South African
Computer
Journal
Number 8
November 1992

Suid-Afrikaanse
Rekenaar-
tydskrif
Nommer 8
November 1992

Computer Science and Information Systems

Rekenaarwetenskap en Inligtingstelsels

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

Die Suid-Afrikaanse Rekenaartydskrif
'n Amptelike publikasie van die Rekenaarverenging van Suid-Afrika en die Suid-Afrikaanse Instituut vir Rekenaarwetenskaplikes

Subscriptions

	Annual	Single copy
Southern Africa:	R45,00	R15,00
Elsewhere:	$45,00	$15,00

to be sent to:
Computer Society of South Africa
Box 1714 Halfway House 1685

Computer Science Department, University of Pretoria, Pretoria 0002
Phone: +27 (12) 420-2504 Fax: +27 (12) 43-6454

Sigma Press

Guest Contribution

The paper below was given as an invited address by Prof Roode at the July 1992 Conference of the South African Computer Lectures' Association. (Editor)

The Ideology, Struggle and Liberation of Information Systems

Dewald Roode
Department of Informatics, University of Pretoria

In 1989, Denning *et al* presented the final report of the Task Force on the Core of Computer Science in an article entitled "Computing as a Discipline" [3]. This was said to present a new intellectual framework for the discipline of computing and proposed a new basis for computing curricula.

In the words of the authors, "an image of a technology-based discipline is projected whose fundamentals are in mathematics and engineering." Algorithms are represented as the most basic objects of concern and programming and hardware design as the primary activities. Although there is wide consensus that computer science encompasses far more than programming, the persistent emphasis on programming "arises from the long-standing belief that programming languages are excellent vehicles for gaining access to the rest of the field" [3].

The new framework sets out to present the intellectual substance of the field in a new way, and uses three paradigms to provide a context for the discipline of computing. These paradigms are *theory*, rooted in mathematics; *abstraction*, rooted in the experimental scientific method and *design*, with its roots in engineering.

Programming, the report recommends, should still be a part of the core curriculum and programming languages should be seen and used as vehicles for gaining access to important aspects of computing.

The following short definition is offered of the discipline of computing [3]:

> The discipline of computing is the systematic study of algorithmic processes that describe and transform information: their theory, analysis, design, efficiency, implementation, and application. The fundamental question underlying all of computing is, *"What can be (efficiently) automated?"*

In the same issue of Communications, tucked away towards the end of the journal, an article by Banville and Landry asked the innocent question "Can the Field of MIS be disciplined?" [1]. It is not clear whether the use of the word "discipline" in both articles was purely coincidental – however, the implications were quite clear: computer science was able to talk about "computing as a discipline," and indeed, could present a report which, in a sense, was a culmination of more than twenty years' efforts. Yet, its sister discipline was still asking questions of a very introvertive nature about itself.

It has become quite clear that the fields (leaving aside for the moment the questions of "disciplines") of computer science and information systems (or MIS, informatics, or whatever other name we want to attach to it) have different aims and objectives, different problems that confront it, and, yes, if we want to be truly scientific, different paradigms. To support the latter statement, it is sufficient to contrast the three paradigms of computing with the four paradigms of information systems development described by Hirschheim and Klein [5]. It can be said that a central activity in information systems is the development of information systems, and that therefore, these paradigms have implications for the field of information systems. The four paradigms can be characterized briefly, as follows:

- The analyst as systems expert
- The analyst as facilitator
- The analyst as labour partisan
- The analyst as emancipator or social therapist.

In the same spirit, Lyytinen sees the "systems development process as an instrument in organizational change" [6] and remarks that analysts' principal problems are "in understanding the goals and contents of such change instead of solving technical problems." Already in 1987 Boland [2] observed that: "designing an information system is a moral problem because it puts one party, the designer, in the position of imposing an order on the world of another."

This is clearly a far cry from Denning *et al*'s statement that the fundamental question is "what can be automated?" At the same time, within the context of the field of computing, there is nothing wrong with this question, and it is probably the right question for practitioners of computing to continually ask themselves. But it is a disastrous question for a practitioner of informatics to ask. And it has taken us quite a long time to realise this – that the two disciplines have fundamentally different roles to play. These roles are complementary and supportive, and not destructively opposed.

The liberation of information systems lies in realising this elemental truth: that information systems are man-made objects designed to effect organisational change and that, as such, they can ill be studied using the paradigms of abstraction and engineering mentioned above.

What then is needed? Banville and Landry offer the consolation that we need not concern ourselves too much about the lack of discipline, and that we can indeed even pride ourselves in being a fragmented adhocracy. It is, in fact, even healthy to continue in all sorts of directions. During this process of finding itself, a discipline should be allowed a considerable degree of latitude, and many avenues should be explored. This obviously makes the field of information systems extremely exciting: it is in the process of discovering remarkable truths, discovering that there are in reality people out there using the systems which analysts design and build, and that the most intriguing problems centre around the role of people in all of this: the analyst, the user, their interaction, the impact of sytems on the work lives of workers on all levels, the impact on organizations. These are questions which have mostly been ignored or lightly treated over the years, but which have emerged as *the* problems to be solved. We do not have the tools to solve them – not yet; but a good starting point would certainly be to first understand more about our field and its research tools, for the empirical, positivist approach so often employed will not suffice to solve the above problems.

In the spirit of contributing to the liberation movement of information systems, we have embarked on a study of research on research in Information Systems, and will report on the results more fully in the near future. We define Information Systems as follows [4]:

> Information Systems is an inter-disciplinary field of scholarly inquiry, where information, information systems and the integration thereof with the organisation is studied in order to increase the effectiveness and efficiency of the total system (of technology, people, organisation and society).

In Information Systems then, we see the fundamental question underlying the entire discipline, to be the problem of balancing the need to contribute, through information systems, to the achievement of the mission of the organisation with the moral responsibility to develop and implement socially accepted information systems.

Each of the fields, computer science and information systems, benefits enormously from the activities of the other. Nonetheless, we must recognize the different approaches used by the two disciplines and allow them to complement each other. It should not be our business to convince one another that the universal truth is that which we use in our discipline – whether that be computer science or information systems. Instead, we should seek out the opportunities for synergy, and for complementing each other. If we succeed in doing this at SACLA, then we could indeed do ourselves proud.

References

1. C Banville and M Landry. 'Can the field of MIS be disciplined?'. *Communications of the ACM*, **32**(1):48–60, (1989).
2. R J Boland and R A Hirschheim, eds. *Critical Issues in Information Systems Research*. John Wiley & Sons Ltd., 1987.
3. P J Denning, D E Comer, D Gries, M C Mulder, A Tucker, A J Turner, and P R Young. 'Computing as a discipline'. *Communications of the ACM*, **32**(1):9–23, (1989).
4. N F Du Plooy, L D Introna, and J D Roode. 'Notes on research in information systems'. Unpublished research report, Department of Informatics, University of Pretoria, (1992).
5. R Hirschheim and H K Klein. 'Four paradigms of information systems development'. *Communications of the ACM*, **32**(10):1199–1216, (1989).
6. K Lyytinen. 'New challenges of systems development: A vision of the 90's'. *Data Base*, pp. 1–12, (Fall 1989).

Editor's Notes: To Compete or Collaborate

Human interaction invariably brings with it a blend of competition and collaboration. Competition means that one enjoys the exhilaration of winning while the other endures the shame of loosing. Because of this reward/punishment mechanism, it is a widely assumed that competition enhances performance and efficiency. This dogma pervades not only commerce, sport and politics, but is found in practically all areas of human endeavour, including research.

The competitive spirit in research is found in the well-known saga of Watson and Crick racing to unravel the double helix structure of DNA. Not so well-known, though equally illustrative, is the intensity of Newton's stratagems to oust Leibnitz from receiving any credit for differentiation. Recently there have been reports of scientists who have either tolerated or manufactured fraudulent results in order to win some or other scientific race. The space race, the arms race, the race for an AIDS cure, the scurry for faster smaller hardware, the race for awards, the drive for publications, Nobel prizes: all of this attests to a profoundly competitive international research culture.

But while competition might be the handmaiden of commerce and sport, it is the harlot of research – an unfortunate concomitant of the silly side of human nature. The archetypal researcher not only rises above the incidentals of human accolades; he disdains them. By tradition, the definitive research qualification is a PhD – a Doctor of Philosophy – a lover of thought. Discovery and thought are not only by their very nature rewarding, they are also humbling. When the archetypal researcher moves outside his interior thought-world, it is to share his discoveries. If he is childish, it is not the little boy flexing his biceps and saying: "I'm stronger than you" but the child rushing to

tell everyone: "Wow – look at this!" He is forgetful of self: Pythagoras, oblivious of the invading enemy and his impending death while he researches in the sand; Archimedes shouting "Eureka" without care for his nudity. The competitive spirit is a crass intrusion into this ancient legacy of innocence and selflessness.

By its nature, collaboration thrives in a climate of easy social intercourse. It may initially feel uncomfortable for researchers, who are inclined to be socially inept and are wont to bury themselves in work away from society. However, once the plunge to collaborate is taken there is ample evidence that it leads to successful research. In maximizing the use of available talent, it brings about a synergy in which two heads are better than one. All participants enjoy its rewards and no individual has to endure the full weight of its failures. In fact, the notion of collaboration is now so commonplace that significant research seems impossible without it. The tendency, however, is to encourage research collaboration within an organisation, but to emphasize competition in relation to outside organisations.

During a forum discussion at the July South African Computer Lecturers' Association (SACLA) conference, an appeal was made for greater collaboration between universities. Not surprisingly, the information technology disciplines at local universities have always had both a competitive and a collaborative relationship. The competitiveness usually takes the form of friendly rivalry, while the very existence of SACLA bears testimony to a rather unique collaborative relationship. In latter years the competitiveness seems to have intensified, while electronic mail and other developments have improved the prospects for collaboration. At issue, then, is whether there is an imbalance between these dual forces. The appeal at the SACLA forum implied that there is, and I would strongly agree. It is my view (my prejudice, if you will) that competition between universities is a self-indulgent and wasteful dissipation of energy.

Those who are inclined to compete should seriously examine what is to be gained. It is unconvincing to argue that winning makes a significant impact on the way in which students select universities: in the main, this is a matter of geography and language preference. To some extent, the same might be said about staff, although research reputation perhaps plays a more important role here. Neither are research funding agencies (e.g. the FRD) influenced by whether X is "better" in some or other sense than Y. On the contrary, it has wisely been decided to fund on the basis of criteria that are believed to be objective, without any reference whatsoever to the performance of competitors. True enough, funds are limited, but it is precisely for this reason that it is wasteful to divide the little there is between divergent research efforts.

It seems to me that there is a wealth of research talent out there, but that each researcher selects an area of interest almost as a matter of whim. There is an urgent need for well-coordinated collaboration on focussed research areas that have been carefully selected as directly relevant to the country. It is especially incumbent on those who finance, manage and lead research to identify such areas and to encourage collaboration in every possible way.

I look forward to the manifestation of such collaboration in SACJ publications authored by researchers from different university departments. To date there have been none of consequence. If we fail to collaborate, we are in danger of becoming little Don Quixotes who spend our lives attacking windmills and defending castles of xenophobia and irrelevance.

PEW: A Tool for the Automated Performance Analysis of Protocols

G Wheeler PS Kritzinger*
Data Network Architectures Laboratory, Department of Computer Science, University of Cape Town, Rondebosch, 7700 South Africa

Abstract

The Protocol Engineering Workbench (PEW) is an integrated software system or tool for the analysis of communication protocols specified using the Estelle *Formal Description Technique or FDT. The execution of systems specified in Estelle are simulated and data about the execution are gathered in the process. The PEW is thus aimed primarily at the performance analysis of such systems but, by virtue of the simulated execution, also ensures that some correctness errors will be detected. The paper describes the design philosophy and various components of the PEW as well as the facilities it provides for data collection and performance analysis. Part of the data gathered is the Markovian transition probability matrix where each transition in the protocol system is considered to be a state in a semi-Markov chain describing the protocol system execution.*

Keywords: *Computer networks, communication protocols, protocol engineering, performance evaluation, correctness of protocols*
Computing Review Categories: *C.2.1, C.2.2, C.4, D.4.9*

Received January 1992, Accepted August 1992.

1 Introduction

Efficient communication *protocols* are needed to maximise the utilisation of limited network bandwidth against varied and unpredictable demands. The design of such protocols is not a simple task. A good protocol must be flexible, reliable and efficient, and able to deal with the unexpected. These requirements have led to the study of various aspects of protocols, especially:

- *specification* - describing a protocol clearly and unambiguously, so that different implementations are compatible [4, 20];
- *implementation* - turning a specification into an implementation; much study is being done on making this process automatic [2, 3, 18];
- *conformance testing* - ensuring the implementation *does*, in fact, conform to the specification [1, 6, 15, 17];
- *verification* - ensuring that the protocol specification and/or implementation guarantees that good things will happen and bad things won't [1, 16, 19, 9];
- *performance evaluation* - determining the efficiency of the protocol, and 'tuning' it, if possible, to perform better [8, 11, 14].

As a result, a number of tools and methodologies have been created addressing these problem areas. The PEW is such a tool, aimed in particular at performance evaluation of protocols.

We begin by describing the Estelle Formal Description Technique used by the PEW. The structure of the PEW is then described, and its use in performance evaluation is illustrated with an example.

*On sabbatical leave in Fachbereich Informatik, Universität Dortmund, Germany

2 The Estelle FDT

Communication protocols lend themselves to being described as extended finite-state machines (EFSM), describing an action and next state for every possible external event (message arriving, timer expiring, and so on) in each possible state. Several different methods for the specification of protocols have been developed. These methods are known as *Formal Description Techniques* (FDTs).

The PEW is based on the Estelle FDT. Estelle (*Extended State Transition Language*) was developed by the ISO and standardized in 1989 [10]. The Estelle FDT is less abstract than most others FDTs, and is geared towards implementation. In fact, Estelle is closely related to Pascal in its statements, procedures and functions, with some additions. Estelle specifications consist of module descriptions, where each module is an extended finite state machine, and the modules communicate with one another by sending messages asynchronously over FIFO channels. A full discussion of Estelle is beyond the scope of this paper, but the major features are briefly outlined:

An executing Estelle specification consists of a tree of *instances* of generic *modules*. This structure may be dynamic; module instances may instantiate or terminate other module instances at any time.

There are two classes of modules in Estelle, called *activities* and *processes*. For simplicity, we will refer in this paper to all module instances as processes, as this term is commonly understood.

A transition consists of zero or more *clauses*, and a transition *body* containing statements which are to be executed when the transition fires. The clauses act as guards, determining whether a transition is enabled or not. They include:

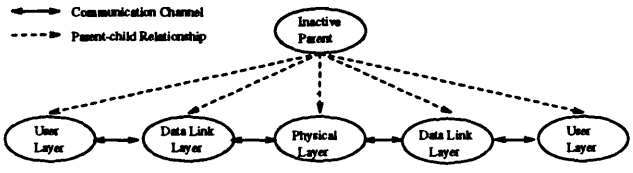

Figure 1. The process structure of a 3-layer point-to-point communication system.

- `FROM` - specifying the state(s) in which the transition begins;
- `TO` - specifying the state in which the transition ends;
- `WHEN` - specifying that a particular message must be at the head of a particular channel queue;
- `DELAY` - specifying how long the other clauses must be satisfied for before the transition can actually fire;
- `PROVIDED` - for any other arbitrary Boolean conditions; and
- `PRIORITY` - for ordering purposes in the case of multiple fireable transitions.

A transition is said to be *enabled* if its `FROM`, `WHEN` and `PROVIDED` clauses are satisfied; it is said to be *offered for execution* if it has been continuously enabled for the duration specified in its `DELAY` clause, and it is said to be *fired* if it is offered and selected for execution. The ISO standard ([10] section 5.3.5) states that the Estelle computational model is time-independent. The only requirement of an Estelle implementation is that time changes uniformly with respect to all the `DELAY` clauses.

Inter-process Communication (IPC) in Estelle is achieved by asynchronous message passing. Messages are input and output via *interaction points*. When a message is sent, it is appended to a theoretically infinite FIFO queue. This means that although an Estelle specification may consist of a number of FSMs, the system as a whole is not a finite state system, as an infinite number of channel states can potentially occur. This characteristic has important implications for state-space explorations of Estelle specifications, and has led to some criticism of Estelle, and the formulation of synchronous communication extensions to the language (see for example, [7]).

In order to represent a system consisting of two users communicating over a network using a protocol, we could use a top-level inactive process with five sub-processes: two user processes, connected to two protocol processes, connected in turn to a network process. The top-level process simply initialises the child processes and establishes their interconnections, after which it remains inactive and the five sub-processes execute concurrently (but asynchronously). For example, Figure 1 shows the process structure for a specification consisting of two user processes communicating with each other over a physical layer process via data link protocol processes. This structure is easily generalised to any number of protocol layers.

3 Structure of the PEW

The PEW is an integrated system containing an editor, an Estelle compiler, an interpreter, and a debugger, as well as separate performance analysis tools. The PEW environment has been described previously in [13]. This section describes its overall structure.

The Meta-Implementation of Estelle

Most compilers for Estelle are aimed at the implementation of protocols. Typically, they generate C code from the Estelle specification, which is linked with a run-time support library and makes use of UNIX's multiprocessing capabilities to support concurrency. Our interest is less in such implemen-tation-oriented systems, and more in using a specification language as a basis from which to investigate the performance of protocols.

Our objective was to provide a complete environment for the execution of a specification. We also monitor the execution of transitions to determine protocol performance measures. Some of the implementation-oriented Estelle compilers provide such features, but invariably the users have to explicitly add code to the transitions themselves. Our aim was to gather all necessary information automatically, without having to change the specification.

In its original version, the software was developed for MS-DOS, since, at the time (1988), UNIX had not yet attained its current universality. We thus could not rely on the operating system to provide concurrency and inter-process communication mechanisms for us; we had to provide that ourselves. In order to do so with the greatest degree of flexibility and portability, a virtual machine approach was adopted. We based our virtual machine on the Pascal p-machine model: a simple stack-based architecture with no accessible memory outside of the stack.

Implementing a symbolic debugger when an interpretive approach is used is straightforward. As the PEW compiler and interpreter are integrated into a single system, the compiler does not destroy the symbol table it creates when compilation is completed, but instead passes this symbol table to the interpreter. The interpreter thus has access to all the symbolic information it may need.

Components of the PEW

The PEW consists of several components, of which the most important are illustrated in Figure 2. The components are shown as rectangular blocks, while the files of information input and output by the components are shown as ovals. It was originally implemented in TurboC under the MS-DOS operating system and more recently on SUN SPARCstations under the UNIX operating system.[1]

Starting at the top, we have an Estelle specification, created using the text *editor*, which includes basic syntax-directed editing features and language help. The specification is then compiled by the *Estelle compiler*, which produces a file of pseudo-code which we call *E-code*, and a symbol table for use by the interpreter. The interpreter provides a meta-implementation of Estelle, and can be used to execute specifications in a controlled environment. Various information is output by the interpreter as a result of

[1] TurboC is a trademark of the Borland corporation. MS-DOS is a trademark of the Microsoft corporation. UNIX is a trademark of AT&T Laboratories. SUN and SPARC are trademarks of SUN Microsystems.

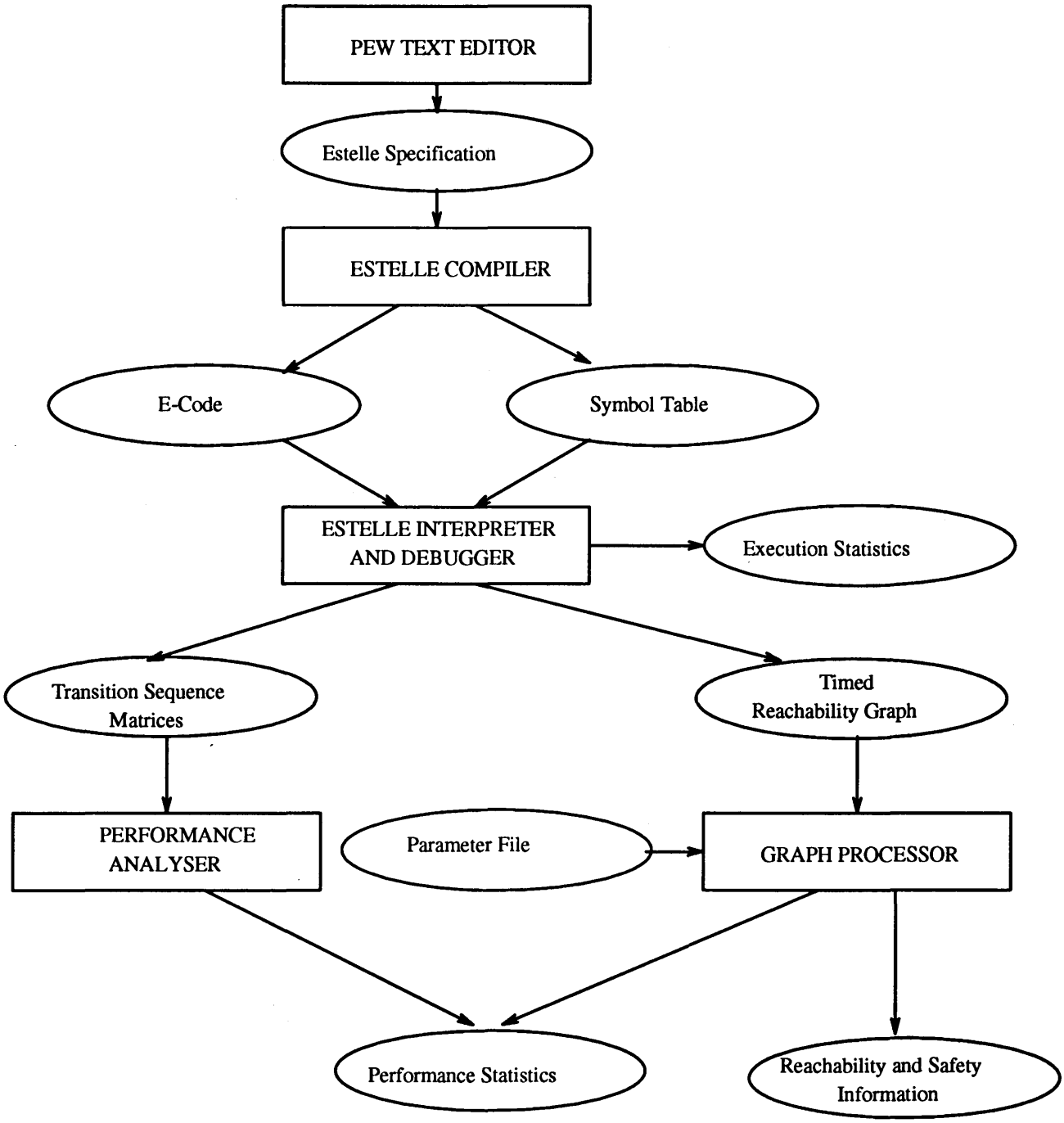

Figure 2. Structural components of the Protocol Engineering Workbench.

the execution. This includes a set of *transition sequence matrices* which can be processed by the *performance analyser*. The interpreter can also be used for the parametric analysis of a protocol; this results in the construction of a *timed reachability graph* which can be processed by the *graph processor* to produce throughput figures for different parametric configurations.

The PEW compiler supports most of the facilities of Estelle as defined in ISO DIS 9074. The compiler is based on the Pascal compiler of Brinch Hansen [5]. Recursive descent parsing is used to produce target code for an instruction set which we have designed for the meta-implementation of Estelle. This *E-code* is a superset of the Pascal p-code used by Brinch Hansen. The target architecture is a set of stack-based virtual machines, one for each executing process.

The PEW performs some checks on a protocol when it compiles the specification. While this is far from a full verification, it assists in finding obvious errors. Amongst other checks, the following errors are reported:

- Interactions that are `OUTPUT` but never referenced in `WHEN` clauses;
- Transitions with `WHEN` clauses that refer to interactions that are never `OUTPUT`;
- States that can never be reached;
- States that can never be left.

The PEW also detects and reports deadlock if it occurs when executing a specification.

As the PEW essentially provides a simulation environment without the need to construct simulation models, Pascal-like I/O procedures were added to the Estelle implementation, as well as a number of other useful functions and procedures. Examples are a function to return the execution time, and one to return the length of message queues. Pseudo-random number functions supporting several different probability distributions were also added. Of particular interest are a number of compiler directives to control aspects of the execution. These include:

- specifying the reliability of message passing commands (`OUTPUT` statements). This is useful for testing systems where the communication channel is unreliable;
- specifying the probability distributions of `DELAY` clauses;

The interpreter is responsible for executing the E-code instructions produced by the compiler. It in turn is controlled by a *scheduler* which evaluates the processes and decides which transitions should be executed.

4 Time and the Scheduler

The scheduling mechanism in the PEW is as follows:

1. Examine all processes, and select the set of non-delayed transitions for execution. Execute these, and return to step 1. If there are no such transitions, go to step 2.
2. Examine all delayed transitions, and choose the enabled delay transition which has the smallest remaining delay. Update the clock by the remaining delay, shorten the remaining delays of all other delayed transitions appropriately, and execute the transition. Go back to step 1.

Time therefore only advances in the PEW when a delayed transition is executed, and will not advance while there are non-delayed transitions to execute. Although the actual execution process is thus necessarily sequential, we found this a satisfactory way of effecting the concurrent execution of Estelle processes which occurs in practice.

The scheduler operates in one of three modes - *interactive, batch,* or *graph-building*. Interactive mode provides a debugging environment for Estelle, including facilities such as single stepping and breakpoints. This environment has been described in [13].

The batch mode is particularly useful for deriving performance estimates from a specification. The user specifies the name of the file containing the Estelle specification and the required duration of execution. The PEW will then execute the specification for this length of (simulated) time, write a file containing execution data, and exit. These data include the number of times each transition was enabled and fired, number of messages passed over channels and mean delays of events. The user may request various levels of detail in this execution record, up to a full execution trace. A sample excerpt from an execution record is shown in Figure 4. The interpreter also outputs the information necessary for a semi-Markov model of the system.

In the graph-building mode, the scheduler executes the specification thousands of times, exploring the possible behaviour of the system under any combination of delay values and/or message loss probabilities. A *timed reachability graph* is built of all possible orders of execution of transitions in this process. This graph is processed by the graph analyser, together with a parameter file specifying the loss probabilities and delay values for the appropriate transitions. The graph analyser performs several steps:

- it propagates the parameter values through the graph;
- it analyses this graph for reachability, and finds potential deadlock and livelock situations;
- it solves the graph analytically, producing throughput figures for the specification.

In order to determine the throughput and safety of the system with a different set of parameters, it is not necessary to rebuild the graph. We simply need to change the parameter file, and re-execute the graph analyser.

5 Facilities of the PEW

There are three general approaches to systems performance evaluation:

- Direct measurement and monitoring of the actual system, or
- simulating the performance of the system or, alternatively
- developing an analytical model of the system.

The PEW is aimed primarily at the performance evaluation of protocols and provides a combination of the latter two approaches, in that

- it executes Estelle specifications in a controlled environment and gathers performance statistics during this process. The specification may be excuted repeatedly in batch mode for various parameter values and the results of user-defined expressions (for example, for throughput) can be tabulated.
- The PEW generates semi-Markov transition probabilities matrices of the transition executions in a protocol layer. The resultant analytical models may be used to predict how changes in the specification will affect its performance.
- in certain cases to be discussed, a parametric analysis of a system can be done and parametric performance predictions can be made.

We will use the simple Producer-Consumer example described in Figure 3 to illustrate these different methods.

There are two processes in that example: a Producer which sends two different message types at different intervals, and a Consumer that receives these messages. The first message type can be sent unreliably, while the second type is always sent reliably.

Executing Specifications

Since the PEW simulates the execution of the protocol specification, it has to be complete. In other words, in the specification of a protocol layer, we need to provide a *user* process and *provider* process to use and provide services to the protocol. The provider service can include the possibility of message losses as well as transmission delays.

When we compile and execute such a specification, the PEW provides a great deal of information on the progress of the execution. A detailed trace of all the scheduling decisions can be printed, as well as a diagram showing the exchange of messages between processes as a function of time. At the end of the execution, the PEW prints a summary of execution statistics. An excerpt from such a summary is shown in Figure 4. Shown in this figure are:

- the *transition sequence count table* showing how often each transition was followed by others.
- the execution summary for each transition, showing how often it was enabled and fired, the time it first fired, the time it most recently fired, the mean delay between it firing and the next transition firing, the mean delay between occurrences of itself, and the number of times the PROVIDED and WHEN clauses were satisfied, if appropriate.
- A summary of the traffic dequeued through the process's interaction points. As the Producer only sends and does not receive, this is zero in the example shown.
- A summary of the interactions sent by the process, including how many were lost.
- The contents of the queues associated with the process's interaction points at the time of termination.

At the end of the figure are shown two further tables produced by the PEW. The *global transition sequence count matrix* shows the sequence counts for all processes internal to the layer, as well as a count of the number of messages

```
{ A simple producer/consumer example. }

SPECIFICATION producer_consumer;

CHANNEL access_point(producer,consumer);
    BY producer:    message1;
                    message2;

{**************************************}

MODULE producer_hdr;
    IP p_ip: access_point(producer)
                    INDIVIDUAL QUEUE;
END;

BODY producer_body FOR producer_hdr;
TRANS
    DELAY (7)
        BEGIN { 90% reliable output }
            {$R90}OUTPUT p_ip.message1;
        END;
TRANS
    DELAY(2)
        BEGIN
            OUTPUT p_ip.message2;
        END;
END;

{**************************************}

MODULE consumer_hdr;
    IP c_ip: access_point(consumer)
   INDIVIDUAL QUEUE;
END;

BODY consumer_body FOR consumer_hdr;
TRANS
    WHEN c_ip.message1 BEGIN END;

    WHEN c_ip.message2 BEGIN END;
END;

{**************************************}

MODVAR
    P_mod: producer_hdr;
    C_mod: consumer_hdr;

INITIALIZE
    BEGIN
        INIT P_mod WITH producer_body;
        INIT C_mod WITH consumer_body;
        CONNECT P_mod.p_ip TO C_mod.c_ip
    END;
END.
```

Figure 3. A simple Producer-Consumer example.

```
======================================================================

Unattributed 1.1 p_mod (producer_body) [Implicit] (Init time: 1)

TRANSITION SEQUENCE COUNT TABLE

        1    2
     ==========
   1:    0:1428:

   2:1428:3570:

TRANSITIONS

 From      To        Enabled Fired First Last    Back    Self    Forward
-------------------------------------------------------------------------
      ANY -> SAME      1428   1428     8  9997   1.500   6.995   0.500
      ANY -> SAME      5713   4999     3  9999   1.571   2.000   1.857


INTERACTION POINTS

 Name           Total Traffic   Mean Length  Max Length  Mean Time  Max Time
-------------------------------------------------------------------------
p_ip                 0               0           0          0          0


MESSAGE SENDS - SUCCESSES AND LOSSES
------- -----   --------- --- ------

IP p_ip           Intr: message2        Sent:  4999  Lost:     0  Total:  4999
IP p_ip           Intr: message1        Sent:  1265  Lost:   163  Total:  1428


CURRENT QUEUE CONTENTS
----------------------------

Ip  p_ip (    0) : empty

================================================================================
Classification Key
==================

c_mod has 2 transitions starting at 1
p_mod has 2 transitions starting at 3

GLOBAL TRANSITION SEQUENCE COUNT TABLE   GLOBAL TRANSITION SEQUENCE DELAY TABLE

        1    2 |   3    4                      1       2 |      3       4
     ======================                 ================================
   1:    0:1265:|   0:   0:              1:       *:  0.496:|      *:      *:
   2:1265:3733:|   0:   0:              2:   1.504:  2.000:|      *:      *:
   3:1265:   0:|   0:1428:              3:   0.000:      *:|      *:  0.500:
   4:    0:4999:|1428:3570:              4:       *:  0.000:|  1.500:  2.000:
```

Figure 4. Excerpt of execution summary statistics for Producer-process

sent and received by the various transitions *linking* the layers. For example, the entry in row 3 column 1 shows that the first Producer transition output 1265 messages which were dequeued by the first Consumer transition.

The *global transition sequence delay table* shows the mean delays that were measured for each pair of transitions following upon one another. For example, the mean delay between the execution of transitions 3 and 4 is 1.5 time units.

One can also perform parametric analyses with the PEW. Transitions can be named (a standard feature of Estelle), and the user can express performance measures of interest using these names. For example, the throughput rate of a protocol in messages per time unit could be expressed as:

```
throughput = GotACK / time
```

where `GotACK` is the name of a transition in which a message that has been correctly received is acknowledged. The user can then specify an identifier as being the variable parameter of interest, as well as a range of values. The PEW will recompile the specification for each value in the range (using the identifier as though it was a `CONSTANT` declaration), execute the specification, and tabulate the resulting values for the expressions set up by the user. It is thus trivial to obtain, for example, a set of throughput figures for different timeout values. The user need only set up the expression for throughput, and specify the range through which to vary the timeout constant.

Generation of semi-Markov Transition Probability Matrices

While executing a specification, the PEW notes the probability of each transition being followed by every other transition, as well as the mean delay between each transition firing and the next transition firing; this information is illustrated in Figure 4. The resulting information is used to parameterise a semi-Markov process model of the protocol system. The execution time must clearly be long enough to ensure that the system has reached steady state.

The semi-Markov models thus constructed can then be solved using the performance analysis tool to obtain the steady state probabilities of every transition in the process. These probabilities, together with the mean delays, can be used to compute the execution rates of the transitions and other performance figures of interest. Furthermore, we can modify the semi-Markov process model to predict the effect that changes on the system will have on its performance.

The model is a simplification of the method proposed by Kritzinger [12], based on the imbedded Markov process contained in an execution trace. The key to the model is the Transition Relation Graph (TRG) structure, which is derived by the PEW from the transition sequence count matrix. The TRGs for the Producer and Consumer processes are shown in Figure 5. The probabilities in these TRGs are based on the set of parameters given in the specification. Using these probabilities, and the delays given in the global transition sequence delay matrix, we can easily compute the mean delay associated with each "state" of the Markov process (the states of the Markov process correspond to the transitions of the protocol), modify the probabilities appropriately, and solve the resulting system to obtain throughput statistics.

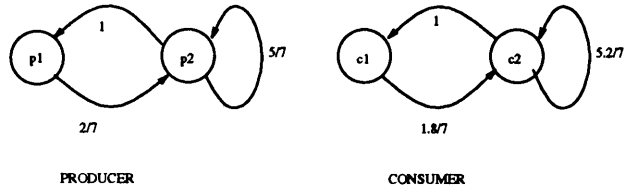

Figure 5. TRGs for the Producer-Consumer process.

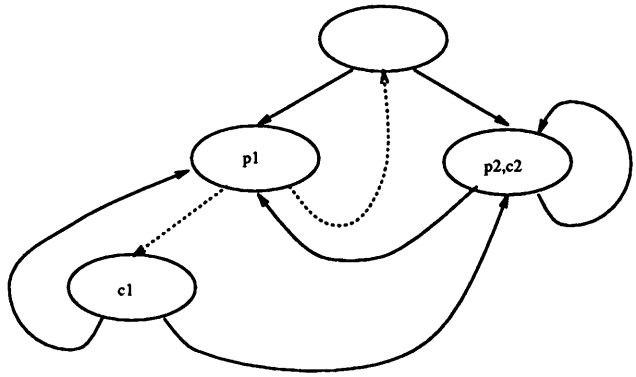

Figure 6. Timed Reachability Graph of the Producer-Consumer process.

Parametric Analysis

If a protocol specification can be executed and reaches steady-state, regardless of the loss probabilities and delays associated with events, then we can do a parametric analysis of its performance by building a *timed reachability graph* as explained next. In order for a protocol to have this degree of stability, synchronism must be enforced between the processes. If not, we can have a process `OUTPUT`ing messages much faster than they can be received, leading to an infinitely long queue, and hence an infinite number of system states.

It is fairly easy to add this type of synchronism to a specification. We can, for example, have the receiver send messages to the sender indicating that it is able to receive. The sender must in turn block until it receives such a message.

When a specification satisfies these requirements, it is possible to build a *timed reachability graph* which is independent of the actual loss probabilities and delays, but specifies all possible transition sequences that can occur. With the PEW one can construct these graphs. Figure 6 shows the timed reachability graph for the Producer-Consumer example. The solid arcs are branches that depend on the relative remaining delays of the Producer transitions *p1* and *p2*, while the dotted arcs are dependent on whether the message output by *p1* is lost (right branch) or not (left branch). Obviously, as with all state explorations, this straightforward method may not work when the state space becomes very large, and one will have to resort to state-space reduction techniques.

Once such a graph has been built, it can be parameterised by the graph processing tool. The user specifies a

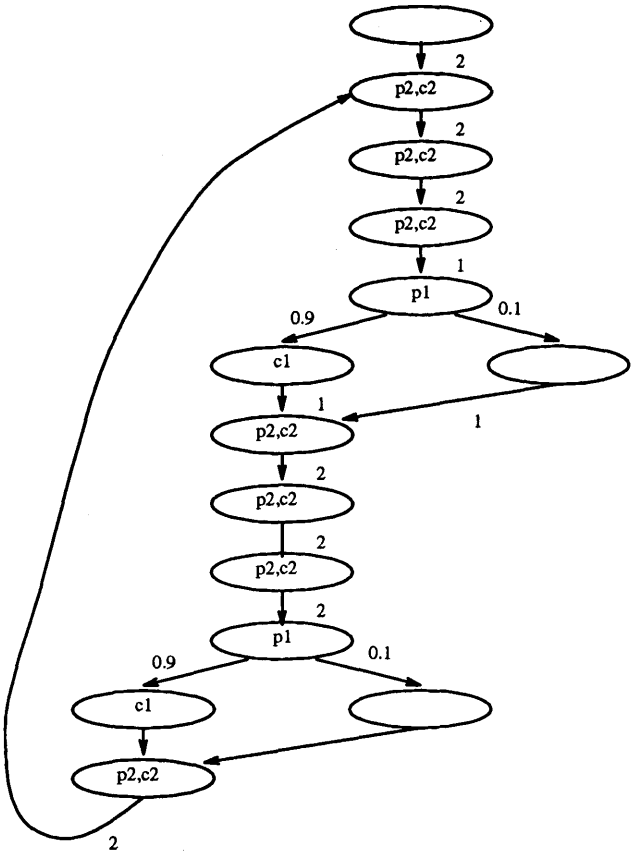

Figure 7. Sample parameterised Timed Reachability Graph of the Producer-Consumer process.

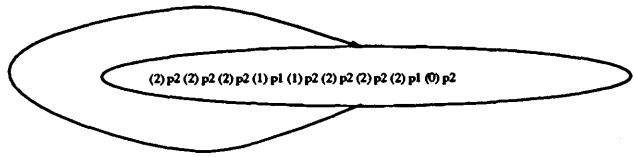

Figure 8. Parameterised Timed Reachability Graph of the Producer process.

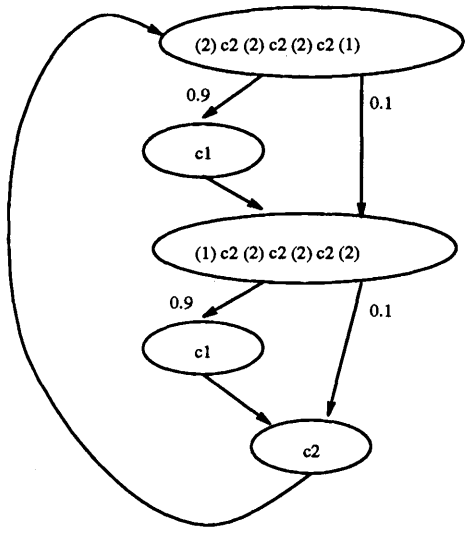

Figure 9. Parameterised Timed Reachability Graph of the Consumer process.

set of parameter values, and the graph processor recursively traverses the graph and labels it with delays and probabilities. One possible complication is that there may be more than one execution history leading up to a node, and the delays and probabilities are dependent on this history. We solve this by making copies of the affected nodes, with a unique copy for each history. The history information that is required is simply the length of time each enabled delay transition has been delayed for. This is sufficient information to determine which one will be executed next, and what the remaining delay is. The parameterisation process results in a larger graph, the actual size increase depending on the number of `DELAY` transitions and the relative primality of the delay values. This is not a simple process, and at this stage the PEW supports the parameterisation process for constant delay values only. Figure 7 shows the timed reachability graph parameterised for the actual delays and reliability given in the original specification. The arc labels are the delays or probabilities of the arc, whichever is appropriate.

The parameterised graph can then be simplified by separating it into different graphs for each process in the specification, and simplifying these by removing empty nodes (by replacing arcs to empty nodes with arcs to the children of the empty node, and adjusting the delays and probabilities appropriately), and merging unique linear paths into single nodes. The resulting graphs can be solved analogously to the solution of TRGs to obtain performance figures. We can simply repeat the parameterisation and solution process to check the performance for a different set of parameters. Figures 8 and 9 show the separate simplified graphs for the Producer-Consumer process. The figures in parentheses are delays, while the arc labels in the Consumer graph are the loss probabilities. It is easy to see from these graphs that in every 14 time units there will be 7 executions each of $p2$ and $c2$, 2 of $p1$, and 1.8 of $c1$.

6 Conclusion

In this paper we provided an overview of the structure and facilities of the Protocol Engineering Workbench (PEW), a software system primarily intended for the performance evaluation of protocols as an aid to their design and the testing of their specifications.

The tool has been deployed by ESKOM, the South African electricity utility, where it was used in the design and development of a three layer protocol as part of a powerstation control network in Southern Africa.

With the experienced gained, several improvements of the current version of the PEW are possible. Of particular advantage would be an Estelle-to-C^{++} translator for semi-automated protocol implementation. Moreover, although the Estelle FDT is a language, the PEW would nevertheless benefit from a higher level graphical interface description, similar to that of SDL, for the development of communication systems.

References

1. A V Aho, B S Bosik, and S J Griesmer. 'Protocol testing and verification within AT&T'. *AT&T Technical Journal*, **69**(1):4–6, (January/February 1990).
2. R Balzer. 'A 15 year perspective on automatic programming'. *IEEE Transactions on Software Engineering*, **SE-11**(11):1257–1268, (November 1985).
3. G Bochmann, G Gerber, and J Serre. 'Semi-automatic implementation of communication protocols'. *IEEE Transactions on Software Engineering*, **SE-13**(9):989–1000, (September 1987).
4. G V Bochmann. 'Specifications of a simplified transport protocol using different formal description techniques'. *Computer Networks and ISDN Systems*, **18**:335–377, (1989/90).
5. P Brinch Hansen. *Brinch Hansen on Pascal Compilers*. Prentice-Hall, 1985.
6. M Bush, K Rasmussen, and F Wong. 'Conformance testing methodologies for OSI protocols'. *AT&T Technical Journal*, **69**(1):84–100, (January/February 1990).
7. S C Chamberlain and P D Amer. 'Broadcast channels in Estelle'. *IEEE Transactions on Computers*, **40**(4):423–436, (April 1991).
8. A E Conway. 'A perspective on the analytical performance evaluation of multilayered communication protocol architectures'. *IEEE Journal on Selected Areas in Communications*, **9**(1):4–14, (January 1991).
9. G L Holzmann. 'Algorithms for automated protocol verification'. *AT&T Technical Journal*, **69**(1):32–44, (January/February 1990).
10. *Final Text of DIS 9074, Information Processing Systems – Open Systems Interconnection – Estelle – A Formal Description Technique based on an Extended State Transition Model*, May 1989.
11. L Kleinrock. 'Performance evaluation of distributed computer- communication systems'. In O J Boxma and R Syski, eds., *Queueing Theory and its Applications*, pp. 1–57. Elsevier Science Publishers, (1988).
12. P S Kritzinger. 'A performance model of the OSI communication architecture'. *IEEE Transactions on Communications*, **34**(6):554–563, (1986).
13. P S Kritzinger and G A Wheeler. 'A protocol engineering workstation'. In S T Vuong, ed., *Formal Description Techniques II, Proceedings of the IFIP TC/WG 6.1 Second International Conference on Formal Description Techniques for Distributed Systems and Communication Protocols, Forte '89, Vancouver, Canada, 5–8 December 1989*, pp. 53–59. Elsevier Science Publishers B.V. (North-Holland), (1990).
14. S S Lavenberg. 'A perspective on queueing models of computer performance'. In O J Boxma and R Syski, eds., *Queueing Theory and its Applications*, pp. 59–94. Elsevier Science Publishers, (1988).
15. R J Linn. 'Conformance testing for OSI protocols'. *Computer Networks and ISDN Systems*, **18**:203–220, (1989/90).
16. B Pehrson. 'Protocol verification for OSI'. *Computer Networks and ISDN Systems*, **18**:185–202, (1989/90).
17. D Rayner. 'OSI conformance testing'. *Computer Networks and ISDN Systems*, **14**:79–98, (1987).
18. D P Sidhu and T P Blumer. 'Semi-automatic implementation of OSI protocols'. *Computer Networks and ISDN Systems*, **18**:221–238, (1989/90).
19. C A Sunshine. 'Survey of protocol definition and verification techniques'. *Computer Networks*, **2**:346–350, (1978).
20. G v. Bochmann. 'Protocol verification for OSI'. *Computer Networks and ISDN Systems*, **18**:167–184, (1989/90).

Notes for Contributors

The prime purpose of the journal is to publish original research papers in the fields of Computer Science and Information Systems, as well as shorter technical research papers. However, non-refereed review and exploratory articles of interest to the journal's readers will be considered for publication under sections marked as Communications or Viewpoints. While English is the preferred language of the journal, papers in Afrikaans will also be accepted. Typed manuscripts for review should be submitted in triplicate to the editor.

Form of Manuscript

Manuscripts for *review* should be prepared according to the following guidelines.

- Use wide margins and $1\frac{1}{2}$ or double spacing.
- The first page should include:
  - title (as brief as possible);
  - author's initials and surname;
  - author's affiliation and address;
  - an abstract of less than 200 words;
  - an appropriate keyword list;
  - a list of relevant Computing Review Categories.
- Tables and figures should be numbered and titled. Figures should be submitted as original line drawings/printouts, and not photocopies.
- References should be listed at the end of the text in alphabetic order of the (first) author's surname, and should be cited in the text in square brackets [1, 2, 3]. References should take the form shown at the end of these notes.

Manuscripts accepted for publication should comply with the above guidelines (except for the spacing requirements), and may be provided in one of the following formats (listed in order of preference):

1. As (a) LaTeX file(s), either on a diskette, or via e-mail/ftp – a LaTeX style file is available from the production editor;
2. As an ASCII file accompanied by a hard-copy showing formatting intentions:
   - Tables and figures should be on separate sheets of paper, clearly numbered on the back and ready for cutting and pasting. Figure titles should appear in the text where the figures are to be placed.
   - Mathematical and other symbols may be either handwritten or typed. Greek letters and unusual symbols should be identified in the margin, if they are not clear in the text.

   Further instructions on how to reduce page charges can be obtained from the production editor.
3. In camera-ready format – a detailed page specification is available from the production editor;
4. In a typed form, suitable for scanning.

Charges

Charges per final page will be levied on papers accepted for publication. They will be scaled to reflect scanning, typesetting, reproduction and other costs. Currently, the minimum rate is R20-00 per final page for LaTeX or camera-ready contributions and the maximum is R100-00 per page for contributions in typed format.

These charges may be waived upon request of the author and at the discretion of the editor.

Proofs

Proofs of accepted papers in categories 2 and 4 above will be sent to the author to ensure that typesetting is correct, and not for addition of new material or major amendments to the text. Corrected proofs should be returned to the production editor within three days.

Note that, in the case of camera-ready submissions, it is the author's responsibility to ensure that such submissions are error-free. However, the editor may recommend minor typesetting changes to be made before publication.

Letters and Communications

Letters to the editor are welcomed. They should be signed, and should be limited to less than about 500 words.

Announcements and communications of interest to the readership will be considered for publication in a separate section of the journal. Communications may also reflect minor research contributions. However, such communications will not be refereed and will not be deemed as fully-fledged publications for state subsidy purposes.

Book reviews

Contributions in this regard will be welcomed. Views and opinions expressed in such reviews should, however, be regarded as those of the reviewer alone.

Advertisement

Placement of advertisements at R1000-00 per full page per issue and R500-00 per half page per issue will be considered. These charges exclude specialized production costs which will be borne by the advertiser. Enquiries should be directed to the editor.

References

1. E Ashcroft and Z Manna. 'The translation of 'goto' programs to 'while' programs'. In *Proceedings of IFIP Congress 71*, pp. 250–255, Amsterdam, (1972). North-Holland.
2. C Bohm and G Jacopini. 'Flow diagrams, turing machines and languages with only two formation rules'. *Communications of the ACM*, 9:366–371, (1966).
3. S Ginsburg. *Mathematical theory of context free languages*. McGraw Hill, New York, 1966.

South African Computer Journal

Number 8, November 1992
ISSN 1015-7999

Suid-Afrikaanse Rekenaar-tydskrif

Nommer 8, November 1992
ISSN 1015-7999

Contents

Sigma Press