J. M Bishop
24/3/91

South African Computer Journal
Number 4
March 1991

Suid-Afrikaanse Rekenaar-tydskrif
Nommer 4
Maart 1991

Computer Science and Information Systems

Rekenaarwetenskap en Inligtingstelsels

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

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

Editor
Professor Derrick G Kourie
Department of Computer Science
University of Pretoria
Hatfield 0083

Assistant Editor: Information Systems
Dr Peter Lay
P. O. Box 2142
Windmeul 7630

Subscriptions

	Annual	Single copy
Southern Africa:	R32-00	R8-00
Elsewhere:	$32-00	$8-00

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

Computer Science Department, University of Pretoria, Pretoria 0002
Phone: (012) 420-2843 Fax: (int) + (012) + 43-6454

Sigma Press

Guest Editorial

Does Today's Industry Need Qualified Computer Scientists?

This guest editorial consists of two contrasting views on the value to industry of a professional degree in computer science. Both authors, one local and one from Germany, are managing directors of well-respected software houses. (Editor)

Viewpoint I

Hans G Steiner
MBP Software and Systems GMBH
Semerteichestrasse 47-49
D4600 Dortmund 1

I would like to begin by recounting from my student days a story that I consider to be relevant. While attending a career forum for computer scientists, mathematicians and physicists, the personnel officer from IBM Germany was asked if he would consider taking on mathematicians. The gist of his answer was as follows: "Of course I must admit that I could just as well give the mathematician's job to a theologian. What is important is the ability to think logically. It is only there, on the job, that he learns how to become productive for us."

This episode occurred 14 years ago at a time when graduating mathematicians did not necessarily learn programming and when computer scientists were few and far between. The situation has improved immensely since then. Mechanical engineers, electrical engineers and physicists, all with programming knowledge, have for the most part taken over many programming jobs. This shows industry that, as time goes by, the answer to the opening question is becoming an ever-louder and more frequent "NO".

I support this opinion and in the remainder of this essay I will expand on my reasons, as well as highlight some exceptions.

An employee who is recruited directly from a university should possess the following four capabilities:
1. An ability to think logically: One of the basic requirements in our business is the ability to recognise, analyze, structure, break down and solve a problem as well as to fully synthesize the solution. The important thing is to break down the problem in such a way that the individual components can feasibly be solved. This is what distinguishes an engineer/scientist from an arts scholar. The latter usually concentrates on the complete problem and tends to settle for a contentious, complex and partially non-feasible solution. In our business, it is not enough to merely ask the "right" questions.

This ability to think logically may be accentuated in computer science; however engineers/scientists will generally possess the ability to an equal extent.

2. Programming skills: Our employees' prime tool of the trade is their ability to encode solutions to problems. Ideally, this ability ought to be held as abstract as possible. In other words, the further away from the "bit", the better. FORTRAN programmers who, for example, concentrate on the multiple use of memory space of all variables will never be successful programmers in an object-oriented programming language.

The difference can be seen, even in today's universities. For example, one only has to read a PROLOG program from a student who learned PASCAL in his first semester and PROLOG in his fifth. On average, this is always a "PASCAL program in PROLOG". The various possibilities offered by a predicate calculus language are only recognised and used by the best students. Again, we do not need the average computer science scholar who has spent between six and eight years writing complicated PASCAL programs, but rather the "thinker" with basic programming knowledge who is capable of abstracting the task. Once again, the ability is independent of faculty.

3. Teamwork skills: Working successfully in a team

This SACJ issue is sponsored by
Department of Computer Science
Rhodes University

requires assertiveness, tolerance, stability and one's own ideas. Very few problems have solutions that can be managed by one person successfully in the allocated time. Out of 700 employees, we can only afford approximately five "lone warriors" who are, in turn, the leading specialists in a wide field. They have a strategic vision which we follow. All remaining employees are evaluated, for better or worse, on their team performance. Some people have an in-built ability to work in teams. A few universities - unfortunately not enough - encourage this team-thinking. Again we see that the ability is independent of university faculty.

4. *Motivation:* The ability to enjoy one's particular job is a major driving force in every employee. Whereas in the sixties everything had to be "bigger, faster and better" and in the eighties "things had to be meaningful to society", the theme for the nineties is self-realization. Those companies who succeed in incorporating different employees (ie employees with different driving forces) into the company culture and who motivate each employee optimally will be successful in the nineties and beyond. There are huge productivity gains to be had from motivating employees. Compared with this, the possibilities offered by CASE tools pale into insignificance.

One basic requirement is thus the recruitment of a self-motivated employee who should at no stage become demotivated, whether it be by company culture, superiors or working conditions.

Again, this is not linked to a specific university faculty and is independent of know-how.

As none of these four capabilities are necessarily restricted to studies in computer science, the technical/-scientific background of new employees who are being recruited is largely irrelevant.

I would now like to point out a few exceptions which might give a computer scientist the upper hand in an interview. I refer exclusively to our own company and our specific company tasks.

1. Porting our COBOL Compiler onto the latest UNIX machine from the manufacturer XY. Knowledge of the UNIX operating systems could be very valuable and enable the new employee to rapidly become productive.
2. Programming the 37th interface (special customer request) for our ISDN card. Knowledge of interface protocols or experience with protocol conversions would be very useful and could be a decisive factor. Such specialized knowledge is usually very rare.
3. Adapting our integrated office automation system to the 17th foreign language. The employee must command the language perfectly. Simply outsourcing the translation would mean that this language version could not be maintained or supported. From this example one can see that specialized knowledge not only refers to knowledge gained from computer science studies.

In the product business, it sometimes happens that computer scientists with specialized knowledge are sought. (This is almost impossible in the project business, due to the variety of tasks to be performed.) However such a "knowledge" advantage over others usually only lasts about a year. After that, the achievements of two different employees (one with specialized knowledge and the other without) tends to even out.

Most applicants who start out do not know our products, as the flow of employees in this industry is almost always from manufacturer to user. Hardware and software manufacturers often lose their products specialist to the products' users. Seldom do employees change in the other direction.

In my opinion, universities can learn two things from this essay:

1. Studies in computer science give basic knowledge that can be used in various jobs. The student should however be careful not to place all his eggs in one basket.
2. Teamwork should be encouraged more. Time allows for very few geniuses, acting as 'lone warriors', to initiate progress in our society.

I have taken the liberty of basing my interpretation and answer to the opening question on my own judgement and experiences. I would be grateful for other opinions and experiences on this topic.

I would like to conclude by expressing my gratitude for having had this opportunity to express my views.

Viewpoint II

Pierre Visser
Grinaker Informatics, P.O.Box 29818, Sunnyside, 0132

The title question currently generates as many viewpoints as a counterpart question: "What is the correct curriculum for a computer science qualification?" Such questions stem from the many and diverse requirements expected to be fulfilled by the still developing applied science. A basic assumption of this editorial is that we need to have an explosion and consolidation in computer science theory. Only after this has occurred will a more general consensus of opinion exist - as is the case in other matured sciences.

An argument is presented here for the current approach of striving towards a balance between immediate industry needs and long term perceived theoretical requirements of industry, even though the balance, as viewed from either side, will always be imperfect.

Industry can, of course, do without qualified computer scientists - that is how it was established. Dedicated mathematicians, physicists, engineers and other scientists will, as in the past, continue to effect improvements. However, as one of those scientists from the

early days, it is difficult for me to understand why one would choose to continue this way.

A computer science qualification is viewed here as a university education (4 years) into theory that is not obtainable otherwise. By definition, therefore, a qualified computer scientist is not trained to conform to specific job requirements. Rather, the computer scientist will possess knowledge that will serve him long past the present day's computing technology.

Whether industry needs qualified computer scientists depends on two issues. Firstly, can an education be provided for computing technology that will serve as a foundation for the student's next 45 years in industry; and secondly, can industry build upon this foundation to created wealth more effectively than without qualified computer scientists.

It is widely accepted that, in broad terms, the teaching of fundamental theory will serve the first purpose. However, what subject matter to include from the wealth of mathematics, physics, OR, and from computing fields such as networking, operating systems and others, remains the illusive issue. Universities can merely strive to select the right mix for the perceived future needs of industry. This requires insight into the evolution of computing technology. I will later discuss such insight as a basic requirement for a qualified computer scientist.

What is important in teaching is to focus on fundamental theory. Just as the natural science student needs to breed fruit flies in order to gain insight into the dynamics of inheritance, so too the computer science student needs to develop software. The purpose should be to create understanding and insight into fundamental theory, and, just as in the case of the breeder of fruit flies, the software developed should never be measured against efficiency requirements from industry.

The second issue is whether industry can build on this theoretical foundation to create wealth.

A depth of insight into computing technology, more so than with other training, can be identified as the focus of the potential value of a qualified computer scientist to industry. Three areas which require such insight are discussed below, namely organisation, product definition and the application of new computing technology in industry.

Computing products form an integral part of an organisation, and represent a significant capital investment aimed at increasing efficiency. These products are incorporated in an evolutionary way to match changing organisational requirements with improving product capabilities. Decisions to use products determine the long term efficiency and cost-effective replacement. Such decisions require insight into computing technology and its evolution. A qualified computer scientist can improve such decisions only if he gains enough insight into computing technology as well as its interaction with business through years of practice.

The success of products in some areas is dependent on market requirements which depend on computing technology and its evolution. The correct definition of characteristics of products that interface to computers is such an example. Insight into computing technology is able to create the versatility, simplicity or other improved selling features which can open new market segments.

The third area where insight into computing technology plays an important role is in the application of new computing technology (or a new trend) in an organisation. Examples include the introductory period for networking, DBMS-technology, distributed processing and document image processing. In areas such as these, the newly qualified computer scientist can be applied effectively and at the same time build up insight through experience which he will require for the other areas of organisational and product decisions mentioned above.

A major dilemma in the continuous development of insight into computing technology by qualified computer scientists is their correct application in industry. The identification of the opportunities within the three areas discussed above, requires insight into computing technology itself. Winning companies that depend on computing technology have this ability. In such companies the insight of the qualified computer scientist into computing technology as well as its contribution to the business is constantly stimulated, turning the qualified computer scientist into a valuable company resource.

What has been neglected in this whole discussion is the role of the "technician" and of the casual user of computing technology. Such personnel are required to implement selected computing technology of the day efficiently, whether in accounting, chemical engineering or other specialised disciplines. Their role and place is unquestioned. However, it cannot be expected of them to evaluate the potential of new computing technology, formulate algorithms from fundamental theory or any such decisions which require insight built upon a sound theoretical knowledge of the field.

The final aspect in answering the opening question is whether the qualified computer scientist can outperform other professionals who build up their own experience in computing technology. Many examples could be cited of improvement brought about by non-computer scientists in the past. However, these individuals formed part of the bootstrapping for computer science theory and education. We should have faith in this bootstrapping of computer science qualifications, because computing technology will increasingly diversify into many directions of specialisation in years to come, each requiring a body of fundamental theory.

This complexity cannot be left to a casual development of insight - industry requires qualified computer scientists to experience interaction with business objectives in order to cope successfully with future computing technology.

An ADA[1] Compatible Specification Language

R. Bosua and A. L. du Plessis
Department of Computer Science, UNISA, Pretoria, 0002

Abstract

Specification languages to be used in conjunction with Ada as the implementation language have been proposed in the literature. Simultaneously, methods for the development of real-time software systems have been introduced, but very few Software Engineering Environments (SEEs) exist which provide automated support for these methods. This paper describes the synthesis of an object-based development methodology for real-time systems, the enhancement and design of a specification language, SALA (acronym for Systems Analysis and Design Language for Ada), and concludes with a short summary on using it in an SEE.

Keywords: *Ada, object-based development methodology, object-relationship-property model, real-time systems, software engineering, specification language, software engineering environment (SEE), System Encyclopedia Manager (SEM) system, system development life cycle (SDLC).*
Computing Review Categories: *D.2.1, D.2.10, D.3.2, D.4.7*

Received September 1989, Accepted November 1990

1. Introduction

A real-time system is a system which controls its environment by means of concurrent processing of multiple inputs, and by exchanging output results and actions within a very short time limit. The time dependency factor and precision of response are of considerable importance for this class of systems [3], [27]. Real-time systems are more difficult to build since functional correctness concerns are compounded by timing correctness constraints [19]. Few system development methodologies include appropriate methods for modelling aspects such as parallelism, task communication and dynamic behaviour [16]. Furthermore there is a need for a life cycle methodology which supports the development of a software system whilst also allowing the system requirements of the environment to be considered [28]. The emergence of CASE tools and increasing use of Software Engineering Environments (SEEs) may lead towards a more rigorous approach to system development. As the real-time system area is still regarded by some as an experimental area of software engineering [16], there is clearly scope for research in the area of SEEs for this application domain. This implies a need for improved methodologies, tools in support of its methods, and particularly also for an appropriate formal specification approach.

This paper describes an investigation during which a methodology for the development of real-time systems was synthesized from various existing methods, and SALA, a specification language originally designed to support Ada, was enhanced in support of this methodology. Section 2 presents problems associated with real-time system development, and Section 3 proposes an approach to overcome some of the problems perceived in this application domain. Section 4 explains how the approach of Section 3 is applied in designing SALA, and concludes with a summary on using the language within the SALA SEE. The development of this environment is described by Bosua [9].

2. Real-time system development problems

During recent years there has been an increased emphasis on more effective and efficient techniques and methods for the requirements analysis and preliminary design phases of the software development life cycle (SDLC) [5], [18]. This trend has been motivated by the realization that accuracy and quality of requirements and design specifications result in implemented systems with fewer defects [1],[2]. While better methodologies for the development of business systems have become available, the same is not true for the real-time domain of application [27]. Only in recent years have appropriate techniques, based on a theoretical foundation, been adopted for real-time systems. This is probably due to the fact that the quantitative demands on the technology, such as critical response times, have been more stringent for real-time applications than for typical business applications. It was only possible to shift from an implementation-oriented to a problem-oriented approach to development when the explosive growth of micro-processor capabilities occurred in the late nineteen seventies. It has become feasible to integrate both the real world issues of real-time systems and its

[1]Ada is a registered trademark of the US Government Ada Joint Program Office

operational characteristics. A further factor hampering the availability of adequate development methods for real-time systems is the intrinsic complexity of such systems due to, for example, the need to model controlling components, communication between components and time dependent behaviour. It is consequently more difficult to establish a comprehensive and coherent methodology for the development of these systems in terms of objects, events and actions.

As for other domains there is a need for a good design method based on recognized software engineering concepts. These concepts are addressed briefly in Section 3.2. Although many automated tools have become available and development environments have been reported there are only a few environments whic h support the development of real-time software [13],[14].

In addition to the above mentioned issues there is also a lack of a formal specification language which supports an analysis and design methodology as well as the implementation language explicitly. A formal language in this sense refers to a computer-processable language with a predefined syntax and semantics, and which is based on a sound conceptual model.

3. Real-time software development

The problems handicapping the development of real-time software that have been addressed in Section 2 may be tackled by following a multi-facetted approach: the design of a specification language for requirements analysis and preliminary design, the adoption of a suitable methodology, and the creation of a development environment in which the language may be used. Each aspect of the approach is described below.

3.1 The specification language

A formal specification language is one which is based on some conceptual model, such as the entity-relationship-attribute model, and has a syntactic and semantic form which allows specifications in that language to be stated in a precise, unambiguous, concise and consistent manner. When the analysis and preliminary design requirements of a system which is to be developed, hereafter called the target system, are specified by means of such a language the specifications may be analyzed for completeness, unambiguity and consistency [1],[15]. The analyzed specifications may then be stored in a central repository, serving as a basic point of reference for the evolving software product during the subsequent phases of the SDLC [22],[25]. This central repository supports the software product development and extends the role of the data dictionary.

The specification language model may be designed in terms of the following: the methodology used for developing the class of target systems, important design information to be represented and retrieved using the methodology, and the implementation language to be used. The design of the SALA language model is described in Section 4. SALA was selected for enhancement to accommodate the specification and design problems of real-time software development. Although other specification languages for Ada have been designed, such as for example ANNA [20], methodological support for systems analysis and design is generally not embodied in these languages.

3.2 A methodology for real-time system development

The value of appropriate methods and techniques for modelling the static and dynamic aspects of software systems, and in this case real-time software systems is generally accepted [3],[4],[27]. Apart from facilitating the management of the software development task a methodology should offer methods based on paradigms for controlling the complexity of a large system. An example is a paradigm for decomposition which enables the factoring of a problem into separable subprograms in such a way that each subprogram is at the same level of detail; each subprogram can then be solved independently and the resultant solutions combined to solve the original problem [21].

During the investigation an object-based development methodology was synthesized from well-established methods. At the *highest level of abstraction* an embedded real-time system can be viewed as an operational environment which resides within a larger monitoring environment. This abstraction is modelled graphically by means of a *Problem Space Profile* [9], as in Exhibit 4. *System requirements* in the operational environment are modelled as high-level objects with the visibility between objects in the problem space, and the operations of each object, identified. This corresponds to the Booch approach [7],[8] and the subject layer of the recently announced Object-oriented Analysis model of Yourdon [30]. The *representation scheme* used at this level is the *Unit Level Profile* [6] (Refer to Exhibit 4). A *top-down decomposition* paradigm is used to obtain smaller scope objects yielding the structure of the high-level objects. A *functional decomposition* method was adopted to model the details of the operations of an object. The *representation scheme* used is derived from the Structured Analysis Method and its associated data flow diagrams according to De Marco [12]. At the *lowest level* of abstraction the internal behaviour of objects, their associated operations and the structure of their data are represented by means of the *Data Level Profile Chart* and *Specification Level Profile Chart* [6]. An example instance of the Data Level Profile is illustrated in Exhibit 3.

The paradigms of this methodology that support the development of real-time systems are : *abstraction*, which is a key concept in reducing the complexity of a system and an *object-based approach* to model the dynamic behaviour of objects. This approach allows for the identification of objects and classes of objects that

SE PRINCIPLE	METHODOLOGY SUPPORT
Encapsulation information hiding and abstraction	An object encapsulates the group of logically related entities or computational resources forming part of the object. Information hiding supports abstraction by suppressing how an object is implemented
Modularity	Identification of objects each with its associated operations enhances modularity at the programming-in-the-small and programming-in-the-large levels
Strong data typing	Each object has a clearly defined set of values and operations. Strong data typing restricts the operations on classes of objects.
Structured development	Strong support for abstraction and top-down development promotes this principle. Objects and data may be decomposed in a natural way
Separation of specification and implementation	The design process is independent of the representation details of the data objects used in the system

Table 1 SE principles supported by the methodology

are modelled as tasks and task types in support of concurrency.

The methodology is based on the notion of objects of the problem space and their associated operations. Each object can be regarded as an abstraction having a set of allowable operations, with its internal details hidden from other objects. Both *abstraction and information hiding* could be viewed as mechanisms for achieving *modularity* [23]. Other software engineering principles and techniques supported include *encapsulation, modularity, strong data typing, structured development*, and the *separation of the specification from the implementation*. Table 1 summarizes the support offered for software engineering principles provided by the methodology. These concepts generally attribute to a better quality end product [7],[8]. The modelling primitives used with the methodology map directly to Ada constructs hence providing a *smooth transition* to the implementation phase. Strong data typing, the definition of abstract objects and their operations, and the use of Ada as an implementation language, promote *portability*. *Verifiability* is served by separation of the specification from the implementation, since support for abstraction and information hiding makes the design process independent of representation details of data objects. This concept is also supported by the presence of strong data typing, since only valid operations are allowed for specific data types. The influence of the methodology on the design of SALA is addressed in Section 4.1.1.

The use of various complementary representation schemes serves to make the evolving software product understandable.

3.3 Implementation language issue

A significant trend in real-time software implementation is the use of high-level languages which provide support for expressing, for example, concurrency, synchronization, interrupts and exceptions [3]. When used in the implementation phase following software development by means of an object-based methodology it is preferable that the implementation language provides primitives and constructs that are complementary to those of the methodology. In this context Ada is a suitable implementation language for object-based software development, allowing objects and operations of the methodology to be mapped easily to the language. This argument may equally be extended to other object-oriented languages. This study however commenced with the original specification language, SALA, which was designed specifically for Ada [11]. Although the distinguishing features of Ada are well-known it is important for this study that it has considerable expressive power, due to its extensive language primitives and constructs, supports a number of software engineering concepts of value to real-time software development, and has the package, task and subprogram primitives as major building blocks [24]. A number of research projects have been devoted to designing a specification language based on Ada [11],[17],[20]. Such a language facilitates the programming effort considerably, since the notation and form used are familiar to Ada programmers. In a suitable SEE it is possible that segments of high-level Ada code may be generated automatically from formal specifications. In addition to these and other arguments in favour of specification languages, the support which a specification language may afford a methodology makes its availability a distinct advantage.

The rest of this paper is devoted to a description of the SALA language, a summary on using it, and a SALA example (Exhibits 4 and 5).

4. SALA for object-based development

The specification language SALA (Systems Analysis and Design Language for Ada) originally developed by Davis [11], was designed to specify the analysis and design requirements for target systems to be implemented in Ada. This original version of SALA has two limitations namely: *it does not support a specific development methodology*, and *it does not address real-time design issues explicitly*. The enhanced specification language supports a synthesized object-based development methodology and the real-time application domain with implementation in Ada [9]. This approach has general appeal particularly since specifying requirements using Ada on its own forces too much implementation detail into the requirements specification document.

4.1 The SALA language model

The language model of SALA is the object-relationship-property (ORP) model, enhanced with text types and value-ranges [26], and is based on the Entity-relationship-attribute model of Chen [10]. An object is a means for classifying data types, while the relationships describe which objects in the model are related to each other, and the nature of the connection. Properties are values of particular types that describe relevant data items for an object type. Text types allow textual information to be entered as additional comments to object instances, that are not subjected to consistency and completeness checking. The ORP model is the meta model of the System Encyclopedia Manager (SEM) System [25], [29] under which SALA was implemented. This meta model allows two, three and four object types as participants in a relationship, while the available connectivity forms between objects are 1:1, 1:M, and M:M for two-part relationships, 1:1:1, 1:1:M, M:M:1 and M:M:M for three-part relationships, and 1:1:1:1, M:1:1:1, M:M:1:1, M:M:M:1 and M:M:M:M for four-part relationships. The object and relationship types for SALA were derived from the following sources: the concepts of the methods of the synthesized object-based development methodology described in Section 3.2, the Ada language primitives and constructs, and the concepts required to model real-time systems.

The object types and relationship types of SALA, and hence the SALA language statements, were structured into a number of System Aspects to facilitate the modelling of the target system. A System Aspect represents a particular perspective of the system. When developing a software system, the system itself is considered from various perspectives, or System Aspects. For example, the Data Specification Aspect would focus on the structure of data objects of the system. The idea of incrementally building the specification by specifying each System Aspect is analogous to the notion of separation of concerns [23]. Exhibit 1 tabulates the SALA object types that may be used when specifying the various System Aspects. Exhibit 2 represents an ORP diagram of the objects and relationships of the Process Control and Dynamics System Aspect. The latter enables the specification of process control requirements and dynamic behaviour of a real-time target system.

4.1.1 The SALA language primitives and constructs

The SALA language primitives refer to object types, property types and relationship types. SALA language constructs refer to the language statements. Since SALA was designed to support the semantics and paradigms of a synthesized object-based development methodology, the Ada programming language for implementation as well as the terminology of the real-time domain of application, meaningful language primitives were chosen.

For example, an object type *package* denotes an object inherent to the methodology and is one of the Ada primitives for an object. Operations of an object may be defined by means of relationship types, such as *causes*, *initiates* and *causes-entry* that are typical operations to be found in modelling real-time systems. Object types to depict a larger monitoring environment and the operations environment are *development environment* and *system*, respectively. The visibility of high-level objects, represented by the Unit Level Profile, determined the choice of relationship types in SALA, such as *imports*, *use* and *use-with*, for example. The top-down decomposition paradigm is supported by means of the *subparts* relationship. The objects and relationships identified so far support programming-in-the-large. At this level, the main objects, the operations, the visibility between objects and the structure of the objects in the problem space can be specified.

Relationships such as *parameter*, *specification* and *private*, for example, were chosen to denote Data Level and Specification Level Profiles, enabling specification at programming-in-the-small level. The meta level ORP diagram for the Data Level Profile, an example instance and the corresponding SALA statements are given in Exhibit 3.

In Exhibit 5, Diagram 0 of Bedside_vdu_process and the decomposition of Monitor_patient_condition are shown together with SALA statements. These statements illustrate the composition of the Monitor_patient_condition package and the behaviour of Handle_treatment task.

SALA object types which stem from the real-time domain are for example, *interface*, *event*, *condition*, *resource*, and *task*. Relationship types for this domain are *use*, *causes*, *calls-entry*, *accepts*, *initiates*, *makes-true* and *makes-false*. Detailed physical design issues may be specified by means of property and text types. Examples are specifying priorities, or handling of exceptions within the body-code of a task. Here *body-code* is a text type

	ORIGIN OF SALA OBJECT TYPES		
SALA OBJECT TYPES	Object-based Methodology & Profiles	Real-time	Ada
Generic Package	X		X
Package	X		X
Task	X	X	X
Generic Subprogram	X		X
Subprogram	X		X
Record	X		X
Array	X		X
Variable	X		X
Type	X		X
Entry	X	X	X
Condition	X	X	
Event	X	X	
Interface	X	X	
SALA RELATIONSHIP TYPES			
Subparts	X		X
Instantiates	X		X
Specification	X		X
Private	X		X
Generic-Parameter	X		X
Parameter	X		X
Use	X	X	X
Use-with	X		X
Generates	X		X
Receives	X		X
Causes		X	X
Calls-entry	X	X	X
Accept	X	X	X
Initiates		X	
Makes-true		X	
Makes-false		X	
Type		X	X
Acquires	X		
Collection	X		

Table 2 The partial SALA language model

of SALA. Time-critical implementation aspects of real-time systems can also be expressed by means of text types.

The choice of object types and relationship types were primarily influenced by the Ada language primitives, in line with the arguments raised in Section 3.3. Table 2 motivates the choice of SALA object types and relationship types as derived from the synthesized methodology, the real-time domain and Ada. The language model makes provision for specifying project management and low-level physical design information, and details regarding the body part of a package or task. This is done by means of property type and text type primitives such as *version preferred* and *body code*, respectively. Enhancements were made with regard to System Aspects, object types, relationship types, cardinality of relationships and text and property types, in line with the objectives of the investigation. Details regarding these enhancements are available in Bosua [9].

4.1.2 Using SALA

Before SALA can be used meaningfully as a specification language a SEE must be created. This was done using the SEM System, which is a generic SEE generator. The customisation of SEM for SALA is described in a dissertation by Bosua [9].

The SDLC support provided by the SALA SEE starts with the application of a synthesized object-based development methodology to the Software Requirements Analysis phase. The analysis and design requirements are specified in SALA statements per System Aspect. These statements are then checked for syntactic and semantic consistency by the SEM/SALA processors. Once the specifications are diagnostically correct the repository for the target system is populated and subjected to extensive verification by means of the Query System of the environment. The Query System enables a developer to enforce methodology and other software management standards. The RSI capability, which has a 4GL type language, was used to generate skeleton Ada source code from the repository.A n example of limited scope is represented in Exhibits 4 and 5. Further information regarding the SEM System and the customization procedure is available in Du Plessis [13].

5. Conclusion

An existing specification language for Ada, SALA, was extended to include support for an object-based development methodology, for Ada Profile Charts enabling real-time systems to be modelled in terminology particular to this problem domain. The object-based development methodology was synthesized from an object-based approach to software requirements analysis and a combination of functional decomposition approach and an object-based approach to preliminary and detailed design. The SEM System, a SEE generator, was customized for SALA yielding a SEE in which real-time software can be developed up to the point where coding in Ada can commence. The SALA SEE provides automated support for a number of software engineering principles.

Bibliography

[1] R J Abbott and D K Moorhead, [1982], Software Requirements and Specifications: A survey of needs and languages, *The Journal of Systems and Software*, **2**, 297-316.

[2] D Alberts, [1976], The economics of Software quality assurance, in *AFIPS Conference Proceedings*, National Computer Conference.

[3] S T Allworth and R N Zobel, [1987], *Introduction to Real-time Software Design*, MacMillan, 2nd edition.
[4] S C Bailin, [1989], An object-oriented requirements specification method, *CACM*, 32(5).
[5] N D Birrell and M A Ould, [1985], *A Practical Handbook for Software Development*, Cambridge University Press, London, p259.
[6] J M Bishop, [1987], Ada Profile Charts in Software Development, *Computer Science Report*, CS-JM-87-01, University of the Witwatersrand.
[7] G Booch, [1987], *Software Engineering with Ada*, Benjamin/Cummings Publishing Company, CA., 2nd edition.
[8] G Booch, [1987], *Software components with Ada*, Benjamin/Cummings Publishing Company, CA.
[9] R Bosua, [1988], SALA: A specification Language for an object-oriented development methodology, *M.Sc. Dissertation*, University of South Africa.
[10] P P Chen, [1976], The Entity-Relationship Model - toward a unified view of data, *ACM Trans. on Database Systems*, 1(1), 9-36.
[11] G Davis, D Block, K C Kang, E Chikofsky and D Teichroew, [1985], Usage of the System Encyclopedia Manager (SEM) System with the Systems Analysis and Design Language for Ada (SALA), *ISDOS Project*, University of Michigan, 157-202.
[12] T De Marco, [1979], *Structured Analysis and System Specification*, Prentice-Hall International, London.
[13] A L Du Plessis, [1985], A Software Engineering Environment for Real-time Systems, Ph.D Dissertation, UNISA.
[14] H Falk, [1988], *CASE tools emerge to handle real-time systems*, Computer Design, 53-74.
[15] N H Gehani, [1986], Specifications: Formal and Informal - A case study, in *Software Specification Techniques*, N. Gehani and A D McGettrick (eds.), Addison Wesley.
[16] R Guth, [1986], *Computer Systems for Process Control*, Plenum Press, N.Y.
[17] A Hill, [1983], Towards an Ada-based specification and design language, *Ada UK News*, 4(4), 16-3.
[18] G F Hoffnagle and W Beregi, [1985], Automating the Software Development Process, *IBM Systems Journal*, 24(2), 102-120.
[19] H Kopetz, [1986], Design of Real Time Systems in *Computer Systems for Process Control*, edited by Reinhold Guth, Plenum Press, New York.
[20] B Krieg-Bruckner, and D C Lucknan, ANNA: Towards a language for annotating Ada programs, *ACM Sigplan Notices*, 15(11), 128-138.
[21] B H Liskov, and J Guttag, [1986], *Abstraction and Specification in Program Development*, McGraw-Hill, N.Y.
[22] C McClure, [1989], 3 R's of Software automation - re-engineering, repository, reusability, Extended Intelligence, Inc.
[23] D L Parnas, [1972], A Technique for Software Specification with Modules, *CACM*, 15(5).
[24] V Rajlich, [1985], Paradigms for Design and Implementation in Ada, *CACM*.
[25] D Teichroew, [1982], The Development of Softwa re Support Environments, *Proc. of the National Conf. of the Canadian Information Processing Society*, Saskatchewan, Canada, 200-210.
[26] D Teichroew, P Macasovic, E A Hershey and Y Yamamoto, [1979], Application of the Entity-relationship Approach to Information Processing Systems Modelling, *Proc. Int. Conf. Entity-relationship Approach to Systems Analysis and Design*, 23-51.
[27] P T Ward and S J Mellor, [1985], *Structured Development for Real-time Systems*, Yourdon.
[28] S M White, [1987], A Pragmatic Formal Method for Computer System Definition, *Ph.D. Dissertation*, Polytechnic University, New York.
[29] Y Yamamoto, [1981], An Approach to the generation of software life-cycle support systems, *Ph.D. Dissertation*, University of Michigan.
[30] Yourdon E , [1990], Object-oriented Analysis, *Case seminar*, Johannesburg.

SYSTEM ASPECTS / OBJECT TYPES	SYSTEM STRUCTURE	PROCESS SPECIFICATION	DATA SPECIFICATION	PROCESS CONTROL AND DYNAMICS	FILE STRUCTURE	PROPERTIES AND CHARACTERSITICS
Record		*	*		*	*
Array		*	*			*
Variable		*	*		*	*
Type		*	*			*
Task	*	*	*	*	*	*
Package	*	*		*	*	*
Generic Package	*	*			*	*
Subprogram	*	*		*	*	*
Generic Subprogram	*	*			*	*
Program	*	*		*	*	*
Directory			*	*	*	*
File			*	*	*	*
Interface				*	*	*
Test					*	*
Development Environment	*					*
Project	*					*
System	*					*
Subsystem	*					*
Job				*		*
Event			*	*		*
Entry		*		*		*
Condition				*		*
Attribute	*	*	*	*	*	*
System Parameter			*	*	*	*

Exhibit 1 Classification of SALA object types into system aspects

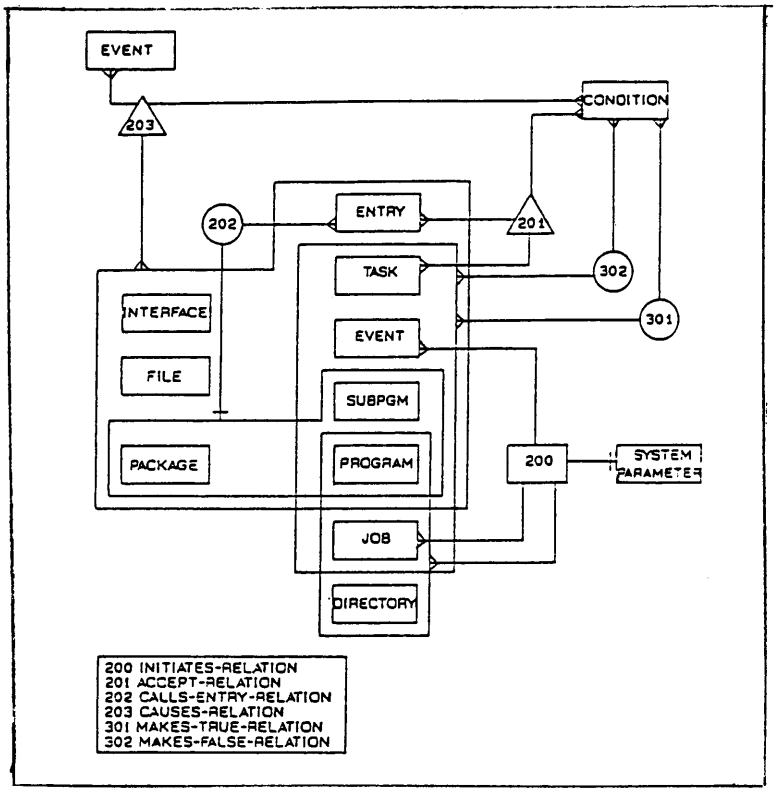


Exhibit 2 ORP Diagram of process control and dynamic system objects

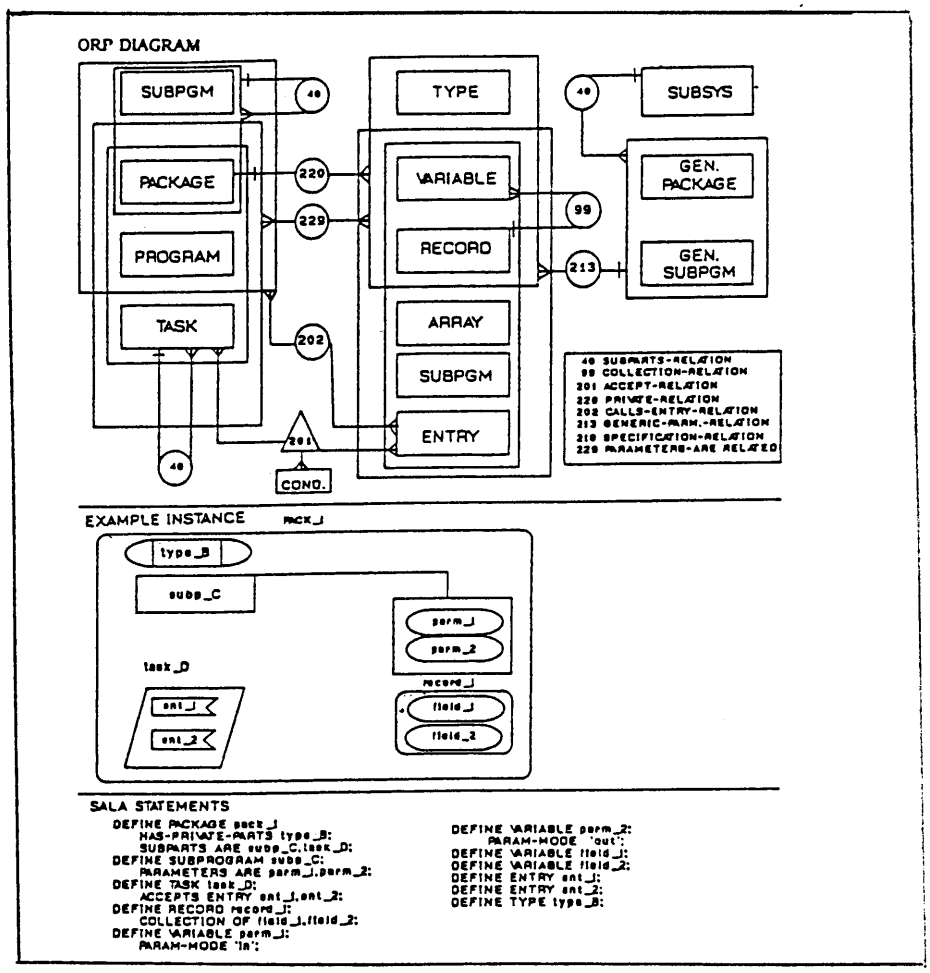
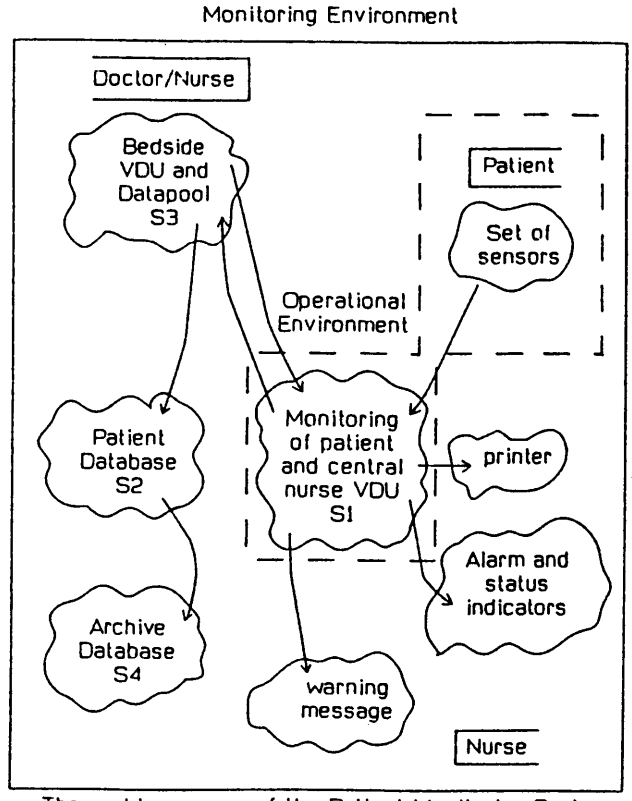
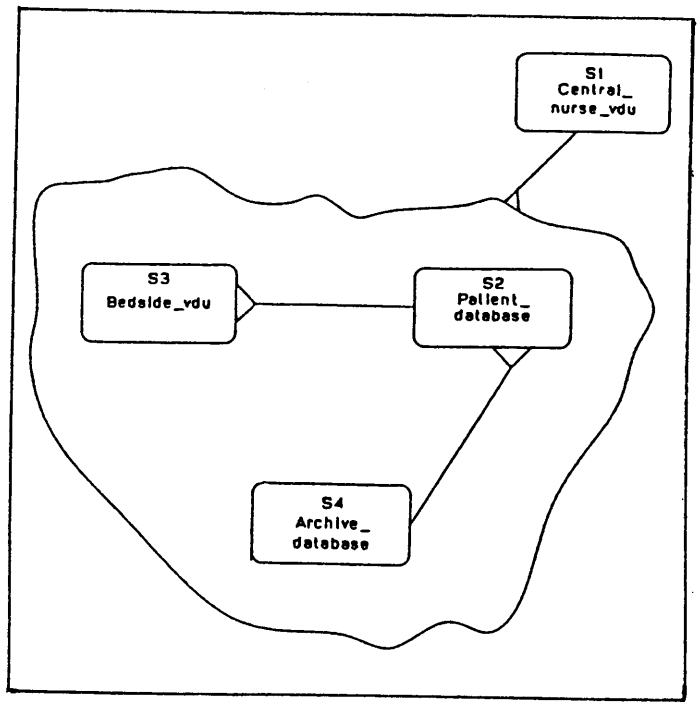


Exhibit 3 Syntax level diagram of data level profile

```
DEFINE DEVELOPMENT-ENVIRONMENT Intensive_care_environment;
  SYNONYM Ice;
  DESCRIPTION;
  This object is the environment outside the monitoring
  environment and has the following subparts;
  SUBPARTS ARE Monitoring_environment,
               Operational_environment;

DEFINE PROJECT Monitoring_software_project;
  SYNONYM Proj;
  DESCRIPTION;
  This is the name of the project that
  represents the software for the monitoring
  of patients;

DEFINE SYSTEM Intensive_care_system;
  SYNONYM Ics;
  DESCRIPTION;
  This system is part of the Monitoring_software_
  project that contains two subsystems: Patient_
  monitor_subsystem and the Intensive_care_
  equipment subsystem;
  SUBPARTS ARE Patient_monitor_subsystem,
           Intensive_care_equipment_subsystem;

DEFINE SUBSYSTEM Patient_monitor_subsystem;
  SYNONYM Pms_sub;
  DESCRIPTION;
  This is a subsystem of the whole monitoring
  environment and the operational environment and
  it has four objects or subparts identified by
  the first step of the methodology;
  SUBPARTS ARE S1, S2, S3, S4;

DEFINE SUBPROGRAM Central_nurse_vdu;
  SYNONYM S1;
  DESCRIPTION;
  This is the main or root procedure of the system
  It controls the operation of the system. 'Imports'
  indicates its dependency on the ohter high-level
  objects;
  IMPORTS S2, S3, S4;
  USING   S2, S3, S4;
  SUBPGM-TYPE 'Procedure';

DEFINE PACKAGE Patient_database;
  SYN S2;
  DESCRIPTION;
  This object is concerend with all the
  operations performed on the patient database.
  The patient database is updated frequently
  using the bedside datapool;
  IMPORTS S3;
  USING S3;

DEFINE PACKAGE Bedsid_vdu;
  SYNONYM S3;
  DESCRIPTION;
  This object monitors the patient condition and
  all the relevant operations of the bedside terminal
  and its associated data pool;
  INSTANTIATES Generic_bedside_vdu;

DEFINE PACKAGE Archive_database;
  SYNONYM S4;
  DESCRIPTION;
  This object concerns all the activities related to
  the archive information. The updating of the
  archive database as well as production of reports;
  IMPORTS S2;
  USING S2;
```



The problem space of the Patient Monitoring System



Unit level profile for the Patient Monitoring System

Exhibit 4 Requirements specification in SALA - Part I

```
DEFINE PACKAGE Monitor_patient_condition;
   SYNONYM S3-2;
   SUBPARTS ARE Handle_treatment, Handle_drugs,
      Handle_commands, Compare_readings_and_set_values;
DEFINE TASK Compare_readings_and_set_values;
DEFINE TASK Handle_commands;
DEFINE TASK Handle_drugs;

DEFINE TASK Handle_treatment;
   SYNONYM S3-2.1;
   DESCRIPTION;
   This object it a Task which handles all treatment
   details for a specific Bed_number in Intensive care;
   SPECIFICATION IS Bed_number, Treatment_text, Time_treat,
   Treatment_given_input, Change_treatment, Prompt_treatment;
   PARAMETERS ARE Bed_number, Treatment_text, Time_treat;
   IS ASSOCIATED WITH Bedside_pool_file;
   UPDATES Bedside_pool_file;
   CAUSES Prompt_treatment WHEN Cond_treat;
   ACCEPTS Treat-input, Change_treatment,
         Query_treat_schedule;
   RECEIVES Treatment_given_input, Change_treatment;
   GENERATES Treatment_data, Prompt_treatment;
   BODYCODE;
    ------ this section contains the body part associated
           with the treatment task.
   Treatment_given_input, Change_treatment : treatment_text
   begin
   loop
     select accept Treatment_given_input(Treatment_text,
                                         Time_treat)
     ----     store the prescribed treatment
           end Treatment_given_input
     or
     select Change_treatment (Treatment_text, Time_treat)
     ---     etc.
   end Handle_treatment;
   DEFINE VARIABLE Bed_number;
     PARAM-MODE 'in';
     TYPE IS 1;
   DEFINE VARIABLE Treatment_text;
     PARAM-MODE 'in';
     TYPE IS 'Penicillen intravenous';
   DEFINE VARIABLE Time_treat;
     PARAM-MODE 'in';
     TYPE IS '8 hourly';
   DEFINE FILE Bedside_pool_file;
   DEFINE EVENT Prompt_treat;
   DEFINE CONDITION Cond_treat;
   DEFINE ENTRY Treatment_given_input;
   DEFINE ENTRY Change_treatment;
   DEFINE ENTRY Query_treat_schedule;
   DEFINE INTERFACE Nurse_monitor;
     CAUSES Treatment_alarm;
```

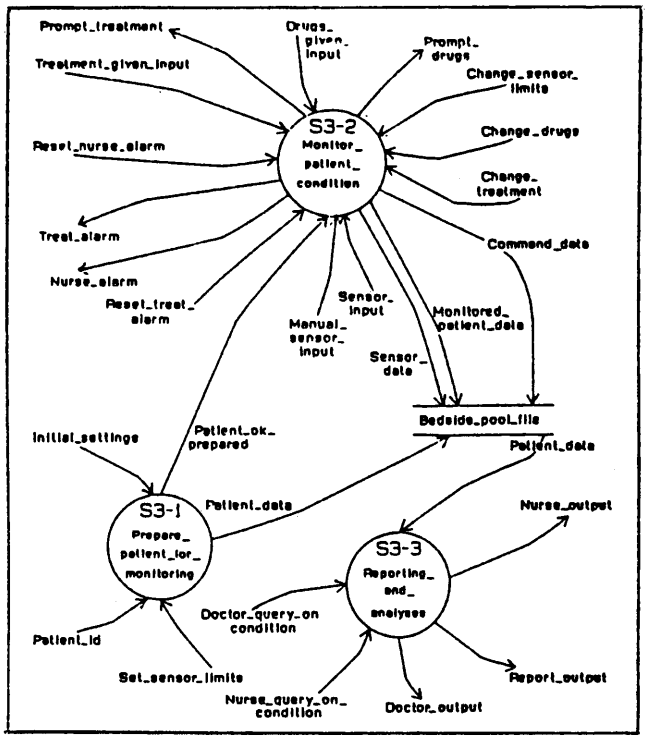
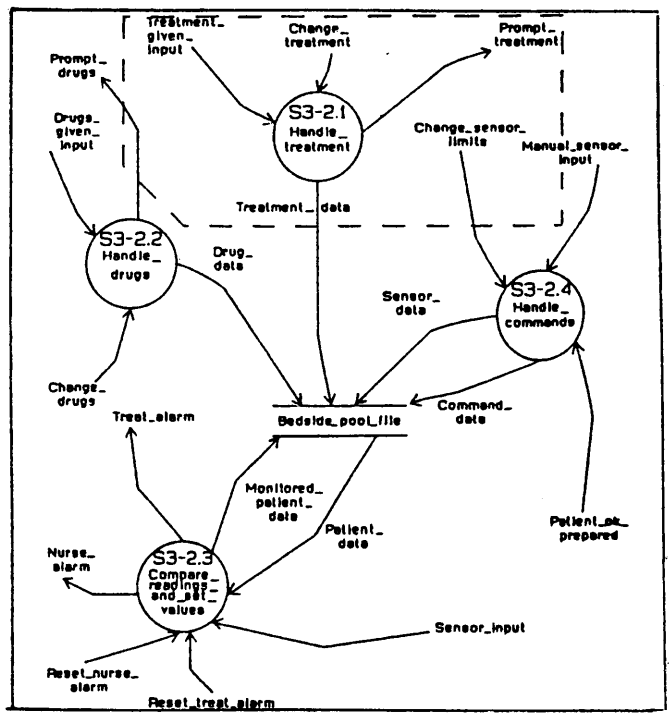


Diagram 0 of Bedside_VDU process



Level 1 diagram of the Monitor_patient_condition process

Exhibit 5 Requirements specification in SALA - Part II

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 a 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 double-space typing on one side only of A4 paper, and provide wide margins.
- 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 on separate sheets of A4 paper, and should be numbered and titled. Figures should be submitted as original line drawings, and not photocopies.
- 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.
- 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. References should thus take the following form:

  [1] E Ashcroft and Z Manna, [1972], The translation of 'GOTO' programs to 'WHILE' programs, *Proceedings of IFIP Congress 71*, North-Holland, Amsterdam, 250-255.

  [2] C Bohm and G Jacopini, [1966], Flow diagrams, Turing machines and languages with only two formation rules, *Comm. ACM*, **9**, 366-371.

  [3] S Ginsburg, [1966], *Mathematical theory of context free languages,* McGraw Hill, New York.

Manuscripts *accepted* for publication should comply with the above guidelines, and may be provided in one of the following formats:

- in a **typed form** (i.e. suitable for scanning);
- as an **ASCII file** on diskette; or
- as a **WordPerfect**, **TEX** or **LATEX** or file; or
- in **camera-ready** format.

A page specification is available on request from the editor, for authors wishing to provide camera-ready copies. A styles file is available from the editor for Wordperfect, **TEX** or **LATEX** documents.

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 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 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 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.

South African Computer Journal

Number 4, March 1991
ISSN 1015-7999

Suid-Afrikaanse Rekenaar-tydskrif

Nommer 4, Maart 1991
ISSN 1015-7999

Contents