for any hardware or software modifications.

The functions of RPAD include:

- BSC communication with the X780 device
- X.25 communication with the network
- mapping BSC control procedures to X.25 according to the 3305 transport protocol
- BSC block to X.25 packet transformation
- X.25 packet to BSC block transformation

In addition, the particular PAD developed offers a number of network management facilities, including statistics, billing, and a centralised network management centre for network/PAD configuration, down-line (through X.25) loading of PAD software, billing and statistics reporting, and on-line network monitoring and.control.

3. THE TARGET HARDWARE SYSTEM

The target system is the Amdahl 4400 series of Network Concentrator (NC) and Network Administrator (NA). RPAD is essentially a suitably equipped (software) NC; the NA, an integral part of the system, is used for network/NC configuration, NC software generation, down-line loading of NC software, network control and monitoring, and billing and statistics.

3.1 Hardware

The basic PAD is a custom-built multilayer board comprising:

- 8 RS-232-C I/O ports
- 2 Z80 I/O processors
- Intel 8086 processor (BP or Block Processor)
- associated support and RAM/ROM

The 8086 processor (BP) performs the higher level protocol functions (eg X.25 packet level, BSC etc), while the two Z80 processors function as I/O processors (IOP), each servicing 4 ports (BSC or X.25).

The basic PAD is expandable to a 40 port unit by the addition of further so-called passive boards.

3.2 Software

Figure 2 illustrates the PAD software environment. Resident software includes:

- BP EXEC, the 8086 executive. Essentially a repeating loop, its main tasks are the despatch of BP processes for the 8086, buffer management, and timer services.
- IOP EXEC, the Z80 executives.
- X.25 utility BP processes: Packet Handler (XPH) and Link Handler (XLL).
- IOP processes: X.25, Async, BSC.
- PMR BP process, an interface (via X.25) to the Network Administrator.

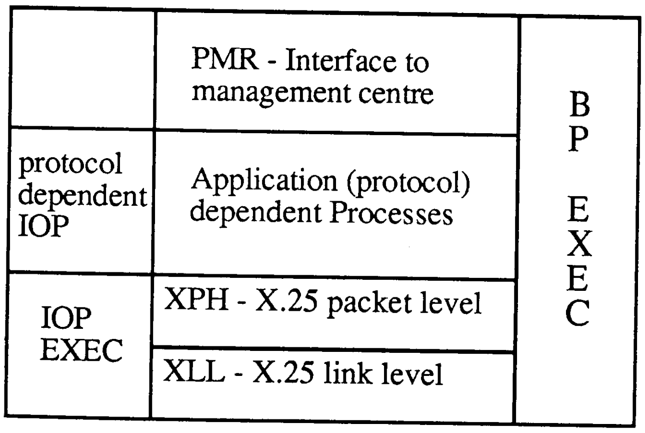


figure 2
PAD Software Environment

In addition to the IOP and BP EXECs, PMR and X.25 utility processes, protocol dependent software (IOP, and BP processes) is added to customise the PAD - eg to support Async, or SDLC, or 3270 BSC, or RJE BSC etc. This is indicated in Figure 2.

The RTX design thus amounted to the design of BP processes that would:

- perform 3780 BSC protocol function
- establish and clear calls through the X.25 network
- translate BSC to X.25 procedures and vice versa (according to 3305 protocol)
- interface with the network service provider (XPH process)
- provide a user interface for parameter negotiation, addressing etc.
- accumulate billing and statistics data
- communicate with the NA via PMR

Before discussing the design, it is instructive to examine the software environment a little more closely.

A single BP process executes on the 8086 block processor (BP) at any one time. Several processes are resident in memory, each dedicated to a very fixed function.

64K memory is available to all processes as shared memory for inter- process communication. This memory is divided up into 70-byte buffers which are managed by the BP EXEC (they are acquired and released by a process, ie a buffer is always owned by one specific process until it releases it). Inter-process communication is further effected by means of 16-byte messages (with tagged buffers), sent from the originating process to the destination process (via the BP EXEC).

BP EXEC maintains a despatch queue of inter-process messages, adds to the queue when a message is sent, and invokes the appropriate destination process when the message reaches the front of the FIFO queue. These inter-process messages (called worklist entries) have a standard format, identifying the sender process, the receiver process, an optional chain of buffers, the type of message and 6 bytes optional of information.

Interfaces between processes are kept clean by defining a small set of message types (events) allowed on each particular interface between two processes.

4. DESIGN, SPECIFICATION, IMPELEMENTATION AND TESTING

As detailed in section 3.2, the resident software of the PAD (independent of the protocol supported) includes:

- X.25 support (physical level: XLL, and packet level: XPH)
- the three executives
- an interface to a network management centre (PMR)

Supplementary to these basic processes residing in each PAD are the processes designed to support the specific protocol which the PAD offers to its user, in this case the RJE 3780 protocol.

4.1 Software Design

The software was designed subject to the following two considerations:

- A transport handler compatible with ISO standards was not feasible in this implementation. This resulted from the fact that RPAD interfaces not only to peer RPADs, but also to existing transport implementations in foreign PADs and IBM host/front-end processors. The transport protocol traditionally designed for higher level communications supporting BSC device connections, is the 3305 (also known as BPAD) protocol [4]
- A single protocol handler process, which functionally seems to be the cleanest design, gave rise to an extremely complex process. Since this single process would have to support both the 3780 protocol to the device, and the 3305 transport protocol to the peer PAD/host, its state machine model would be represented by the cross-product of the state machines of the two protocols.

For these reasons the protocol software of RPAD was designed as three distinct processes: two protocol handler processes (one each for 3780 BSC and 3305), and a user interface process. The following processes were thus defined :

- Device Manager (RDM): A multi-instance process emulating the 3780 protocol to the device. Each RDM handles a single device. It interfaces with the IOP resident process for data to/from the device, and to the transport handler for data to/from the network.
- Transport Handler (RTH): A single-instance process supporting the 3305 transport protocol to its peer on the remote side of the network. It interfaces with the RDM process on the device side and XPH (X.25 packet level process) on the network side. RTH also performs connection establishment and clearing, addressing, application hunting, block/packet transformations and error recovery procedures.
- User Interface (RUI): A single-instance process assisting with outgoing call establishment. RUI intercepts call establishment data from the user, allows for negotiation of certain attributes and eventually translates the user call establishment data from the user's format to the format expected by the 3305 protocol.

The design of the RTX processes and their inter-relationships are illustrated in Figure 3.

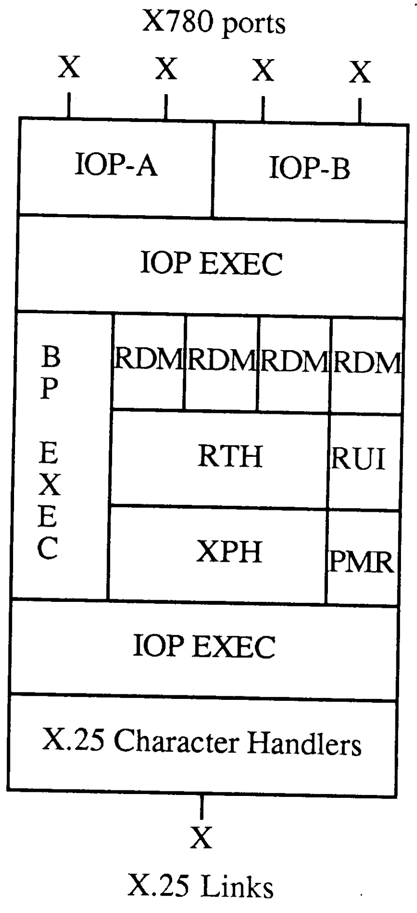


figure 3
RTX Software Architecture

4.2 Protocol Specification Methods

There is now almost universal recognition of the need for a thorough specification of a software product before proceeding to the coding/ implementation phase. In general, the effort involved in specification helps to clarify the nature of the problem at hand, and enhances the software system in terms of its modularity, portability, reliability, maintainability, etc. [1,2].

It is also worth noting that a good product specification greatly facilitates the coding effort. In terms of text-book descriptions of the software development process, completion of the product specification marks a point at which the systems analyst is able to disengage somewhat from the process, while the programmer then proceeds with coding [2].

4.2.1 Specification as a Top-down Process

In practice the level of specification may vary greatly, depending on the purpose for which it is done. At one extreme, the mere breakdown of the product into functional processes accompanied by a verbal description of each process function (as, for example, the functional units described in the preceding section (4.1)) may be regarded as a specification. The purpose of this specification is to clarify the way in which the major high-level functions of the product are to be achieved.

At a more detailed level, the variations in expected input to the (functionally determined) processes may be described together with the expected outputs. In communicating processes (be they sequential or concurrent), the time dependencies between input and output should also be carefully specified. The way in which this was done for the product at hand is set out in 4.2.2 below.

Frequently, such an input/output analysis may suggest how each functional process may be broken down into logical units or modules, each of which caters for its own level of functionality This level of specification for the product at hand is described in 4.2.3.

Finally, the way in which each smaller module achieves its functionality may be described. The level of detail given at this point should be sufficient for the programmer to commence his task. Clearly this will be determined by the resource environment in the sense that a programmer with a sound knowledge of the problem at hand may require only the briefest of detail, in contrast to an inexperienced novice. At any rate, it is somewhat Utopian to imagine that the systems analyst may be entirely removed from the scene after this level of specification has been provided. In a practical non-standard application (such as the product being described) a degree of post-specification interaction between systems analyst and programmer is inevitable. The approach taken to specification at this level is described in 4.2.4.

Each level of specification is accompanied by an associated verification process. Usually this is at a fairly informal level, and involves thinking about test-case inputs to the design which might render unacceptable outputs. However, if appropriate specification languages are used it may be possible to formally verify that each module, as well as the entire system, will react according to specification - ie yield predicted output for all possible inputs. This formal verification may also occur at the code level, where statements of the specification language serve as imbedded assertions in the code (in the form of comments) and are such that at each point an assertion can be shown to be valid provided that foregoing assertions are valid. At the limit, the formal verification process may be automated [3].

During the project at hand, no large-scale formal verification proofs were attempted, although loop-invariants were sometimes defined and used in manual exercises to prove the correctness of particularly complex sections of code. Neither was\a formal specification language used to specify the product. In retrospect, it would have been both interesting and challenging to have done a detailed specification of the product in terms of LOTOS, the specification language proposed by ISO FDT subgroup C. In [6] an introduction to LOTOS is given, together with an outline of how aspects of the product might be specified in this language.

4.2.2 Message Flow Specification

Expected inputs (and outputs) to the RPAD functional processes (RUI, RTH and RDM) are inter-process messages whose format and meaning depend, on the one hand on the hardware/software environment, and on the other hand, on the protocol being implemented. Syntactically they appear as so-called worklist entries - ie a specific data-structure imposed by the operating system which also imposes a semantic interpretation on certain fields of the structure. The residual semantic interpretation of each worklist entry is determined by the product designers, so that each worklist entry acquires a unique name and code combination to designate its meaning (eg Connect Request <2-3.01>).

These worklist entries are limited in number so that the specification challenge is not so much the determination of expected inputs and outputs, but resolving the inter-relationship between them. (Note, however, that it is obviously important to list and document each possible worklist entry, together with a description of its meaning, as part of the specification.) The asynchronous full-duplex nature of the communication renders it particularly important to precisely specify the time-dependencies between input and output in a clear but concise manner. The specification method explained below was considered particularly appropriate for this purpose.

The method starts with the identification of discrete protocol functions (eg connection establishment for successful autocall, connection establishment for unsuccessful autocall, etc.). Each such function involves the exchange of messages between two entities (A and B) representing the two communicating RPADS. These entities in turn contain processes which exchange messages not only internally between themselves, but also between the local X780 device(s) and themselves. For the present purposes, these X780 devices will also be referred to as processes.

As a general comment, it should be noted that while the present design did not follow the precise layered design proposed by ISO [7] because of the peculiarities of the environment (eg the 3305 protocol is a non-ISO transport layer), the functional processes defined above are in fact layered, in the sense of being distinct entities which are oriented towards communicating with a peer entity. Hence, XPH communicates with a peer XPH on the other side of the network (or with an equivalent process if communication is through a device implementing the 3305 protocol) and 'services' its local RTH process. Similarly RTH communicates with a peer RTH (or equivalent process) on the other side of the network, and services RDM.

At this point, there is a deviation from the conventional layered model, in that RDM may be regarded as being divided into two halves. The 'top' half of RDM communicates with its peer BSC process located in the local X780 device, while the 'bottom' half communicates with a peer 'bottom' half RDM across the network. The top half uses IOP to relay its messages, and the bottom half, the lower layer levels (RTH and XPH). (RUI and PMR are considered to be service processes.) By virtue of the above, there is a logical ordering of message flows through the processes, in that outbound messages from a device flow from device to IOP to RDM to RTH to XPH to network, and vice-versa for inbound messages. For the present discussion purposes, therefore, the highest layer process will be regarded as IOP, followed by RDM, RTH, then XPH.

Each protocol function may now be specified on a so-called message flow diagram, illustrated in figure 4. In such a diagram, entity A and its associated processes are represented on one side of a page, and entity B with its processes on the other. Each process involved in the protocol function being described is represented by a vertical line drawn down the page. Such 'process lines' are ordered by drawing the highest layer process line of entity A on the extreme left, followed by the next highest layer process line, etc. The ordering for entity B's process lines is the mirror image of that of entity A on the right hand side of the page. The location of the process line for a service process is somewhat arbitrary, but should be as close as possible to the layered process with which communication is most frequent.

Message flows (ie the sending and receipt of worklist entries) are represented by horizontal lines (called message lines) between process lines, with an arrowhead designating the receiving process. Each message line is annotated by the unique worklist entry name and code combination, so that the type of message may be easily identified. The higher up on the diagram that a message line appears, the earlier in time the message is considered to have been sent. However, there is no particular time scale associated with the vertical axis, so that the only temporal statements which can be deduced from the diagram are of the type 'message P was sent (received) before (or after or simultaneously with) message Q'.

A vertical line alongside a process line is used to designate a timer which starts at the highest point of the line and elapses at the lowest point (where an arrowhead is placed). Further brief annotations are permitted on the diagram to remove possible ambiguous interpretations. (Eg 'ignore' to denote that a process has ignored a message.)

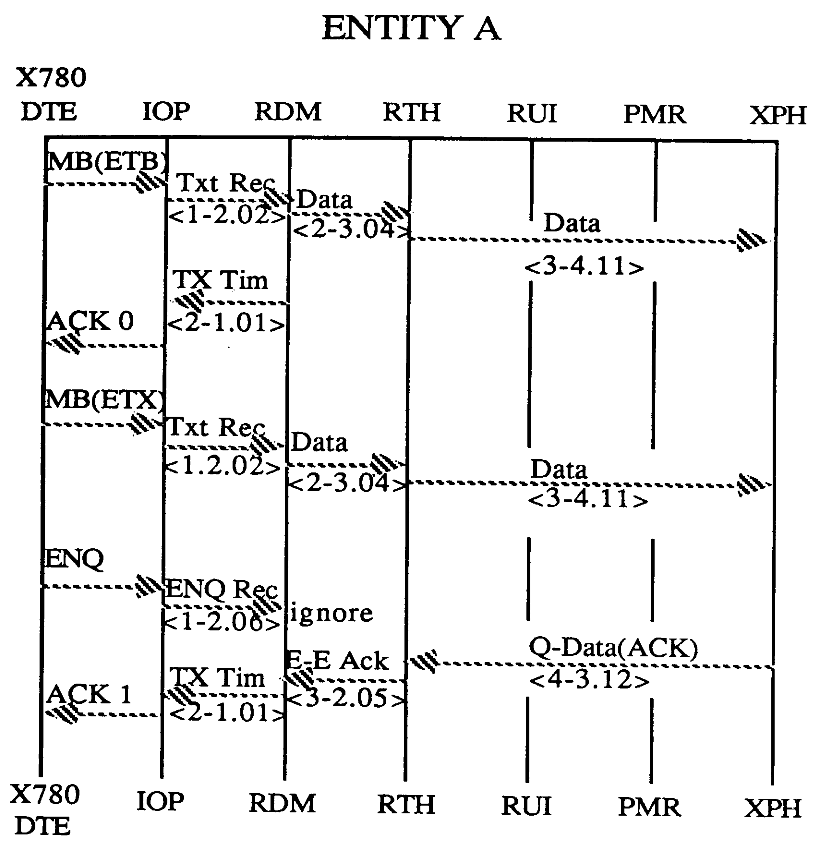


figure 4

Message Flow Diagram

4.2.3 State Diagrams

While message flow diagrams give a good visual representation of the temporal relationship between the messages which are exchanged, they do not provide clear guidelines for further decomposition of the processes involved. State diagrams fill this vacuum for the protocol-handling processes, namely RTH (handling the 3305 protocol) and RDM (handling the BSC protocol).

In a state diagram each node (or state) may be viewed as a distinct submodule. A transition to a new (or possibly the same) node is indicated by a directed arc, labelled by the message sent out by the originating node. Note that it is not implied that the destination node receives this message; rather, the destination node indicates which submodule will handle the next message received by the process. The actual destination of the message is apparent in the message flow diagrams, not in the state diagrams.

There is a relatively simple heuristic method for moving from the message flow diagrams to a set of state diagrams. (Note, however, that other equivalent state diagrams might be possible.) This heuristic is encapsulated in the following propositions which hold (with some exceptions) :

a) Each process in the present implementation runs to completion. Hence, once it has received a message it cannot be interrupted by another message.
b) Because of proposition a) a process may be considered to be in a given state until it sends out the last of its messages to non-service processes. Usually one and only one such message is sent out.
c) By virtue of propositions a) and b), a process line in a message flow diagram may be

divided into segments which end just after a message line from that process to a non-service process. These segments represent process states.

d) Within each segment determined in c) there will ordinarily be only one message line from a non-service process entering the segment, followed at a later point in time by only one message line from the segment to another non-service process. These message lines together with the state of the previous segment provide a firm guide to the state of the current segment.

e) Exceptions to the propositions a) to d) occur when, for example, an incoming message is ignored by a process. Such situations are dealt with on an ad hoc basis.

In this way, state diagrams were deduced from message flow diagrams for the aforementioned processes. It is interesting to note that in the case of RTH, the full 3305 protocol was not implemented, but a subset of the protocol which would still respond sensibly to other existing implementations (eg the Comm-Pro 3305 implementation) at a remote location. Hence, the state diagrams obtained for the present implementation differ slightly from the published version [4].

These state diagrams could form the basis for describing the state of an entire communicating system (including all non-service processes within two communicating RPADS together with the state of the communications channel). Tools exist for examining the progression of communicating systems from one such global state to the next to identify potential deadlocks, non-reachable global states, etc. [10]. Without these tools, verification for these conditions was done manually on a per process basis.

The fact that ISO FDT subgroup B has focused on the specification of protocols in terms of state diagrams demonstrates the wide acceptance that this method of specification has gained. Indeed, a Pascal-like language called Estelle has been designed to specify the semantics of a state diagram representation [9]. In the present study, specification in this way was not considered.

Finally, it is worth noting that, although the RUI process is not state-driven, the message flow diagrams nevertheless provided a valuable guide to decomposition of this process into submodules.

4.2.4 Pseudo Code Specification

Each submodule was specified in terms of structured English or pseudo code. It was considered appropriate to standardise some aspects of this effort to improve readability and facilitate the coding of the product. These guidelines are now given:

For each of the functional processes, RDM, RTH and RUI, all significantly complex data-types were initially identified and diagrammatically represented.

Guided by the state diagrams, each functional process was divided into procedures. All procedures start with a p_; all global variables (ie global with respect to one of the functional processes RDM, RTH or RUI) start with a gv_; all global types start with a gt_ and all global constants start with gc_. An early proposal to include a numbering system within this naming convention was initially followed, but proved too cumbersome to maintain and was later dropped.

All procedures start with a prologue containing the following information: procedure name, date, specifier's name, a brief abstract describing the procedure's functions and peculiarities, a list of called and calling procedures, the inputs catered for by this procedure and the outputs generated. Apart from the usual conventions of indentation, further specification was left to the discretion of the specifier.

The documents produced along these lines formed the basis for the program specifications. These specifications were heavily relied upon during the coding. They were such that they could be used by a small team of selected students who, under project-team guidance, assisted in the less complex coding areas.

There was some debate as to whether these program specifications should be amended to reflect adaptations made during coding. The motivation for doing this would be to provide a reliable entry-point for maintenance programmers at a future date. In the present context it was felt that the specifications up to state diagram level, together with the actual code (itself clearly documented by comments reflecting at least part of the program specifications) provide an adequate entry-point. This was motivated by the fact that those likely to do maintenance (personnel not involved in development) were already sufficiently informed of the hardware and software design of the product, had developed similar products, and would therefore find a program specification unnecessarily detailed. Subsequent experience, at a stage just prior to handing over the product during which the maintenance programmers were afforded the opportunity of acquainting themselves with the product, tended to vindicate this view. However, there may well be circumstances in which an up-to-date version of the program specifications should be provided.

4.3 Implementation

As previously mentioned, the protocols (more accurately the processes driving the protocols) were specified by means of state machines. From the pictorial representation defining the states, procedures were developed handling the cross-product of each state and each possible input message to the process. Although a substantial part of the input message set (process interface) of the process is invalid for any particular state, the code caters explicitly for each possible combination. Most of the input messages cause a branch to one of a much smaller set of error procedures.

Both the device manager (RDM) and the transport handler (RTH) are largely state driven, with the exception of a few functions (management, statistics and billing) which are independent of the protocol state machine. These functions are handled in separate procedures.

RUI is basically a parser and a translator. It takes the contents of the so-called 'Call Request Block' received from the user (in which several different options exist for the user to specify his connection destination and parameters) and fills out the required parts of the X.25 Call Request Packet, ie it fills out the fields with default information as well as with information explicitly supplied by the user. RUI provides all addressing translation information, except local and remote hunting (rotary destination lists) which is done by RTH.

The implementation language is largely PASCAL, chosen for features including programming ease, speed of development, debugging time and certain compiler features. Minor sections of the code are implemented in 8086 Assembler, mainly used for time-critical code and facilities not available in PASCAL (eg operations on register and stack contents).

The environment/hardware used for implementation includes the target hardware (as described earlier) and an Intel Microprocessor Development System (MDS). The MDS development provides a primary level of unit testing and debugging facilities before the code is transferred and tested on the target hardware. All code is developed, compiled and linked into loadable modules before it is transferred.

4.4 Testing

4.4.1 Testing Overview

The matter of adequate code testing tends to be sorely neglected in many software implementations. This is borne out by the proverbial scepticism towards computers so frequently encountered outside the world of computer-specialists. Reasons for this neglect may be found at many levels, some of which are suggested below:

- Firstly, there is the psychological disposition of the programmer who has applied himself with such devotion to his coding task, that he finds it difficult to imagine that he may have overlooked various logical errors.
- There is the fact that products tend to be developed against tight time schedules. Hence, when time-limits for the various phases are overrun, management may be tempted to cut back on testing time, rather than postpone the delivery date.
- Also, testing is not always the most exciting activity, and may be done in a shoddy fashion, merely to ' get the product off our back' .
- There is an art to designing tests, particularly tests to trace those paths through the code which only arise in the rarest of circumstances. Acquiring this art takes time and effort.
- It is extremely difficult to test communications software comprehensively, especially to

simulate the time inter-dependencies which arise from the asynchronous nature of communication in the real environment.
- Finally, test facilities are sometimes expensive, both in terms of man-hours, and in terms of hardware/software resources either to facilitate the test process, or to accurately reflect the environment in which the product is to function.

Clearly, in any system of significant size, a totally comprehensive test is simply not feasible. Even a formally verified system merely verifies correctness with respect to a specification which may itself be flawed in some way. However, even though the notion of what constitutes an adequate test procedure may be somewhat fuzzy, to ignore or underplay the importance of adequate testing is to run the risk of unreliable software. This must inevitably rebound on management in the form of dissatisfied clients, loss of reputation and loss of profits. Hence it is important that the software manager not only carefully budgets for testing (both in terms of time and cost), but that he insists that the testing be rigorously done.

There are several possible approaches to organising and scheduling tests. At the start of the project under discussion, several discrete test phases were identified and scheduled into the overall project plan. The test phases were designated by the following names, and scheduled to occur in the order given, once coding had been completed.

- Unit Testing
- Integration Testing
- Alpha Testing
- Acceptance Testing
- Beta Testing

The issues addressed during each of these test phases will now be discussed.

4.4.2 Unit Testing

Once a compilable module of code is affirmed to be syntactically error-free, it may be fed with a well-chosen set of test inputs and verified to produce expected output. In the MDS/PASCAL environment a particularly useful development tool was available for this type of debugging, known as PSCOPE-86. This affords the user such facilities as inserting breakpoints, single-stepping, high-level code patches, access to program symbols, etc. PSCOPE was used extensively during this phase of testing.

In general, unit tests are designed and carried out by the programmer responsible for the unit. However, in the case of one of the functional processes (RUI), an extensive set of test-cases was designed by an independent party. This was considered necessary because this particular process, being a user interface, is in practice likely to be subjected to a wide spectrum of unacceptable human inputs as its main source of stress.

4.4.3 Integration Testing

Integration testing commences when the unit-tested modules are ready to be integrated with one another and tested as a system or part thereof. Inter-process communication errors are sorted out during this phase of testing.

As with unit testing, the integration tests are designed and executed by the programmers. The purpose of this series of tests is to ensure clean interfaces between all RTX processes. They basically consist of putting all the relevant processes together into the RPAD, using an X780 device to feed in messages to the RPAD, examining the resulting messages on the network side, and vice-versa. Programmable line monitors are used to capture and generate messages on the network side.

There are two levels of interfacing problems which arise here.

In the first instance, the four processes specific to RTX (ie IOP, RDM, RTH and RUI) must have clean interfaces. Although the close liaison between members of the project team might appear to militate against errors at this level, testing inevitably brings to light several unforeseen problem areas. (As a general comment, it is noted that the fact of working in a small team greatly facilitated communication. This is in contrast to previous experience of large project teams liaising mainly through formal meetings and documentation, where the scope for misunderstandings and misinterpretations is increased.)

However, the foregoing processes form only part of the RTX software, the remainder consisting of the pre-existing operating system and utility processes. Because information about the interfaces to these processes is largely documentation-dependent, the scope for misunderstandings widens, and the consequent imperative of thorough testing increases.

4.4.4 Alpha Testing

In the alpha test procedures, the prime purpose is to verify that the implementation as described by the Functional Specification operates correctly. This means that all ' normal' modes of operation described by the specification should be exercised. Where the Functional Specification makes claims about dealing with abnormal conditions, tests for these conditions should also be devised.

The general approach is to try to identify so-called equivalence partitions [10] , each of which represents a set of scenarios which are roughly equivalent, such that if one scenario (or a representative sample of scenarios) from the equivalence partition has been tested, then others within this partition are likely to work as well. In deciding on sample choices to represent an equivalence partition, an attempt is made to incorporate the notion of boundary-value analysis - ie to choose scenarios which are in some sense ' at the fringes' of the equivalence partition [10] .

Ideally the alpha tests should be designed by an independent party not directly involved in the project. Since no such individual could be found, the design was left to a team-member who had not been heavily involved in programming or in unit and integration testing. An alpha test plan was set up, designed to test such aspects as user services, BSC interface, 3305 protocol support, network control and management, and the stress limits of the PAD. For each test in the alpha test plan, the purpose of each test was outlined, the recommended test procedure given, and the expected results of the test were stated.

The test environment should ideally allow for maximally configured RPADS to communicate through an X.25 network, using each of the members in the X780 family of devices. Monitoring and logging facilities should be available to analyse traffic at the external (BSC and X.25) interfaces, together with mechanisms to generate and/or delay test traffic at these points. Furthermore, it is desirable to have facilities for the easy re-configuration and swapping of equipment.

Since the actual facilities available to the project team fell short of these requirements, a number of improvisations had to be made to approximate the required test environment.

4.4.5 Acceptance Testing

This level of testing may not be required in a conventional development environment. However, in the present case, the product was to be marketed and maintained by the client. The latter party therefore felt the need to test the product at this stage. Furthermore, it had not been possible to carry out the complete set of alpha tests due to lack of local facilities. Hence, it was recommended that the alpha tests be completed at the client's site as part of the acceptance test procedures.

4.4.6 Beta Testing

The purpose of the beta tests is to test the product in an operational environment. The tests should ideally be performed by a potential client, with only minimal assistance from the project team members. This affords the opportunity of, inter alia, verifying that the user and system manuals are clear and comprehensible.

5. DOCUMENTATION

Approximately half of the time spent on the entire development effort was devoted to documentation. Thrashing out requirements with the client and drawing up an acceptable Requirements Specification are never easy tasks. Though fundamental to the usefulness of the developed end product, these tasks are often sadly neglected. Preceding any design, the functionality of the end product was reviewed and approved by both the client and the developers; another time-consuming but essential effort.

In any development effort of substantial size, it is imperative that the design is done soundly, and is divorced from syntactical complexities during the coding effort. Additionally, if the designers delegate the coding effort partly to assistants, the consistency of the design documentation is of fundamental importance for the duration of coding.

The hierarchy of development documents includes:

- Requirements specification (14 pages): A thin, factual document listing the features to be supported by the end product.
- Functional specification (166 pages): Detailed description of the features supported and functionality of the product. This document can be used as a basis for an operation manual (if required).
- Design specification (292 pages): Bulky, technically detailed document intended at the development and maintenance staff only. It details the implementation of each function as put forward in the functional specification. There is one global design specification for the entire project.
- Program specifications (RDM 154 pages, RTH 182 pages, RUI 57 pages): One document per process, detailing its implementation to the level of specification language. It contains all information required for the coding and maintenance of the process.

Although all the processes were initially largely developed in specification language, this proved to have served its purpose once coding was completed. It seemed unreasonable to maintain the specification language in the Program Specification document after coding and testing were completed. Well commented code alone should assist maintenance sufficiently and should be seen as part of the final Program Specification.

In addition to the above hierarchy of design documents, test specifications are supplied, listing the functions required of the product and test scenarios and data required for the testing of each function. A test specification was drawn up for each of the alpha and beta tests. Ultimately, these test specifications should be drawn up by an independent party, so as to ensure unbiased testing procedures.

Internally, during the development effort, loose points were documented as encountered (and distributed among the members of the team) using a set of design notes. This is a convenient way of unambiguously documenting changes affecting the design.

6. CONCLUSION

The most prominent conclusion, when thinking about a project like the RTX development, is that the size of the project was grossly underestimated.

The full development involved approximately 18 months, nearing two years if preparatory work preceding the official start of the project is taken into consideration. A full-time staff of three people were involved, with help from various other people only involved on a part-time basis for short durations of time during the development.

The initial discussion and assimilation of requirements, the preparation of the functional specification, the design and documentation of the design took approximately half of the overall time invested in the project. This was, however, time and effort well invested and paid off in the long run during the development. The design was clean, sound and unambiguous. Complex coding issues could be referred to the Design Specification without the need to have the document updated at any time during the coding and testing phases. It is of fundamental importance that the design issues of a large project should be divorced from the coding issues. Each in itself is complex enough to require the full concentration of the person involved at any specific time.

Complete and consistent documentation assisted in developing solid, reliable and well structured (maintainable) code which performed well under test conditions. It also contributed substantially to making accurate estimates during the project for the phases yet to be completed, and the meeting of set deadlines.

As an indication of the effort involved, the number of pages per document was given in the previous section. The final code size (in bytes object code) per process was:

- RDM: 14337
- RTH: 19975
- RUI: 8106

The size of the resident software is given by:

- IOP: 10633
- XPH (X.25 packet level): 12607
- PMR: 13392

The advantages of a small team (three people) were considerable in terms of (i) time saved because no scheduled meetings were necessary, (ii) effective and instant communication of design issues and/or changes, and (iii) scheduling of the development and testing resources for various individual needs during initial coding and unit testing phases.

One handicap of the development effort was the single-user MDS. This system was heavily used during the coding, compilation and unit testing phases (before the code was transferred to the target hardware) and tight time schedules resulted. Furthermore, the development operating system (ISIS on the MDS) and the operating system of the target hardware (CP/M during loading stages) differed. This resulted in some transfer incompatibilities and tools facilitating this transfer were developed.

7. REFERENCES

1. Bochman G. & Sunshine C., Formal Methods in Communication Protocol Design, *IEEE Transactions on Communications,* Vol. COM-28, **4**, April 1980, pp 624-631.
2. Gane C. & Sarson T., Structured Systems Analysis: Tools and Techniques, Englewood Cliffs, New Jersey: Prentice Hall, 1979.
3. Polak W., An Exercise in Automatic Program Verification, *IEEE Transactions on Software,* **SE-5**, No. 5, September 1979, pp 453-457.
4. GTE TELENET, A Packet Assembly/Disassembly Protocol Specification for Binary Synchronous Communications (BPAD/3305), April 1980.
5. ISO Information processing Systems - Open Systems Interconnection - LOTOS - A Formal description Technique Based on the Temporal Ordering of Observational Behaviour. March 1985.
6. Kourie D.G., A Partial RJE Pad Specification to Illustrate LOTOS. Presented at the 1985 Computer Science Lecturers' Association Conference (to appear in *QI*).
7. Day J.D. & Zimmerman H., The OSI Reference Model, *Proceedings of the IEEE,* **71**, No. 12, December 1983, pp 1334-1340.
8. Rudin H., An Introduction to Automated Protocol Validation, *Proceedings of the IFIP Conference on Data Communications,* September 1982.
9. Vissers C.A., Tenney R.L. & Bochman G.V., Formal Description Techniques, Proceedings of the IEEE, **71**, No. 12, December 1983, pp 1356-1364.
10. Myers G.J., The Art of Software Testing, New York: John Wiley & sons, 1979.

A PARTIAL *RJE* PAD SPECIFICATION TO ILLUSTRATE *LOTOS*

D G Kourie
Department of Computer Science
University of Pretoria, 0001 Pretoria

LOTOS (Language Of Temporal Ordering Specification) is employed to give a partial specification of a system to connect RJE devices across an X.25 network. The system's implementation has been described fully elsewhere [5]. The present purpose is to introduce LOTOS as a specification language, showing how fairly complex time-dependencies may be described in an unambiguous fashion, and also pointing to the way in which LOTOS specifications may be verified.

Key concepts of LOTOS are surveyed, and the underlying model which abstracts the RJE system to be specified is presented. The LOTOS specification of this model is given in outline, with particular emphasis on aspects of the connection phase. The type of verification to which a LOTOS specification may be subjected is briefly discussed and an indication is given of how such verification may be approached.

1.OVERVIEW OF *LOTOS*

1.1 Background

After several years of development (from 1981 to 1984), a draft proposal for the syntax and semantics of LOTOS [9] was produced by ISO FDT Subgroup C in March 1985. Currently, this proposal is undergoing minor revisions. A provisional LOTOS Tutorial has also been provided [12].

LOTOS derives from the seminal work of Milner who describes a Calculus for Communicating Systems (CCS) [15]. It is based on a modification of this formal mathematical calculus, referred to as CCS*. Data structures and value expressions (ie language expressions that describe data values) in LOTOS are represented in the same way as in ACT ONE - an abstract data type language described by Ehrig et. al. [6]. It should be noted that the currently available ISO draft proposal document [9] does not describe ACT ONE, but gives a self-contained though somewhat turgid description of CCS*.

An informal introduction to highlight some of the important CCS* concepts now follows in 1.2 to 1.7. This introduction also serves to describe LOTOS, in that the concepts defined in CCS* apply directly to LOTOS, and the way in which these concepts are represented in LOTOS differs only slightly from the CCS* representations. Some of the main differences are discussed in 1.8.

It should be mentioned that LOTOS bears a strong resembelance to CSP developed by Hoare [8], since both are influenced by Milner's work.

1.2 Processes and interactions

In CCS* processes are viewed as abstract entities which may interact with each other at abstract shared resources called interaction points. At such points primitive atomic synchronised interactions occur which are called events.

1.3 Behaviour Expressions

The behaviour of a process in CCS* is described by a so-called behaviour expression. This expression is primarily (but not exclusively) aimed at describing the way in which a process appears to its external environment. Essentially this means that a behaviour expression describes events and their temporal ordering at interaction points shared by a process and its external environment.

In general, a behaviour expression is built up by conjoining smaller behaviour expressions using operators described below. Hence, if B1 and B2 are behaviour expressions, and one of the legitimate CCS* operators, denoted by <op> is applied to B1 and B2 to obtain B1 <op> B2, then this represents a new behaviour expression, say B3 which may in turn be conjoined by some operator to another behaviour expression to obtain yet another behaviour expression, etc. The final behaviour expression thus obtained represents the description of some process of interest to the specifier, and is shown in CCS* by the notation :

p(x1,...,xn) := B

Here, p is called a behaviour identifier which gives a name to the process being described. x1,...,xn are variables which occur in the behaviour expressions used to describe the process, and B is the actual behaviour expression which describes the process. The representation p(x1,...,xn) := B is called a process abstraction.

By substituting value expressions E1,...,En for x1,...,xn in B, a specific instance of the behaviour expression B is obtained which is denoted by p(E1,...,En), and which is called a process instantiation. Such a process instantiation is itself a behaviour expression, which may be used as a building block in the construction of another, larger behaviour expression as described above.

1.4 'Atomic' Behaviour Expressions

At the lowest level of the aforementioned essentially recursive procedure to construct behaviour expressions, two 'atomic' building blocks for constructing behaviour expression are found, namely the behaviour expression stop, and a part of a behaviour expression called an action (or event offer) denotation.

The appearance of stop in a behaviour expression, B, signifies that the process described by B terminates at the point where stop is encountered. Of course, B may simply be a behaviour expression which has been conjoined into a larger behaviour expression, say B', so that the appearance of stop in B need not necessarily denote a point where the process described by B' terminates. Whether this is so or not will depend on the operators used to incorporate B into B', as well as on the other behaviour expressions which are incorporated into B'.

An action denotation is used to indicate the potential occurrence of an event. Such a denotation consists of a label, representing an interaction point, followed by zero or more value expressions, each preceded by an exclamation mark symbol. For example :

ip !expA1 !expA2

is an action denotation with label ip representing some interaction point and with expA1 and expA2 being value expressions. The way in which the potential event represented by this action denotation actually occurs is now discussed.

1.5 Process Synchronisation

Suppose the action denotation given above appears in the behaviour expression of a certain process (say A), and that another action denotation :

ip !expB1 !expB2

occurs in the behaviour expression describing another process (say B). If the value of expA1 = the value of expB1 and the value of expA2 = the value of expB2, then a matching (event) offer is said to have occurred at the interaction point labelled by ip. The action denotation appearing in A's behaviour expression signifies that A will wait at this point until a matching offer is made by some other process (ie B in the example at hand). When this occurs, then both processes involved continue to function, each possibly waiting at the next point at which an action denotation appears in its respective behaviour specification.

1.6 Input/Output Representation

At first sight it may appear that an action denotation as described above is simply a mechanism for describing process synchronisation. This is indeed one of its uses, but seen in conjunction with CCS* operators, it may also be used to describe more complex interactions. To describe such interactions the action-prefix operator (denoted by ;) and the choice operator (denoted by []) are now considered.

If a is an action denotation and B is a behaviour expression, then a;B is a behaviour expression which describes a process behaving exactly like the process which B describes, but only after the action denotation a has received a matching offer.

If B is the set of behaviour expressions {B1, B2,...,Bn}, then

B1 [] B2 [] ... [] Bn

is a behaviour expression which is identical to a non- deterministically chosen element of the set B. Note that an alternative way of writing this expression is by using the so-called summation operator (Σ), applied to B to get ΣB.

Suppose now that x is a variable which can assume values over a given domain s = {s1, s2,...,sn}. (in CCS* terminology x is said to have the sort s.)

The behaviour expression ip!s1;[s1/x]B describes a process which waits for a matching offer of value s1 at interaction point ip, and then behaves as described by the behaviour expression B in which all occurrences of x are substituted by s1. (This substitution is expressed by the [s1/x] notation.) Suppose that a process is meant to input a value for x at the interaction point ip, and then proceeds to function as described by the behaviour expression [si/x]B if the value of x turned out to be si (i = 1,...,n). This may be expressed in CCS* as :

ip!s1;[s1/x]B [] ip!s2;[s2/x]B [] ... [] ip!sn;[sn/x]B

This is clearly a rather cumbersome way of saying that a process must input a value for x, substitute this value for all occurrences of x in the behaviour expression B, and then behave as described by B. CCS* allows for a neater, but equivalent notation, namely ip?x:s;B. This is an example of a so-called extended action-prefix expression.

1.7 CCS* Operators

Apart from the action-prefix operator and the choice operator discussed above, CCS* allows for the following other operators to be applied to behaviour expressions : parallel composition (denoted by |), disabling (denoted by [>), restriction (denoted by \), relabelling, and guarding. These are now informally described.

Consider two behaviour expressions B1 and B2. B1 | B2 is a behaviour expression which describes a process where the processes described by B1 and B2 run independently and in parallel to one another. This means that the order in which subexpressions of B1 and B2 are interleaved cannot be stated a priori. However, the alleged independence of the processes described by B1 and B2 mentioned above must be qualified by the fact that they also will synchronise under the following circumstances : if the behaviour of the processes interleave in such a manner that at some point they are both ready to interact with the same event, then it is assumed that the interaction takes place (since this state of affairs implies that a matching event offer has occured). From the point of view of the environment external to these two processes, the interaction is not directly observed, so that the event that occurs is regarded by the environment as a so-called internal event (denoted by i).

B1 [> B2 is a behaviour expression which describes a process which functions exactly like the process described by B1, unless a matching offer occurs which starts B2 off. If this happens, then the process associated with B1 immediately halts (ie is disabled) and the process described by B2 continues to function to its termination.

Suppose B1 is a subexpression in a larger behaviour expression B2. Suppose too that A is a set of labels denoting interaction points where interactions to B1's external environment occur, but which are not part of B2's external environment. These interaction points and their associated labels are said to be hidden from B2. Such hiding is shown by means of the restriction operator applied to B2 and A thus : B2\A.

Frequently it is necessary to relabel the interaction points labelled within a behaviour expression. Suppose S is the relabelling function. Then B1[S] is the behaviour expression obtained from B1 by relabelling each label as per S.

Finally, suppose E is some boolean expression. The behaviour expression [E] -> B1 means that the associated process stops if E is false, and is described by B1 if E is true.

Parentheses are used to associate behaviour expressions with their appropriate operators, but may be omitted if allowed by a priority ordering defined on operators as follows : (The ordering is from highest to lowest priority) restriction and relabelling, action-prefix, guarding, choice-composition, parallel-composition, and finally disabling.

Hence, for behaviour expressions B1,...,Bn, the following holds : (B1 | B2) [> (B3 [] B4) is equivalent to B1 | B2 [> B3 [] B4.

1.8 LOTOS

The LOTOS syntax is described in [9] in terms of BNF notation. Syntactically and semantically the language constructs closely parallel representations used in CCS*. Some important differences are now described.

The enable operator (denoted by >>) is introduced at a lower priority than disabling. If applied to two behaviour expressions B1 and B2, then B1 >> B2 means that once the process that B1 describes has terminated successfully (which is denoted by the special process 'exit'), then the process that B2 describes will begin.

The equivalent of the CCS* | operator is a || symbol in LOTOS. However, LOTOS also allows for operators which lock out the possibility of synchronisation at selective gates. If these gates are a1,a2,...,an, then the operator |[a1,a2,...,an]| may be used. If synchronisation is excluded at all gates then the operator ||| is allowed. Note that these latter two operators imply non-determinism in the following sense : if two processes are in a potentially synchronising position (ie both prepared to interact through a mutually shared gate with the same event) and the environment offers that event at that gate, then one of the processes (chosen non-deterministically) will interact, and the other will not.

Interaction points in LOTOS are referred to as gates. One of the syntactically admissible forms of a process abstraction in LOTOS appears as follows :

p[a1,...,an](x1:t1,...,xm:tm) := B

where a1,...,an are gates at which interactions occur, and x1,...,xm are variables occurring in the associated behaviour expression indicated above by B. The variables x1,...,xm have sorts t1,...,tm respectively. A process instantiation for this process is p[a1,...,an](E1,...,Em), where E1,...,Em represent value expressions replacing corresponding occurrences of x1,...,xm in the associated behaviour expression. Hence LOTOS differs slightly from CCS* in its representation of process abstraction and process instantiation, in that the gates are explicitly shown in LOTOS, while the corresponding labels do not appear explicitly in the equivalent CCS* expression.

Note that the parenthesised parts of the above representations are optional so that a process abstraction may also appear as p[a1,...,an] := B, with the corresponding process instantiation being p[a1,...,an]. Process abstractions and process instantiations may also be more complex than those given above, but these forms are not discussed here.

LOTOS requires that all sorts, operations, equations be rigorously 'declared'. Loosely speaking LOTOS may be said to be 'strongly typed', in that the sort of all variables must be declared, together with operations which may be performed on these sorts. These matters are not dealt with in detail here, and while they are fully specified syntactically in the relevant draft proposal [9], the semantic explanation is given in [4]. However, earlier LOTOS proposals [10,11] will provide an informal insight as to how typing proceeds.

Other syntax requirements in LOTOS should be deducible from the example specification in 3. below. Note, for example, the occurrences of so-called local definition expressions, which are in actual fact process abstractions preceded by the keyword 'where'. These process abstractions define the process instantiations which occur 'locally' in a higher level process abstraction.

Note that comments in LOTOS are contained within the symbols (* and *).

2. A MODEL OF THE *RJE* SYSTEM

The RJE system to be specified is described by a process abstraction with behaviour identifier RTX. RTX consists of a number of communicating processes physically located in a hardware device known as an RPAD. The RPAD operates in under control of a (possibly remote) device called the network administrator (NA), which configures each RPAD in an X.25 network, gathers statistical data, enables and disables ports, etc. For the present purposes, a process within RTX will be posited which deals with most of the messages from the Network Administrator. It will be described by the process abstraction with behaviour identifier NA.

Two other processes within RTX are relevant to the present specification. The first is device manager which interfaces to a so-called X780 device. (Such a device may be any IBM 2770, 2780, 3770, 3780, 3740 or equivalent device emulating the BSC transmission protocol.) This process will be described by a process abstraction with behaviour identifier RDM. The second process is a 3305 transport handler, which is described by a behaviour abstraction with behaviour identifier RTH. (cf. [7] for a description of the 3305 protocol.)

The specification identified as RTX_SPEC describes how the processes NA, RDM and RTH interact with each other, as well as with their external environments. The single gate through which RDM and RTH interact is designated by b in the specification. RDM interacts with RTX's external environment through two gates designated a and na. RTH interacts with RTX's external environment through the single gate designated by c. NA also communicates with RTX's external environment through gate na. These gates and their environments are schematically shown in figure 1. Note that the fact that the NA interacts with other RTX processes in the actual implementation does not materially affect the present specification.

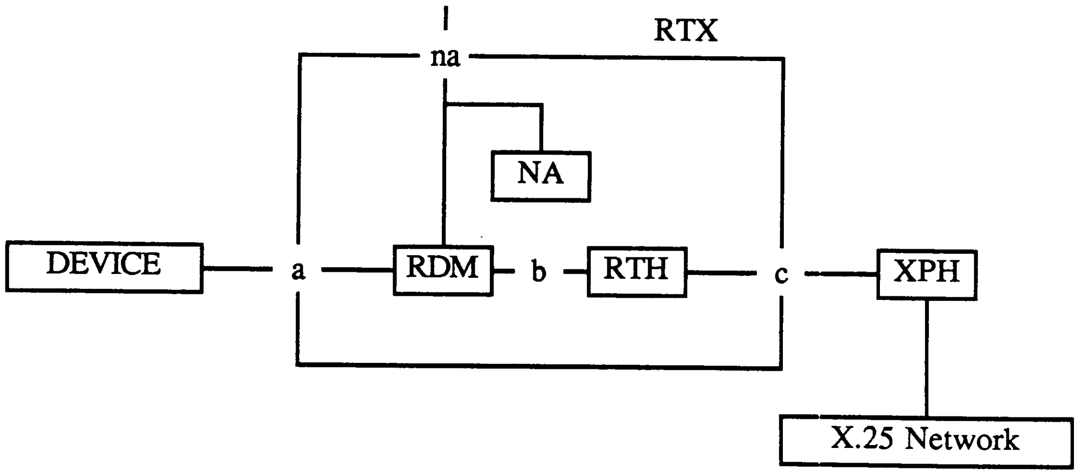


figure 1

RDM interaction through gate a is to the X780 device which communicates with a (remote) peer across an X.25 network using the BSC interface provided by the RPAD. Hence, the specification of the interactions at gate a amounts to a specification of the BSC protocol. Note that from an implementation point of view, a process is needed to assemble the serial incoming message-stream from the device into data structures which RDM accepts and, conversely, to serialise RDM's output to the device. This process is not relevant in the present specification.

RDM interaction through gate na only concerns two Network Administrator commands, namely those which enable and disable communication through gate a. Other interactions through this gate (relating for example to requests for billing or statistical data) do not impinge on the overall communication process, and may essentially be viewed as being handled independently and in parallel to interactions at other gates. These interactions are dealt with by the process NA.

RTH interaction through gate c is to the so-called XPH process, the X.25 packet handler process which ensures that messages received from RTH are appropriately translated into X.25 packets and sent to the X.25 network, and that X.25 messages received from the network are passed appropriately to RTH. XPH therefore provides the usual network services to the 3305 transport handler, RTH, and is not further specified below.

It should be noted that when the specification mentioned below is implemented, several implementation issues will need to be addressed. In particular, the set of peer X780 devices to which a given device can connect across the X.25 network has to be decided upon, and parameters which characterise the way in which communication is to take place may also have to be provided.

Hence, a device may be configured to connect in a so-called autocall mode, meaning that it always uses the same connection parameters relating to such matters as which application is to be accessed, whether the call will be reverse-charged or not, etc. Alternatively, the device may be configured to send a so-called Call Request Block (CRB) which prescribes what these parameters should be for a specific connection. The temporal ordering of messages between processes in general, and through gate a during connection establishment in particular, will differ according to which configuration option is used. In the specification below, an autocall configuration for the device is presumed.

Furthermore, the way in which a device connects to an application may vary. In some cases, the connection will always be to the same remote port address which offers the application. In this case, the application is designated as a mapped application. On the other hand, a device may be connected to any one of a set of ports offering the desired application. In the latter case the application is referred to as a session application. Which applications are to be characterised as mapped and which are to be considered as session applications is a configuration issue. Part of the function of RTH is to implement a reasonable algorithm to select an initial port offering a desired session application during connection set-up phase, and to re-select a port if a previous connection attempt was unsuccessful. Since RTH is not specified at a detailed level below, the matter of whether the application requested by the autocall device is of type session or mapped is not relevant in the present partial specification of RTX. A full specification of RTH would, however, have to accommodate these alternative application categories.

3. A PARTIAL *LOTOS* SPECIFICATION OF *RTX*

```
(*****************************************************************)
(* Two type definitions are given below. Each conform strictly *)
(* to LOTOS syntax.                                             *)
(* The first (process_messages) defines five groups of          *)
(* messages which may occur during interactions.                *)
(* The second (process operations) defines the various          *)
(* messages which may occur, categorised by sort-group to which*)
(* these messages belong.                                       *)
(* Note : LOTOS type syntax also provides for the               *)
(*      semantic definition of operators, using the             *)
(*      type definition for equations. Hence one could          *)
(*      define operations such as PUSH and POP on various       *)
(*      sort categories. This is part of ACT ONE. However,      *)
(*      no such definitions have been attempted here,           *)
(*      since such operations were not required for the         *)
(*      present specification.                                  *)
(*****************************************************************)
type process_messages
  is sorts dev_msg,
       rth_msg,
       xph_msg,
       na_msg,
       dev_int_msg
 endtype
```

```
  type process_operations
  is opns
     enq,     eot,
     ack0     : -> dev_msg

     connect_request,
     connect_confirm,
     disc_ind,
     disc_req   : -> rth_msg

     clear_ind,
     clear_req  : -> xph_msg

     bill_req,
     bill_respond,
     stats_req,
     stats_respond,
     disable,
     enable     : -> na_msg

     finish_enq,
     finish_eot,
     finish_error : -> dev_int_msg
 endtype

(***************************************************************)
(* Abstract : RTX                                              *)
(* This process allows for the interleaving of two major       *)
(* subprocesses. The first (NA) deals with all routine messages*)
(* from the network administrator. The second reponds          *)
(* appropriately to 'stray' messages at gates a and c,         *)
(* recursively returning to process RTX. However, it is also   *)
(* prepared to engage in an enable or disable event at gate na,*)
(* thereby transforming to an corresponding enabled or disabled*)
(* state.                                                      *)
(***************************************************************)
process RTX[a,c,na]
    :=
    NA[na]
    |||
    ( ((XPH_EVENT[c] [] DEV_EVENT[a]) >> RTX[a,c,na])
     []
     (na!disable; RTX_DISABLED[a,c,na])
     []
     (na!enable; RTX_ENABLED[a,c,na])
    )
where
(***************************************************************)
(* Abstract : NA                                               *)
(*  This process is partially specified.                       *)
(*  It suggests how responses would be made for billing and    *)
(*  statistics information, requested at the na gate.          *)
(***************************************************************)
process NA[na]
    :=
    ( (na!bill_req; na!bill_repond; exit)
     []
     (na!stats_req; na!stats_respond; exit)
     []
     :
     :
    ) >> NA[na]
 endproc (* NA *)
```

```
(*****************************************************************)
(* Abstract : XPH_EVENT                                          *)
(* This process deals with xph messages which occur before       *)
(* RTX is ready to connect.                                      *)
(*****************************************************************)
 process XPH_EVENT[c]
    :=
    c?xph:xph_msg;
       ([xph = clear_ind] -> exit
        []
        [xph <> clear_ind] -> c!clear_req; exit
       )
 endproc (* XPH_EVENT *)

(*****************************************************************)
(* Abstract : DEV_EVENT                                          *)
(* This process deals with device messages which occur before    *)
(* RTX is ready to connect.                                      *)
(*****************************************************************)
process DEV_EVENT[a])
    :=
    a?dev:dev_msg;
       ([dev = eot] -> exit
        []
        [dev <> eot] -> a!eot; exit
       )
 endproc (* DEV_EVENT *)

(*****************************************************************)
(* Abstract : RTX_DISABLED                                       *)
(* When RTX has been disabled, it must receive an enable         *)
(* message from the gate na before any connection may be         *)
(* established. Messages from gates a and c which arrive         *)
(* before this point are dealt with appropriately.               *)
(*****************************************************************)
process RTX_DISABLED[a,c,na]
    :=
     (na!enable; RTX_ENABLED[a,c,na])
     []
     (XPH_EVENT[c] [] DEV_EVENT[a]) >> RTX_DISABLED[a,c,na]
endproc (* RTX_DISABLED *)

(*****************************************************************)
(* Abstract : RTX_ENABLED                                        *)
(* Once RTX has been enabled its future description is given     *)
(* by the processes RDM and RTH functioning in parallel and      *)
(* sychronising with each other at gate b (which is hidden       *)
(* from the envirnoment of RTX). However, a disable event        *)
(* at gate na interrupts these processes, and places RTX back    *)
(* into a disabled state.                                        *)
(*****************************************************************)
process RTX_ENABLED[a,c,na]
    :=
    (RDM[a,b,na] |[b]| RTH[b,c])\[b]
    [>    (na!disable; RTX_DISABLED[a,c,na])
where
(*****************************************************************)
(* Abstract : RDM                                                *)
(*  RDM applies the choice operator to the two processes         *)
(*  RDM_CALLING (which deals with calls issued at gate a)        *)
(*  and RDM_CALLED (which deals with a call to gate a            *)
(*  from gate b)                                                 *)
(*****************************************************************)
```

```
process RDM[a,b,na]
   :=
   RDM_CALLING[a,b,na] [] RDM_CALLED[a,b,na]

where
(*************************************************************)
(* Abstract : RDM_CALLING                                    *)
(*  Here the first message at gate a is rejected (eot        *)
(*  at gate a if appropriate), and control is passed back to *)
(*  RDM, unless an enq is issued at gate a.                  *)
(*  In the latter case, a connect_request command is issued  *)
(*  at gate b, and CONNECT is invoked.                       *)
(*************************************************************)
process RDM_CALLING[a,b,na]
    :=
    a?dev:dev_msg;
    (
     [dev = eot] -> RDM[a,b,na]
     []
     [dev = enq] -> b!connect_request;CONNECT[a,b]
     []
     [dev <> enq and dev <> eot] -> a!eot; RDM[a,b,na]
    )
where
(*************************************************************)
(* Abstract : CONNECT                                        *)
(*  This process consists of two subprocesses which          *)
(*  synchronise through an internal gate (hidden from the    *)
(*  environment) called int. The first subprocess is called  *)
(*  DEV_MSG and is described below. The second subprocess    *)
(*  may either interact with an event at gate b and some     *)
(*  time later synchronise with an internal message from     *)
(*  DEV_MSG sent through gate int, its subsequent behaviour  *)
(*  then being described by CONNECT_RESPOND;                 *)
(*  However, the second subprocess might (alternatively)     *)
(*  be informed via gate int (before interaction through     *)
(*  gate b) that an error condition has occured. It then     *)
(*  rejects the next message from RTH (passed through gate   *)
(*  b) and returns to RDM.                                   *)
(*************************************************************)
process CONNECT[a,b]
    :=
    ( DEV_MSG[a,int]      |[int]|
     ( (b?rth:rth_msg;
        int?fin:dev_int_msg;
        CONNECT_RESPOND[a,b](rth,fin)
       )
       []
       (int!finish_error;
        b?rth:rth_msg;
        ([rth <> disc_ind] -> b!disc_req; RDM[a,b,na]\[na]
         []
         [rth = disc_ind] -> RDM[a,b,na]\[na]
        )
       )
     )
    )\[int]
```

```
where
(******************************************************************)
(* Abstract : DEV_MSG                                            *)
(*  This process describes the generation of an arbitrary        *)
(*  string of enq and eot messages of arbitrary length.          *)
(*  The string describes the sequence of interactions at         *)
(*  gate a which occur before synchronisation at gate int        *)
(*  becomes possible.                                            *)
(*  If the string length is 0, or if the string ends in an       *)
(*  enq, then a finish_enq message is used to synchronise at     *)
(*  gate int.                                                    *)
(*  If the string ends in an eot, then a finish_eot message      *)
(*  is used to synchronise at gate int.                          *)
(*  If, at any point, a message other than eot or enq is         *)
(*  issued at gate a, then the string generation process         *)
(*  stops and a message finish_error is used to synchronise      *)
(*  at gate int.                                                 *)
(*  Note that a subprocess DEV_MSG1 accounts for most of the     *)
(*  description of DEV_MSG.                                      *)
(*  Note also that after interaction at gate int, this           *)
(*  process deadlocks.                                           *)
(******************************************************************)
process DEV_MSG[a,int]
    :=
    (int!finish_enq;stop)
    []
    DEV_MSG1[a,int]

where
 process DEV_MSG1[a,int]
    :=
    a?dev:dev_msg;
    ([dev = eot] -> (DEV_MSG1[a,int]
                     []
                     (int!finish_eot;stop) )
    []
    [dev = enq] -> (DEV_MSG1[a,int]
                    []
                    (int!finish_enq;stop) )
    []
    [dev <> eot and dev <> enq] -> (a!eot;int!finish_error;stop)
  endproc (* DEV_MSG1 *)
endproc (* DEV_MSG *)

(******************************************************************)
(* Abstract : CONNECT_RESPOND                                     *)
(* This parameterised process must be invoked with specific       *)
(* values for rth and fin of the specified sort. If fin has       *)
(* value finish_enq, and rth has value connect_confirm then       *)
(* an ack0 is issued at gate a and the process is further         *)
(* described by DATA_TX. In all other cases, appropriate          *)
(* disconnection action is taken before returning to RDM.         *)
(******************************************************************)
process CONNECT_RESPOND[a,b](rth:rth_msg,fin:dev_int_msg)
    :=
    ([fin = finish_enq] ->
     ([rth = connect_confirm] -> a!ack0;DATA_TX[a,b]
      []
      [rth = disc_ind] -> a!eot;RDM[a,b,na]\[na]
      []
      [rth <> connect_confirm and rth <> disc_ind] ->
               ((a!eot;exit) ||| (b!disc_req;exit))>> RDM[a,b,na]\[na]
      )
      )
      []
      ([fin = finish_eot or fin = finish_error]
       ([rth <> disc_ind] -> b!disc_req; RDM[a,b,na]\[na]
       []
       [rth = disc_ind] -> RDM[a,b,na]\[na]
       )
      )
   )
where
(****************************************************************)
(* Abstract : DATA_TX                                          *)
(*  This process is not specified.                              *)
(*  It should describe the interactions which occur during     *)
(*  the data transfer phase once a connection has been          *)
(*  established.                                                *)
(****************************************************************)
process DATA_TX[a,b]
        :=
        ..........
endproc (* DATA_TX *)
endproc (* CONNECT_RESPOND *)
endproc (* CONNECT *)
endproc (* RDM_CALLING *)

(****************************************************************)
(* Abstract : RDM_CALLED                                        *)
(*  This process is not specified.                              *)
(*  It should describe the interactions which occur in order   *)
(*  to deal with an incoming call to the device.                *)
(****************************************************************)
process RDM_CALLED[a,b,na]
   :=
    ..........
endproc (* RDM_CALLED *)
endproc (* RDM *)

(****************************************************************)
(* Abstract : RTH                                               *)
(*  This process is not specified.                              *)
(*  It should describe the interactions which occur at gates   *)
(*  b and c - ie the inputs and outputs to the transport        *)
(*  handler.                                                    *)
(*  Note that any action denotation of the form b!rth           *)
(*  which occurs in this process should have a counterpart     *)
(*  action denotation of the form b?rth:rth_msg, or b!rth       *)
(*  in the above RDM processes.                                 *)
(****************************************************************)
process RTH[b,c]
   :=   ..........
endproc (* RTH *)
endproc (* RTH *)
endproc (* RTX_ENABLED *)
endproc (* RTX *)
endspec
```

4. VERIFICATION ISSUES

Verification is generally understood to mean the procedure of proving that some predicate (called a post-condition) will hold at the end of a program given that some other predicate (called a pre-condition) holds at the start. Indeed, if this proof can be carried out successfully, then an enunciation of the pre-condition and post-condition, together with a qualifying statement about conditions under which the program terminates, can be regarded as a correct specification of the program. These matters represent some of the most challenging aspects of Computer Science, and have been intensively studied, resulting in several systems which automate the proof procedure to prove sequential programs correct. (cf. [13,17]) The proof procedure becomes considerably more difficult in the case of concurrent programming, but studies of how verification may take place, and even be automated, in these cases have proceeded apace. (cf. [14,1])

However, a specification of a system in LOTOS is clearly different from a specification in the aforementioned sense, even if at some philosophical level the two types of specification are equivalent. (There seems, for example, to be some underlying commonality between the Modal Logic described by Manna and Pnueli [3] and the semantics of a LOTOS specification.) A LOTOS specification is in fact a specification of the temporal ordering of events at gates within a system, together with a description of what these events are. If a system is implemented in such a way that it exhibits the same behaviour towards its external environment (ie at its gates) as is prescribed by a LOTOS specification of the system, then the implementation may be deemed to be correct. Mechanisms for proving correctness at this level is not what is meant by verification in the present context. Rather, what is sought is a means of verifying that a LOTOS specification itself actually specifies what was intended. Indeed, the terminology used by Hoare [8] is perhaps somewhat clearer. He distinguishes between the definition of a system, (for example a CSP or LOTOS definition) the specification of the defintion (in terms of pre- and post- conditions) and the verification that the definition conforms to the specification. Often the specification is in terms of a statement about the nature of the so-called derivations (Hoare calls them traces) which characterise the system.

A system of inference rules are proposed in CCS* based on so-called action predicates. This provides an approach to identifying and generating derivations of a system defined in LOTOS. An example of an action predicate is the following statement :

The process described by B1 (a behaviour expression) interacts at an interaction point labelled g by virtue of a matching offer g!v1,...,vn arising at the interaction point, and then subsequently behaves as prescribed by the behaviour expression B2.

This action predicate may clearly be true or false, and is abbreviated to the notation :

B1 -gv1...vn-> B2

Note that v1,...,vn are values (eg 8, 10, 'string' etc.) and not value expressions (eg 4*2, x+5, etc.). gv1....vn is called an experiment .

An example of a CCS* inference rule, where a denotes some experiment and B1, B2 and B2' are behaviour expressions, is the following :

$$\frac{B2 \text{ -a-> } B2'}{B1 \text{ [> } B2 \text{ -a-> } B2'}$$

This states that if the action predicate B2 -a-> B2' is true, then the action predicate B1 [> B2 -a-> B2' is true. The inference rule in effect defines the semantics of the interrupt operator, [>. As an example of how this inference rule may be used, consider the behaviour abstraction RDM_ENABLED[a,b,na] given in the LOTOS specification in 3. above, where the interrupt operator is utilised. Appropriate translation of this LOTOS specification to CCS* (the translation procedure is described in [9]) will show that RDM_ENABLED describes a process which is always prepared to interact at the na gate with the event 'disable'. Should such an interaction be offered, the specification states that RDM_ENABLE will henceforth be described by the behaviour abstraction RDM_DISABLED, ie (using the notation [[B]] to designate the translation of the LOTOS expression B to its corresponding CCS* expression) :

[[RDM_ENABLED]] -na!disable-> [[RDM_DISABLED]]
since :
[[na!disable;RDM_DISABLED]] -na!disable-> [[RDM_DISABLED]]

This verification procedure may be forward-propagated by submitting another experiment to [[RDM_DISABLED]], deriving the resulting behaviour expression, and using the inference rules to draw further conclusions about what holds by virtue of the sequence of experiments. The sequence of experiments is referred to as a derivation. Hence, by examining a variety of derivations one may investigate whether they lead to legitimate (ie expected) behaviour expressions or not.

Because all concepts and inference rules in CCS* have been formally defined and because there is a formal relationship between CCS* and LOTOS, there appears to be a firm basis for formal verification of systems defined in LOTOS. Such verification would seek to establish that the set of derivations for a given definition conforms to expectations.

Note, however, that these are basically reachability properties which have also been studied from the finite state-machine point of view [18]. LOTOS does not currently provide for the expression and verification of fairness and liveliness properties (expressible in modal logic [3]), but these issues are being researched [16]. However, LOTOS does provide for the expression of non-determinism, a theme not emphasised in this paper. Verification of non-deterministic systems defined in LOTOS would involve examining not only the set of possible derivations, but also the set of those derivations which could also be refused by the system, by virtue of some non-deterministically chosen action.

5. CONCLUSION

It is interesting to note that, to date, LOTOS has been primarily used as a specification mechanism. (cf. [2] for a specification of the OSI Transport Service, and [4] for a specification of a Presentation Service.) These studies demonstrate the language's expressive power, its modularity and conciseness, all of which properties were specifically aimed for by LOTOS' designers.

In [2], mention is made of a project at Twente University to develop a number of LOTOS tools, including a syntax checker, single-step simulator, trace checker, specification integrator, test generator and possibly a specification compiler. At Pretoria University a syntax checker has been written [19] and progress is being made towards a trace generator.

It is to be anticipated that the future availability of such tools will stimulate a shift in the research focus from specification to verification issues, in which the formal well-defined character of LOTOS will play a central role.

REFERENCES

1. Babich A. F., .br; Proving Total Correctness of Parallel Programs. *IEEE Transactions on Software Engineering*, SE-5, No. 6, November 1979, pp558-574.
2. Brinksma E. and Karjoth G., A specification of the OSI transport service in LOTOS. *Proceedings of the IFIP WG 6.1 Fourth International Workshop on Protocol Specification, Testing, and Verification*, (Editors : Yemini Y., Strom R., and Yemini S.) North-Holland (1985).
3. Boyer R. S. and Moore J. S. (Editors), *The Correctness Problem In Computer Science*. Academic Press (1981) .
4. Carchiolo, V., Le Moli G., Palzzo S. and Pappalardo G. Modelling and Specifying a Presentation Protocol by Temporal Ordering. *Proceedings of the IFIP WG 6.1 Fourth International Workshop on Protocol Specification, Testing, and Verification*, (Editors : Yemini Y., Strom R., and Yemini S.) North- Holland (1985).
5. Carey C. W., Hattingh C., Kourie D. G., Van den Heever R. J. and Verkroost R. F, .br; The Development of

an RJE / X.25 PAD : A Case Study., Quæstiones Informaticæ, **4**, No 3, 1986.
6. Ehrig H., Fey W. and Hansen H., *ACT ONE : An Algebraic Specification Language with two levels of semantics*, Bericht-Nr. **83-03**, Technical University of Berlin (1983)
7. GTE TELENET, *A Packet Assembly / Disassembly Protocol Specification for Binary Synchronous Communications* (BPAD/3305), April 1980.
8. Hoare C.A.R., *Communicating Sequential Processes*, Prentica-Hall (1985).
9. *ISO* 8807, Information Processing Systems - Open Systems Interconnection, LOTOS - A Formal Description Technique Based on the Temporal Ordering of Observational Behaviour, March 1985
10. *ISO* TC97/SC16 WG 1 N299, Definition of the Temporal Ordering Specification Language LOTOS, May 1984.
11. *ISO* TC97/SC16 WG 1 N157, Draft Tutorial Document. Temporal Ordering Specification Language, January 1984.
12. *ISO* TC97/SC 21 N 938, Provisional LOTOS tutorial, November 1985.
13. Johnson S. D., *A Computer System for Checking Proofs*, UMI Research Press, Ann Arbor, Michigan 1982.
14. Lubechevsky B. D., An Approach to Automating the Verification of Compact Parallel Coordination Programs. (I). *Acta Informatica*, **21**, 1984, pp 125-169.
15. Milner R., *A Calculus of Communicating Systems*, Springer-Verlag. (1980) .
16. Parrow J. and Gustavsson R., Modelling Distributed Systems in an Extension of CCS with Infinite Experiments and Temporal Logic. *Proceedings of the IFIP WG 6.1 Fourth International Workshop on Protocol Specification, Testing, and Verification*, (Editors : Yemini Y., Strom R., and Yemini S.) North-Holland (1985).
17. Polak W., Compiler Specification and Verification, *Lecture Notes in Computer Science*, No 124, Springer-Verlag, 1981.
18. Rudin H., An Introduction to Automated Protocol Validation, *Proceedings of the Joint CSSA / IFIP Conference on Data Communications*, Johannesburg, September 1982.
19. Van der Vegte L., A LOTOS syntax checker, Paper presented at the 1986 South African Computer Science Lecturers' Association.

NOTES FOR CONTRIBUTORS

The purpose of the journal will be to publish original papers in any field of computing. Papers submitted may be research articles, review artilces and exploratory articles of general interest to readers of the journal. The preferred languages of the journal will be the congress languages of IFIP although papers in other languages will not be precluded.

Manuscripts should be submitted in triplicate to:

Prof. G. Wiechers
INFOPLAN
Private Bag 3002
Monument Park 0106
South Africa

Form of manuscript

Manuscripts should be in double-space typing on one side only of sheets of A4 size with wide margins. Manuscripts produced using the Apple Macintosh will be welcomed. Authors should write concisely.

The first page should include the article title (which should be brief), the author's name and affiliation and address. Each paper must be accompanied by an abstract less than 200 words which will be printed at the beginning of the paper, together with an appropriate key word list and a list of relevant Computing Review categories.

Tables and figures

Tables and figures should not be included in the text, although tables and figures should be referred to in the printed text. Tables should be typed on separate sheets and should be numbered consecutively and titled.

Figures should also be supplied on separate sheets, and each should be clearly identified on the back in pencil and the authors name and figure number. Original line drawings (not photocopies) should be submitted and should include all the relevant details. Drawings etc., should be submitted and should include all relevant details. Photographs as illustrations should be avoided if possible. If this cannot be avoided, glossy bromide prints are required.

Symbols

Mathematical and other symbols may be either handwritten or typewritten. Greek letters and unusual symbols should be identified in the margin. Distinction should be made between capital and lower case letters; between the letter O and zero; between the letter I, the number one and prime; between K and kappa.

References

References should be listed at the end of the manuscript in alphabetic order of the author's name, and cited in the text in square brackets. Journal references should be arranged thus:

1. Ashcroft E. and Manna Z., The Translation of 'GOTO' Programs to 'WHILE' programs., *Proceedings of IFIP Congress 71*, North-Holland, Amsterdam, 250-255, 1972.
2. Bohm C. and Jacopini G., Flow Diagrams, Turing Machines and Languages with only Two Formation Rules., *Comm. ACM*, **9**, 366-371, 1966.
3. Ginsburg S., Mathematical Theory of Context-free Languages, McGraw Hill, NewYork, 1966.

Proofs and reprints

Proofs will be sent to the author to ensure that the papers have been correctly typeset and *not* for the addition of new material or major amendment to the texts. Excessive alterations may be disallowed. Corrected proofs must be returned to the production manager within three days to minimize the risk of the author's contribution having to be held over to a later issue.

Only orginal papers will be accepted, and copyright in published papers will be vested in the publisher.

Letters

A section of "Letters to the Editor" (each limited to about 500 words) will provide a forum for discussion of recent problems.