CONTENTS

RESEARCH PAPERS
1-13 A Comprehensive Approach to Specification Languages
   J.K. STOEGERER
14-21 Protocol Translation for Packet Network Interconnection
   J.I. FERNANDEZ
22-26 Using the UNIX System on the AAEC Network
   G.W. PEADY

DISCUSSION PAPER
27-32 Database Retrieval Technology and Subject Access Principles
   G.W. GREENLEAF and R.A. CLARKE

SHORT COMMUNICATION
33-35 PREPOL: A Method to Convert a Mixed Data Set to all Numeric
   D.J. ABEL and W.T. WILLIAMS

SPECIAL FEATURES
35 Call for Papers Next Special Issue on Expert Systems
36-40 Book Reviews
Introducing the world's most powerful on-line transaction processing system: The Tandem NonStop TXP.

TANDEM COMPUTERS' powerful new NonStop TXP™ system significantly improves price/performance ratios in existing on-line applications, and can also be configured to handle hundreds of transactions per second with absolute data integrity. Based on a 32-bit multiple processor architecture, it delivers two to three times more processing power than the NonStop II™ system, reduces the cost per transaction by up to 50 percent, and maintains full software and hardware compatibility with NonStop II systems.

TANDEM Pty. Ltd.

3 Bowen Crescent,
Melbourne, Vic. 3004
Telephone: (03) 267 1577

22 Atchison Street,
St. Leonards, N.S.W. 2065
Telephone: (02) 438 4566

S.G.I.O. Building,
Cnr. Turbot & Albert Sts.,
Brisbane, Qld. 4000
Telephone: (07) 229 3766

3rd Floor,
151 South Terrace,
Adelaide, S.A. 5000
Telephone: (08) 212 1110

148 Adelaide Terrace,
Perth, W.A. 6000
Telephone: (09) 325 4044

20th Floor, William City Centre,
Boulcott Street,
Wellington, N.Z.
Telephone: (04) 72 3286
Advances in Production Management Systems
Edited by G. Doumeingts, Université de Bordeaux, France and W.A. Carter, C.A.M.I., U.S.A.
1984 xiv — 528 pages
Price: Dfl. 145.00/A$55.10
IFIP Price: Dfl. 108.75/A$41.40

The papers in this volume may be classified into four themes:
1) Design of Control Systems;
2) Design of Information Systems;
3) Implementation of automated production control systems;
4) Flexible Manufacturing Systems.

In addition to these topics, the book includes the state-of-the-art in production management presented from various points of view, by five authors from various countries covering the industrial world.

Security, IFIP/Sec '83
Edited by VIIVEKE FÅK, Linköping University, Sweden.
1983 xxxvi + 328 pages
Price: Dfl. 95.00/A$36.10
ISBN: 0-444-86698-8
IFIP-price: Dfl. 71.25/DM 27.10

This book represents the documentation of IFIP's first security conference. All contributions are written by professionals specifically for the conference and them show the true state-of-the-art in the field of data security. The book contains papers and discussions on security management, society and EDP security, risk management, auditing, cryptography, access management, society and EDP security, risk management, auditing and cryptography, access management, society, and EDP security. The book also includes papers discussing the development of systems for protocol implementation and testing that are being developed by a number of computer manufacturers and national laboratories.

Formal Description of Programming Concepts, II
Edited by DINES BJØRNER, Technical University of Denmark, Lyngby, Denmark.
1983 xii + 456 pages
Price: Dfl. 190.00/A$54.40
ISBN: 0-444-86619-1
IFIP-price: Dfl. 129.50/A$34.20

This volume presents 370 pages of 18 scientific papers, 21 pages of questions and answers in connection with 15 of these papers, and 42 pages of discussions in connection with 6 sessions in which 16 papers were delivered. The session and papers deal with:
- techniques and theories of programming language semantics definition and semantics directed compiler generation, both denotational and algebraic;
- mathematical models of functional programming languages;
- fixed-point, temporal-logic, algebraic, and structured operational semantics theories; definition and modelling techniques for communicating and concurrent, parallel and non-deterministic languages;
- denotational models and modal/dynamic logic investigations into database systems;
- general models of proof systems, algebraically specified abstract data types, and term-rewriting systems;
- petri-net-based graph models for concurrency and event-structure (Petri/Scott) models for observable concurrency.

CAD Systems Framework
Edited by KETIL BQ, SIU, Norway and FRANK M. JILLEHAGEN, ICAN A/S, Norway.
1983 x + 342 pages
Price: Dfl. 115.00/A$43.70
IFIP-price: Dfl. 86.25/DM 32.80

This work focuses attention on the design of CAD systems. The earlier books based on IFIP WG 5.2 conferences analyse the different methods/areas of CAD, such as: databases, man-machine interaction, artificial intelligence, system architecture. This book, CAD Systems Framework, is a synthesis of the experiences from the earlier conferences, and is organized in six major areas. Each presentation is followed by a discussion. For each of the areas international experts discuss the state-of-the-art and new aspects of modern CAD systems design based on design theory, frameworks and well defined modules.

Satellite and Computer Communications
Edited by JEAN-Louis GRANGE, NADIR Project, Rocquencourt, France.
1983 xii + 360 pages
Price: Dfl. 120.00/A$45.60
IFIP-price: Dfl. 90.00/A$34.20

Recent advances in electronic and space technologies have allowed satellite systems to be designed to provide new facilities for business communications including computer communications.

This book is a result of an international symposium sponsored by IFIP Technical Committee 6, and contains twenty-seven papers from highly qualified experts from more than ten different countries. Contributions in this Proceedings cover such aspects as systems design, experimentation, measurements, as well as performance evaluation using simulation tools or analytical models.

Protocol Specification, Testing and Verification, III
Edited by HARRY RUDIN, IBM Zurich Research Laboratory, Rüschikon, Switzerland and COLIN H. WEST, IBM Corporation, Armonk, NY 10504, USA.
1983 x + 532 pages
Price: Dfl. 170.00/A$64.60
ISBN: 0-444-86769-4
IFIP-price: Dfl. 127.50/A$48.50

This book is a collection of recent papers on specification, verification, implementation and testing of protocols. The more theoretical papers address topics including temporal logic, petri nets, and other formal techniques for protocol specification and verification. A number of papers discuss the development of systems for protocol implementation and testing that are being developed by a number of computer manufacturers and national laboratories.

Information Systems Design Methodologies: A Comparative Review
1982 x + 648 pages
Price: Dfl. 170.00/A$64.60
IFIP-price: Dfl. 127.50/A$48.50

There is a lack of comparative information concerning the abundance of information systems design methodologies. This book presents a series of thirteen methodologies applied to one and the same non-trivial problem case that deals with the question how to support a working conference organized by an IFIP working group. By actually applying a specific information systems design methodology to this case, its strong and weak points emerge. Each contribution gives an in-depth treatment of the background, development and performance of the methodology. As such, the book presents a very up-to-date survey and assessment of available, real world,
Information Systems Design Methodologies

A Feature Analysis
1983 x + 266 pages
Price: Dfl. 100.00/A$38.00
ISBN 0-444-86705-8
IFIP-price: Dfl. 75.00/A$28.50

This proceedings contains several papers each, of which analyses the capabilities and features provided in many widely used information systems design methodologies. Each paper treats the analysis from a different perspective. Most of the methodologies analysed are presented in a preface to this book entitled "Information Systems Design Methodologies: A Comparative Review". A study of the papers in this book should leave readers with valuable insight into the similarities and differences between methodologies.

Systems Design For, With, and By the Users
Proceedings of the IFIP WG 9.1 Working Conference on Systems Design for, with, and by the Users, Riva del Sole, Italy, 20-24 September, 1982
1983 x + 424 pages
Price: Dfl. 130.00/A$49.40
IFIP-price: Dfl. 97.50/A$34.20

The contributions of this book (papers and working group reports) are assembled under the common denominator to reflect participation problems in computer systems design. In this context, participation predominantly but not exclusively means the participation of workers and their organizations in the process of computer systems design. The emphasis thereby is laid on the participation of the most numerous group of the users, the workers, who are not users but used in systems design. The papers comprise a wide range of approaches towards participatory systems design. In addition to the papers presented, the volume contains the reports of the working groups, some of them with explicit recommendations to the IFIP community and other parties concerned.

VLSI '83
Proceedings of the IFIP TC10/WG 10.5 International Conference on Very Large Scale Integration, Trondheim, Norway, 16-19 August, 1983
edited by F. ANCEAU, University of Grenoble, France and E.J. AAS, University of Trondheim, Norway
1983 x + 468 pages
Price: Dfl. 145.00/A$55.10
ISBN 0-444-86751-1
IFIP-price: Dfl. 108.75/A$41.40

This book describes how the advances of integrated circuits technology to VLSI have a major influence on the design of computer and other digital systems. This influence will take several forms which deal with the interaction between semi-conductor fabrication and digital system design in the fields of new computer-based design tools for VLSI, special purpose VLSI systems and mathematical modelling of VLSI circuits and systems.

Computer Hardware Description Languages and their Applications
edited by T. UEHARA, Fujitsu Laboratories Ltd., Japan, and M. BARBACCI, Carnegie-Mellon University, U.S.A.
1983 x + 244 pages
Price: Dfl. 100.00/A$34.20
ISBN 0-444-86833-7
IFIP-price: Dfl. 75.00/A$25.70

The quality of the material submitted for this book speaks highly of the state-of-the-art in the field. The trend towards more formal aspects of CHDLs continues. One third of the papers presented here deal with formal verification aspects. Other topics include the use of formal descriptions in the synthesis of digital systems as well as the design of new hardware description languages.

Main features of this work are:
- Verification (Formal Techniques, etc.)
- Synthesis (VLSI Systolic Design, etc.)
- New CHDLs and Special Simulation Tools

Beyond Productivity: Information Systems Development for Organizational Effectiveness
edited by TH.M.A. BEMELMANS, University of Technology, Eindhoven, The Netherlands
1984 about 450 pages
Price: Dfl. 145.00/A$55.10
ISBN 0-444-86832-9
IFIP-price: Dfl. 108.75/A$41.50

The proceedings address the problem of designing and implementing information systems that really support organization effectiveness. Developing an information system is basically a modelling process from several perspectives. Therefore all contributions in this book are dealing with a multiperspective view. Especially the organizational, social, political, psychological and systems theoretical viewpoints are discussed. The ideas of philosophy behind all articles is that the design of an information system demands for multi-disciplinary or even interdisciplinary design, capabilities in which informatics of computer science is only one of the necessary disciplines.

Main features of this work are:
- The extended scope of information system design, with a main emphasis on the very early and extremely difficult steps in IS-design
- The operational ideas, methods and techniques which are described
- The multi-disciplinary approach (which is not only relevant for computer science specialists but also for disciplines such as psychology, social sciences, organizational theory and systems theory)

25% DISCOUNT ORDER FORM FOR IFIP-PUBLICATIONS

Please send me the following book(s);
Qty Author/Title ISBN IFIP-members price:

□ Please add my name to your mailing list.
Specific field of interest

□ Payment enclosed:
(Bank Draft/Eurocheque/International Money Order/Personal Cheque/Postal Cheque)

□ Charge my credit card:
□ Master Card □ Diners □ Bankcard □ American Express □ VISA
Card No.____________________________Valid until___________ Date________________ Signature

NOTE: New members will be charged at bank conversion rate Dfl. Non Discount price.
A Comprehensive Approach to Specification Languages

By J. K. Stoegerer†

Specification languages in their various forms provide a basis for a formalised, well engineered specification process and a foundation for the application of computerised tools. Due to their application within the development process three major types of specification languages — requirements specification languages (RSL), design specification languages (DSL) and program design languages (PDL) are distinguished. Three major goals for specification languages are postulated from which a comprehensive set of principles is derived. Finally, we give a very short overview of the most important specification languages and propose some likely future directions.

Keywords and phrases: software specification, specification languages, requirements specification languages, design specification languages, program specification, software development tools, non-procedural languages, automated analysis tools.


1. INTRODUCTION

The fundamental activity during software development is the production of various descriptions of the target system. The start of this process may be a very abstract and vague definition of the problem which is refined and elaborated in successive phases. More and more information is added, the level of detail increases and typically the process culminates in the production of a collection of executable computer programs with some accompanying documents (user manuals, maintenance guidelines, ...). These successive descriptions differ fundamentally in their goals, vocabulary and form of presentation. This paper is concerned with languages which can be used to describe a target software system from the first phases of requirements definition up to the physical specification of program design. As a basis for our discussion we will use the software life cycle as shown in Figure 1.

The initial phase of requirements engineering is brought about by a need of the user/customer for which a computerised information processing system seems to be the answer. The identification of user requirements comprises functional and qualitative descriptions of the system as described by Yeh and Zave (1980). The system concept along with financial, structural and performance constraints is determined. Based on this concept a development plan can be worked out and, based on a cost/benefit analysis, the project steering committee can decide whether or not to proceed to the logical design phase. Typically, the users, managers and systems analysts are involved in this phase. Thus, a requirements specification may be characterised as a problem oriented description of a (software) system. We will term languages that can be used best in this phase as requirements specification languages (RSL).

In the next phase, working from the requirements, an overall or architectural design of the target system is devised. The system functions, the major system components and their relationships are identified. In the former phase, the systems analyst (or problem definer) has specified what the system is intended to do. In contrast, a design characterises how this is to be achieved (see e.g. Enos and Van Tilburg, 1981). Thus, a design specification may be characterised as a solution oriented description of a software system and we will call the languages used to specify implementation independent design information design specification languages (DSL).

In the physical design phase the overall and rough structures are refined. Algorithms and data structures are developed and precise interfaces between the various modules are established. The vocabulary of such a description is implementation or software domain oriented. Typically, statements in such a language are specifically related to data structure specification and properties of module activity and interaction. We will call them program design languages (PDL). Several levels of refinement may be necessary and this phase stops when there is a precise specification of all modules from which the programming process can start.

As is seen in Figure 1, the programming activity is central to the whole development process and can be considered as the axis of reflection between definition/design and test phases. Figure 1 suggests a top-down design and a bottom-up validation approach. After the elementary modules are tested and debugged (module test phase) they are integrated into subsystems and finally these are combined to form the target system (integration/system test phase). Before the resulting product is handed over to the user it is tested in an operational environment (operational test). After its installation the delivered system has to be maintained. This includes modifications of existing functions, location and correction of residual errors, addition of new features and performance tuning (adaptive, corrective and perfective maintenance).

Copyright ©1984, Australian Computer Society Inc.

General permission to republish, but not for profit, all or part of this material is granted, provided that ACJ's copyright notice is given and that reference is made to the publication, to its date of issue, and to the fact that reprinting privileges were granted by permission of the Australian Computer Society.

†Department of Computer Science, University of Melbourne, Parkville, Vic. 3052; present address: Institute fuer Informationsverarbeitung, Tu Graz, Schlesstatgasse 4a, 8010 Graz, Austria. Manuscript received June 1982, revised June 1983.

2. DESIRABLE FEATURES OF SPECIFICATION LANGUAGES

Specifications may be stated in various forms. Ideally, one would think that any text in a natural language should be acceptable. However, firstly, these systems face tremendous technical problems and are still considered to be in their infancy. Secondly, specifications in natural language might become too voluminous and hence, comprehensibility and understandability is hindered rather than increased. Another alternative is the formal language consisting of predefined words and phrases with pre-assigned meanings. An example are the relational languages such as PSL (Teichroew and Hershey, 1977a) and RSL (Bell, Bixler and Dyer, 1977). These languages have specific structures, are relatively easy to use and implement and offer reduced complexity for resolving errors, omissions, etc. A third alternative is the specialised language consisting of predefined words and phrases with pre-assigned meanings. An example are the relational languages such as PSL (Teichroew and Hershey, 1977a) and RSL (Bell, Bixler and Dyer, 1977). These languages have specific structures, are relatively easy to use and implement and offer reduced complexity for resolving errors, omissions, etc. A third alternative is the specialised language consisting of predefined words and phrases with pre-assigned meanings.

Within our context and despite the differences between RSL, DSL and PDL we postulate three common major goals for specification languages (see also Balzer and Goldman, 1979).

1. Understandability: specifications have to serve different audiences as mentioned above. When information must be transferred and understood across an interface of widely different activities and people, poor communication may occur ultimately causing lost time and excessive costs. Therefore specifications must be clearly and unambiguously understandable by all parties.

2. Analysability: this should enable us to predict the impact of design decisions, to simulate system behaviour, to ascertain the equivalence of the various descriptions (implementation — physical/logical design specification — requirements definition) and the consistency and validity within each specification. Because of improved analysis techniques it should be possible in the long run to produce automatically efficient computer programs directly from a requirements statement simply by adding some necessary information at the subsequent levels.

3. Maintainability: programs and their specifications have to cope with changes over time which arise from both inside and outside of the development process. For example, within an IBM organisation it was detected that the average project experiences a 25% change in requirements alone during its period of development (see Climis, 1979).

From these three major goals it is possible to derive a set of principles for specification languages which are described below. In the following, four categories A, B, C and D are introduced to establish a framework. These categories are strongly related to each other and do overlap (e.g. modularisation is dealt with in the understandability category although this could also be treated in the maintainability category. In fact, one could argue that understandability itself is a subcategory of maintainability as was done for example in the software quality characteristics tree by Boehm, Brown and Lipow, 1976). Thus, the categorization presented below should not be taken in the literal sense and, besides language questions, environmental features are sometimes considered also.

A. Basic Description Facilities

1. Language basis: Existing specification languages are based on a variety of concepts. We distinguish between model-based, representation-based and methodology-based languages, a criterion which was used first by Smith and Tripp (1981) to survey software design tools. There are a variety of formal models which can be used as a basis:
The actions within a real-time system might occur either sequentially or concurrently. As Wirth has noted there is a need for describing sequences of actions which are the focus of consideration for real-time systems. Real-time systems are mostly obtained by implementing the target software system in a multiprocessor environment or not (implementation in a co-ordinated manner). System behaviour might be affected by the relative timing among actions. A real-time software system is one in which the relative speed or duration of some activity is relevant to the behaviour of the system. The actions within a real-time system might occur either sequentially or concurrently. As Wirth has noted there is a considerable increase of complexity as one moves from a sequential to a concurrent or from a concurrent to a real-time system. The type of system for which a specification language is intended heavily influences the language characteristics. For example, requirements for business data processing applications differ dramatically from requirements for real-time systems. Real-time systems are mostly specified by describing sequences of system responses activated by a series of external stimuli, i.e. they describe the dynamic nature of the system. Data translation processes, documentation formats and similar system properties are the focus of consideration for business data processing applications. For example, requirements for engineering design applications, i.e. these specifications describe the static nature of the target system. Generally, of key importance are language features that allow us to specify time dependencies and temporal relationships (see also category A9 below) between stimuli/responses or system components (modules and their interaction).

A second aspect involves systems where the processing ultimately is distributed among several geographically distant nodes or the processing is done only within one centralised single processor. The more complex requirements for distributed data processing systems (DDPS) include for example the specifications for the network of processors, decisions for allocating the various processes to the nodes within the network, communication links, co-ordination requirements etc. Alford (1979) distinguished two types of DDPS according to (a) different binding times of processing to processors and (b) the nature of application. The allocation of functions and associated performance to various DDP nodes and the communication links can be carried out at a number of points. The extremes are the system design phase and at execution time. Allocation at execution time seems to be advantageous when processing is supplied by the user. Hence an unknown rate and duration of user jobs can be expected requiring dynamic allocation at execution time. In contrast allocation at system design time requires well-known processing requirements and loads. The second aspect, nature of application, yields a further point of view to divide DDPS into two significantly distinct classes. Supplied DDPS (the user supplies the programs and processing instructions) handle a variable workload of various jobs on processors at different locations. Dedicated DDPS on the other hand are those specifically aimed at a certain job and therefore both hardware and software are usually specifically designed for accomplishing it. Specification languages that include language constructs for allocating DDP requirements to various nodes (time binding requirements) should methodically identify possible solutions and alternatives and compare them to the systems preference rules, i.e. the system-level performance requirements. The following category, A3, becomes of vital importance specially for DDPS (specification of hardware configurations and their network of interactions,...).

3. Environment description: System descriptions should encompass all interacting parts including both the digital subsystem (see Figure 2) and the environment subsystem (see Figure 3).

This implies that requirements specification languages should include some sort of real world model to describe the environment in which the data processing system is required (see Zave, 1979). This enables us to...
specify the software subsystem "interface" in the same way as the software subsystem itself rather than introducing another formalism. As it is seen in Figure 3, this comprises description of users (the various people of the customer who will work on the target system, the development team, . . . ) and their tasks, the organisations (offices, departments, . . . ) which are affected, existing software systems with which the target software subsystem will interact and hardware description facilities (the host computer and others). Along with descriptions as to how these environmental components are interrelated and related to the software subsystem and how they interact, this yields a precise statement in which no external impulses remain undefined.

These environmental description facilities which will be most useful in the requirements specification phase provide the basis for the automatic generation of various user views (e.g. customer manager document, customer bookkeeper document, . . . ) through associated automated report generators and documentation packages. The remaining points 4-9 of category A are more related to the digital subsystem.

4. Data flow: In some systems the flow of data is so dominant that in order to have a clear picture of the system the specification of the data flow between the processes almost completely tells us everything about the system. Mostly the control aspects follow naturally from the data flow requirements. This observation also led to the proposal of development methodologies claiming that the identification of the inherent data structure and the data flow permits a convenient derivation of the program structure (see e.g. Orr, 1978). An example is the SANN scheme which models graphically the data flow between activities (see Lamb et al., 1978).

5. Control flow: Techniques for specifying the control flow of systems are less developed and numerous than the data flow description techniques (see Ramamoorthy and So, 1978). Many aspects of a software system are characterised by its control flow (operation sequence, interaction pattern of subsystems, deadlock situation, . . . ). Example techniques are petri nets (see Peterson, 1977) and finite-state machines (see Salter, 1976).

6. System structure: This refers to techniques that depict the system in terms of its major components and their hierarchical decomposition. Example representation schemes are structure charts as presented by Brown (1977) and many others surveyed by Peters and Tripp (1976).

7. Module structure: Similar to system structure methods these techniques describe the structure of a module. This includes description of the inputs and outputs, transformation processes and interface definition.

8. Data structure: Methods which represent the data items used by the processes and their hierarchical interrelationships.

9. Performance aspects: To date, three major performance aspects are known: accuracy, timing and load. Accuracy refers to the precision with which some results are required (very important for example in numerical problems). Stavely (1978) stated that timing aspects depend strongly on the class of software system under consideration (sequential/concurrent/real-time). Timing aspects may be stated in relative (e.g. ordering of events within time) or absolute terms (e.g. elapsed time between events). Examples are response time, process deadlines, turnaround times, etc. Load aspects consider data volume, frequency of inputs, etc. Other performance aspects such as reliability, resource, cost etc. are of a more complex nature and are not addressed explicitly in current languages. In many languages performance characteristics may be stated independently of functional requirements in order to reduce the overall complexity. The systems analyst/designer first defines hierarchical relationships of modules and other functional properties independent for example of time and then adds the performance aspects.

B. Understandability

1. Dimension of language: In his survey, Jones (1979) classifies specification languages as character string languages (one-dimensional), graphical languages (two-dimensional) and hybrid languages (made up of both graphics and text). As it is noted elsewhere (Teichrhew and Sayani, 1980) this distinction is not necessary because the form of input can be separated from the representation in the data and that in turn can be separated from the form of the output (input/external/output representation). However, the speed of information transfer depends considerably on the way things are expressed. Based on studies in the domain of perceptual psychology, Jones (1979) concludes that graphics-based languages are superior to character string languages in both speed of information transfer and overall clarity. The major problem with graphics-based languages is their limited "processability". Until recently, most required manual drawing and updating and therefore the associated costs were high and production rates were low. Recent
achievements consist of compilable graphics which automatically resolve former problems of drawing, modification, layout, abstraction (zooming out of details), etc. (See e.g. Willis, 1981).

2. Variation of detail: Until recently the importance of detail variation — the possibility of neglecting irrelevant detail at the current phase of development — was not recognised. Detail variation consists of two aspects. Firstly, it means the incremental development of something. The language should allow analysts and designers to start with vague statements and help them to increase detail. For example, the designer could start with the specification of the normal case and receive feedback on both validity and completeness (what is missing?) no matter how sketchy his design is. Secondly, at some stage it may be necessary to throw away all irrelevant detail that has already been stated in order to see the overall structure. These simplified models gained through abstraction facilities should allow greater insight and separation of the essentials of a situation from the inessential. A particular abstraction method of interest is projection which permits the designer to focus upon and highlight interesting behaviour and to suppress irrelevant details. The first aspect refers to the input representation and the second to the output representation of the language. Incremental and abstraction facilities both take into account the fact that the development of software is an inherently iterative process. A good example is PDL of Caine and Gordon (1975), which allows the program designer to write PDL specifications in whatever detail level is appropriate to the designer and problem at hand. In SPECLE (Biggerstaff, 1979) the designer may start with a skeletal design and incrementally elaborate it until it is complete.

3. Factoring of specifications: Systems can be seen from various viewpoints where each perspective may depict possibly overlapping subsets of properties of the entire system. This “paraphrasing” capability which is commonly used in other disciplines (e.g. architecture, . . .) implies the support of various alternative input representations. Firstly, different people with different needs may state these multiple viewpoints which are combined into the central internal representation. Secondly, these descriptions may view a system with respect to different sets of concerns (e.g. desired properties of overall operation as well as desired behaviour of individual modules). These various viewpoints may overlap and the resultant redundancy offers additional analysis capabilities. Furthermore this is a promising strategy to reduce and decompose complexity. This fits nicely into category B2 because incrementally developed specifications and abstractions can be viewed as special paraphrasing facilities of the input and output representations. Simple paraphrases are inversions of relationships (“A sends data to B” — “B receives data from A, D, . . .”). A good example is the IDAP system which allows incremental specification of different perspectives (see Hammond, Murphy and Smith, 1978).

4. Modularisation: Modularisation can be viewed as a special abstraction mechanism and refers to the decomposition of systems into small clusters of actions (modules). Its main objective is to reduce complexity whose major sources are the number of elements in a system and their relationships. Modular specifications admit that different properties are independently described and interrelationships among these properties may be defined separately from the specification of the properties themselves. Specifi-
6. **Procedural vs. Non-procedural specifications**: From the statement that RSLs describe "what" a system is intended to do, we can conclude directly that they should use only non-procedural constructs (see Leavenworth, 1977). The system is viewed as a black box and how it performs is described with no reference to any internal workings and ignoring software-oriented problems and all questions of efficiency. This encourages concentration on the user organisation and on what the system is intended to do (see Leavenworth and Sammet, 1974). Procedural constructs would not yield the same powers of abstraction needed during these early phases. Furthermore, they would be too constraining for subsequent phases. Non-procedural specifications also seem desirable in the logical but not in the physical design phase. In order to depict e.g. the structures of algorithms a limited set of control constructs (if then/else, case, sequence, loop) should be allowed. The background is that most specification languages are not intended to produce an executable program but to represent a description of the target system (e.g. the logical design) clarifying all relevant aspects. However, some PDLs provide a combined program design specification and implementation language. In these languages one proceeds from a physical design specification down to the detailed implementation level. These "implementation specification languages" are certainly ideal for the designer and programmer but fairly useless to either the user or manager due to their formal and technical vocabulary. One example is GYPSY (Ambler et al., 1977) which allows the incremental development of specifications into program code thereby blurring the distinction between specification languages and programming languages.

7. **Degree of formality**: Specifications written in formal notations are largely incomprehensible to the vast majority of persons who contract for the design and development of software systems. On the other hand, formal specifications eliminate virtually all sources of imprecision and ambiguity through precise syntactic and
Specifications
— in Spec.Lang.#i

Table Def.
Manager

Table Driven
Processor

Specifications
in Spec.Lang.#1

Table Driven
Processor

Figure 5. Table-driven Processor.

Semantic definitions and provide a basis for constructing a correct program and mathematically verifying its equivalence to the specification. In contrast, informal specifications will state only some information explicitly; the rest is left implicit and must be extracted from the context. This is a form of abstraction: attention is focused on essential parts increasing understandability. In fact, this way of recording partial rather than complete descriptions is similar to the normal mode of human communication. A system dealing with informal specifications must include strong inference, assumption detection and context extraction mechanisms. Otherwise the quality of the specifications produced will be minor. Some progress has been made in dealing with imprecise specifications. Mostly, artificial intelligence techniques such as fuzzy systems (Zadeh, 1971), semantic networks (Mikelsons, 1975), etc. are used to interpret informal information at the system level (Martin and Bosij, 1976). The resulting question is the proper balance between formal and informal: how far to push formality (see Žemanek, 1974).

One potential advantage of specification languages seems to be that they appear capable of merging formal and informal descriptions of a system as advocated by Wasserman (1977). The informal information acts primarily as a supplement and serves to enhance the clarity whenever formal mechanisms are inadequate to represent design aspects. As a rule of thumb, formal specification languages are adequate when high reliability is required and the requirements seem to be very stable. SPECIAL (Roubine and Robinson, 1976) and AFFIRM (Musser, 1979) are examples of formal specification languages which are beginning to be used with some success on non-trivial practical software projects. Some progress in the construction of informal specification languages has occurred (see Balzer, Goldman and Wile, 1978) but major breakthroughs are still missing. An important negative effect of informal specification languages might be that large volumes of information are required to improve the understanding and to eliminate all sources of ambiguities. Jones (1979) states that unrestricted English is almost always "prolix" (excessively wordy) abrogating the advantage of informality. It can be concluded that a mixture of formal and informal specifications combined within a graphics-based language would be paramount.

8. Alternatives: To improve the reviewability of designs it is important to record design problems. Alternative solutions should be formulated along with the evaluations and arguments that led to the choice of a particular alternative (see Freeman, 1975). This aspect seems to be most valuable in the logical design phase where major architectural decisions are made. Explicit enumeration of the alternatives and their impact will greatly enhance understandability. Moreover, the explicit requirement to provide several solutions and evidence for and against them will promote broader thinking leading to better solutions which might have been overlooked otherwise.

C. Analyzability

1. Completeness: In his survey Jones (1979) revealed that generally completeness of specifications is a function of program size, i.e. the larger the system the more likely there are omissions in specifications. It is an inherent nature of specifications to be incomplete. Specifications are always some sort of model or abstraction of some real or envisioned situation. Furthermore, it should be noted that completeness is always a relative question and depends on what level of detail is desired. The differences between the various stages (requirements definition — logical/physical design — implementation) are obvious but also within each phase several refinements are likely due to the incremental development of specifications. Hence, specification language processors must be capable of dealing with incompleteness and the specifier should be allowed to ask at each phase of development: "Is the design complete?" or "What things are missing?"

2. Static validity: One of the most useful feedback aids checks the static validity of a specification. This includes examination of consistency ("are there any conflicting and incompatible statements?"), ambiguity or vagueness ("unique interpretation possible?"), overspecification ("too much detail stated in this phase? Overconstraining?"), redundancy ("repeated specification, possibly in different terminology?"), etc.
3. Tolerance: Commonly, an error within a program leads to its execution being terminated. The special nature of specifications does not necessitate immediate completeness and correctness. Thus, the associated analysis system should be forgiving and guide the specifier in dialogue form through the provision of immediate feedback. This not only refers to sources of errors but also to additions. Internal quality assurance procedures should enable the system to give suggestions, to evaluate alternatives etc.

4. Quality metrics: Many current development methodologies provide subjective qualitative guidelines. Given enough experience the specifier is able to interpret them correctly and produce high quality specifications. Without enough experience, however, they become ambiguous and open to subjective interpretation. Hence, what is really needed are quantifiable measures of "goodness". This means e.g. measures of complexity, connectivity, path analysis etc. The quality metrics should view the specifications from different perspectives. For example, how easily may likely future modifications be accommodated, how reliable will the system be, etc.

5. Traceability: Traceability refers to the capability of verifying a specification against its successor or predecessor. The different phases of software development have varying orientations and demand different sets of language capabilities to express the required properties and formats. Hence, any systematic approach clearly requires more than one language (e.g. one RSL or a DSL and the programming language). The major problem now is to relate these descriptions to ensure correspondence and equivalence (intertraceability). Additionally, as we are allowed to develop specifications incrementally it must be possible to verify that each refined description is a more detailed expression of the former statement (intra-traceability). Thus, explicit language constructs for tracing are necessary. For example, design decisions should be traceable to both the originating requirements and to special programming modules which implement them.

6. Feasibility: One major disadvantage of a top-down approach to specification definition is that successive levels of detail may reveal that the chosen approach is infeasible or that major negative effects and problems are associated with it that could not be anticipated. Methods that allow the demonstration of feasibility (or implementability) would substantially reduce the amount of backtracking in the development process. Closely coupled with this is the question of appropriateness of a system ("Is this really the system we need?"). Current methods include prototyping and simulation of system behaviour through modelling constructs. The key idea of prototyping is that "...exploration of what is possible to implement provides guidance ..." in the development process (C. Hewitt, in Riddle and Fairley, 1980, pp. 142-144). This implies that during development a series of more detailed and sophisticated models (specifications) and prototypes (implementation versions) are produced. It is expected that co-existence and co-evolution of both specifications and prototypes will stimulate each other in a very fruitful way. Experience and insight gained through the construction of prototypes will permit a better view of the real needs and reveal shortcomings or avenues for improvements.

7. Dynamic validity: In contrast to static properties, dynamic correctness examines the behaviour of a software system over time. The objective is to evaluate and predict the behaviour of specifications before implementation starts and to support specification modification until a required performance goal is met. Feedback about dynamic properties often is the most useful and powerful information for the specifier but also very difficult to obtain. Some benefits are: early feedback about user convenience and system responsiveness, avoidance of prescriptive problems, performance and efficiency prediction, etc. Methods to predict system behaviour are of paramount importance especially for real-time systems. To achieve the above mentioned objectives, mostly logical models are used that describe the intended system behaviour as opposed to the details of its operation. This allows an abstract representation in pseudo-procedural terms. The SREM project (Davis and Vick, 1978) has shown that it is possible to provide a simulation model during the requirements phase without entailing premature decisions about system structure. Another example is the translation of PSL requirements statements into a standardised discrete event simulation model described by Willis (1978).

D. Maintainability

1. Non-prescriptive specifications: This aspect is applicable to the requirements definition phase and the logical design phase and addresses the undue enforcement of constraints on subsequent activities. Non-prescriptive specifications serve to prevent the premature selection of (a) design solutions in the requirements phase and of (b) implementation mechanisms in the logical design phase which can unnecessarily limit the range of potential system realisations. For example, the requirements document may limit the range of solutions by specifying a particular algorithm or data structure. The architectural design document may already define the needed data in an implementation oriented format. Good examples are two of the early requirements specifications of the Ada language (see US DoD, 1976 and 1977) which have been criticised by Dijkstra (1978) to be not only contradictory but also to be a mixture of requirements, rough design statements and implementation decisions. Generally, it is harder for procedural modelling techniques to be non-prescriptive than for non-procedural ones (see Riddle, 1980). Riddle and Wieden (1978) identify as an important aspect of non-prescriptive specifications orthogonality: "... forming associations among the elements of the system which may be completely different from those found in the system's implementation. Therefore, they express a logical rather than a physical system organisation". Postponement of implementation dependent information as long as possible may have a major influence on the maintainability of a system. Firstly, failure to do so may cause an increased amount of reworking and redesign during the development time. Secondly, a major aspect of implementation dependent information is efficiency but optimisation spreads information throughout the components thereby increasing their interdependence and the overall complexity. Hence, optimisation is a major deterrent to maintainability and understandability.

2. Modifiability: Modifiability addresses the ease or difficulty with which a software product can be changed. This may include extension or contraction, adaptation, the correction of errors and improvement of performance. Some criteria have been discovered to be useful in facilitating software updates which are also applicable to specifications. One is information hiding developed by Parnas (1978). Ideally, it should be possible to localise modifica-
3. Compliance to rules or standards: The recognition of a discipline within the programming process had a lasting impact on the computer community (see e.g. Dijkstra, 1976). We accepted constraints on our programming freedom in order to achieve a more reliable and well-understood product (Stoeger, 1982a). The same degree of discipline is required for the specification phases. This implies the standardised application of techniques and methods that have been found beneficial. Although all specification approaches suggest useful rules of thumb, procedures, standards, etc., there remain many open questions when using them and no guarantee exists that they have been used at all or in the right way. Thus engineered specifications have to include additional mechanisms to enforce complete and consistent application of these criteria and conventions. Obviously specification languages can help a lot in conjunction with automated analysis tools. For example, a top-down approach can be ascertained through the requirement that the specification of a module is prohibited until it is called somewhere. Such an enforcement to certain rules does not necessarily restrict us to one method. Alternate strategies and different methods may be enforced in different stages. Also essential are precisely defined configuration control procedures. For example, each problem definer should have the “ownership” of his part of the total specification and updates by others should be forbidden.

4. Documentation generation: Several surveys of software maintenance have confirmed that software maintenance costs outweigh development costs (see e.g. Lientz, Swanson and Tompkins, 1978 and Liu, 1976). The main expenditures of effort are user enhancements (42%), adaptation (17%) and correction (22%) (Boehm, 1979). The basis for a quick location of errors in modules and all affected components that have to be modified is a comprehensible and up-to-date documentation. To avoid a repetition of the same painful lessons of software maintenance in requirements and design maintenance, it should be very easy to extract automatically some information from the various specifications. This includes the generation of reports of various specification parts on various peripheral devices (hardcopy printer, plotter, . . .) and in different levels of detail. These output paraphrasing capabilities of specification systems are sometimes called “feedback analyses”. This means that alternative representations of characteristics of the system under design are derived (this may include some informal analysis) from the internal representation and presented to the developer who uses human intelligence to check the system (user validation).

5. Analogous specification: Sometimes parts of specifications may deal with components that are very similar to each other. Balzer and Goldman (1979) stated that in such cases it is more convenient to specify one part in terms of another outlining the similarities and differences between them. This results in the facilitation of maintenance because changes of the base part would be automatically reflected in the analogously specified components. Those parts where a promulgation of the change is not desired could be easily excluded.

6. Reusability: One method for the improvement of systems development productivity is to reduce the amount of code that must be produced. This may be achieved through reusable modules as propagated by Lanergan and Poynton (1979). The foundation for the reusability of software pieces is their storage in a central data base along with a library system describing their function and properties. In future, realisations of systems will be based on such a library system by synthesising them from existing elements. However, until now this has proven to be a myth because currently there is no large-scale reuse of code. There are some trends in the USA and USSR to standardise modules and to foster software sharing between federal agencies (see e.g. Teichroew, Hershey and Yamamoto, 1977, pp. 340). However, besides technical problems, human and institutional resistance may also have to be faced. It is well known that programmers generally tend to be hostile towards using other programmer’s work, because “. . . it is worse, not efficient, clumsy, etc.” Specification languages for the first time offer module descriptions that are separated from their implementation. Hence, their internal descriptions could become the basis for a library system and modules could be shared between various machines or development support systems as the specification language processors are able to deal with various external (input/output) representations as described above. Principally, software components at all levels can be made reusable (e.g. generalised software packages, turn-key systems, . . .) but the obvious level with the most potential for mass usage is the module level. Many of the above listed objectives will contribute to reusability of specifications (compliance to standards, analysability to guarantee certified catalogued modules, analogous specification, . . .).

3. OVERVIEW OF SOME SPECIFICATION LANGUAGES

The above presented framework provides an excellent classification scheme to survey in depth virtues and scope of existing specification languages. This has been done in detail by the author (Stoeger, 1982b). Here we restrict ourselves to a very short overview of some of the most important representatives of each of the three types. The overview is supplemented by Table 1.

1. PSL: The most widely known and used requirement specification language is PSL — problem statement language — within the ISDOS (information systems design and optimisation system) project (see e.g. Reifer, 1978; Teichroew and Sayani, 1971; Teichroew and Hershey, 1977a, 1977b; Teichroew et al., 1980; Winters, 1979). PSL had substantial influence on the development of subsequent specification languages and is based on the ERA (entity, relationship, attribute) approach, i.e. a system is considered to be composed of “objects” (entities) which have certain properties (attributes) and between them there are particular relationships. PSL is intended to be used in business applications and therefore provides excellent facilities for the specification of static system properties (data definition, data structure, data translation, . . .).

2. RSL: In contrast, RSL — requirements specification language — emphasises the control aspects of a system. It is also based on the ERA model but permits the specification of all stimulus/transform/response relationships of a system and organises them into particular “structures” (R-Nets, subsets and paths). Furthermore, RSL can deal with parallelism and problems of synchronisation (see
### TABLE 1. Comparative table of some specification languages

**Explanation of Table 1:**
- blank fields: not addressed
- hatched fields: partly addressed
- black filled fields: sufficiently supported

**basis:**
- Mo = model-based
- Me = methodology-based
- Re = representation-based

**system type:**
- S = sequential software systems
- C = concurrent software systems
- R = real-time software systems

**language dimension:**
- n = dimension of input representation

**degree of procedurality:**
- N = non-procedural
- P = procedural
- M = mixed

**degree of formality:**
- F = formal
- I = informal
- M = mixed

**phase of application:**
- R = requirements specification language
- I = design specification language (logical design phase)
- P = program design language (physical design phase)
- C = programming language (coding)

**status:**
- I = implemented
- R = used for research
- P = used within a production environment
- C = ongoing project (changes are likely)

#### LANGUAGE

<table>
<thead>
<tr>
<th>CHARACTERISTICS</th>
<th>RE</th>
<th>EP</th>
<th>RP</th>
<th>SPEC</th>
<th>CVSB</th>
<th>DMD</th>
<th>PDL</th>
<th>PLL</th>
<th>SLL</th>
<th>SPECIAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>language basis</td>
<td>Mo</td>
<td>Mo</td>
<td>Mo</td>
<td>Re</td>
<td>No</td>
<td>Mo</td>
<td>Re</td>
<td>Mo</td>
<td>Ne</td>
<td>Mo</td>
</tr>
<tr>
<td>system type</td>
<td>Mo</td>
<td>Mo</td>
<td>Re</td>
<td>Re</td>
<td>SCR</td>
<td>SCR</td>
<td>S</td>
<td>S</td>
<td>S</td>
<td>SC</td>
</tr>
<tr>
<td>environment description</td>
<td>Mo</td>
<td>Mo</td>
<td>Re</td>
<td>Re</td>
<td>SCR</td>
<td>SCR</td>
<td>S</td>
<td>S</td>
<td>S</td>
<td>SC</td>
</tr>
<tr>
<td>data flow</td>
<td>Mo</td>
<td>Mo</td>
<td>Re</td>
<td>Re</td>
<td>SCR</td>
<td>SCR</td>
<td>S</td>
<td>S</td>
<td>S</td>
<td>SC</td>
</tr>
<tr>
<td>control flow</td>
<td>Mo</td>
<td>Mo</td>
<td>Re</td>
<td>Re</td>
<td>SCR</td>
<td>SCR</td>
<td>S</td>
<td>S</td>
<td>S</td>
<td>SC</td>
</tr>
<tr>
<td>system structure</td>
<td>Mo</td>
<td>Mo</td>
<td>Re</td>
<td>Re</td>
<td>SCR</td>
<td>SCR</td>
<td>S</td>
<td>S</td>
<td>S</td>
<td>SC</td>
</tr>
<tr>
<td>module structure</td>
<td>Mo</td>
<td>Mo</td>
<td>Re</td>
<td>Re</td>
<td>SCR</td>
<td>SCR</td>
<td>S</td>
<td>S</td>
<td>S</td>
<td>SC</td>
</tr>
<tr>
<td>data structure</td>
<td>Mo</td>
<td>Mo</td>
<td>Re</td>
<td>Re</td>
<td>SCR</td>
<td>SCR</td>
<td>S</td>
<td>S</td>
<td>S</td>
<td>SC</td>
</tr>
<tr>
<td>performance aspects</td>
<td>Mo</td>
<td>Mo</td>
<td>Re</td>
<td>Re</td>
<td>SCR</td>
<td>SCR</td>
<td>S</td>
<td>S</td>
<td>S</td>
<td>SC</td>
</tr>
<tr>
<td>language dimension</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>variation of detail</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>factorization</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>modularization</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&quot;natural&quot; language constructs</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>degree of procedurality</td>
<td>M</td>
<td>N</td>
<td>P</td>
<td>P</td>
<td>M</td>
<td>M</td>
<td>M</td>
<td>N</td>
<td>P</td>
<td></td>
</tr>
<tr>
<td>degree of formality</td>
<td>M</td>
<td>M</td>
<td>F</td>
<td>F</td>
<td>M</td>
<td>M</td>
<td>M</td>
<td>F</td>
<td>F</td>
<td></td>
</tr>
<tr>
<td>alternatives</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>completeness</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>static validity</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>tolerance</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>quality metrics</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>traceability</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>feasibility</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>dynamic validity</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>non-prescriptiveness</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>modifiability</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>standard compliance</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>documentation generation</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>analogous specifications</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>reusability</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>data base system</td>
<td>R</td>
<td>R</td>
<td>L</td>
<td>P</td>
<td>P</td>
<td>P</td>
<td>L</td>
<td>P</td>
<td>L</td>
<td>P</td>
</tr>
<tr>
<td>phase of application</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>status</td>
<td>IPC</td>
<td>EPR</td>
<td>IP</td>
<td>IP</td>
<td>IP</td>
<td>BB</td>
<td>ERC</td>
<td>IP</td>
<td>IC</td>
<td>IPC</td>
</tr>
</tbody>
</table>

3. RLP: The RLP - requirements language processor — was developed at the GTE Laboratories and is based on a model very similar to an R-net. It uses stimulus-response sequences and allows the use of application oriented terminology within each application area. This multi-lingual facility is achieved by making minor language modifications which are accepted by the table-driven language processor (Davis et al., 1979). Thus far two languages, P2 and RTRL, have been put into operation and a third, COSS-RL, is in development (Davis, 1982).

4. RDL: On the borderline of requirements and design specification languages is the RDL — requirements and development language (Heacox, 1979). It combines
specification languages still has to be advanced.

4. SUMMARY AND CONCLUSION

will support the development process. Hence, the process of generating software will be a continuous definition, concurrent or real-time systems and was developed within the DREAM (design realisation, evaluation and modelling) system. It has a very formal character and customers and users will have problems to understand its constructs. However, it is a very precise and advanced tool in the hands of designers and permits the automatic examination of many desirable properties of software designs.

6. Others: A mixture of a DSL/PDL is SPECLE (Biggerstaff, 1979). SPECLE was developed as part of the IDAP (integrated design analysis programs) system at the BOEING Computer Services Company (see Hammond, Murphy and Smith, 1978). Its emphasis lies on simplicity, understandability, and readability. A similar specification language is FLEX (Sutton and Basil, 1981). Two examples of program design languages are PDL — program design language (Caine and Gordon, 1975) and SSL — software specification language (Buckles, 1977), while GYPSY (Ambler et al., 1977) and SPECIAL (Chandersekaran and Linger, 1981) are implementation specification languages. Some more examples of specification languages are presented by Bodart and Pigneur (1979), De Remer and Kron (1975), Everhart (1980), Hamilton and Zeldin (1976), Ludewig (1982), Macasovic (1979) and Winchester (1980).

Table 1 summarises in tabular form the observed characteristics of some of the above mentioned languages. Each vertical column provides a quick overview of the description facilities and characteristics of one particular language revealing its strong and weak points. On the other hand, the horizontal lines exhibit how sufficiently a particular aspect is addressed in current specification languages. It becomes evident from inspection of Table 1 that aspects which are most closely related to concepts of programming languages (data structure, module structure, control flow, . . . ) are best addressed while the other, relatively "new" characteristics such as environment description facilities, performance aspects, quality metrics, etc. are not sufficiently supported.

4. SUMMARY AND CONCLUSION

We can add up the whole bulk of information presented by citing the old saying: a problem properly stated is half solved. However specification languages should not be regarded as a panacea. They are tools that offer many benefits for the user and software engineer but the current state-of-the-art which may be called the first generation of specification languages still has to be advanced.

In the future, a coherent family of languages whose purpose, semantics and syntax are derived from the needs and requirements of each phase of the development cycle will support the development process. Hence, the process of generating software will be a continuous definition, transformation and validation process supported by an integrated specification system. After the target system has been defined and validated within one language we proceed to the next phase. The establishment of explicit relationships between the various forms of description will permit semi-automatic proofs of correctness. All information of the whole development process will be stored in a flexible internal format within a central data base. This software development data base will be a centre of information storage and exchange among the various languages and integrated tools.

Currently, specification languages are relatively restricted to one particular phase of software engineering (RSL, DSL, PDL, programming languages). It may be argued that differences between these phases will become blurred due to extensions of the current language types. Progress of technology in one particular phase will have fall-outs in the other phases. A first indication of this evolution is the movement of program design languages towards programming languages and vice versa.

Furthermore, the characteristics of specification languages will change. In order to permit increased speed of information transfer new forms of mixed formal/informal and graphical/textual input representations will be created. A simple comparison is reading a book (textual, one-dimensional representation) and seeing its performance in the theatre (multi-dimensional representation). How often one overlooks important details or fundamental ideas stated explicitly or implicitly in a text. Seeing the same "alive", with all expression, means of mimerry, gesture and modulation, will provide the audience with deeper and better understanding. Similarly the dimension of specification languages has to be expanded. This means removal of limited processability of graphical languages, human engineered user interfaces which give a feeling of presence, involvement and participation (improvement of usability through immediate simple or paraphrased feedback, maybe in natural language), improvement of feedback tools which analyse the system behaviour over time, etc.

Another area of specification languages which has not been tackled yet is the development of specification languages for project plans (MSL — management specification language). They will be the core of the management support subsystem interfacing with the other subsystems via the central data base and/or directly via explicit language constructs. Such an MSL will permit precise measurement of progress, definition of milestones, description of budgets and schedules, accumulation of project statistics, etc. Hence, it will not be restricted to a particular development phase, but rather will span the entire software development life-cycle.

Finally, it should be noted that all specification languages which are currently in use or development have one characteristic in common. They address development of new software by supporting the specification and construction of new software systems. They share this characteristic with existing programming languages. As described in this paper, the current state-of-the-art of these languages has already elevated from alchemy to science (Alford, 1980b). However the reverse approach seems to be also promising, considering the vast amount of software that already exists. Such a "specification producing language" will accept as input an existing software system and its output will consist of a variety of analysis (performance, reliability, suggested modifications, . . . ) and documentation reports.

Software lags behind hardware and this arrear can
only be mastered by increasing productivity. Reusability of software and automatic programming offer the most promising avenues in achieving this goal. Specification languages are a prerequisite to both.

ACKNOWLEDGEMENT

The preparation of this paper was made possible through a postgraduate scholarship from the Australian European Awards Program of the Australian Government. I would like to thank Professor H.A. Maurer, Professor G. Haring and Professor H. Florian (Technical University of Graz, Austria) for their support of my application. Especially, I would like to thank Professor P.C. Poole for his support and valuable suggestions throughout the preparation of this paper.

REFERENCES


STOEGERER, J.K. (1982a): An annotated bibliography of soft­

design techniques, specification systems and software de­


codeaided structured documentation and analysis of information processing systems, IEEE Trans.

SMITH, G.L. and TRIPP, L.L. (1976): Software design represen­


SALTER, K. (1976): A methodology for decomposing system re­

ORR, K.T. (1978): Introducing structured systems design, in

PETRICK, L.J. and TRIPP, L.L. (1976): Software design represen­


RAMAMOORTHY/YEH (eds.): Tutorial: software methodology,

PARNAS, D.L. (1972): On the criteria to be used in decomposing


PETERS, L.J. and TRIPP, L.L. (1976): Software design represen­


Protocol Translation for Packet Network Interconnection

By J. I. Fernandez*

Two or more packet switching networks can be connected together by implementing a gateway that translates between the generally incompatible protocols of the different networks. One such implementation of protocol translation is considered in detail. This is used to illustrate the problems involved and possible solutions of them.

Keywords and phrases: Packet switching networks, communications protocols, gateways.
CR categories: C.2.1, C.2.2.

1. INTRODUCTION
During the last decade, packet switching has changed from an experimental technique, used for computer communication by a few research organisations, to a widely accepted operational technique for wide area computer networking. A very large number of diverse organisations have developed packet switching computer networks. With the basic technique of wide area computer networking well understood, the research emphasis moved in the last half of the decade to the interconnection of the packet networks developed.

Even the earliest attempts at developing packet switching networks saw the definition of sets of rules for computer communications, which were given the name protocols. Protocols were developed for "internal" network communication (between the nodes of a packet switching network) and for "external" communication (to interface the network to host computers). Research in the field of packet switching during this period was directed at defining these protocols, understanding their behaviour by theoretical and empirical studies, making refinements, and more recently, formally specifying standard protocols and verifying them. The proliferation of packet networks, during the period when protocols were still being experimented with, resulted in a diversity of protocols. Packet network interconnection or "internetworking" therefore depends on achieving communication between packet networks that use a variety of protocols.

An analogy can be drawn between computer communications protocols and human languages. A single computer network represents a community of computers speaking the same language. In general, computer communications protocols, like human languages, differ widely. In addition, just as there are languages with multiple dialects, so there are groups of similar (but not identical) protocols with a common derivation, e.g. different computer manufacturers' versions of the ISO HDLC protocol (ISO, 1979) and IBM's SDLC (IBM, 1979).

To achieve internetworking, one is faced with the same choices as someone attempting to achieve communication between people who speak different languages. Firstly one can use an interpreter, who, for example, translates between English and French. Secondly one could define a new standard language for international communication, like Esperanto, and require people to learn this language as well as their own native languages. This is analogous to the standard protocols being defined by international bodies like the International Organisation for Standardisation (ISO) and the International Telegraph and Telephone Consultative Committee (CCITT). Thirdly one could consider that a popular language such as English or French constitutes a de facto standard by virtue of its widespread use, and adopt it as the required one for international communication. IBM's Systems Network Architecture (IBM, 1982) and Digital Equipment Corporation's DECNET (DEC, 1982) could be regarded as fitting into this category in the field of computer protocols.

Early attempts in either field invariably use the first approach, since they occur before a defined or de facto standard emerges. If an English speaking person wishes to communicate with a number of French and German speakers, there is little motivation for learning Esperanto unless the others are known to speak that as well; the more attractive approach is to obtain an English-French interpreter and an English-German interpreter.

Thus packet network interconnection is commonly achieved by implementing a system that translates between the protocols of the networks it interconnects. This interfacing system has come to be commonly known as a gateway.

2. INTERNETWORKING AT CSIRO
The CSIRO work on internetworking began in 1979. At this stage, the CSIRO packet switching network, CSIRO-NET, (Palandri, Gilbert, Paine and Vezis, 1978) had been in operation for some time and provided Australia-wide access to a Control Data Cyber 76 located in Canberra. This host provides a facility for interactive editing and remote job submission for a large number of users. A number of other networks had been developed, in Australia and overseas, and some of these had facilities that were complementary to the CSIRO facilities. For example some of the universities had DECsystem-10 based networks, which offered excellent support for interactive program development. A large number of scientific data bases were accessible on some overseas networks. Some research networks offered
corresponding to a different level in the hierarchy. The set of protocol layers could be defined, with each layer functions could be conveniently arranged in a hierarchy and communications protocols. The various communications which currently consists of some one hundred and forty 3. THE ANF-10 TO CSIRONET CONNECTION

The ANF-10 connection to CSIRONET has previously been described (Fernandez and Gibbons, 1983). For the purpose of discussing the protocol translation, we will therefore merely summarise the main features of the networks and the interconnection.

CSIRONET is a private packet switching network which currently consists of some one hundred and forty PDP-11 nodes. The nodes run a dedicated program, NODECODE, and use communication protocols developed by the CSIRO Division of Computing Research. The host computers now supported include Control Data, DEC and IBM-compatible machines.

ANF-10 (Advanced Network Features-10) is a DEC network product designed for remote access to, and interconnection of, DECsystem-10's (Digital Equipment Corporation, 1977). The nodes are PDP-11s and the protocols used are similar to those of the vendor's other, more well-known network product, DECNET.

The gateway is a PDP-11 that runs the DEC operating system RSX-11S and that has synchronous lines connecting it to each network. It appears to be both a host and a node on each network. It thus implements the ANF-10 protocols and the CSIRONET protocols and translates appropriately between them.

3.1 The Communications Protocols

One of the concepts that became accepted during the development of packet networks was that of using layered communications protocols. The various communications functions could be conveniently arranged in a hierarchy and a set of protocol layers could be defined, with each layer corresponding to a different level in the hierarchy. The International Organisation for Standardisation has recently produced a Reference Model for Open Systems Interconnection (OSI) to "provide a common basis for the co-ordination of standards development for the purpose of systems interconnection" (ISO, 1982). This model has seven layers defined. The protocols used in CSIRONET and the ANF-10 networks were developed long before the ISO work on this model. As a consequence, there is not an exact correspondence between the layering of these protocols and the layering of the OSI model. However, in view of the fact that the OSI model is becoming fairly widely known, a comparison of these protocols with the reference model may provide a better understanding of them. It should be borne in mind in the discussion below, that the protocols are retrospectively being fitted to the model, and hence the correspondence is not always clear and one-to-one. In fact, alternative mappings of the protocols to the model could perhaps be made.

The seven layers of the OSI Reference Model are shown in Figure 1. The lowest, the Physical Layer, is responsible for the mechanical and electrical characteristics of the data transmission and can be classed as dealing with transmission of bits between adjacent nodes. The Data Link Layer provides detection and possible correction of errors in this transmission. The Network Layer performs routing and switching functions to provide network connections. The Transport layer provides transparent end-to-end data transfer. The Session Layer provides the organisation and synchronisation for two end user processes to establish connections and exchange data. The Presentation Layer deals with data formatting and transformation. Finally, the Application Layer can be regarded as the highest level catch-all that "provides all services directly comprehensible to the users" (ISO, 1982).

Three protocol layers are defined in the ANF-10 specifications. DEC describes these layers as providing physical link management, logical link management and device control, respectively. The "physical link" layer is implemented by a protocol DDCMP, Digital Data Communications Message Protocol, and has the function of providing error free communication between adjacent nodes.

![Figure 1. Layering of OSI Reference Model.](image-url)
It thus corresponds to the "data link" layer of the OSI model and lies above the model's "physical" layer. The ANF-10 specification does not explicitly define the physical layer; the specification of the ANF-10 "physical link management" layer assumes an underlying layer corresponding to the OSI model's Physical Layer. The DEC terminology, which conflicts with that of the OSI model, was chosen in the context of viewing networks as having physical links between nodes with logical links multiplexed over the physical links. Thus the next layer up is known as the logical link layer, an ANF-10 logical link being what is now commonly known as a virtual circuit. The protocol used is called NCL, Network Control Language. This protocol is responsible for managing virtual circuits and providing error free transmission of data over them. It therefore corresponds to the "network" and "transport" layers of the OSI model. ANF-10's Device Control Protocol, DCP, controls the formatting of data exchanged and is responsible for setting parameters for remote network devices. As an example of the latter, it controls interactive terminal parameter settings. Thus DCP performs the functions of the presentation and application layers of the OSI model. The three ANF-10 protocols layers are compared with the OSI model's layers in Figure 2.

The internal protocol used in CSIRONET does not have the formal layering of the ANF-10 protocols. However, this protocol is not visible to the gateway software that performs the internetwork connections. Instead, the gateway software includes a CSIRONET interface package, CNIO, (Paine, 1980) which presents a higher level protocol interface. The CNIO interface provides facilities for creating and terminating virtual circuits and for transmitting data over them, including some capabilities for data formatting and control functions. It thus performs functions that are defined in the network, transport, presentation and application layers of the OSI model. It does not include a comprehensive set of these functions and is now progressively being enhanced with a set of layerd higher level protocols. However, since these had not been defined at the time of gateway implementation, the gateway was constrained to use just the basic functions provided by the CNIO protocol.

### 3.1.1 ANF-10's NCL Protocol

DEC’s NCL protocol defines three categories of messages: unnumbered control messages, numbered control messages and data messages. The unnumbered control messages consist of the following:

- **ACK** (acknowledgement)
- **NAK** (negative acknowledgement)
- **REP** (enquiry about last received message)
- **START** (link initialisation)
- **STACK** (acknowledgement of START)
- **NODE ID** (node identification)

When a node first comes up, it sends out **NODE ID** messages to its immediate neighbours, identifying itself. **START** and **STACK** messages are then exchanged with every node in the network to establish communications. NCL messages exchanged subsequently are acknowledged by the recipient, either by including an acknowledgement number in the appropriate field of a data message or by sending an **ACK** to explicitly acknowledge receipt. When an NCL message is not acknowledged for a predetermined period, a **REP** message is sent as an enquiry. The destination responds with a **NAK** indicating the last message received.

The numbered control messages can be grouped as follows:

1. **NEIGHBOURS**
   - **REQUEST CONFIGURATION**
   - **CONFIGURATION**
   - **STATION CONTROL**
2. **CONNECT**
   - **DISCONNECT**
   - **DATA REQUEST**

When an ANF-10 node sees a link to a neighbouring node come up or go down, it broadcasts a **NEIGHBOURS** message to every other node in the network, with information on its changed path tables. After a **START-STACK** exchange with a node that has just come on line, a node sends its own device information in a **CONFIGURATION** message and requests similar information from the other node with a **REQUEST CONFIGURATION** message. The device information includes the number of interactive terminals and batch devices of each kind that the node supports, whether or not it is a host, and, if so, whether or not it supports process-to-process communication. **STATION CONTROL** messages are used to perform privileged functions, such as causing nodes to be rebooted or dumped to host storage facilities.

The first group of numbered control messages together with the unnumbered control messages can be considered as providing the Network Layer of the OSI Reference Model.

A logical link (virtual circuit) is established between

---

**Figure 2. Comparison of ANF-10 protocols to OSI modes.**

<table>
<thead>
<tr>
<th>OSI Reference Model Layers</th>
<th>ANF-10 Protocol Layers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application</td>
<td>DCP</td>
</tr>
<tr>
<td>Presentation</td>
<td></td>
</tr>
<tr>
<td>Session</td>
<td></td>
</tr>
<tr>
<td>Transport</td>
<td>NCL</td>
</tr>
<tr>
<td>Network</td>
<td>DDCMP</td>
</tr>
<tr>
<td>Data Link</td>
<td></td>
</tr>
<tr>
<td>Physical</td>
<td></td>
</tr>
</tbody>
</table>
any pair of devices on different nodes following the exchange of a CONNECT request and a CONNECT confirm message. (Devices are taken to include host processes as well as terminals and batch devices.) The DATA REQUEST message enforces flow control in an ANF-10 network. Data messages may not be sent on any logical link until the recipient has sent an appropriate DATA REQUEST message. The sender decrements his count of data requests for each message he sends and increments it suitably on receipt of each DATA REQUEST message. This second group of numbered control messages, together with the data message category, can be considered as providing the Transport Layer of the OSI Reference model. The data messages are further subdivided into Device Control Protocol (DCP) messages.

3.1.2 ANF-10's DCP Protocol

The Device Control Protocol specifies the following message types:

- DATA
- DATA with EOR (End of Record)
- STATUS
- CONTROL
- USER ID
- FILE SPEC

Note that all of these message types are considered data messages at the NCL level. The names of the first two message types explain their functions. The STATUS and CONTROL messages are used to exchange information on current device parameters and to change them. The USER ID message is meant to carry accounting information and the FILE SPEC to carry file information. In practice, these last two message types were never implemented by DEC. The Device Control Protocol provides the functions of the Presentation and Application Layers of the OSI Reference Model.

3.1.3 CSIRONET's CNIO Protocol

Before discussing the CSIRONET CNIO protocol in detail, we should point out a distinction between this protocol and the ANF-10 protocol. The CSIRONET CNIO interface used by the gateway defines a set of "items" which allow a user process to carry out various communications functions at the Network, Transport, Presentation and Application levels. Items are defined to create and clear virtual circuits and to transmit data and control functions. Thus the items correspond logically to particular message types in a communications protocol. However, the CNIO protocol is an interface protocol that defines the format of these items, not the format of the actual packets transmitted through the network to perform the corresponding communications functions. Having made this distinction, we will refer to the CNIO items as messages in the discussion below, to allow convenient comparison with the ANF-10 protocol.

The CNIO protocol defines two kinds of messages, initialisation messages and communications messages. The initialisation messages consist of:

- RESET
- SET TASK LIMITS
- SET HOST NAME

The RESET message is used to re-initialise a group of virtual circuits. The SET TASK LIMITS message allows a program to select parameters used such as the number of virtual circuits supported and the length of the timeout period that causes a virtual circuit to be broken. The SET HOST NAME message identifies the CSIRONET host facility. These messages can be considered to belong to the Network Layer of the OSI model.

The communications messages consist of:

- CONNECT
- CONNECT RESPONSE
- DISCONNECT
- PLEASE SEND DATA
- DATA
- INTERRUPT
- SET STREAM PARAMETERS

The first three establish and terminate virtual circuits. The PLEASE SEND DATA is a request for a certain number of data messages and enforces flow control. The INTERRUPT message is used to send a single byte of information at high priority. SET STREAM PARAMETERS is used to dynamically change buffering used on a virtual circuit. The gateway treats these messages as providing a service comparable to that provided by the Transport Layer of the OSI Reference Model.

The DATA messages are further subdivided into the following categories:

- DATA W/O EOR (data without end of record)
- DATA W EOR (data with end of record)
- MODE CHANGE ITEM
- DEVICE INFORMATION
- NETWORK MESSAGE
- NORMAL EOF (end of file)
- ABNORMAL EOF

The first two messages allow exchange of data. The MODE CHANGE ITEM and DEVICE INFORMATION messages are used to read and set device parameters. The NETWORK MESSAGE, used to transmit general network status messages, is an anomaly at this level, since it belongs to the Network Layer. The EOF messages are used to signal termination of file transfers. These messages provide functions of the Presentation and Application Layers of the OSI Reference Model.

3.2 The Protocol Translation

The gateway translates the NCL and DCP protocols to and from CNIO protocol. In some cases, the messages in one protocol have an exact counterpart in those of the other; in other cases an equivalent can be found; and in a few cases no analogue can be found and the messages concerned are "terminated" at the gateway.

3.2.1 The Network Level

The network level messages in each protocol are generally terminated at the gateway. The NCL unnumbered control messages and the first group of numbered control messages deal with network level functions and the gateway merely implements them as a node of an ANF-10 network. Similarly the SET HOST NAME and SET TASK LIMITS messages of CNIO are used by the gateway in its role as a CSIRONET node, and not translated to or from any ANF-10 message. A CNIO RESET message indicates a group of virtual circuits has been broken and the gateway, on receiving this, breaks the associated virtual circuits to the ANF-10 network.
3.2.2 The Transport Level

The gateway translates basically between the Transport, Presentation and Application layer messages of each protocol. In general, a message of a particular kind in one protocol is translated to a message of the same kind in the other protocol. However, there are a few exceptions, where the gateway on receiving a message of a particular type, does not map it into its counterpart in the other protocol, but into some other message type. An example of this is the connection scheme at the Transport level for interactive terminals.

The gateway appears to be a single host on each network. When an NCL CONNECT request message is received, the gateway normally replies with an NCL CONNECT confirm message which confirms the creation of the virtual circuit. The ANF-10 device or process which initiated the connection is now connected to the gateway. The DATA message received following this is treated by the gateway as a connection request for a CSIRONET host. The gateway translates the message text of this DATA message to create a CNIO CONNECT request message. It is thus translating the DATA message to a CONNECT message. On the other hand, when a CNIO CONNECT message arrives at the gateway, initiating a new connection to it, the gateway replies with a CNIO CONNECT RESPONSE accepting this connection, but in addition creates an NCL CONNECT request message and routes it to the default ANF-10 host for the CSIRONET user making the connection. It is thus translating between CONNECT messages in the two protocols in this case.

The reason for the different actions in the two cases is that different schemes are used by the two networks for connecting interactive terminals and the gateway emulates these schemes as appropriate. On CSIRONET, the first line of text typed at an unconnected terminal is scanned for a host name. On the other hand, on an ANF-10 network each terminal has a default host associated with it, and typing any character on an unconnected terminal results in a connection to this host being attempted. The gateway's role is to make CSIRONET terminals appear like terminals connected to an ANF-10 network and to make ANF-10 terminals appear like terminals connected to CSIRONET. As part of the first function, the gateway assigns a home host on the ANF-10 network to CSIRONET terminals, and when one of these terminals connects to the gateway, the latter attempts to create a connection to the home host. The gateway emulates the other connection scheme as follows: when an ANF-10 terminal connects to the gateway, the gateway accepts the connection but awaits further information on the host desired. This information is assumed to arrive in the text of the first DATA message. The gateway implementation thus maintains consistency with the connection scheme of each network, so that no modification to either network is involved.

NCL uses CONNECT with suitable parameters to accept CONNECT requests and DISCONNECTS to reject them. CNIO uses explicit CONNECT RESPONSES in each case. A connection reject in either network is passed as a DATA message to the other network. Again the gateway protocol translation involves mapping a message of one type to a message of a different type here. The user remains connected to the gateway but is informed that his host connection attempt (explicit or otherwise) has failed.

Flow control is imposed by DATA REQUEST packets in NCL and by PLEASE SEND DATA packets in the CNIO protocol. These packets are very similar — each carries a count of data packets the sender is willing to receive. When a node is able to receive data packets, it creates and sends one of these flow control packets and adds the number requested to an internal counter. This counter is decremented for each data packet received.

The simplest form of translation here would be for each DATA REQUEST packet received from the ANF-10 network to cause a PLEASE SEND DATA with the same count value to be sent to CSIRONET. Similarly a PLEASE SEND DATA from CNIO would result in an NCL DATA REQUEST being sent. In this scheme the gateway would not need to keep track of the actual values received. However since the unit of data requested is a packet, this does not take into account possible differences in maximum packet sizes in the two networks. A better solution is to scale the count value in each case, to take into account different maximum packet sizes. Fractional values would have to be truncated, not rounded up, to preserve flow control. There are difficulties with this approach — if one network's packet size is much larger than the other's, this scaling might result in never being able to request a message from the network with the larger packet size. In addition, this scheme would lead to inefficient operation of the links: since the gateway would only request data packets from the source after it was informed that the destination could accept them, network delays would cause uneven transmission of data, leading to poor response time and throughput.

Due to these problems, it was necessary to implement a more sophisticated management of flow control in the gateway. The gateway makes a decision, on the basis of the total buffer pool available, as to how many buffers can be used in each direction on each virtual circuit. It then attempts to have these buffers full, whenever possible, in order to achieve maximum throughput. Consider for example data flow from a CSIRONET host to an ANF-10 terminal. The gateway decides that a certain number of buffers can be used for this purpose. It therefore sends a PLEASE SEND DATA packet to CSIRONET requesting the equivalent number of data packets. When one or more of these packets arrives, it enters a queue of packets destined for the ANF-10 terminal. Whenever an NCL DATA REQUEST packet is received, the appropriate number of data packets is sent in response, to the ANF-10 terminal. Following this, a PLEASE SEND DATA packet is sent to CSIRONET to keep the queue as close to full as possible.

Thus the gateway, rather than translating directly between the two flow control messages, acts as an intelligent buffering device, using the flow control protocol messages to supply a steady stream of data to the destination while ensuring that it is not itself swamped by the source.

3.2.3 The Presentation and Application Levels

As stated earlier, two of the DCP message types are in fact not implemented in ANF-10 networks. The messages that have to be handled are DATA, DATA with EOR, STATUS and CONTROL. Similarly one of the CNIO messages is not implemented, leaving the following to be handled: DATA-W/O EOR, DATA-W-EOR, MODE CHANGE ITEM, NETWORK MESSAGE, NORMAL EOF, ABNORMAL EOF.

Each network supports the same two kinds of data messages. However, since maximum message lengths may be
different in the two networks, a single data message in one network may become multiple data messages in the other. Hence, for example, CNIO DATA-W-EOR may need to be translated into a series of DCP DATA messages, terminated by a DCP DATA with EOR message.

The STATUS and CONTROL messages in ANF-10 are used to determine and change parameters such as carriage return delays and local/remote echoing on terminals. CNIO uses MODE CHANGE ITEM for changing parameters. Hence the gateway maps from STATUS/CONTROL to MODE CHANGE ITEMS and vice versa. Inevitably, some parameters allowed for on one network are not recognised on the other. For example some of the terminal characteristics supported and used by the ANF-10 network were not supported by CSIRONET and the CNIO protocol when the gateway was being implemented. In a number of these cases, the parameters were regarded as important enough to justify addition of support for them on CSIRONET and new MODE CHANGE ITEMS were defined in the CNIO protocol to allow their selection. In other cases, the features were regarded as "cosmetic" and no additional support was provided. In these cases, the gateway cannot perform the translation. Neither protocol allows for the destination to "refuse" a parameter change. To avoid an infinite loop with the source repeatedly checking status, finding it unaltered and re-attempting the change, the gateway flags the change as accepted, even when it cannot be translated.

The NETWORK MESSAGE in CNIO is "terminated" at the gateway, without translation, since it belongs to the Network level. The EOF messages, used by CNIO to signal the end of file transfers, rather surprisingly have no equivalent in DCP. In fact ANF-10 uses a DISCONNECT-initiate and expects a DISCONNECT-confirm response to terminate a file transfer. A "reason" field in the DISCONNECT message is used to indicate successful termination. The CNIO protocol uses an exchange of NORMAL EOF messages to perform this function. Hence for a file transfer link, a CNIO EOF message is translated to an ANF-10 DISCONNECT and an ANF-10 DISCONNECT is translated to a CNIO EOF. In these sequences the gateway waits for acknowledgements from the destination before returning the appropriate acknowledgement to the source.

4. DISCUSSION

While the description above has been of a mapping between two specific sets of packet network protocols, we feel that most of the problems and their solutions are common to the general case of packet network interconnection. The two networks considered were developed with somewhat different goals. DEC developed ANF-10 to link machines that excel in interactive time-sharing and considerable effort has obviously gone into the design of protocols for allowing "dumb" terminals to be switched between hosts and to continue to operate in a satisfactory way. CSIRONET also supports a large number of dumb terminals, but the original emphasis was on job submission to a large number-crunching machine and the network was designed to efficiently transport large volumes of batch device traffic. Despite the different approaches, similar functions in the different protocols could be identified in most cases and protocol messages could be mapped either directly or indirectly. Only a small number of cases required additions to the protocols to provide the desired functionality.

The general approach to the protocol translation performed here has been to understand the significance of the messages received from one network and either immediately or at a later time, generate a suitable message to the other network where possible. The alternative approach would be to attempt to directly map each message of one protocol into its equivalent in the other protocol, so that receipt of a message from one network would immediately result in the corresponding message being sent to the other network. The choice made can be considered analogous to an interpreter translating the meaning of a sentence or phrase rather than the literal meaning of the words. The second approach is easier to implement in each case but often something is lost in the translation. If the gateway were to try to translate each message of one protocol directly into a message of the other protocol, a simple table look up would be involved. No intelligence would be required in the gateway and, in fact, a simple hardware implementation might be feasible. However, as we have illustrated in our discussions of virtual circuit establishment and flow control, there are severe problems with this approach and it does not appear feasible for the general case of the implementation of network interconnection. A corollary is that implementers of protocol- translating gateways need to acquire a high degree of familiarity with the protocols involved, extending to a thorough understanding of the significance of the messages in each protocol, in order to successfully map between them. In the past, higher level protocols have been poorly defined and documented and this understanding has not come easily.

It might be thought that a gateway could be implemented as a simple low level protocol converter that translates protocols at the Network and lower levels. While it may be possible to achieve basic internetwork communications with this facility, in practice much of the functionality built into each network is usually lost. This is because the higher level user services are usually implemented very differently on different networks. For example, the way in which application programs communicate with user terminals varies widely. A protocol translation that deals only with the lower protocol layers might, in this case, produce a situation where an application program can communicate with a terminal through an internetwork link, but most of the user friendliness is lost. Unless user friendliness is preserved, internetwork links will attract little usage. In the initial implementation of the ANF-10 gateway, a number of protocol messages that dealt with the higher level functions could not be translated into suitable equivalent messages. This typically led to the loss of a user friendly feature. Effort was therefore put into making the relatively small number of modifications necessary to support some of these features, and the equivalent protocol messages were defined. This kind of minor modification to a network may be necessary when implementing a protocol-translating gateway.

When the translation between two protocols is first being designed, it usually becomes clear that certain messages in one cannot be translated into equivalents in the other. In order to avoid major modifications to each network, it is best to "terminate" these messages at gateways, wherever this can be done without significant impact on functionality (including user friendliness). However the diverse nature of the devices and processes supported on various networks makes it difficult to determine in advance where this can be done satisfactorily. While the decision...
This was due to the complex nature of the interactions between the features of the two networks, a result of their being designed differently. We believe this approach of initially minimising modifications, learning from usage and then using experimentation to decide on a minimal set of necessary modifications, is the best way to undertake translation of different protocols.

A considerable amount of research and development work on network interconnection has been carried out by the US Defense Advanced Research Projects Agency (ARPA) and by Xerox Corporation's Palo Alto Research Center. Most of this work had its beginnings in the discussions of the International Network Working Group (IFIP Working Group 6.1) in 1973, and it is therefore not surprising that a similar approach has been taken by the two organisations. In each case internetworking is based on the definition and implementation of protocols specifically for internetwork communication. The ARPA protocols evolved from suggestions by Cerf and Kahn (1974). Two protocols were defined, an Internet Protocol (IP) and a Transmission Control Protocol (TCP). These protocols have been implemented on a number of hosts and gateways in an ARPA-sponsored system of interconnected networks, known as the ARPA-Catenet (Postel, Sunshine and Cohen, 1981). The Xerox researchers have defined an internetwork protocol named "PUP", which they have used to link a number of research networks (Boggs, Shoch, Taft and Metcalfe, 1980). The Arpa and Xerox internetworks do not use protocol translation, but instead use the Internet protocols, along with a scheme known as encapsulation or wrapping (Pouzin and Zimmerman, 1978) for transmission through each network. Each host that engages in internetwork communication and every gateway must implement the appropriate internetwork protocol, IP or PUP. The source host initially encapsulates the data in an internetwork protocol packet, which is in turn encapsulated in a "local" protocol packet for transmission through the first network. The first gateway removes this network's protocol layers, processes the internetwork protocol layer and uses this to select options for transmission through the next network. The gateway encapsulates the internet protocol packet in the appropriate protocol layers for transmission through that network and then transmits it. This process is repeated at each gateway. Finally the destination host processes and removes the internetwork protocol layer.

The ARPA-Xerox approach depends on the various sites agreeing to implement a new internetwork protocol over their existing network protocols. Each group had to define a "standard" internetwork protocol for its own domain of interconnection. The IP and PUP protocols are not compatible and the two groups have had to address this problem for mutual internetworking. A scheme of mutual encapsulation has been designed (Shoch, Cohen and Taft, 1981) leading to further overheads in both communications bandwidth and processing, and to further complexity.

We felt that our environment did not justify the major effort involved in defining and implementing an internet protocol, one that would in any case be incompatible with other internet protocols, as shown by the ARPA-Xerox experience. An indication of the effort involved in developing a reliable standard protocol is provided by the Xerox researchers statement that their work diverged from the ARPA work due to "TCP's long and sometimes painful standardisation process" (Boggs, Shoch, Taft and Metcalfe, 1980).

The ARPA and Xerox internets assume a datagram service within each network for packet transmission, i.e. individual packets are injected into a network without any link set-up and it is assumed that within each network packets can arrive at their destination out of sequence, or can be lost or duplicated by the network. End-to-end recovery must be performed by the source and destination hosts. In our case, the two networks, ANF-10 and CSIRO-NET, both use a virtual circuit approach, where an end-to-end logical link is set up within the network, data is transmitted over it with the network performing error recovery and sequencing, and the link is then shut down. The gateway sets up an internetwork link as a concatenation of virtual circuits and relies on each network performing error recovery on the corresponding virtual circuit. The Xerox-ARPA datagram scheme can be used in the more general case where some of the networks linked do not provide a virtual circuit service. However, in practice, networks designed to carry interactive terminal and file transfer traffic are very likely to provide a virtual circuit service, since, as Sunshine (1977) points out, this kind of service is the best basis for such traffic. On networks which only provide a virtual circuit service, an unnecessary overhead is incurred in having error recovery handled at both the network and internetwork levels. This can be very expensive in terms of communications bandwidth, particularly for the short packets generated for interactive terminal traffic.

When our own work on internetworking and that of many research groups began, there was an absence of well defined communications protocol standards. However, during the intervening period the CCITT has been developing and refining recommendations for such protocols. X.25 is now accepted as a standard protocol for connecting hosts to public packet switched networks and a similar protocol, X.75, is used for the interconnection of such networks. (CCITT, 1980; Unsoy, 1982). Like the approach we have taken, the X.75 recommendation is based on a concatenation of virtual circuit services over different networks.

The OSI model will have a positive effect on the ease of achieving network interconnection, even before standard protocols are defined for the higher layers. New protocols that have been recently developed, and that will continue to be developed before standards are established, will conform to the common framework provided by the model. In addition, organisations that have already developed networks are publishing specifications that relate their protocols to the OSI model and in some cases modifying their protocols to achieve better conformity. All these developments make it easier to identify equivalent messages in different protocols and therefore to carry out protocol translation.

While the ISO and CCITT efforts at standardisation will lead to more compatibility between network protocols, we do not believe that the need for protocol-translating gateways will disappear. Since the original develop-
Apply now for the American Express Card and help the Australian Olympic Team.

American Express will donate $2 to the Australian Olympic Team for every approved application from February 1 to May 31.

The Card of the 1984 Los Angeles Olympics
Dear Prospective Cardmember,

You can help the Australian Olympic Team by becoming an American Express Cardmember. American Express has established the Green and Gold Olympic Appeal to raise $225,000 for our athletes.

It is an appeal with a difference. It isn’t asking you for any donations; all it requires is that you enjoy yourself with American Express.

When you join American Express between now and May 31 this year, American Express will donate $2 on your behalf. Then, every time you use the Card it will donate another 5 cents.

And American Express is giving you even more reason to be excited about the Green and Gold Olympic Appeal. They are also running the BE THERE Olympic Sweepstakes, and your name will be entered not only when you join, but also every time you use the Card. You could win a trip for two people to see the Games in Los Angeles, a complete Sharp home entertainment centre or Adidas sports gear!

It’s a great feeling knowing you are helping the Olympic effort – just by having a good time!

This is your opportunity to do something for Australia in a very positive way. In my opinion, there will never be a better time to apply for the American Express Card.

Best wishes,

John Konrads, Australian Olympian and American Express Cardmember.

Participated in Melbourne Olympics (1956), Rome Olympics (1960) and Tokyo Olympics (1964). Won three medals in swimming in Rome:
GOLD - 1,500m freestyle
BRONZE - 4 x 200m relay
BRONZE - 400m freestyle

The Card of the 1984 Los Angeles Olympics

John Konrads, Australian Olympian and American Express Cardmember.
"The world's most respected Card offers many sound, practical benefits."

Worldwide Convenience.
In over 734,000 establishments including restaurants, hotels, airlines, travel offices and stores both at home and 160 countries overseas, the American Express Card is gladly accepted as payment.

Emergency Card Replacement.
If the Card is ever lost or stolen, it can be replaced in as little as 24 hours at any American Express Travel Service Office in major local and overseas cities.
A truly reassuring benefit when you consider that many other cards cannot be replaced at all overseas.

Emergency Personal Cheque Cashing.
You can cash an emergency personal cheque for up to A$1000 at most American Express Travel Service Offices worldwide. And A$50 at participating hotels where you use the Card.

A Better Alternative to Cash.
Whereas cold hard cash can run out, the American Express Card will never leave you empty handed.
The Card has no pre-set spending limit. You are given the flexibility to spend as you see fit. And if the Card is lost or stolen, your maximum liability is limited to just A$50.

The Card also takes the worry out of exchanging currency. You see, all foreign charges are automatically converted to Australian dollars.

And to help you keep track of spending, you receive monthly statements and signed records of charge that can be used as expense account and tax verification.

A Friend Around the World.
The 1,000 Travel Service Offices of American Express Travel Related Services Company, Inc., its representatives and affiliates around the world act as your 'homes away from home'.
They'll help with travel arrangements, local currencies, Travellers Cheques and advice on where to stay, shop and dine.
They'll even hold or forward your mail.

Consider these benefits:
- Convenience.
- Emergency Card Replacement.
- Signed Receipts Monthly.
- Widespread Acceptance throughout Australia.
- Extended Payment on Air Tickets.
- Cardmember Accommodation Plan.
- Exclusive Personal Travel Account.
- International Currencies Automatically Converted.
- A Friend Around the World.
- Personal Cheque Cashing Worldwide.
- No Pre-Set Spending Limit.
- Worldwide Mail Service.
- Automatic Travel Accident Insurance.
- Company Membership.
- Family Supplementary Card Membership.

See back panel for more benefits of the world's most respected card.
The American Express Card may be more difficult to obtain, but now it’s easier to apply.

DECLARATION: I hereby apply for the American Express Card. I warrant that the information volunteered by me is true and correct. I authorise American Express to confirm and exchange credit information concerning my financial affairs. I acknowledge and understand that subject to acceptance I will be billed at a later date for a joining fee of $30 and an annual fee of $40 for the Basic Card and an annual fee of $30 for each Family Supplementary Card and $35 for each Company Supplementary Card. I and all of my Supplementary Cardmembers hereby agree to be bound by the Terms and Conditions notified to me by American Express at the time of issue or replacement and any amendments thereto subsequently so notified. I acknowledge the right of American Express to vary the joining fee and annual fee in respect of the Card without notice. I agree that a microfilm or other reproduction of this application may be produced by American Express as evidence of my application for the Card.
Simply fill in this application and mail to: American Express, G.P.O. Box 1582, Sydney N.S.W. 2001. Phone 239 0888 in Sydney or 008 230 100 from anywhere in Australia for the cost of a local call.

Upon review of your application, it may be necessary to telephone you to obtain further information to assist us in assessing your application.

MR □ MRS □ MISS □ MS □ DR □ Date of Departure ___/___/
Surname

Given Names

Home Address

Postcode Date of Birth

Tel. Business (_____) Tel. Private (_____) 

Occupation Income

If salary is less than $15,000, specify other sources of income or assets

Employer's Name

Years there

Employer's Address

Postcode

Your Bank and Branch

SUPPLEMENTARY CARD MEMBERSHIP Do you require an American Express Card for members of your family or company?
Name(s) of Supplementary Cardmember(s)

Yes □ No □

Refer to declaration overleaf.

Applicant's Signature X Date ___/___/

For any Company Card the signature of the Managing Director or Company Secretary is also necessary. Please sign below.

Signature X Title

TEAR ALONG PERFORATION AND INSERT INTO POST PAID ENVELOPE BELOW.

Don't leave home without it.
Apply Now.

No postage stamp required if posted in Australia

BUSINESS REPLY POST
Permit No 8633 issued at Sydney
Postage and fee will be paid on delivery to:

American Express International Inc.
Travel Related Services
G.P.O. Box 1582
SYDNEY N.S.W. 2001
Cardmember Accommodation Plan.

By quoting your American Express Cardnumber when booking accommodation at over 10,000 motels and hotels around the world, you can ensure that a room will be kept for you, regardless of how late you arrive.

And by paying with the American Express Card you'll also enjoy the many benefits of the American Express Check-in/Check-out Service.

There will be no more delays, queues or congestion when you register or leave your hotel.

Company Membership.

The American Express Company Card gives you both the benefits of the personal Card as well as tighter control over your company's travel and entertainment expenses.

This helps reduce administrative costs and improve cash flow.

The Benefits of the Card to Members of Your Family or Company.

As an American Express Cardmember, you are entitled to give anyone in your family or Company, a Card of their own.

These Cards carry their own name and signature so that they may also enjoy the security, convenience, recognition and emergency protection the American Express Card gives you.

The charges incurred are billed to you, but listed separately and identified by the relevant Cardnumber.

For each Personal Supplementary Card there is an annual fee of $25 and $30 for each Company Supplementary Card.

The World's Most Respected Card.

Whether you're travelling, staying in hotels, shopping or dining both at home or overseas you'll discover that the American Express Card is the world's most respected Card.

An invaluable credential wherever you travel.
ment of packet switching for wide area networks, a great
variety of networks has sprung up. Packet radio networks
and satellite networks compete with or complement net-
works using terrestrial leased lines. The last few years have
seen an explosive growth of local area networks using dif-
ferent techniques such as Ethernet CSMA, token rings and
token buses. It is unlikely that a single set of standard
protocols will be optimum for all these types of networks.
In the last decade, dramatic advances in large scale
integration of circuitry have made it possible to move more
and more communications functions from software to hard-
ware and to achieve substantial cost savings in doing so. At
the same time, much greater effort is being put into for-
mally defining protocols, and this makes it easier to imple-
ment them in hardware. We can expect to start seeing
VLSI communications interfaces that implement large
portions of standard protocols such as X.25 and other pro-
tocols emerging from the work of ISO, CCITT and IEEE
802 committees. The current generation of gateways has
software that implements the protocols of each network
and software that translates between the protocols. In
future, the protocol implementation may be all done in
hardware, leaving the gateway software to perform just the
translation. This should lead to simpler, cheaper, more
reliable gateways. In the longer term the protocol transla-
tion function may itself migrate to hardware for the more
popular protocols.

ACKNOWLEDGEMENTS

Parts of the ANF-10 gateway software, including ex-
tremely useful facilities to assist in down-line loading and
remote debugging, were developed by John Gibbons.
Cecilia Chan produced the system documentation and per-
tremely useful facilities to assist in down-line loading and
remote debugging, were developed by John Gibbons.

REFERENCES

BOGGS, D.R., SHOCH, J.F., TAFT, E.A. and METCALFE, R.M.
CCITT (1980): Yellow Books, Data Communication Networks,
Vol. VIII, Fascicle VIII.2, VIII.3.
CERF, V.G. and KAHN, R.E. (1974): A protocol for Packet Net-

Digital Equipment Corporation (1977): Networks Programmer's
Digital Equipment Corporation (1982): Introduction to DECNET,
AA-J05SC-TK.

BIographies

Joseph Fernandez is a Senior Research Scientist with the
CSIRO Division of Computing Research in Sydney. He
received a BSc degree from Makerere College, Uganda in
1965 and a PhD from Washington University in 1972. His
current research interests are computer network intercon-
nection, computer based message systems, and the applica-
tion of VLSI to computer communications interfaces.
Using the UNIX System on the AAEC Network

By G. W. Peady

An outline is given of the Australian Atomic Energy Commission computer network which connects the many computers at the Lucas Heights Research Laboratories. This network allows high speed data transfer between any two connected computers.

A description is given of the software interface which allows communication between processes running under the UNIX timesharing system and other computers on the network. The need for easy communication with the network is essential, although the protocols required are quite complex, they should not concern users. This paper shows how the complexities involved can be hidden in the system handling software, allowing the user ease of programming the network in a high level language yet retaining the complete flexibility offered by the network.

Keywords and phrases: Interface, network, protocol, UNIX.

CR categories: C.2.1, C.2.2.

1. INTRODUCTION

Modern communication networks are mostly high speed serial links between nodes, e.g. Ethernet, Cambridge Ring (for example, see Clark, Pogran and Reed, 1978 or Wolf and Sideris, 1982). At the Australian Atomic Energy Commission (AAEC), a parallel bus communication network called the Dataway was designed and constructed and was first documented by Richardson (1971). This network, probably the first network of its type in Australia, has been in use for over 12 years and provides for intercomputer communication for more than 16 computers. The Dataway has a 24-bit wide data transfer at a maximum rate of 0.5 Mbits/s, which, although slow by modern standards (e.g. 10 Mbits/s for Ethernet), is still reasonably fast considering its technology. The Dataway is a local area network with a total length of approximately 2 km of cable within the Lucas Heights Research Laboratories (LHRL). Attached to the network are several different types of computers including many Digital Equipment Corporation (DEC) PDP11s, several Data General Novas and a DEC PDP9L computer (see Figure 1). The PDP9L is an interface to the network for the central computer, an IBM 3033S. This paper discusses the network and the connection of these systems into it.

Until recently most of the traffic on the network was directed to the central computer and the connected computers acted as terminal concentrators and/or remote job entry/exit stations for it. However, the connection of several larger PDP11s capable of running their own operating systems meant that access to them from the network (and vice versa) was needed. All these computers run level 6 UNIX (or IDRIS - a UNIX look-alike). UNIX is a time sharing operating system which provides a sophisticated environment for program development (Thompson and Ritchie, 1978). It is gaining wide acceptance in the mini- and micro-computer field as a vehicle for software development by both commercial and scientific users.

2. COMMUNICATION CONVENTIONS AND PROTOCOLS

To try to put it in a current framework, the network will be described in the context of the International Standards Organisation Open Systems Interconnection model (ISO, 1982) which, although not entirely applicable, does provide a suitable basis for discussion.

2.1 The Physical Layer

At the lowest level, the Physical, the Dataway consists of 37 parallel lines (plus earth lines), 24 of which are used for bi-directional data transfer and address selection, three are used for parity and ten are used as control lines. For each computer there is a control unit which checks for valid addresses for that computer and monitors the control lines. At this level no protocol is imposed on data transfers. However, there is a hardware monitor which will force a termination of an active data transfer sequence if no activity has occurred for a fixed time. The hardware also contains circuitry for the detection of parity errors in a data transfer. A complete description of the hardware can be found in Ellis (1970).

2.2 The Data Link Layer

The OSI second layer protocol, the Data Link layer, is handled by a hardware interface for each computer. This interface manipulates under computer control the control lines on the network and handles direct memory access.
between computers. On PDP11s the interface is a normal device with registers for status, direct memory access and address selection. At this layer a protocol is imposed for data transfer and requires:

1. Initial connection between specified addresses (computers),
2. Optional data transfer,
3. Exchange of error status on completion,
4. Release of the network.

2.2.1 Initial Connection
The initial connection between two computers is made by one of the computers placing the address of the required computer, its own address and a command into the network control registers and then requesting the use of the network. If the network is available, the signals are placed on the network and the requested computer can indicate its acceptance of the request with another signal, thus completing the connection. If either the network is unavailable or the requested computer does not reply then these conditions are signalled by the hardware. The interface to the network is, in modern terminology, multiple access with collision detection (MA/CD).

The Dataway has 256 possible commands where a command is a code which accompanies each sequence and is usually used to indicate the type of data which is sent in the transfer. For example the command 5 indicates that the data accompanying the command is to be printed and followed by carriage return and line feed characters, while the command 6 indicates that the data is to be requested from the terminal with no echoing of entered characters. There are also 256 addresses for communication and each computer has a unique set of these addresses assigned to it.

2.2.2 Optional Data Transfer
After the initial connection the computer which received the request can determine a suitable reply to the command. If the command requires some data transfer, this is indicated to the network hardware interface and the transfer occurs without processor intervention and is followed by an interrupt on completion.

2.2.3 Status Exchange
Each network sequence is completed by an exchange of status between the two machines. This exchange conveys information on the acceptance of the data, successful receipt of the data or suitability of the command. Since the higher level protocol is derived from a magnetic tape system, terms like Unit Check, Channel End and Device End are commonly used as indicators of status on completion of a sequence.

2.2.4 Release of Network
After the status exchange each computer must release the network and a hardware monitor on the network will generate a disconnect signal to all units if a sequence has not completed properly. This ensures the availability of the network for other computers.

2.3 The Network Layer
This layer provides for the establishment, maintenance and termination of network connections. Since the central mainframe computer at LHRL was for a long time the only active computer on the network the protocol for this layer follows the protocol for an IBM magnetic tape unit (Richardson, 1973). This protocol is easily implemented on the central computer where each address on the network appears as a magnetic tape unit. This historically based convention provides all necessary error correction and detection protocols and has been followed in the interfacing of the UNIX computers to the network.

Unfortunately, this protocol results in asymmetric communication between the two computers. That is, since tape units do not usually request any data transfer but simply indicate that a particular condition has occurred, a tape unit based protocol expects that one of the partners will be passive during the transfer. Consequently, on the network the concept of active and passive partners is well established. The active partner, usually a network host machine, is the computer controlling the type of data transfer, with the type of transfer being affected by the occurrence of certain conditions in the passive partner. All computers (except the central computer) on the Dataway can take a passive role but, currently, only a few computers can take an active role.

Another aspect of this protocol is that, since there are 256 different commands possible and these all occur at the Data Link layer rather than at a higher level, then the interpretation of these commands must occur at this lower level. This aspect could be overcome by using only one type of Dataway command for reading, another for writing and embedding the real command in the data for use at a higher layer in the protocol. This would then be much closer to the more usual protocols for local area networks.

Since the UNIX system may assume both active and passive roles on different occasions, both types of access are supported. However on those occasions when it was the passive partner, the protocol on the UNIX system had to be contrived. A fairly detailed discussion of how the UNIX interface is implemented can be found in Peady (1981).

2.4 The Session/Transport Layers
The OSI layers of Session and Transport are not distinguishable in the Dataway network and can be considered under the one heading. A session on the network is established by the initial communication between two cooperating computers. It is usually started with a Rewind command and terminated by a Rewind/Unload command, reflecting its magnetic tape origins.

The session protocol was described by Richardson and Sanger (1978). Briefly it is that a computer (the passive computer) which wishes to use the services of another computer (the active computer) sends an initial request to that computer. If this request is accepted then a Rewind command is sent to the passive computer to indicate the acceptance of the request and the start of the session. If the request is not accepted then a No Access command is sent instead.

Once the session is established, the active computer sends commands as needed. The flow of these commands can be affected only when the passive computer returns an appropriate status or issues a single command called an Attention. These methods are the only ones the passive computer can use to affect the flow of commands from the active computer. They correspond to exception conditions on a magnetic tape unit. Certain error conditions can occur during the transfer of data and the protocol allows for the detection and correction of these by use of the ending status. For example the ending status of Unit Check may indicate that there was an error detected in
The passive computer can indicate that it no longer wishes the received data. It is also possible to detect an error in the sending of the ending status. At the conclusion of the session, the active computer sends a Rewind/Unload command to indicate termination. The passive computer can indicate that it no longer wishes to keep the session active by sending an Attention command with an appropriate data pattern.

2.5 The UNIX Implementation

The problem of interfacing UNIX to the Dataway network are in many ways similar to those of the UNIX/Cambridge Ring (Collinson, 1982). There are multiple hosts on each system with different operating systems and environments to contend with. The Cambridge Ring interface has the advantage of an intelligent controller to handle most of the Network and Link layers. Other UNIX network systems (Martin, 1982; Schmidt, 1981) use a serial connection between computers and provide the network support using well defined protocols. The main difference in most of these systems is that the addressing information is interpreted at a higher level than the Data Link layer, usually being contained as part of the data package.

Since control of the hardware and error recovery is fairly complex and many of the responses must be handled immediately, there was no real possibility that the network handler could be a user process as is possible with most other communication networks (Collinson, 1982). Consequently, the handling was performed in the kernel.

The Dataway handler is part of the UNIX kernel and treats the network connection as a character device when it is the active partner and as a raw block device otherwise. The reason for this dichotomy will be seen in the following sections. In the active mode, the usual character queues are used for data transmission. In the passive mode data is transferred, using direct memory accesses by the Dataway hardware, from the user's address space in the normal fashion for raw block devices. The original implementation of the Dataway handler provided a special system call for user programs to get access to the handler variables and also some modifications to the system. When it became necessary to implement the handler on the IDRIS system (White-smiths, 1981) it was not easy to provide this new system call, as only a binary license had been purchased, consequently the handler was modified to allow access using the system call used for modifying character device characteristics (i.e. stty/tty). However, no other modifications to IDRIS were necessary besides those implemented in the device driver. This new implementation means that the handler can be easily transported to later versions of UNIX (or IDRIS).

2.5.1 Active Control

For communication between two UNIX machines, one machine must adopt an active role and the other a passive role. The active role is illustrated by the handling of network terminals by a network UNIX host machine. The control of these network terminals was designed with the aims of ease of use and to involve no change to the usual terminal handling software. These aims have been achieved and no difference between directly connected terminals and terminals connected by the network is apparent to any system software. The differences are hidden at the system call level. The system call to open the 'terminal' is only completed when a computer on the network indicates that it requires the services of the UNIX computer. The system call to write characters to the terminal sends these characters with a Write command on the network to the other computer. The system call to read characters obtains the characters from the UNIX character queues which are filled by network read commands issued by the UNIX network driver. This method is used to simulate the normal asynchronous entry of characters onto the UNIX character queues by only issuing the network read command when the network 'terminal' indicates that a character has been entered. This method also has the added advantage that a full line can be assembled on the network 'terminal' and passed, fully processed, to the UNIX computer.

On the network there is a method for prematurely terminating a session which is indicated by sending a HANGUP signal to those UNIX programs that are currently communicating with that network 'terminal'. Signals in UNIX are effectively asynchronous interrupts to a program, and were implemented initially to notify error conditions. They may be used for interprocess communication provided certain precautions are taken (see below). The UNIX program may elect to handle these interrupts if desired.

It is still possible to completely control the network commands used for reading and writing data if the defaults are not sufficient. It is also possible to force a System read request to issue a Network read request. For example the following sequence of commands executed by a UNIX user program would use the default network command for the initial System write, change to use the network command SPECIAL for the next System write, and force a network read command (FREAD) to be sent for subsequent System reads.

```
write(dwflid, buffer, bufsize);
dwcmd(dwflid, SET_WRITE, SPECIAL);
write(dwflid, buffer, bufsize);
dwcmd(dwflid, SET_FORCE_READ, FREAD);
read(dwflid, buffer, bufsize);
```

The routine 'dwcmd()' provides the access to the System network handler to enable the user to change the handler characteristics. As mentioned earlier it is implemented using a standard UNIX system call.

2.5.2 Passive Control

Implementation of the passive use of the network was more difficult than the above for two reasons:

1. To be able to establish a connection between two computers on the network it is necessary for these to be specified at some stage in the data transfer. This would preferably be at the opening so that the connection could be made for the duration of the session.

2. The network command which the active computer sends is not known in advance to the passive computer, as the sequence and type depend entirely upon the active computer.

There appear to be two general methods of solving this problem. The addressing and command information could form part of the data to be read or written. This method was rejected as clumsy as it would require the setting aside of buffers of indeterminate size in the kernel address space and possible error returns to system calls if the data was sent to the opposite direction, i.e. if the user program in the passive computer issued a Read system call and the network would only allow the equivalent of a
write then the Read would have to be given an error return
to indicate that a Write was really needed. Also, the data
would need to be transferred from the kernel buffer to the
user area and would have to include several extra bytes for
routing and command data. More importantly this
method was not consistent with existing network software.

The second method is the one which has been adopted. Every command from an active computer is given
a Busy status and a signal is sent to the controlling UNIX
process to indicate that a Dataway request has occurred.
However the use of signals for this type of activity is
fraught with danger because the signals in Standard level 6
UNIX are not stacked and, as the network operates asyn-
cronously, a signal may be lost if it coincides with another
signal. To avoid this, either signals must not be used with a
process in the passive state or UNIX must be changed so
that it stacks the signals. Once either of these remedies is
adopted, the UNIX computer is able to adopt a passive
role fairly easily.

The initial connection between the two machines on
the network occurs in two stages. Firstly the UNIX
program must indicate the two network addresses which are
to be used for the connection, that is the address of the
active computer and the address to be used on the UNIX
machine. Usually the UNIX machine has a range of address-
es which can be used and, for the sake of portability of
software between machines, the addresses on each machine
are referred to by the generic names of base0, base1 etc.
The address of the active computer can be any of the host
machines on the network (other addresses maybe be used but
of course will not reply). Currently there are four host
machines available, an IBM running MVS and three PDPlIs
running UNIX/IDRIS. The logical connection required is
made known to the UNIX network handler by opening a
special device with an appropriate name. For instance,
opening /dev/IBM/base2 would make the logical connection
between the IBM host machine and the network address
base2 on the UNIX machine.

The second stage of the establishment of the connect-
ion occurs when the UNIX program sends a network write
command containing data to indicate the type of service
requested of the host (or active computer). Once this write
command is sent, all further communication between the
machines is under the control of the active computer.

After this initial connection has been made the hand-
ling UNIX program, on obtaining a signal from the UNIX
network handler, determines the Dataway command,
selects the buffer and transfer count for that command and
issues the appropriate Read or Write UNIX System call. The
UNIX network handler accepts these Reads or Writes and
then indicates to the active computer that it is no longer
busy and that the data is now ready for transfer. The active
computer then re-issues the Dataway command and the
data transfer occurs.

The session can be terminated in two ways: either the
active computer decides that the session is completed and
sends a network command to that effect; or, generally in
the case of an abnormal completion, the UNIX program
can call the System Close routine which indicates to the
other machine that the connection is now no longer
required.

2.5.3 Error Conditions

Certain errors may arise during a sequence, and those
detected by the network hardware are handled by the
UNIX network handler routines. The user program is
notified of an error only after a large number of retries has
been unsuccessful. Other error conditions occasionally
occur which cannot be detected by the network hardware.
Such conditions as apparently lost sequences may interrupt
further communication. The UNIX handler corrects these
errors by using timers to detect lost sequences and it takes
the appropriate corrective action.

Other errors can occur which are more subtle and
generally caused by hardware malfunctions. These are diffi-
cult to detect but certain precautions exist. Events such as
interrupts out of sequence and differences in transfer
counts are handled appropriately and a record is kept of
their occurrences. Certain diagnostic tests are built into the
network handler to allow other computers on the network
to cycle against the UNIX computer, to try to detect any
errors which may be occurring.

2.6 Higher Layer Protocols — Presentation and
Application

These higher level protocols are not well defined on
the network and are only valid for a particular use. There is
one common protocol used for interactive terminals on the
network and this allows terminals connected to any com-
puter on the network to communicate with any active net-
work computer (Sanger, 1976). This is the basic protocol
which has been implemented to allow terminal access to
the UNIX system which is then in the role of an active
computer on the network. To give a terminal already con-
ected to a UNIX computer further access to the network
requires the UNIX system to use the network passively.
This software allows access to all other active computers,
including other UNIX computers, and it provides similar
facilities to those available on other UNIX networks (Joy
and Leffler, 1982). Thus any UNIX user is able to login to
any other UNIX computer as a normal terminal and have
extremely fast access to that computer. The program also
enables files and other information to be transferred
between the two computers.

Other protocols have been developed for particular
transfers of files between computers on the network. These
protocols involve error detecting mechanisms using check-
sums and special commands for checking sequencing of
blocks in the data transfers etc. These protocols are devel-
oped for particular applications and are not generally valid
except for these applications. Further protocols have been
developed for interactive graphics, loading of remote com-
puters, etc. Currently work is being done on extending the
use of the network so that files on one machine can be
accessed easily from another machine in a reasonably trans-
parent fashion.

3. CONCLUSION

An easily controlled software interface for UNIX
systems to the AAEC network has been designed and imple-
mented to permit network access to and from any UNIX
computer. This allows UNIX systems to use the network as
equal partners with the centre's IBM computer system.

4. REFERENCES

1497-1517.

COLLINS ON, R.P.A. (1982): The Cambridge Ring and UNIX,
Using the UNIX System


BIOGRAPHICAL NOTE
Glynn W. Peady received a BSc with first class honours in Mathematics from the University of New South Wales in 1970 and a PhD in Applied Mathematics from the University of Melbourne in 1974. Dr Peady was a lecturer in Mathematics at Mitchell College of Advanced Education in 1973-74 and has been a Research Scientist at The Australian Atomic Energy Commission since 1974. His research interests are computer networks, graphics and microcomputer applications.

REQUEST FOR AN EARLY COPY OF THE JOURNAL

Part of my ‘inheritance’ as editor of the Journal was a collection of back issues reaching back to about 1979. Thanks to the generosity of Professor Murray Allen, I have been able to extend the collection back to Volume One, so that it is now almost complete except for one issue —

Volume One, Number Three, November (?) 1968

If any reader has a copy of this issue and would be willing to donate it, I would be extremely pleased to hear from him (or her).

The contents of the issue (one of two for 1968) were as follows:

P.R. Masters — Evaluating programmer performance.
I.N. Capon — Computing science curricula for Australian universities.
E.S. Burley — Real-time in retrospect.
J.A. McGeorge — CARBINE — a Computer Automated Real-time Betting Information Network.
N.W. Bennett — ABACUS — a fast Fortran system for the IBM/360.
J. Oliver — Tree-searching school timetables.
C.E. Wallington — Display systems in numerical meteorological experiments.
J.F. Temperly — On the handling of a particular data structure.
G. Gregory and K. Sharpe — A generalisation of the rejection procedure for the generation of random variables.
R.S. Andersen — On the reliability of finite difference representations.
J.M. Bennett, P.C. Cooley and Jennifer Edwards — The performance of an integer programming algorithm with test examples.

J. Lions, Editor
Discussion Paper

Database Retrieval Technology and Subject Access Principles

By G. W. Greenleaf and R. A. Clarke*

Thom and Thorne (1983) questioned the efficacy of proposed subject access to personal information because of features of relational databases. Those arguments are extended and a similar argument applied to free-text retrieval DBMS. Three sets of information privacy principles, those of the NSW Privacy Committee (1977), the OECD (1980) and the Australian Law Reform Commission (1983) are assessed against these difficulties. Weaknesses in these principles are identified. Suggestions for rationalisation of the Australian Information Privacy debate are made.

Keywords and phrases: right of access; subject access; individual access; data-subject access; record-subject access; privacy; data protection; freedom of information; deductive database; relational database; free-text retrieval.

CR categories: E.0, F.4, H.3, I.7, J.1, K.5.

1. INTRODUCTION

The recent papers by Thom and Thorne (1983) and Bushell (1983), published in this Journal, are among the first papers published in Australia which subject information privacy principles to the type of detailed scrutiny that they deserve. The Australian Law Reform Commission’s (ALRC’s) Privacy Report (ALRC, 1983) has since been published, containing a new set of Information Privacy Principles and a Draft Privacy Bill.

Thom and Thorne raise problems involved in applying data protection principles to relational databases. We argue that similar problems arise from recent developments in free-text retrieval technology. They doubted that protection principles and regulatory mechanisms, particularly the individual access principle, are capable of adequately dealing with these problems. We examine the principles espoused by the NSW Privacy Committee (1977 and 1983) and the OECD (1980) and conclude that, if taken as a whole, and sympathetically implemented, they may be more effective than Thom and Thorne suggest. We conclude that the ALRC’s principles face greater difficulties in coping with these problems.

2. RELATIONAL DATABASES

2.1 The Subject Access Principle

Thom and Thorne raise an important difficulty in defining the meaning and operation of the principle of subject access when applied to recently developed relational databases. The principle, that an individual should have access to all information concerning himself is found in some form in all known formulations of information privacy principles (see ALRC, 1983, para 1235).

The OECD (1980) refers to the “Individual Participation Principle”, the NSW Privacy Committee (1977) to “subject access” and the ALRC (1983) to “Access to Records of Personal Information” and “record subjects”. Because we have some reservations about the use of “records” rather than “data” or “information” (see para. 6), and for brevity, we have simply referred to “subject access”, the “Subject-Access Principle”, and “information subjects”.

2.2 Explicit and Implicit Information

Thom and Thorne argue that the operation of this principle is reasonably clear for a database structured on a hierarchical model, but is unclear for databases structured by logic programming (relational and deductive databases, defined in Thom and Thorne, 1983, p. 146). The reason is that hierarchical databases are “accessed by a single primary key”, by which we take them to mean that every record which contains information relevant to an individual will also contain a data item, the “primary key”, capable of identifying that individual, and so all information concerning the individual is “stored explicitly” and can be easily accessed. In contrast to this “explicit information”, “implicit information may be retrieved only by accessing and combining the information held in different records”, possibly by using rules which posit relationships between data items held in different records. Deductive databases are extensions of relational databases by the inclusion of such stored rules. We use ‘relational’ to include both the relational and deductive models. Relational databases are characterised by the relative case with which the stored rules enable the extraction of information stored implicitly, in contrast to the considerable difficulties encountered in manual systems or hierarchical databases.

A further distinction is made between two types of
explicit information, that "which can be derived from the database using the stored rules and a single predicate query" (inferable) and that "which can be accessed only by the use of more complex queries and some external rules which are not stored as part of the database" (potentially inferable). The terms "inferable" and "potentially inferable" are used, as Thom and Thorne do not provide any, and they use "derivable" in a different sense from our use of "inferable".

2.3 Problems with Access to Implicit Information

This discussion leads to Thom and Thorne's fundamental questions: how is implicit information included in "the information held concerning an individual" for the purposes of the principle of subject access; and how can any regulatory mechanism hope to make a right of subject access effective in the face of such sophisticated databases?

They conclude that "the individual must have access to rules and implicit facts as well as the explicit facts" but "due to the possibly dynamic nature of rules, it may not be possible to determine all information attributable to an individual".

In illustrating these problems Thom and Thorne use an uncontentious example of a "family tree" database, but they mention the Costigan Commission enquiries (The Royal Commission on the Activities of the Federated Ship Painters and Dockers Union) as an illustration of the potential privacy dangers of the use of relational databases.

The Costigan Commission developed "a structured database" from "public and government records, the records of financial institutions and the personal records of the people being investigated and the people with whom they dealt" (Meagher, 1983a). The system's "personal indexing system" captures against a person's name virtually any information known about the person's characteristics, history, associates or actions. "By use of link analysis, the system can be employed to produce all known associations of a specified individual, whether the association is direct or indirect. Indeed, if all links between two specified persons are needed to be known, the system can produce all of the paths between the two, even if there are several intervening persons".

The Costigan Commission is a small organisation set up to investigate the affairs of "upwards of 2,000" people and in doing so it has established a database capable of interrelating information in over 2.5 million folios. Meagher's opinion is that the subject access principle "is revealing are a clear illustration of the potentially sensitive and damaging nature of such implicit information."

The ALRC has recognised that "concern might be expressed at the reach of the net that has been cast and the very large number of apparently law-abiding Australians who are caught up in that net, by reason of some association with others in activity which in most cases is probably innocent or trivial" (ALRC, 1983 para 533).

Criminal intelligence raises special problems for the operation of any data protection principles. However, Meagher's opinion is that the individual access principle "is not appropriate where the data base is being used to investigate whether that person is involved in a criminal organisation" (Meagher, 1983b p. 140). It is beyond the scope of this paper to discuss this issue (see Meagher, 1983b, pp. 87-93, 138-40; ALRC, 1983 para 533, 1418).

2.4 Latent Information

To answer "what is data relating to an individual?", we consider that as well as explicit and implicit information a further category of "latent information" must be identified and defined.

For example, consider an allegation that Smith attends the races with Jones, which information is held only in the comments field of a record "about" Jones in the sense that it is only accessible by fields containing identifiers to Jones. If the system also has no stored rules of inference concerning racing attendance or associates, Smith's involvement is not inferable. Also, the database structure or existing software may not support such potential derivation rules. For searches designed to find information about Smith, the racing attendance information is currently unretrievable ("latent") without replacement of software or database restructuring. Nevertheless, the racing attendance allegation is clearly information in the database about Smith, and is personal and potentially prejudicial. It may be discovered and used to his detriment by accident, or by exhaustive search.

We define "latent information" as information in a database which is about a person and contains an identifier to that person, but which:

(i) is not explicit in that it cannot be accessed by use of that identifier;
(ii) is not inferable by any stored rule;
(iii) is not potentially inferable by any external rule; and
(iv) is only discoverable accidentally or by exhaustive search.

Although the dangers to privacy posed by accidental retrieval, software replacement or database redesign are less than those posed by implicit information, it is important that the existence of latent information be acknowledged as a limitation on the value of the subject access principle and other privacy protection principles. It also constitutes an important distinction between relational and free-text databases, as will be discussed later.

2.5 Categories of Database Information

We are now able to expand the categories of information which may be contained in databases beyond those proposed by Thom and Thorne by addition of the categories discussed in 2.2 and 2.4 above.

The information content of each exclusive ring comprising the database is shown in Figure 1.

Figure 1.

A = explicit*  
B = inferable  
C = potentially inferable  
D = latent (or non-attributable)  
A+B = derivable*  
B+C = implicit*  
A+B+C = attributable*  
A+B+C+D = total personal information in database

(Terms marked * follow Thom and Thorne)

Considered from the perspective of the rules and capacities of the system allowing information retrieval, circle 1 represents the stored rules, circle 2 the external
rules (potential rules), circle 3 the structure of the software and the database, and circle 4 the distinction between information contained in the database and other information.

3. FREE-TEXT RETRIEVAL TECHNOLOGY

Free-text systems are characterised by the database over which they operate containing the full text of documents (newspaper articles, letters, telephone transcripts, court judgements), with very limited structuring of the texts during data capture, (e.g. markers to identify paragraph commencement). In this sense the texts stored are "raw data", and are in stark contrast to the data definition language used by relational and other structured databases. However, either highly structured indexes or data compaction techniques are used in conjunction with the "raw" text files.

Such databases are similar to relational databases in the use of logic as a data manipulation language. Boolean search logic, using the familiar Boolean connectors "or", "and" and "not", in combination with a variety of contextual or positional operators, allows the retrieval of all documents in the database in which a string, word, phrase or particular logically specified combination of words occurs. "There are two kinds of free-text systems: all-software inverted file systems, and the hardware driven, associative file processors" (Stephen, 1983). The more common software products include STAIRS, STATUS and BASIS which are of general application and LEXIS which is specific to legal material. The Datafusion Associative File Processor and ICL's Content-Addressable File Storage (CAFS) are hardware examples. The differences between the two types are not of significance to this article.

3.1 Uses of Free-Text Retrieval Technology

Unstructured, discursive information has until now resisted widespread inclusion in computerised databases. The high cost of manual summary extraction and index construction constituted a de facto privacy protection for the extensive personal information contained in such texts. Free-text retrieval systems automate indexing or data structuring.

Until now, at least publicly-accessible databases have not included significant amounts of personal information (see Infogrow, 1982). With data capture and storage costs also decreasing, more applications involving such information will result, including:

(i) newspapers and periodicals, such as in the Australian Financial Review Information Service (AFRIS);
(ii) court reports;
(iii) criminal and national security intelligence systems;
(iv) documents of organisations which give and receive written advice e.g. policy oriented government organisations, solicitors;
(v) customer and employee records (see Clarke, 1983).

3.2 Information Privacy Problems

The ALRC has suggested nine most prominent sources of concern for privacy which have been generated by computerisation (ALRC, 1983 para 118). Consideration of these makes it arguable that widespread application of free-text systems to personal information would constitute a greater danger to privacy than any other type of database. We limit ourselves here to a consideration of the implications of relational and free-text systems for the Subject Access Principle, a matter not discussed by the ALRC.

In one sense, a free-text system facilitates subject access by a more powerful retrieval method than is available in any other type of database. An individual can search the whole database for every occurrence of his or her name or any other personal identifier, and inspect all contexts in which such occurrences occur. It makes no difference that the information may be in a record which is primarily about someone else, and there are, thus no latent facts. In free-text systems, therefore, "explicit information" includes every record which contains an occurrence of an individual's identifier. This feature of free-text systems also demonstrates their potential for invasion of privacy when utilised by those other than the individual information-subject.

Although 'explicit', the information may be so discursive and extensive as to be virtually meaningless unless the information subject has some way of knowing which of it the user regards as relevant.

The distinction between explicit and implicit information made by Thom and Thorne is therefore of different significance with free-text systems. This is because every search command in a free-text system other than simple searches for every occurrence of a person's name can be said to generate "implicit information" in that the rule (the search command) must be known before what is considered to be "the information held concerning an individual" can be determined. For instance, the search may be to find all instances of "Smith" and "bribe" occurring in the same paragraph. Most information could be said to be only "potentially inferable", in that with free-text retrieval any search may involve the construction of "new rules", because the Boolean search used may be unique to the search and might not be "added" to the "stored rules", but discarded. In the extreme, there may be no "stored rules" except the handful of basic Boolean search logic rules, which are used to generate an indeterminate number of new queries. In practice, commonly used command sequences are likely to be stored in a library of procedures. Similarly, the results of any search may be used (to the subject's prejudice, perhaps) and then discarded, but the prejudicial information will remain in the database.

The resulting privacy problems are problematic for relational databases. If there are no or few stored rules, then "complex associations" may be made in a unique search, which the data subject has little hope of anticipating or duplicating. A data protection authority may also have difficulty in determining "all information attributable to an individual" when even its skilled investigators have no stored rules to work with, but just a mass of free-text and the rules of Boolean logic. There is indeed a "potential to make the right of access principle unenforceable".

These problems will be exacerbated by systems which combine features of free text and relational databases.

4. THE NSW PRIVACY COMMITTEE'S GUIDELINES

We now consider the personal data system principles of the NSW Privacy Committee (1977), to examine whether they allow access to implicit information. We argue that it isn't possible to consider the principle of subject access in isolation, without considering whether other principles may remedy the problems raised. The Privacy Committee "Guidelines" were the earliest comprehensive information privacy principles published in Australia and...
the Committee announced in its 1983 Annual Report that "Legislation is now necessary . . . laying down privacy protection standards" and that "it is expected that any legislation adopted will codify the central principles of collection, storage, access and amendment already embodied in the Committee guidelines" (NSW Privacy Committee, 1983).

Guideline 7, "Subject Access" states:
"Every person should be able to know of the existence and of the content of data which relates to himself."
"Personal data" is defined as "particulars concerning any characteristic of an identifiable natural or legal person, or the affairs of that person" (emphasis added).

Data, or at least its content, is to be accessible without qualification as to the nature of its storage (although 'uncirculated personal notes' and 'personal memory' are excluded). The data is to be accessible not only if the person to whom it relates is directly identified, but also if he is indirectly identifiable. The "Subject Access" Guideline is, therefore, arguably broad enough to require disclosure of implicit information about a person, or at least inferable information, but this is hardly clear.

However, Guideline 6, "Public Access", states:
"The interested public should be able to know of the existence, purposes, uses and methods of operation of personal data systems" (emphasis added).

The principle seems to justify an information subject obtaining details of stored rules, and perhaps regularly used but still "external" rules, and thereby, of implicit information.

Another Committee policy proposes a more extensive right where a person is actually to be affected by adverse use of data about him: "Before an individual is adversely affected by data he should have the opportunity for personal discussion to verify accuracy and comment on the information on which the decision is being made" (NSW Privacy Committee, 1980, p. 18). This policy seems to make no distinction between explicit and implicit information. Disclosure of reasons for adverse decisions after the decision also makes no such distinction.

5. THE OECD'S BASIC PRINCIPLES

During the period 1978-1980, the Club of the 'advanced western nations', the Organisation for Economic Co-operation and Development (OECD) constituted an Expert Group to prepare a Recommendation concerning Privacy Guidelines. Although other international bodies have undertaken similar exercises, notably the Council of Europe, it is the OECD which is the most relevant reference point for Australia, particularly as the ALRC Chairman was also Chairman of the Expert Group.

The Recommendations of the Council of the OECD (1980) included the following principles:

- The "Individual Participation Principle": "An individual should have the right:
  (a) to obtain from a data controller, or otherwise, confirmation of whether or not the data controller has data relating to him;
  (b) to have communicated to him, data relating to him . . .

- "Personal data means any information relating to an identifiable or identifiable individual (data subject)"

- The "Openness Principle": "There should be a general policy of openness about developments, practices and policies with respect to personal data" (emphasis added).

The NSW Privacy Committee Guidelines and OECD Basic Principles therefore correspond fairly closely in this area of access by the information subject. The OECD do not deal directly with the question of disclosure of reasons for decisions adverse to the data subject's wishes.

Although Thom and Thorne's conception of new problems, and our own further development of their thesis, postdate the NSW Privacy Committee and OECD proposals, their frameworks for information privacy protection appear capable of catering for the newly foreseen difficulties, if sympathetically implemented, but it could be said that this is largely due to their degree of generality rather than anticipation of the problems.

6. THE ALRC'S PROPOSALS FOR SUBJECT ACCESS

In April 1976 the Commonwealth Government referred the question of privacy issues arising under Commonwealth and Territorial laws to its Law Reform Commission (ALRC). The Commission's Report (ALRC, 1983), released in December 1983, deals with many aspects of privacy which are not necessarily related to information systems, including intrusive conduct and surveillance.

The ALRC (1983, para 1236), proposes ten Information Privacy Principles and recommends "a right, enforceable under Commonwealth law, for an individual to have access to records of personal information held about him by record keepers". The Report includes a Draft Privacy Bill which embodies this recommendation, and includes the ten Principles as a Schedule.

We propose to discuss these Principles (see ALRC, 1983, paras 1193-1385); in isolation from their incorporation in the draft Bill. It is important to do so because the Commission's Report will and should become the major focus of discussion about privacy issues in this country, irrespective of whether or not the draft Bill is ever enacted, or even placed before Parliament. We will discuss the adequacy of the draft Bill in dealing with implicit information in a future issue of this Journal.

The Information Privacy Principles recommended by the ALRC draw primarily on the OECD guidelines, but their similarity to other formulations of information privacy principles is acknowledged (ALRC, 1983, para 1195, and para 638, footnote 171).

Principle 5, Access to Records of Personal Information states that:
"Where a person has in his possession or under his control records of personal information, the record-subject should be entitled to have access to those records" (paras 1230-1277, our emphasis).

"Personal information [is] information about a natural person from which, or by use of which, the person can be identified" (para 56, summarising paras 1196-1198, our emphasis).

"A Record [is] any written document . . . [and] photographs, drawings, films, tapes and other devices for conveying or transmitting information . . . [except] . . . published documents, library material, mail and non-business records" (para 64, summarising para 1237).

We were unable to find any discussion of the novel term 'record-subject'.

There is a potentially highly significant difference between the Commission's Principles and those of the OECD because of the substitution of the term "record" for "data". "Data" (like "information") is an intangible concept, enabling the OECD Guidelines to indicate a desirable condition without concern for the manner in which the data is physically stored or communicated. A record, on the other hand, no matter how widely defined, is a tangible entity which has moreover a large number of pre-existing and specific meanings, both in the law and in computing.
practice. This narrowing of the classification of accessible 'things' may be the cause of the difficulties discussed below.

The ALRC acknowledges that "Freedom of Information legislation, untrue to its title, is generally framed in terms of access to documents (however defined) rather than access to information" (ALRC, 1983, para 1407, emphasis in original) but does not discuss any implications that a parallel shift from the intangible to the tangible may have for its information privacy principles and their operation.

The ALRC's motivation in using the term 'record' is clear: "every effort should be made to ensure the compatibility between the entitlements ... under both the Freedom of Information and Privacy Acts" (ALRC, 1983, para 1408). That this consideration should be given weight in the preparation of the Draft Privacy Bill is undoubted, but we doubt its merit in the formulation of the general principles.

A further significant discrepancy between the OECD and ALRC Principles is the absence in the latter of any Openness Principle. Consequently, there can be no argument for access to stored rules based on such a principle. Principle 2(c) does require disclosure by the record-keeper, at the time of information collection, of "his usual practices with respect to disclosure", but this seems to have little relevance here. We are unable to find any explicit statement in the Report as to the reason for this important omission. It could be that the Commission has assumed that the Openness Principle is one of freedom of information, not privacy, or that the Freedom of Information Act 1982 has already implemented that principle. Our view is that if openness concerning such aspects of system operations as the existence of stored rules is necessary for adequate privacy protection, then such a principle should be part of a comprehensive set of Information Privacy Principles as well as part of Freedom of Information principles. If not, in those areas where Freedom of Information has no application, the Information Privacy Principles will be incomplete and misleading.

The Freedom of Information Act's objects are:

(a) making available to the public information about the operations of departments and public authorities . . .; and

(b) creating a general right of access to information in documentary form . . . (Freedom of Information Act 1982, Section 3, emphasis added).

The reference to 'documentary form', is somewhat limiting and the focus of 'operations' is narrower than the OECD's 'developments, practices and policies'. The main deficiency is, however, that the scope of Freedom of Information is restricted as yet to the public sector. The Commission asserts that its 'proposals adopt and, so far as relevant, apply to private sector record-keepers . . . the basic entitlements and exemptions under the Freedom of Information Act 1982' (para 1409), but we can not find any such proposals beyond the individual access provisions. This means that the ALRC makes no allowance for individuals to discover 'stored rules' or 'implicit information' by requiring that private sector organisations disclose their practices and policies.

The Commission makes no reference to the question of the communication to data subjects of the reasons for adverse decisions. They did however give consideration to the need to introduce a general requirement that where such decisions were made the person should be notified of the decision and of their rights including inspection of the "relevant record". While regarding this as "thoroughly desirable as a good administrative practice", they deferred the matter for future re-consideration (ALRC, 1983, para 1397). This would in any case fall short of the Privacy Committee's proposal that the person should be able to "comment on the information on which the decision is being made" and the emphasis on access to records may again exclude access to stored rules and implicit information.

One intriguing future possibility raised by the ALRC (1983, para 1406) is that of "direct access by a record-subject to the terminal to interrogate the information base and thereby to secure access to the required information". Such access would clearly be limited without access to at least stored rules.

7. A NEW STAGE IN THE AUSTRALIAN INFORMATION PRIVACY DEBATE

7.1 Confusion Over Principles

A major problem in the information privacy debate which has smouldered in Australia for over a decade is that it has suffered from a number of competing sets of principles being advocated. As Bushell (1983) points out, confusion has arisen in South Australia, where as late as June 1983 the State Government's Data Processing Board issued a set of interim principles which bore little relationship to those under discussion elsewhere. This is even more confusing than Bushell states because in doing so the South Australian Government was ignoring the recent recommendations of its own Law Reform Committee (SALRC, 1980) that the recommendations of the UK Lindop Committee (Lindop, 1978) be substantially adopted.

The problem is not that different principles, or different methods of implementation, are being advocated, but rather that the debate has been carried on with too little common structural framework. When developed in full, general principles of information privacy comprise many sub-principles or exceptions and are inter-related at many points. They are designed to deal with an enormous variety of information systems, and many aspects of a full set of principles will only be relevant to some types of systems. Different approaches to such principles can only sensibly be compared point by point to determine where overlaps, differences and omissions occur. Such comparison, and resulting informed debate, is greatly facilitated if competing principles are constructed on a common framework.

Differing terminologies are just as confusing as differing structures. When differing principles use "data", "information" or "record", it is often difficult to know whether real distinctions are being made, and comparisons are difficult.

7.2 1984: A Brave New World?

The ALRC Privacy Report (1983) brings a new stage in this debate. The Information Privacy Principles recommended by the ALRC have broad similarities to previous proposals. We have criticised some aspects of the ALRC's Principles, and a fuller consideration of all the ALRC's Principles and the draft Bill is obviously necessary.

The ALRC proposals are supported by more extensive argument and documentation than previous proposals. In our view, the principal item on the agenda of the Austra-
alian information privacy debate should be for all the parties to that debate to determine whether the ALRC Information Privacy Principles provide an adequate framework and terminology. If there are reservations about the precise content of different principles, or the best method of implementation or enforcement, that does not matter so much if the participants are working within a common framework, and their differences are thus identifiable and debatable. If, of course, the ALRC framework or terminology is regarded as fundamentally inadequate, then it should not be accepted. Perhaps ACS could take the lead and consider whether the ALRC Principles (or some modification of them) should be adopted in substitution for its own privacy principles (see ACS, 1982).

REFERENCES


NSW Privacy Committee (1977): Guidelines for the Operation of Personal Data Systems, Privacy Committee, Attorney-General's Department, Box 6, GPO Sydney, 27 pp.


SalRC (1980): Fifteenth Report Of The Law Reform Committee of South Australia Regarding Data Protection, the Committee, c/- Supreme Court, Victoria Square, Adelaide.


BIOGRAPHICAL NOTES

Graham Greenleaf, currently tutor in the School of Law, Macquarie University, received Arts and Law degrees from the University of Sydney in 1975. He subsequently undertook legal research and was engaged in private practice. He is Foundation President of the New South Wales Society For Computers And The Law. His research interests include information law, privacy and property law.

Roger Clarke has recently joined the Australian National University as Reader in Information Systems following 17 years in private industry in Sydney, London and Zurich. He holds an M.Com. degree from the University of NSW. His particular interests are in software development technology and social implications of computing. He was a consultant to the Australian Law Reform Commission in relation to the Commission's Privacy Reference.

The two authors were Research Officers for the NSW Privacy Committee in 1975-1978 and 1976-1977 respectively, and were closely involved with the drafting of the Committee's information privacy guidelines.
Short Communication

PREPOL: A Method to Convert a Mixed Data Set to All Numeric
By D. J. Abel† and W. T. Williams*

There are available in the literature classificatory techniques which are attractive for the analysis of problems with large numbers of individuals because their solution times are linearly proportional to the number of individuals. They are, however, restricted to numeric attributes only. This paper presents a procedure to transform the non-numeric attributes in a mixed data set to all numeric, which can then be processed by such techniques. Preliminary experience suggests that this composite algorithm provides solutions whose ease of interpretation is comparable to those generated by a commonly-used agglomerative technique.

Key words and phrases: classification, mixed data, conversion.

CR categories: G.m, 1.5, J.3.

1. INTRODUCTION

Most classificatory pattern analysis techniques have solution times which are proportional to the square of the number of individuals present in the data set, typically because an inter-individual distance matrix must be computed. This tends to make solution of large problems (with more than a few hundred individuals) impractically expensive. However, there do exist requirements to treat large problems. For example, pattern analysis is potentially a valuable approach to describing plant genetic resource collections which can consist of several thousand accessions. A not atypical collection is that of Phaseolus vulgaris at the Centro Internacional de Agricultura Tropical which has 28117 accessions of cultivated forms alone (Amaya, 1982).

Programs with better characteristics are AXOR (Lambert, Meacock, Barrs and Smart, 1973) and POLYDIV (Williams and Lance, 1975) which have solution times that are linear in the number of individuals. The strategies adopted are similar. However, both are limited in the attribute types that may be considered: AXOR permits qualitative and numeric attributes only, while POLYDIV is restricted to numerics only.

It is often necessary in practice to deal with mixed data sets where, in addition to those attribute types, multistate attributes are present. A multistate attribute is one defined by a number of states (e.g. red, blue, green). If the order in which the states are numbered is significant, then the attribute is an ordered multistate; otherwise it is a disordered multistate. For example, 'yield' for pasture plants might be coded as 'high/good/average/poor' and would be appropriately taken as an ordered multistate. On the other hand, in a coding of 'red/blue/yellow' for 'flower colour' the sequence of coding is arbitrary so that 'flower colour' must be considered a disordered multistate. Disordered multistates can further be considered as exclusive or non-exclusive. An exclusive disordered multistate is one for which the individual may be described by only one state, while a non-exclusive disordered multistate permits more than one state to be used for an individual. For example, on admission to hospital a patient might be suffering from a number of diseases so that 'disease on admission' could be taken as a non-exclusive disordered multistate. As he can only be assigned to a single ward at a time, 'first ward assigned' would be an exclusive disordered multistate.

In this paper we present the basis for a pre-processor for POLYDIV, which transforms a mixed data set of qualitative, numeric, disordered multistate and ordered multistate attributes to a numeric-only data set. Experience to date suggests that this approach provides classifications whose ease of interpretation and consistency with external data is comparable to those from the program MULCLAS (Lance and Williams, 1967).

2. THE TRANSFORMATION TECHNIQUES

Clearly the numeric attributes of the mixed data set can be passed directly to the all numeric data set to be formed, so that transformations to numerics need be devised only for the qualitative, ordered multistate and exclusive and non-exclusive disordered attributes.

It is desirable that a distinction be drawn between asymmetric and symmetric qualitative (binary) attributes. As this differentiation is rarely made in pattern analysis, it is useful to examine the characteristics of the two types. (For a more detailed discussion, see Williams, 1976.) A symmetric binary is a special case of a disordered multistate in that the state codes (0 and 1) are arbitrarily assigned to the two states. No ordering of states is implied and it is not meaningful to consider the two states as representing the extremes of a range of values. Examples of attributes appropriately represented as symmetric binaries are 'leaf-pinnate/leaf-palmate' and 'flower-red/flower-blue'. An asymmetric binary, on the other hand, is a special case of an ordered multistate in that the state codes (0 and 1) are arbitrarily assigned to the two states. No ordering of states is implied and it is not meaningful to consider the two states as representing the extremes of a range of values. Examples of attributes appropriately represented as asymmetric binaries are 'abundant/common/scarce/absent' and it is meaningful to aggregate the attribute over a set of sites to derive an estimate of the abundance of the species over those sites as a group. We also note that some pattern analysis techniques, such as the...
Conversion of mixed data to all numeric

Bray-Curtis measure (Bray and Curtis, 1957), place significance on the codes used for asymmetric binaries.

An asymmetric binary then can be operated upon considering it as a discretized numeric: it is appropriate to standardise it as if it were numeric. Consider an asymmetric binary attribute for which $n$ individuals have values recorded. Let $a$ individuals have a value of 1. Then the mean $\bar{x}$, the deviance $D$ and the standard deviation $SD$ are given by:

$$\bar{x} = \frac{a}{n}$$

$$D = \frac{a(1-a/n)^2 + (n-a)(-a/n)^2}{n} = \frac{a(n-a)}{n}$$

$$SD = \left(\frac{D}{n}\right)^{1/2} = \left[\frac{a(n-a)}{n}\right]^{1/2}/n$$

The standardised value $x'$ for the attribute for an individual can be determined from the raw value $x$ in the usual manner:

$$x' = \frac{x - \bar{x}}{SD}$$

This reduces to:

$$x' = \left[\frac{(n-a)/a}{n}\right]^{1/2}, x = 1$$

$$x' = -\left[\frac{a/(n-a)}{n}\right]^{1/2}, x = 0$$

Symmetric binaries require a different treatment. As the state values cannot be taken as implying an ordering of states reflecting a discretized property, it is inappropriate to apply the form of standardisation leading to (4). Rather the two states of a symmetric binary can be considered as two linked asymmetric binaries which can both be standardised. The standardised value for a symmetric binary can then be taken as:

$$x' = \left[\frac{(n-a)/a}{n}\right]^{1/2}, x = 1$$

$$x' = \left[\frac{a(n-a)}{n}\right]^{1/2}, x = 0$$

Ordered and exclusive disordered multistates can be similarly treated as sets of linked asymmetric binaries. Let the $j$th attribute be an ordered or exclusive multistate of $k$ states with $n_j$ individuals having values recorded for the attribute and let $a_{jm}$ be the number of individuals in the $m$th state of the $j$th attribute. Then, by reference to (5), $x_j$ the transformed value of the $j$th attribute for an individual in its $m$th state can be taken as:

$$x'_j = \left[\frac{(n_j-a_{jm})/a_{jm}}{n_j}\right]^{1/2}$$

Non-exclusive disordered multistates can be transformed by taking the standardised value to be the mean of values computed using (6) for those states present for an individual. Applying the same notation as before, we define $y_{jm}$ as taking the value of 1 if an individual has a state $m$ for the attribute $j$ and 0 otherwise. Then the contribution to the standardised value for the attribute for the individual from the $m$th state is given by:

$$c_{jm} = \left[\frac{(n_j-a_{jm})/a_{jm}}{n_j}\right]^{1/2}, y_{jm} = 1$$

where $n_j$ and $a_{jm}$ are defined as above, and the transformed value for the attribute by:

$$x'_j = \left[\Sigma_{m=1}^{k} c_{jm}\right] / \left[\Sigma_{m=1}^{k} y_{jm}\right]$$

Some observations on the approach can be made. This treatment of ordered multistates departs from their consideration in many pattern analysis procedures directly as numerics. Treatment as numerics assumes that there exists a linear relationship between the codes used and the physical property. In practice, however, many multistate codings correspond to the set of states able to be reproducibly recognised by an inherently non-linear human observer. For example, 'leafiness' might be recorded on a scale of 1 through 10. While that scale might correspond reasonably well with the actual leaf cover in the middle part of the range, it could well be compressed towards the top (where leaf cover is dense) and expanded towards the bottom (where there are few leaves). While the approach offered here avoids the assumption of linearity, it does so at the expense of loss of any information inherent in the ordering of state values. Another loss of information can occur when for a symmetric binary, for example, the two states have equal frequencies. As with many other pattern analysis techniques, these deficiencies must be recognised as inherent penalties in a heuristic approach seeking to provide a solution of some value within acceptable solution costs with the only real assessment coming from trials on realistic problems.

We suggest that selection of the most appropriate treatment of particular attributes is a matter for the judgement of the user with the characteristics of his data set in mind. Treatment of an ordered multistate as numeric, for example, can be forced simply by including it with the numerics in the input data set.

A program applying the transformations would be expected to develop the transformed values for each state for each attribute in a first stage rather than to evaluate the various expressions for each individual. Clearly allowance must then be made for two anomalous cases where the denominators of expressions vanish. These are where $a = 0$ or equal to $n$ (i.e. where an attribute is single-valued) and where $n = 0$ (i.e. where all individuals have missing values for the attribute). In these cases the attribute can effectively be excluded from consideration.

Finally, we note that as programs such as POLYDIV perform standardisation as a first step and as the transformation of asymmetric binaries is simply standardisation, the asymmetric binaries can simply be passed without transformation to the all numeric data set.

3. EXPERIENCE

The transformation procedures have been implemented as a FORTRAN program, PREPOL, which generates a file able to be input directly into the TAXON system (Ross, 1982).

The acceptability of classifications generated by POLYDIV operating on PREPOL output was tested in two trials, using the MULCLAS algorithm (Lance and Williams, 1967) as a benchmark. While MULCLAS follows an agglomerative strategy in contrast to the divisive strategy of POLYDIV, it was selected as the technique most likely to be used for problems with large numbers of individuals and as a commonly-used general-purpose classificatory algorithm. The TAXON implementations of the MULCLAS and POLYDIV algorithms were used.

The two data sets were provided by Dr. R.L. Burt who, together with Mr. B.C. Pengelly, provided a comparative assessment of the MULCLAS and POLYDIV classifications. The first set was a collection of 150 Stylosanthes
Conversion of mixed data to all numeric

accessions with 14 binary, 23 numeric, 7 ordered multistate and 11 disordered multistate attributes. The PREPOL/ POLYDIV and MULCLAS classifications were reported as similar but with the MULCLAS solution favoured. The collection is characterised by a large group and several small groups of accessions. PREPOL/POLYDIV was unable to split the large group into smaller groups able to be readily interpreted, but MULCLAS did so on the basis of a pair of attributes which were well-correlated. The second set was a collection of 69 Desmanthus accessions, with 17 numeric, 1 disordered multistate and 4 ordered multistate attributes. Here the two classifications were reported as roughly equivalent with no clear preference.

Clearly the PREPOL/POLYDIV system transforms the mixed data set to an all numeric once only. Logically the transformations could be performed for each group which is split within POLYDIV at the time of the split. This was tested but failed to give passable results as group sizes fell. This appeared due to the excessive contributions of attributes which, in small groups, had a sharply-peaked distribution of individuals across states.

REFERENCES


BIOGRAPHICAL NOTES

D.J. Abel is with the Information Systems Section of the CSIRO Division of Computing Research. His publications include topics in operations research, spatial databases, pattern analysis and biological systems. His PhD was awarded by the James Cook University in 1979 for studies in operations research. He was previously with the James Cook University Computer Centre.

W.T. Williams, OBE, PhD, DSc (Lond), Hon. DSc (Qld), FAA, was born in 1913 and took his first degree at Imperial College, London. He was in the British Army (Major, REME) 1940-1946, and was Professor of Botany, Southampton University 1951-65. He joined CSIRO Division of Computing Research in 1966, transferred to Division of Tropical Pastures in 1968, retired in 1973, but continued for some years as part-time Research Fellow. He is now part-time consultant to the Australian Institute of Marine Science. Originally a plant physiologist, he transferred to numerical classification in 1958. He is also a pianist (LMusA), specialising in two-piano recitals of 20th century music.

CALL FOR PAPERS

SPECIAL ISSUE ON EXPERT SYSTEMS

The Australian Computer Journal will publish a special issue on "Expert Systems" in February 1985. Research papers, tutorial articles and industry case studies on all aspects of the subject will be welcome, and both full papers and short communications will be considered.

Prospective authors should write as soon as possible to:

Dr J.R. Quinlan,
School of Computing Sciences,
New South Wales Institute of Technology,
PO Box 123, Broadway, NSW 2007

to notify him of their intention to submit material for the issue and provide a brief summary of their intended contribution.

In order to allow adequate time for refereeing and editorial review, complete manuscripts will be required by 28 September, 1984 at the latest.

Papers should be prepared in accordance with the guidelines published in the November 1982 issue of the Journal. Authors are requested to pay particular regard to the Journal's preferred style for references.
Book Reviews


This introductory text is designed for anybody working in a computing systems environment. Its stated purpose is to present a performance control system that will ensure that a computing centre delivers an agreed service with reasonable resource usage. The discussion is mostly concerned with MVS based systems running major online applications.

Early chapters of the book present the fundamentals of performance control systems and service objectives, and lead on to service agreements based on system availability and internal response times. Tools to follow up the service agreements are discussed, including techniques to measure and control the resources of the complex MVS operating system. Some IBM software products useful for performance control are briefly described in the text, even though some of these are not generally available for public use.

Later chapters cover system monitoring, anomaly handling and the total control system, with a final section on management type reports for a case study produced using colour presentation techniques.

The style of the book was rather lifeless — not exactly bedtime reading standard — and parts of the material seemed redundant — what would one expect from an introductory text?

The previous concise performance control discussed in the text are important to the survival of every large computer centre, making the book a valuable addition to any computing systems centre library.

P. Sanger, AAEC Lucas Heights Research Laboratories


It seems strange to review a foreign-language publication in an Australian journal. It shouldn't be, and not only because of our newly-discovered 'multi-cultural heritage', but also because a great deal of valuable material originates in languages other than English.

A case in point is the Graef and Greiller work on Computing Centre Management. The authors claim that "there is no comparable work in English" and include an 8-page Table of Contents in our language. The original edition of 1975 resulted from the authors' experiences in industry conferences and seminars over a period of many years. The opportunity was taken in the revised edition to cater for both increased knowledge and the development of decentralised computing operations.

Unfortunately the authors show little evidence of appreciating during the preparation phase how popular such a book could be in the English-speaking world. There are many lists, but too few tables and diagrams, and a great deal of fairly erudite German text. Some terms are fairly comprehensible (Kaltstartprozedur, Programmldokumentation and Magnetband) but I doubt the Durchsichtigkeit (transparency) of such terms as Erfassungsgeraete (data capture peripherals), Unzuverlaessigkeit (unreliability), Arbeitsstandkontrollregister (job status register), Storungsmeldung (fault report) or perhaps best of all Datenfernuebertragungsverfuegbarkeit (data communications up-time). Managers of computer centres, provided they have some fluency in the German language, would find a great deal of worthwhile material in this reference work.

Data Centre Operations is American, less impeccable and more usable. The Appendices comprise a hundred pages of check-lists; there are seventy-five sample forms, fifty sample printed reports and myriad tables and lists. Not being an operations manager myself, I adopted a sampling approach and found that most of the obvious issues like hardware purchase contracts, site layout, organisation structure, training, backup and recovery and charge-out were dealt with comprehensively: it is a reference work that ought to be on the shelf in every large installation.

The bibliography is rather limited and a little aged, comprising mainly 1972-1977 publications. A more significant complaint is that there seems to be no discussion of the management of transition between software versions and hardware upgrades — one of the more difficult and most poorly performed functions in many data centres. The book seems to assume a stable state in computer technology. It will therefore date quickly, and even during its lifetime will provide no framework within which the operations manager can absorb and evaluate the shifts of emphasis (and downright revolutions) going on around him.

Roger Clarke, XAMAX Consultancy Pty Ltd. Sydney


This modest little book is supposed to give the hobbyist reader, already familiar with BASIC, an entry into the same programming world as seen through the eyes of a FORTRAN programmer on a microcomputer. However because BASIC is such a simple language, it is hard to get too enthusiastic about statements and constructs perceived in BASIC being actually implemented in FORTRAN in a similar way. Your reviewer finds it difficult to believe that if the hobbyist BASIC programmer were to be given a FORTRAN manual then he couldn't find equivalents to his familiar statements in it. This having been said, the text does contain some gems which make it at least worthwhile reading. These I'll highlight as they emerge in the following detail of the book's structure.

Our book consists of six chapters and two appendices. The first chapter is eight pages of an "Introduction to FORTRAN" which is quite good at placing FORTRAN in its historical perspective without trying to state its glories. In fact, if anything, FORTRAN is treated quite humbly.

The next two chapters cover as much Microsoft FORTRAN 80 as the author thought necessary for the above BASIC programmer, taking 25 pages to do so. Each of these two chapters concludes with five and six self testing questions respectively.

Chapter 4 opens with our author being almost apologetic for FORTRAN's existence on micros but claiming in this chapter to present some applications which bring out the strength of FORTRAN. Without preempting the content, I think this chapter should be interesting to the reader of our book, because it does present a couple of microcomputer applications capable of being handled quite naturally by FORTRAN. The chapter concludes with some suggested references.

Chapters 5 and 6 are two more gems. The first of these introduces us to the widely available GINO-F library of general purpose graphics subroutines, accessible via FORTRAN. This chapter should be an eye opener to our BASIC programmer who is looking for a hardware independent path to worthwhile drawing feats. Chapter 6 is worried about the FORTRAN programmer's ability to write structured code, and gives a brief introduction to the RATFOR preprocessor, with an obvious reference for further reading.

Appendix 1 collects the details of compiling and linking on CP/M, while Appendix 2 gives answers to the self test questions.

An annoying section at the end of each chapter is the summary. In my opinion, a summary of these small chapters is unnecessary when all can be gleaned by flicking through the pages.

It's amusing to read of FORTRAN treated in such a modest fashion as to be compared to BASIC, Moreover it's refreshing. The latter half of the book although interesting, probably won't entice the reader away from BASIC, but should let him know that there are other worlds. As a vehicle for getting the student familiar with FORTRAN quickly, we have a tutorial slightly overpriced for the amount of time it'll take to work through it. While not at all a book for programmers or computer scientists, it may however stimulate the hobbyist or teenage student to further reading. Borrow this one from your local library.

Peter J. Mason, University of NSW


At my first quick look through the contents of this book, I was inclined to dismiss the publication as a text which teaches neither BASIC nor thermodynamics adequately. However, the pref-
ace claims that it doesn't pretend to be a comprehensive treatise on either, but rather "... aims to help readers to become proficient at BASI programming by using it in an engineering subject and to use computing as an aid to understanding the subject". With this I was relieved and continued my examination in more earnest.

Of the stated aims, it is the first which is the more successful as it comes nearer to the result. In my opinion, I recommend this book completely. A level of proficiency in the most rudimentary form of BASIC will be achieved, but the appetite will most certainly be whetted for more.

The book consists of seven chapters. Chapter 1 is an introduction to the BASIC language. Chapter 2 treats the laws and low level concepts of thermodynamics. Chapter 3 introduces us to the state properties of substances, steam tables and Langrangian interpolation. Chapter 4 covers processes and cycles commonly found. Chapter 5 concerns fluid flow. Chapter 6 titled "Applications, plant, etc." contains more sophisticated thermodynamic models and their problem solution. Chapter 7 is about aspects of heat transfer.

The text works well mainly because of its layout. Each chapter after the second starts with some theory and then treats work example problems based on the previous theory. Listings, results and program notes. As well each chapter concludes with a bibliography and further problems for the student to work through. The worked problems are suitably small, interesting, practical and uncluttered. The reader should have little difficulty in finding the thermodynamic theory contained in each program listing.

This book could have been written 15 years ago, but with the recent popularity of the micro and the increased accessibility to the BASIC language, the demand is now. The author has solved the problem of different dialects and implementations of BASIC by using only the lowest common denominator form. We are confronted with one character or one character plus one digit variable names, files and file manipulation are not mentioned at all, nor is graphics of any sort. This is a strength.

More of a tutorial than a reference book, this text should appear on most college library shelves.

Peter J. Mason, University of NSW


There has been a need for several years now for suitable textbooks that treat digital image processing for remote sensing applications at a level that is not unreasonably mathematical and yet is sufficiently detailed to be meaningful. Hitherto remote sensing image analysis have had to depend either upon superficial treatments of image processing techniques found in general remote sensing texts, or alternatively upon detailed mathematical treatments of image analysis found in the electrical engineering and computer science literature. It is pleasing therefore that those who review, emerge at a time when digital processing and analysis of remote sensing image data is gaining in popularity, largely because of the increasing availability of dedicated and mainframe-based image analysis systems. This is particularly the case in Australia.

The book is arranged in six chapters and includes appendices containing information on data availability and data processing agencies in the United States. The author, in his preface, states that his treatment is a practical one, directed towards those wishing to get started in using digital methods of image analysis. The treatment is described as being characterised by breadth, being an overview of the techniques and algorithms. Apart from passing comment parallelepiped classification is not covered nor are any alternative classifiers. Unsupervised classification by clustering is described, as is feature reduction, however no techniques are given. The very important multispectral linear transformation of principal components analysis is only mentioned but not treated in any detail.

Chapter 5 presents a set of case studies. These are very useful and serve to demonstrate how the techniques presented are applied in practice. The final chapter discusses three so-called research topics. To the reviewer these do not seem typical of contemporary image processing research for remote sensing applications, apart from a treatment of shape detection. The other topics here are image enhancement using linear programming techniques and edge detection.

Notwithstanding the above the book at $US27.50 represents reasonable value for money and, in view of the need for texts of this type, is a useful if somewhat limited treatment.

ALD. Richards, University of New South Wales


Here it is at last! A book even your mother-in-law will be proud to see on your bookshelf. Yes, it’s yet another of those nice North-Holland conference proceedings (this time the 1982 IFIP TC-2 conference on formal description concepts) with the pretty red cover and the title all in lower case so even the kids can read the words.

If you’re wondering why you didn’t know the conference was being held, participation was by invitation only, and there were enough impressive people (35 authors all told, of whom many are leading researchers) without you or me. If you manage to read beyond the contents page, you’ll find 18 distinguished papers on topics covering every imaginable facet of formal description concepts. The six sessions covered the areas of:

- programming language semantics, semantics of abstract data types, correctness of programming language translations and compiler generation from semantic descriptions,
- abstract algebraic semantic characterisation of functional languages and a semantic theory for networks of concurrent programs,
- fairness assumptions in a subset of communicating sequential processes (CSP), algebraic and operational semantics views of CSP, transition of an Ada subset into a calculus of communicating systems (CCS),
- a denotational description of programming languages for the manipulation of relational databases and models of conceptual and external database schemas,
- a formal proof theory for program equivalence, fixpoint and initial algebra semantics of recursive program schemes and a description of recursive decomposition ordering,
- graph models for concurrent programming and a model for observable concurrency.

Of course, there may only be one or two papers of specific interest to you (they aren’t aimed at the novice computer scientist), but when you find them they’ll be worth the effort (but perhaps not the money). There is, as usual, the added bonus of almost every paper appearing in a different font — as well as inducing the odd coronary when you consider the cost had it been properly typeset, the change gives a welcome break from the monotony of conference papers.

When you become bored with the papers you can always read the ‘Question and Answer’ segments which follow each paper, and the “Discussion” which was held at the conclusion of each session. These are enlightening, these are enlightening and occasionally humorous and often provide an insight into the character of an author you’ve only met in a journal. An excellent inclusion.

I have ceased attempting to justify the cost of these proceedings. I can only imagine it is due to the demand — perhaps present only because reviewers recommend, as I now do, that you urge your
local library to obtain a copy, as the book is a worthwhile reference. At the price, I certainly would not buy the book for myself. (I don’t need to own it and besides, I don’t have a mother-in-law.)

T. Vickers, University of New South Wales


This book was published in conjunction with the BBC TV series ‘The Computer Programme’ as part of the BBC Computer Literacy Project. I have not seen the TV series and cannot comment on how well the two match together. However, the book is quite self-contained. Presumably the book and the TV series reinforce each other’s messages and generate interest in each other, but are largely independent products otherwise.

Besides not being a TV watcher, I should confess to a second deficiency as reviewer for this book: I am personally very skeptical about the value of the currently fashionable computer literacy movement. Certainly the movement has informed about the world around us, and computers are an important part of that world and are therefore worth knowing about. However, I do feel dubious about attempts to make national issues out of such needs. It is misleading to equate technological advances and social progress, since countries like South Korea and Taiwan made major strides with computer technology while their level of computer literacy has been, if anything, lower than that of Britain or Australia. It is equally misleading to equate “educating the public” with “serving the public”, without first looking into the motives of the educators. For computer literacy, it is only too easy to accept the cynical view that the aim is to persuade the technological “have-nots” into passive acceptance of new technologies which will put them selves further at a disadvantage. While I do not doubt the good intentions of the advocates of computer literacy, I would advise a greater level of detachment: what is obviously good to them may not appear so to others.

The book itself contains six chapters: Setting the scene, Problems and computers, The hardware and the software, Understanding programming, You and your microcomputer, and The limits to growth. It appears that these have been individually contributed but were subjected to common editing. Stylistically the chapters are quite uniform, and rather reminiscent of De Bono’s lateral thinking bestsellers. Unfortunately this saneness tends to carry over into the contents of the various chapters too, with each author striving to illustrate his discussion with layman-comprehensible examples of the applications of computers. Individually it is only fair that Setting the scene should start with a few examples of computer applications before going on to the history of computers and general background, that Problems and computers should illustrate computer problem solving with examples, that You and your microcomputer should follow descriptions of microcomputer components with applications, and that The limits to growth should discuss what computers can do in the future. Reading four chapters all in the vein “Computers are useful things in life” does make the theme wear rather thin however. The fault does not lie with the authors; indeed, I would commend the final editors for having ensured that all the examples in the various chapters are reasonably distinct. It is simply not easy to speak at length about computer problem solving with examples, that You and your microcomputer was discussed and the input/output section was more extensive. Current developments in VLSI design, thirty-two bit microprocessors, and reduced instruction set microprocessors are not generally available.

The book commences with a coverage of the basics of combinational and sequential logic. Circuits are drawn using British Standards Institute symbols, which may be unfamiliar to some readers. Algorithms for the four basic arithmetic functions and hardware implementations of these are discussed in some detail. These are followed by chapters on: computer memories, the NOVA minicomputer is discussed throughout the book, giving it a dated feeling (the reviewer studied this processor in an undergraduate course in 1972). However, as the author points out, the basic principles of microprocessor are the same. Also, details of the internal workings of microprocessors are not generally available.

The book is good value for those wishing to understand the internal workings of a simple processor. The text is clear and includes a wealth of illustrations. However, it would have greater appeal if a modern microprocessor was discussed and the input/output section was more extensive. Current developments in VLSI design, thirty-two bit microprocessors, and reduced instruction set architectures will quickly date some sections of the book, but many of the concepts and circuits will be with us for a long time to come.

P.J. McKerrow, University of Wollongong


These are the proceedings of the IFIP WG 9.1 Working Conference on Systems Design for, with, and by the Users, held in Italy on 20-24 September, 1982. The book comprises 36 papers and seven working group reports assembled into the following sections:
- Participative System Design — from the Past into the Future
- Contributions to a Theoretical Framework of Participative System Design
- Experiences with General Users’ Participation
- Experiences with Participative System Design in Different Sectors and Functions
- Models and Methods for Participative System Design
- Trade Union Experience with New Technology
- Evaluation Studies on Trade Union Experiences
- Towards Trade Union Action and Alternatives
- Trade Union Experiences in Education for Participative System Design
- Different Actors’ Roles in Participative System Design
- Reports from the Working Groups
- Addendum

Of these headings imply, the main focus is on the participation of workers and their organisations in the process of computer systems design. These are the most numerous group of users, who often are not direct users, but are used in systems design.

The papers vary widely in nature and quality. Some are theoretical, others give case histories of specific system design projects.
Book Reviews

Most are well written, but some use conventional grammar or spelling, which can be distracting. However, even the latter are generally worth reading, and certainly the better papers are a veritable gold mine of information on this topic, on which there is such a paucity of information and especially of practical experience in Australia. The papers represent views from several countries, including Germany, Sweden, the UK, France, Norway, Denmark, Austria, the USA and, of course, Italy.

The numerous case histories and practical discussions relate to such areas as:
- public administration
- engineering (esp. CAD, and NC programming by machinists)
- nursing
- videotex (esp. the involvement of the general public)
- labour related legislation (esp. in Norway, Sweden, Germany)
- printing
- and many others.

This is an invaluable book for any DP library, consultant, manager, union representative, researcher and project manager. It may be relatively expensive, but it represents excellent value for money. If I were an analyst/programmer I probably wouldn't buy it myself at $US55. However, I would pester the DP manager into buying it for the office.

Those interested in similar work based on (relatively scant) Australian experience, will find two particularly relevant papers in the proceedings of 10 ACC (pp. 346-360, 487-499).

Andy Bennett
Ministry of Technology, South Australia


This is a curious little book, as it is not at all clear what class of readers the authors are addressing. The preface suggests that it is directly applicable to computer graphics and additionally to other programming tasks where geometric operations are required.

But by the middle of the nine chapters discuss elementary geometrical concepts, initially in two dimensions and later in three. These range from the distance between two points to the centre of a tetrahedron. A number of alternative formulae are given for each concept, and the computational merits of each discussed. A program in FORTRAN 77 is then given for the more meritorious expression. The final chapter discusses drawing pictures, and gives an elementary account of line and pixel devices, the drawing of lines, arcs and circles on these devices and the role of clipping.

The authors have discussed these topics from the most elementary possible level, and have deliberately eschewed as much linear algebra, vector and matrix theory as possible. As a consequence they have given a catalogue of formulae with FORTRAN implementations, but no derivation. This approach is akin to teaching someone to ride a surf-board before they can swim. They might manage to learn something of board riding, but if they fall off they'll be in trouble.

By the same token, a study of the geometry of lines, surfaces and polyhedra, and their projections, cannot sensibly be undertaken without a working knowledge of linear algebra, vectors and matrices.

The book appears to have been typeset on a Sanders printer or equivalent, which has resulted in a particularly easy to read text. The figures are also rather primitively presented, and the authors managed to misspell his own name!)

High Level CHDLs

1. SÅ: A Language for Describing Computer Architectures, by Dasgupta. A very seminal paper, which addresses the difficult issue of architecture description. That is, the interconnection of information processing systems. This is to be distinguished from the organisation description, the interconnection of digital processing elements. It is the organisational description that CHDLs usually define. Dasgupta uses the terms exo-architecture and endo-architecture to refer to a similar distinction. He uses a heavily declarative version of pseudo-Pascal to describe the exo-architecture, which, it is argued, can then be semi-automatically translated to an instantiated CHDL. More power to his elbow, but he hasn't quite succeeded in demonstrating that this paper.

2. The Application of CONLAN Assertions to the Correct Description, by Eveking. Another interesting paper, drawing this time from the work of Dijkstra (1976) for a semantic definition models for the description of Register Transfer Systems. The author does acknowledge that "We do not know if our techniques can be scaled up to 'real examples', so practising digital designers can, for the nonce, ignore this paper.

3. DDV Verifier, by Uehara et al. A reduction ad absurdum approach to the CHDL DDV description, via cause and effect chains, in order to prove correctness. The authors of this paper were less modest than the previous two.
cEDURE body grouping is with proc ... retn [yecch toot!]; and module grouping is with mech ... endmech (passable). The arbitrary introduction of bastardised keywords like glovar and privar [why not global var and private var?] did nothing for me, either.

2. MODLAN - A Language for Multilevel Description and Modeling of Digital Systems, by Pawlak and Jezewski. A much more pragmatic approach to the problem addressed by Dasgupta. I didn't feel that this paper deserved to be classified under the 'High Level' heading.

3. HILO Mark 2 Hardware Description Language, by Flake, et al. A straightforward no-nonsense description of yet another CHDL. The adjectival 'High Level' is more warranted here. It refers not to the level of description, but the structure of the language. It is a pity that the authors did not also try to apply it to the level of description of the language.

4. A Classification of Program and Data Flow Control in Computers, by Cordonnier and Tourse. An interesting taxonomy of computer architecture based upon a model of both control and data flow. The classic von Neuman and data flow architectures become special cases of a range of architectures in which the function and data components of an instruction are treated consistently. Recommended reading for all computer architects.

Computer Aided Design Systems

1. A CHDL directed CAD system and its Design Data Base, by Blair et al. Buy Australian! This paper describes the organisation of a data base for a digital design system developed at UNSW, and is competently presented. I could find no evidence of the mythical Cultural Centre here. This paper appears to be wrongly classified. It is a much better written paper, and is structurally more sound. Its main claim is the ability to mix levels of description, which is certainly possible, but it does so in a way that does not readily reflect the hierarchical relationship between these levels of description. A readable description of the PDP-8 architecture demonstrates the conciseness of the notation.

2. On a Top-Down Design Methodology for Packet Systems, by Leung. To fully appreciate this paper, the reader needs first to be conversant with ADL, the CHDL upon which the methodology is based. The language contains data flow constructs in which the hardware design is specified. It is claimed that a design in ADL can be automatically translated into a self-timed logic circuit, although the evidence for this is rather sketchy.

3. Parameterised Modules and Interconnections in Unified Hardware Descriptions, by Klein and Sastry. Describes a formalism (what language elements there are are based on Pascal) that gives a neat view of the structural properties, and hierarchical organisation of a digital circuit. A novel feature is the use of a data-driven model. It is claimed (no evidence given) that the notation is also suitable for simulation.

4. Object Oriented Description Environment for Computer Hardware, by Takeuchi. Apart from the occasionally poor English constructions, this paper is well worth reading. Drawing upon the object based features of SIMULA and SMALL-TAL, and using the message passing concepts of the latter, it provides a natural and readable (for LISP programmers, at least) notation for hardware description. Particularly nice is the way in which the design support system merges with the language concept itself, a feature that APL and BASIC programmers will appreciate.

Modularity and Communication with CHDLs

1. A New Multi-Level Hardware Design Language (LALSD II) and Translator, by Su, Huang and Fu. This paper is in marked contrast to the 'High Level CHDLs' discussed above. Indeed, this paper appears to be wrongly classified. It is a much better written paper, and is structurally more sound. Its main claim is the ability to mix levels of description, which is certainly possible, but it does so in a way that does not readily reflect the hierarchical relationship between these levels of description. A readable description of the PDP-8 architecture demonstrates the conciseness of the notation.

2. On a Top-Down Design Methodology for Packet Systems, by Leung. To fully appreciate this paper, the reader needs first to be conversant with ADL, the CHDL upon which the methodology is based. The language contains data flow constructs in which the hardware design is specified. It is claimed that a design in ADL can be automatically translated into a self-timed logic circuit, although the evidence for this is rather sketchy.

3. Parameterised Modules and Interconnections in Unified Hardware Descriptions, by Klein and Sastry. Describes a formalism (what language elements there are are based on Pascal) that gives a neat view of the structural properties, and hierarchical organisation of a digital circuit. A novel feature is the use of a data-driven model. It is claimed (no evidence given) that the notation is also suitable for simulation.

4. Object Oriented Description Environment for Computer Hardware, by Takeuchi. Apart from the occasionally poor English constructions, this paper is well worth reading. Drawing upon the object based features of SIMULA and SMALL-TAL, and using the message passing concepts of the latter, it provides a natural and readable (for LISP programmers, at least) notation for hardware description. Particularly nice is the way in which the design support system merges with the language concept itself, a feature that APL and BASIC programmers will appreciate.

Application and Comparison of CHDLs

1. The CAP/DESC System: Simulator and Case Study, by Dachauer et al. This paper is poorly organised and written. It purports to show (i) how the system can be used to support the design of a 16-bit computer, and (ii) the principles of a simulator used to implement the simulation aspects of the language. It seems that the two aspects would have been better treated as separate papers: as one paper, neither topic is handled well. Rather than 'show how the system is used to design a computer', the design is presented as a fait accompli. The principles' part suffers from poor English, poor spelling, and poor expression. Not recommended.

2. Simulator of a Horizontal Bit Sliced Processor: The MICE Experience, by Van Dam, et al. An interesting paper, describing the experience of actually using a CHDL (ISPS) in earnest. Well written and readable, it is a pity that not more of these sorts of papers appear.

3. Development of Comparison Features for Computer Hardware Description Languages, by Singh and Tracey. A comparison between 8 CHDLs, at a conceptual level (i.e., there are no remarks about implementations and the like). The title is a little misleading, in that the paper is really a comparison as such, and not a development of comparison features, although these features are clearly identified. Because of this, the paper suffers through not addressing a wider range of CHDLs.

Microprogramming and Control

1. Firmware Description Languages for a Microprogram Meta-Assembler, by Mezzalama and Prinetto. Conventional assembly languages have been used for assembling code for non-native machines, but this system is (to the reviewer's knowledge) the first meta-assembly language, designed to provide an assembly environment for a range of target machines. Competently presented.

2. Statistical Studies of Horizontal Microprograms, by Marwedel. Here is evidence for those academics (like myself) who are trying to convince university administrations that computer science is an experimental discipline. The paper presents experimental evidence for the effect of microprogrammed processor organisation upon processor performance. Factors identified (with percentage improvement, where relevant, in parentheses) are:

   a. Micro-Instruction Pipelining (12.5%)
   b. Variable Duration of Microsteps (69%)
   c. Speedup of Conditional Statements (2.8%)
   d. Nested Conditions
   e. Multiple Use of Hardware
   f. Bandwidth of Memory
   g. Critical Path Analysis

   Competently written.

3. Step-Assignment Method using Decomposition for Realising AHP-like Control Sequence Descriptions, by Kereszes and Pacher. A rather obscure paper, that shows 'the possibility of using classical decomposition methods of switching theory' on a parallelism-free control sequence. If the paper were better written, this may be quite obvious: as it stands, I have trouble in deciding quite what it is that the paper is discussing.

In addition, there were three invited papers included in the volume (six others are identified in the Introduction, but didn't quite make it into the proceedings). The three given here have obviously been transcribed from presentation notes and diagrams, and don't read well in paper form.

References


John Hurst,
Australian National University
A CONFERENCE ON
CHALLENGE AND CHANGE
Implications of Technological Development for Policy and Planning in Tertiary Education
will be held at DARLING DOWNS INSTITUTE OF ADVANCED EDUCATION – TOOWOOMBA from 5 – 8 July 1984

The conference is being convened jointly by DDIAE and ACS. Invited papers will be delivered in the following areas:

- Technological Change in Australia: Its Present State and Future Prospect
- Employment and Social Consequences of Changing Technology
- Educational Needs of the Information Society
- Technological and Social Change: the Issues for Government in a Democratic Society
- The Significance of Communications Technology for:
  (a) Distance Learning;
  (b) The Educationally Disadvantaged at Tertiary Level
- The Implications of Technological Change for:
  (a) Higher Education (University and CAE);
  (b) Technical and Further Education
- Significance of Technological Change for the Future Structure and Organisation of Australian Tertiary Education

The Keynote Address will be delivered by Sir Bruce Williams, KBE, Director, The Technical Change Centre, United Kingdom.

Substantial provision will be made for plenary and small group discussions, and arrangements will be available for comments of particular interest groups to be forwarded to government and related bodies.

Further information is available from:
Mr P.R. Chippendale, Conference Director, or Ms C. Cottman, Darling Downs Institute of Advanced Education, PO Darling Heights, Toowoomba, Qld 4350. Telephone (076) 30-1300, Telex AA 40010.

COMPUTER SCIENCE TEXTS
A MAJOR NEW SERIES FROM BLACKWELL SCIENTIFIC PUBLICATIONS
This important new series of textbooks has been created after a careful appraisal of the current needs of students, teachers and others involved in the computer sciences.
Each book is based on a specific topic of direct relevance to computing and its associated subjects, and is designed for use as a valuable aid in teaching.

Titles now available are:
MICROCOMPUTERS AND THEIR COMMERCIAL APPLICATIONS
D. E. Avison, University of Aston in Birmingham
$13.95
LISP PROGRAMMING
I. Danicic, University College of Wales, Aberystwyth
$16.50
A FIRST COURSE IN FORMAL LANGUAGE THEORY
V. J. Rayward-Smith, University of East Anglia
$21.50
FROM LOGIC TO COMPUTERS
P. J. Thewlis, Leicester Polytechnic
$19.50

Distributed in Australia by:
BLACKWELL SCIENTIFIC BOOK DISTRIBUTORS PTY. LTD.,
31 Advantage Road, Highett, Vic. 3190
Telephone (03) 553 3517

This Publication is available in Microform.

University Microfilms International

Please send additional information to...
Name: ___________________________
Institution: ______________________
Street: ___________________________
City: ____________________________
State: ___________________________
Zip: ____________________________

300 North Zeeb Road
Dept. P.B.
Ann Arbor, Mi. 48106


SUBSCRIPTIONS: The annual subscription is $20.00. All subscriptions to the Journal are payable in advance and should be sent (in Australian currency) to the Australian Computer Society Inc., Box 4944, GPO, Sydney, 2001. A subscription form may be found below.

PRICE TO NON-MEMBERS: There are now four issues per annum. The price of individual copies of back issues is now available. Some are already out of print. Issues for the current year are available at $5.00 per copy. All of these may be obtained from the National Secretariat, Box 4944, GPO, Sydney NSW 2001. No trade discounts are given, and agents should recover their own handling charges.

MEMBERS: The current issue of the Journal is supplied to personal members and to Corresponding Institutions. A member joining partway through a calendar year is entitled to receive one copy of each issue of the Journal published earlier in that calendar year. Back numbers are supplied to members while supplies last, for a charge of $2.00 per copy. To ensure receipt of all issues, members should advise the Branch Honorary Secretary concerned, or the National Secretariat, promptly, of any change of address.

MEMBERSHIP: Membership of the Society is via a Branch. Branches are autonomous in local matters, and may charge different membership subscriptions. Information may be obtained from the following Branch Honorary Secretaries, Canberra: PO Box 446, Canberra City, ACT, 2601, NSW: 1st Floor, 72 Pitt Street, Sydney, NSW, 2000. Qld: Box 1484, GPO, Brisbane, Qld, 4001. SA: Box 2423, GPO, Adelaide, SA, 5001. WA: Box F320, GPO, Perth, WA, 6001. Vic: PO Box 98, East Melbourne, Vic, 3002. Tas: PO Box 216, Sandy Bay, Tas, 7005.

CONTRIBUTIONS: All material for publication should be sent to Associate Professor J. Lions, Editor, Australian Computer Journal, Department of Computer Science, University of New South Wales, Kensington, NSW 2033. Prospective authors may wish to consult manuscript preparation guidelines published in the November 1982 issue. The paragraphs below briefly summarise the essential details.

Types of Material: Four regular categories of material are published: Papers, Short Communications, Letters to the Editor and Book Reviews. Generally speaking, a paper will discuss significant new results of computing research and development, or provide a comprehensive summary of existing computing knowledge with the aim of broadening the outlook of Journal readers, or describe important computing experience or insight. Short Communications are concise discussions of computing research or application. A letter to the Editor will briefly comment on material previously appearing in the Journal or discuss a computing topic of current interest. Descriptions of new software packages are also published to facilitate free distribution.

Refereeing: Papers and Short Communications are accepted if recommended by anonymous referees. Letters are published at the discretion of the Editor, and Book Reviews are written at the Editor's invitation upon receipt of review copies of published books. All accepted contributions may be subject to minor modifications to ensure uniformity of style. Referees may suggest major revisions to be performed by the author.

Proofs and Reprints: Page proofs of Papers and Short Communications are sent to the authors for correction prior to publication. Fifty copies of reprints will be supplied to authors without charge. Reprints of individual papers may be purchased from the Associated Business Publications, PO Box 440, Broadway, NSW, 2007. Microfilm reprints are available from University Microfilms International, Ann Arbor/London.

Format: Papers, Short Communications and Book Reviews should be typed in double spacing on A4 size paper, with 2.5cm margins on all four sides. The original, plus two clear bond-paper copies, should be submitted. References should be cited in standard Journal form, and generally diagrams should be ink-drawn on tracing paper or board with stencil or Letraset lettering. Papers and Short Communications should have a brief Abstract, Key word list and CR categories on the leading page, with authors' affiliations as a footnote. The authors of an accepted paper will be asked to supply a brief biographical note for publication with the paper.

This Journal is Abstracted or Reviewed by the following services:

Publisher
ACM Bibliography and Subject Index of Current Computing Literature.
ACM Computing Reviews.
AMS Mathematical Reviews.
ENGINEERING INDEX INC.
INSPEC Engineering Index.
INSPEC Computer and Control Abstracts.
INSPEC Electrical and Electronic Abstracts.
ISI Current Contents/CompuMath
ISI CompuMath Citation Index.
SPRINGER-VERLAG Zentralblatt fur Mathematik und ihre Grenzgebiete.

Copyright ©1984, Australian Computer Society Inc.

Production Management: Associated Business Publications, Room 104, 3 Small Street, Ultimo, NSW 2007 (PO Box 440, Broadway, NSW 2007). Tel: 212-2780, 212-3780.
All advertising enquires should be referred to the above address.

Printed by: Ambassador Press Pty Ltd, Parramatta Road and Good Street, Granville, NSW 2142.