Special Issue: SAICSIT '99
Contents

Preface
P. Machanick ................................................................................................. 1

Research Articles
Active Learning: Issues and Challenges for Information Systems and Technology
RD Quilling, GJ Erwin and O Petkova .......................................................... 5

A Generic Modelling Framework for Interactive Authoring Support Environments
Paula Kotze .................................................................................................... 15

O Petkova and JD Roode .............................................................................. 26

An Information-Theoretic Semantics for Belief Change
T Meyer .......................................................................................................... 33

A Complexity Metrics Model for Software Correction
A Törn, T Andersson and K Enholm ............................................................ 40

A Conceptual Design for High-Volume Data Processing of Warehouse Database into Multidimensional Database
Paisarn Trakulsuk and Vichit Avatchanakorn ............................................. 49

A Pragmatic Approach to Bitemporal Databases: Conceptualization, Representation and Visualisation
Chiyaba Njovu and WA Gray ........................................................................ 58

A Building Recognition System
SP Levitt and B Dwolatzky ........................................................................ 68

Computer Programming and Learning to Write
John Barrow .................................................................................................. 77

Co-operating to Learn using JAD Technologies
TA Thomas .................................................................................................... 87

Critical Success Factors for the Implementation of DSS at a Selection of Organisations in Kwazulu/Natal
URF Averweg and GJ Erwin ....................................................................... 95

Enhancing the Predictability of Two Popular Software Reliability Growth Models
Peter A Keiller and Thomas A Muzzuchi .................................................. 105
Generalised Unification of Finite Temporal Logic Formulas
Scott Hazelhurst ................................................. 110

Harmonizing Global Internet Tax: A Collaborative Extranet Model
E Lawrence and B Garner ..................................... 119

Improving Object Oriented Analysis by Explicit Change Analysis
Lui Yu, Siew Chee Kong, Yi Xun and Miao Yuan ............. 128

Reconciling the Needs of New Information Systems Graduates and Their Employers in Small, Developed Countries
Rodney Turner and Glenn Lowry ............................... 136

Shortest Delay Scheduling Algorithm for Lossless Quality Transmission of Stored VBR Video under Limited Bandwidth
Fei Li, Yan Liu, Jack Yiu-Bun Lee and Ishfaq Ahmad ......... 146

Software Croma Keying in an Immersive Virtual Environment
Frans van der Berg and Vali Lalioti .......................... 155

Some Automata-Theoretic Properties of $\cap$-NFA
Lynette van Zijl and Andries PJ van der Walt ............... 163

Statistical Analysis of an Automated Computer Architecture Learning Environment
JT Waldron, J Horgan and G Keogh ........................... 168

The CILT Multi-Agent Learning System
Herna L Viktor .................................................. 176

The Development of a Generic Framework for the Implementation of Cheap, Component-Based Virtual Video-Conferencing System
Soteri Panagou and Shaun Bangay ............................ 185

The Role of Experience in User Perceptions of Information Technology: An Empirical Examination
Meliha Handzic and Graham Low .............................. 194

What are Web Sites Used for: Cost Savings, Revenue Generating or Value Creating?
Man- Ying Lee .................................................... 201

New Ideas Papers

Approaches to Video Transmission over GSM Networks
Bing Du and Anthony Maeder .................................. 210

From Information Security Baselines to Information Security Profiles
Rossouw von Solms and Helen van der Haar .................. 215

Grounded Theory Methodology in IS Research: Glaser versus Strauss
J Smit ................................................................ 219

Introducing a Continuum of Abstraction-Led Hierarchical Search Techniques
Robert Zimmer and Robert Holte ............................... 223
Multimedia as a Positive Force to Leverage Web Marketing, with Particular Reference to the Commercial Sector

**Stan Shear** .......................................................... 229

Understanding HCI Methodologies

**Peter Warren** .......................................................... 234

Electronically Published Papers


**Experience Papers**

A Java Client/Server System for Accessing Arbitrary CANopen Fieldbus Devices via the Internet

**Dieter Bühler, Gerd Nusser, Gerhard Gruhler and Wolfgang Küchlin** ......................... 239

An Object-Oriented Framework for Rapid Client-side Integration of Information Management Systems

**Ralf-Dieter Schimkat, Wolfgang Küchlin and Rainer Krautter** ................................ 244

Distributed Operating Systems: A Study in Applicability

**Jürgen Prange and Judith Bishop** ........................................ 249

Formal Verification with Natural Language Specifications: Guidelines, Experiments and Lessons so far

**Alexander Holt** .......................................................... 253

Introducing Research Methods to Computer Science Honours Students

**Vashti Galpin, Scott Hazelhurst, Conrad Mueller and Ian Sanders** ......................... 258

Visualising Eventuality Structure

**ST Rock** ................................................................. 264

Electronically Published Papers

Prepare for the New Millennium

Is there life after y2k?

Mount Amanzi Lodge, Hartebeespoort, South Africa

Sponsors

http://www.cs.wits.ac.za/

Think different.
Preface

Philip Machanick, Overall Chair: SAICSIT’99

Running SAICSIT’99, the annual research conference of the South African Institute for Computer Scientists and Information Technologists, has been quite an experience.

SAICSIT represents Computer Science and Information Systems academics and professionals, mainly those with an interest in research. When I took over as SAICSIT president at the end of 1998, the conference had not previously been run as an international event. I decided that South African academics had enough international contacts to put together an international programme committee, and a South African conference would be of interest to the rest of the world.

I felt that we could make this transition at relatively low cost, given that we could advertise via mailing lists, and encourage electronic submission of papers (to reduce costs of redistributing papers for review).

The first prediction turned out to be correct, and we were able to put together a strong programme committee.

As a result, we had an unprecedented flood of papers: 100 submitted from 21 countries. As papers started to come in, it became apparent that we needed more reviewers. It was then that the value of the combination of old-fashioned networking (people who know people) and new-fashioned networking (the Internet) became apparent. While the Internet made it possible to convert SAICSIT into an international event at relatively low cost, the unexpected number of papers made it essential to find many additional reviewers on short notice. Without the speed of e-mail to track people down and to distribute papers for review, the review process would have taken weeks longer, and it would have been much more difficult to track down as many new reviewers in so little time.

Even so, the number of referees who were willing to help on short notice was a pleasant surprise.

The accepted papers cover an interesting range of subjects, from management-interest Information Systems, to theoretical Computer Science, with subjects including database, Java, temporal logic and implications of e-commerce for tax.

In addition, we were very fortunate in being able to invite the president of the ACM, Barbara Simons as a keynote speaker. Consequently, the programme for SAICSIT’99 should be very interesting to a wide range of participants.

We were only able to find place in the proceedings for 36 papers out of the 100 submitted, of which only 24 are full research papers. While this number of papers is in line with our expectation of how many papers would be accepted in each category, we did not have a hard cut-off on the number of papers, but accepted all papers which were good enough, based on the reviews. Final selection was made by myself as Programme Chair, and Derrick Kourie, as editor of the South African Computer Journal. Additional papers are published via the conference web site.

We believe that we have put together a quality programme, and hope you will agree.

Acknowledgments

I would like to thank the South African Computer Journal production team, Andries Engelbrecht and Herna Viktor, respectively from the Department of Computer Science and Informatics, University of Pretoria, for their work on producing the proceedings.

The reviewers listed overleaf did an excellent job: many wrote very detailed reports, sometimes after being called in on very short notice. Inevitably, there were some glitches resulting from the unexpected workload, but the buck stops with the programme chair: I promise to do better next time.

I would also like to thank my own department for putting up with the extra work and expense that running a conference entails. I tried not to burden them with too much extra work, but our secretaries, Zalm Gowar and Leanne Reddy, inevitably had to take on some extra work. John Ostrowick provided valuable assistance with design of our web pages and call for papers poster. Carol Kernick, who handles our finances and membership records, did a fine job of keeping up with the demands of the conference.

Finally, I would like to thank our sponsors, whose contribution made this conference been possible:

- PricewaterhouseCoopers – sponsored generous prizes and the conference banquet
- National Research Foundation (NRF) – provided financial support
- University of the Witwatersrand – provided financial support
- Programme for Highly Dependable Systems, University of the Witwatersrand – provided financial support
- Standard Bank – provided financial support
Editorial

• Apple Computer – provided equipment for the conference
• Qualica – provided technical support including helping with the conference web site

Web Site

For more information about SAICSIT, including a pointer to the conference site, see <http://www.saicsit.org.za>.

Referees

• Department of Computer Science, University of Pretoria
  – Derrick Kourie
  – Bruce Watson
  – Vali Lalioti
  – Andries Engelbrecht
  – Ivan Mphahlele

• German National Research Center for Information Technology - GMD
  – Gernot Goebbels

• School of Math Stat and IT, Natal University Pietermaritzburg
  – Peter Warren

• Graduate School of Business, University of Cape Town
  – Kurt April

• CSIR
  – James Jardine

• Department of Computer Science, Rhodes University
  – Shaun Bangay
  – Peter Clayton
  – John Ebden
  – Richard Foss
  – George Wells
  – Peter Wentworth

• Information Technology Division, Rhodes University
  – Caro Watkins

• Department of Information Systems, Rhodes University
  – Brenda Mallinson
  – Dave Sewry

• Department of Informatics, University of Pretoria
  – Carina de Villiers
  – Herna Viktor
  – Niek du Plooy
  – Elsje van Rooyen
  – Machdel Matthee
  – Alan Abrahams
  – Jackie Phahlamohlaka

• School of Information Technology & Engineering, University of Ottawa
  – Stan Szpakowicz
  – Dwight Makaroff

• Department of Information Systems, Victoria University
  – Glenn Lowry
  – Peter Shackleton
  – Tas Adam
  – Alastair Wallace
  – Stephen Burgess
  – Dave Burgess
  – Julie Fisher
  – Jerzy Lepa
  – Geoff Sandy
  – Rod Turner
  – Alastair Wallace
  – Andrew Wenn

• Peninsula School of Computing and Information Technology, Monash University
  – Ainslie Ellis

• School of Information Management and Systems, Monash University
  – Angela Carbone

• School of Computer Science & Software Engineering, Monash University
  – Dianne Hagan
  – Judy Sheard

• School of Management Information Systems, Deakin University
  – John Lamp
  – David Mackay
  – Philip Joyce
- Marie van der Klooster
- Bill Hewett
- Jennie Carroll
- Rodney Carr
- Computer and Automation Research Institute, Hungarian Academy of Sciences
  - Ferenc Vajda
- Department of Computer Science, University of the Witwatersrand
  - Ian Sanders
  - David Lubinsky
  - Conrad Mueller
  - Yinong Chen
  - Bob Baber
  - Vashti Galpin
  - András Salamon
  - Scott Hazelhurst
  - Philip Machanick
  - Zoltán Fazekas
- Department of Electrical Engineering, University of the Witwatersrand
  - Barry Dwolatzky
  - Farzin Aghdasi
- Department of Mathematics, University of the Witwatersrand
  - Jonathan Burgess
- School of Law, University of the Witwatersrand
  - Victoria Bronstein
- Edward Nathan & Friedland Attorneys, Johannesburg
  - Justine White
- Department of Information Systems & Technology, University of Durban-Westville
  - Geoff Erwin
- Department of Computer Science, Clemson University, Clemson, SC
  - Karen Hay
- Logical SA
  - Philip Green
- Department of Computer Science, University of Bristol
  - Alan Chalmers
- Department of Systems and Computer Science, Howard University, Washington, D.C
  - Todd Shurn
  - John Trimble
- Department of Computer Science, University of Cape Town
  - Sonia Berman
- King’s College, London
  - Peter Wood
- Department of Information Systems, University of Cape Town
  - Paul Licker
  - Steve Erlank
  - Alemayehu Molla
- Department of Department of Mathematics and Applied Mathematics, University of Cape Town
  - Ingrid Rewitzky
  - Peter Jipsen
  - Renkuan Guo
- Department of Information Systems, University of the Western Cape
  - Andy Bytheway
- Department of Electrical Engineering and Computer Science, The George Washington University, Washington, DC
  - Rachelle Heller
- School of Biophysical Sciences and Electrical Engineering, Swinburne University of Technology
  - Ian Macdonald
- Electrical and Electronic Systems Engineering, Queensland University of Technology
  - Anthony Maeder
- Computer Science Department, Åbo Akademi
  - Aimo Törn
  - Kaisa Sere
  - Mats Aspnas
- Division of Informatics, University of Edinburgh
  - Graeme Ritchie
  - Arturo Espinosa-Romero
  - Jon Oberlander
Editorial

- Chris Brew
- Alexander Holt

• Department of Computer Science and Engineering, Helsinki University of Technology
  - Tommi Junnila
  - Nisse Husberg

• Technical University of Budapest, Department of Measurement and Information Systems
  - Andras Pataricza

• NASA-Goddard Space Flight Center
  - Nigel Ziyad

• Department of Computer Science & Information Systems, University of Port Elizabeth
  - Janet Wesson
  - Leon Nicholls

• Computer Technology Department, Indiana University Purdue University Indianapolis
  - Tim Price

• Department of Electrical Engineering and Computer Science, The University of Michigan
  - Trevor Mudge

• Department of Electrical Engineering and Computer Sciences, University of California, Berkeley
  - David Forsyth

• School of Computer Science, University of Birmingham
  - Mark Ryan

• Faculty of Mathematics, Computer Science, Physics & Astronomy, University of Amsterdam
  - Carlos Areces

• Department of Computer Science, Universidade do Vale do Rio dos Sinos - UNISINOS Rio Grande do Sul
  - Marcelo Walter

• School of Information Technology & Mathematical Sciences, University of Ballarat
  - Binh Pham
A Generic Modelling Framework for Interactive Authoring Support Environments

Paula Kotze

Department of Computer Science, University of South Africa, Pretoria, South Africa, kotzep@alpha.unisa.ac.za

Abstract

Existing modelling frameworks for the design of interactive systems emphasise the relationship between the system’s internal state and the rendering of this state at the user interface. Modern programming and multimedia design tools do, however, involve a critical area not covered by these models — they are used to create another interactive system which has its own internal state and its own user interface when executed. This paper proposes a framework that can be used to model these types of systems. The framework focusses on the relationship between the state of the development system and the state of the delivered product. Interactive authoring support environments aimed at the development of computer-based instructional software are used as case study example.

Keywords: Formal modelling, interactive systems, generic models, usability, authoring support environments.

Computing Review Categories: D.2, H.1, H.5, I.7, K.3

1 Introduction

Software reuse traditionally has meant the reuse of code, extracted from subroutine libraries. In recent years there has been an increasing interest in domain analysis and domain model reuse, extending the scope of reusability to earlier in the development life cycle. At earlier stages the developer is able to look at the structure of a component without it being masked by implementation details. Such models typically employ mathematical notations in order to express selected parts precisely and concisely. Storing components in simple generic form extends the range of requirements satisfied by a component and can therefore be more widely applied. Such models can be used to rapidly prototype and render the user interface of an application, and also to reason about its usability attributes.

This paper focusses on domain-specific design of the user interface components for a specific class of interactive visual programming environments and multimedia development systems, namely interactive authoring support environments (ASEs). ASEs are widely used for the development of computer-based instructional (CBI) systems and other types of presentations, such as DVD discs with embedded controls.

Domain-specific design for this class of interactive systems requires a high correspondence between design concepts in the application domain and the interface components to enable efficient implementation to take place. The domain model proposed in this paper therefore consists of two parts: the application framework modelling the products of the ASE and the interface framework to model the process of authoring.

The proposed modelling framework uses a combination of formal frameworks exemplifying functional aspects and frameworks focussing on interactive issues, all based on the Z notation to ensure compatibility. The framework represents the first attempt at formal domain modelling for the specified class of interactive systems.

2 Choosing a modelling approach

The usefulness of abstract models in comprehending the complex composition of interactive systems has been widely illustrated in past research. Examples of such models include:

- Frameworks exemplifying the functional aspects of interactive systems, such as the operational approach including model-based specification using Z [12], VDM [8], etc. and process-based specification such as CSP [7], and the axiomatic approach or property-based specification such as OBJ [5].

- Frameworks exemplifying the interactive properties of interactive systems, for example non-formal frameworks such as Abowd and Beale’s interaction framework [1], as well as abstract mathematical frameworks such as Dix’s red-PIE framework [2], and Sufrin and He’s framework [13].

- Specification frameworks including the interactor notation [4], ICO [10], GRAPLA [14], etc.

These modelling frameworks for the design of interactive systems emphasise the relationship between the system’s internal state and the rendering of this state at the user interface. Modern programming and multimedia design tools do, however, involve a critical area not covered by these models. These systems are used to create another interactive system which has its own internal state and its
own user interface when executed. This paper proposes and illustrates a two-phase method to address this problem.

In order to model ASEs, the first phase involves the development of a generic framework for modelling CBI systems — the products of ASEs. A model of this kind can be expressed using the frameworks exemplifying functional aspects. The choice of specification notation and approach depends on the type of functional properties of the system to be specified. Model-based methods focus on the data paradigm and the system state model, while process-based methods focus on the communication between different processes within a system as well as communication between the system and its environment. The former is more applicable to modelling CBI and multimedia systems [9].

Using the application framework as a basis, the next step involves the development a design framework for the interactive system itself. In choosing an appropriate modelling framework one needs to examine the tradeoffs between operationality, expressiveness, and reusability of such frameworks.

Non-formal frameworks are generally considered to be non-operational; thus their efficiency, reliability, and generalizability cannot be guaranteed. General mathematical frameworks for the specification of interactive systems, on the other hand, are limited in expressing interaction properties specific to the requirements of a particular domain, and may also sacrifice operationality. To provide a generic framework for interactive ASEs, such frameworks will have to be refined and expanded to include aspects explicitly belonging to the domain of authoring CBI systems. System-specific specification frameworks are, in contrast, considered too specific to support wide reuse for classes of interactive systems. To provide a generic framework for the design of the interactive ASEs class, a more general type of specification would be required, avoiding implicit assumptions about a small number of systems.

A level of abstraction lying between the general mathematical frameworks and the system specific specification frameworks, could contribute to solving the problem of finding a generic framework for modelling aspects relating to a specific class of interactive ASEs.

3 Application framework

The basic data model for CBI applications is illustrated in Figure 1. A computer-based instruction environment consists of a set of courses on a certain topic, for example, a course on databases. Each course is subdivided into a number of lessons, for example a lesson on data modelling, a lesson on normalisation, etc. Lessons are ordered according to some prerequisites. Each lesson consists of a number of related and integrated networks, for example a central network carrying the main instructional activities connected to a network containing remedial or enrichment material, a network representing a glossary of terms, and a network containing on-line help. A network consists of a set of nodes connected by means of links. Nodes carry the actual material used for instruction. We argue that the functional characteristics of ASEs should reflect all these levels of a CBI environment.

The lowest level includes node or instructional content creation. It refers to the input, formatting, and modification of media objects carrying instructional content and any other information to be displayed or stored in relation to a particular node (such as student-response processing and branching sequences). The second level refers to network definition. It involves functions specifying the structure and behaviour of the individual networks of which a lesson consists, based on the principles of the instructional strategy they represent. The third level refers to lesson definition and includes functions which involve specifying the combination of a number of related instructional networks to act as a unit. Lesson management functions include the capability to define or select a particular instructional strategy (test, tutorial, simulation, etc.), the collection of response data, the control options available to a student, etc. The effect of these functions is local to a specific lesson.

Global effects across lessons are specified at the course level. Fundamentally, operations at this level allow authors to manage the authoring and instructional process for a whole course under development. The records accumulated for each lesson in a particular course are integrated and provide a very detailed trace of a student’s progress through a particular course.

An object-oriented model-based approach using the Z notation is used to model the application framework. All objects are modelled using the same basic structure. Each object has a unique identifier, belongs to a particular type, and has a set of content material associated. Object content represents the typical content material associated with the object under consideration and differs between object types. The unique identifiers are used during the authoring process to retrieve component objects.

The part of the framework referring to the definition of a node is used as example to illustrate the modelling approach. The complete framework can be found in [9].
The African Elephant

Question:
About twice the circumference of the front foot of the African Elephant gives the approximate height of the elephant.

True [!] False [ ]

3.1 Node objects

Nodes are organised, functionally defined units of information and activities, representing the instructional content of CBI systems. Instructional content is subdivided into manageable chunks of information, each representing some instructional transaction (event). Each node is identified by means of a unique node identifier, values of which are chosen from a given set (NODE_ID). Figure 2 illustrates an example of a CBI node with node_id 'QUES12'. Furthermore, each node also has a type and a content:

| Node | node_id : NODE_ID
|      | Node_Type
|      | Node_Content |

Node types

Node types are used to assist authors and students to draw semantic distinctions between different kinds of nodes in a CBI network. For this reason, distinctive representations of the various types at the user interface of both the ASE as well as the resulting CBI system, are especially important. The views and use of types most relevant to nodes are those of type classification, control over access, the display orientation, presentation, and data gathering [9]. The data gathering aspect is handled by node monitors. Four typed partitions are therefore identified to form the node type:

| Node | node_type : NODE_TYPE
|      | orientation : ORIENTATION
|      | controlled_by : CONTROL
|      | presentation : PRESENTATION |

For example, a tutorial node may in some cases occupy a window larger than the standard screen display, be interactive in nature by allowing the student to manipulate some of the media objects forming part of the node, but be inactive in that the student must tackle fixed content. Another instance of a tutorial node might be a dynamic, animated, full screen display, controlled by the system. Figure 2 illustrates an example of a static, system-controlled, frame-based, question node being developed.

Node content

Node content refers to the actual, instructional, subject-content material a student will receive, as well as certain hidden control information for monitoring progress through the system.

The instructional content, comprising the material displayed to the student, is in the form of multimedia objects. Each multimedia object is of a particular type, for example text, graphic, video, animation, etc. Each media type is described in terms of a number of attributes, to which values can be assigned when authoring a media instance. The instructional content of the node illustrated in Figure 2 consists of the text objects, question answer options, and the bitmap of the elephant (displayed above the line containing the ◊s).

To create effective instructional and training material, one needs some special features to assess student progress. These features are implemented by means of node_monitors. This information is 'concealed' within the node content and the student is usually unaware of its existence. Node monitors are used for keeping student records as well as determining which path to follow through the CBI network, and change dynamically as a result of a student's interaction with the instructional system. The node monitors of the node illustrated in Figure 2 refers to all the text lines starting with a ◊.

The node content is seen as a linear ordering of the disjoint data types Media and Monitors:

Node_Content_Type ::= med((Media)) | mon((Monitor))

| Node | node_content : seq Node_Content_Type
|      | instructional_content : seq Media
|      | node_monitors : seq Monitor |

instructional_content = node_content;  
{ ic : P Node_Content_Type |  
ic = ran(node_content > med) }  
node_monitors = node_content;  
{ nm : P Node_Content_Type |  
nm = ran(node_content > mon) }

Node instances

Finally, ASEs keep a central repository of all addressable objects in the CBI system under construction:

| Nodes | nodes : NODE_ID → Nodes |
|       | ∀ nid : NODE_ID |  
nid ∈ dom(notes) |  
(node_instances(nid)).node_id = nid |
Media, Monitors, Networks, Lessons, and Courses require similar detailed specifications identifying their unique attributes. Each of these object types also requires the specification of the set of events or operations in which they may participate.

4 Interface framework

Most of the features discussed thus far dealt with the concepts and requirements of the output of an ASE, namely the actual multimedia instructional system on its different composite levels. However, an ASE is an application itself. The communication between author and system during the development of the CBI material also needs to be considered.

For the application framework an object-oriented, model-based approach was used to model the essential components of CBI systems. Despite the fact that object-oriented, user interface technology has been touted as a major step towards increasing design and programming productivity for modern interfaces, since it clearly decomposes component structure and functionality, object-oriented interface implementations fail to address several key issues. They neither directly represent users' tasks nor map users' intentions to application procedures. Nor do they allow one to specify static and behavioural elements in a modular way that would facilitate their reuse.

On the other hand, task-oriented interface specification in which task relationships, not just the task itself, are captured, is an effective way to represent user requirements, but there are few strategies for implementing this approach. This lack of research has resulted in limited knowledge about how to identify the content of each interface component and how to keep component functions separate. The knowledge required in such implementations for dialogue control is very complex and intermixed with application and presentation aspects.

Thus the ideal interface design approach should provide the modeling ease of an object-oriented implementation, yet simultaneously preserving task relationships and explicitly representing them as constraints. The proposed framework aims to fulfill this need.

4.1 Context

Tasks are operations to manipulate the concepts of a domain, while a goal is the desired output from a performed task [3]. An intention is the specific action required to meet the goal. Generally, an ASE can be defined as comprising a number of goal-directed processes and a number of events driving these processes. The goal of the processes and events is to produce an operational CBI system.

Task analysis is the identification of the problem space for the user of an interactive system, in terms of the domain, goals, intentions, and tasks [3]. The original intention behind task analysis was to capture what had to be known about using a system to achieve goals. Some of these activities may involve aspects that are not, and are not expected to be, part of the computer system under design or development.

In contrast to this user-oriented approach to tasks, a task in the system-oriented approach can be defined in terms of accesses to the functionality of the interactive system in order to determine or modify its state. According to this approach, tasks depend on the part of the interactive system to which they refer [11]:

- Presentation-related tasks do not require access to the functional core of the application and refer to the set of functionalities which are independent of the interaction objects — tasks such as selecting, resizing, or scrolling windows, selecting visualised objects, customizing dialogue boxes, etc.

- Application-related tasks require access to the functional core of the application and refer to parts of the interactive system which are independent of the general components used to interact with the user. In our domain, this refers to the general functionality of ASES.

The first of these aspects relates to the appearance of interaction objects, while the second relates to the application semantics. In developing the interface framework the focus is on the latter, yet without denigrating the appearance-related factors. Inappropriate use of windowing techniques, scrolling, and colours can result in tedious and confusing interaction. Detailed domain knowledge in the design of these is thus just as important [6].

4.2 The required interface model

Any fundamental model of interactive behaviour requires states and commands that transform states [3]. The abstract model of the system used here consists of a set of states, relationships between these states, as well as certain permitted operations.

Using an interactive system generally involves two state sets: a state set relating to the internal state of the system and a second relating to its external display. All the abstract frameworks types for interactive systems listed in section 2 are based on this assumption. In a detailed study

Figure 3: The relationships between the four state sets

Resulting CBI State (RESULT)  CBI Interface (CBI-DISPLAY)

Internal ASE State (STATE)

ASE Interface State (ASE-DISPLAY)

Key: Mappings Relationships
program elephant;
var
  x:array[1..500] of char;
  RV1:integer;
  name:string[40];
begin
  RV1:=0;
  name:='QUES12';
  setpal(3,0);
  clear;
  upimg(elephant.bmp);
  movgabs(40,20);
  setclr(0);  
  settxt(4,2,0,0);
  writeln('The African Elephant');
  x[RV1]:='A';
  y[RV1]:=RV1+1;
  x[RV1]=name;
end;

Figure 4: The CBI result state for the example in Figure 2

of the interactions involved in developing CBI material by means of an interactive ASE [9], it was revealed that there are actually four state subsets of interest when interacting with an ASE, as illustrated in Figure 3 — one for the internal state of the ASE, a second one representing the external displays of the ASE during authoring, a third representing the state of the resulting CBI system, and a fourth for the external display of the CBI system.

Our main interest lies in the relationship between the external ASE display and the CBI result (how the resulting CBI system is reflected at the ASE interface during the authoring process) and between the external ASE display and the CBI student display (how the student's perception of the resulting CBI system is reflected at the ASE interface during authoring).

The internal state (STATE), or system state, refers to the functional state of the ASE, including aspects such as the internal state of the CBI objects under development, as well as the variables needed to control the interaction with the author (for example the current object being edited, the templates and default values used for specifying attribute set values, etc.). The display state (ASE_DISPLAY) of an ASE refers to the external perceivable rendering of the internal state as reflected at the ASE interface.

The result state (RESULT) refers to the actual state of the CBI program under development — the program that will determine what the student will receive in the delivered product and the rules for its presentation. The result display state (CBI_DISPLAY) refers to what the student will perceive at the interface of the CBI system — i.e. instructional content, but not the rules governing the sequence in which the instructional content is displayed. Figures 4 and 5 illustrate the associated result state and the result display state of the example in Figure 2, respectively.

A general interaction model consisting of these four state subsets is used to demonstrate, in the first place, how the central functionality of an authoring support environment is reflected in the external display of the system, and secondly how the external display can be subdivided into different conformant subsets to facilitate the understanding of what the delivered CBI system would entail.

4.3 Applicable frameworks

All but two of the existing modelling frameworks for interactive systems mentioned in Section 2, have to do with the effects of interacting with a single system. Only the red-PIE [2] and Sufrin and He's [13] frameworks contain a reference to the result of an interaction. The result in the case of the red-PIE framework is, however, seen as a static object (like the result we get from using a word processor), not as a dynamic object. Also, over and above other general difficulties in specifying interactive systems, the red-PIE framework is too general to be applied directly to the specification of important principles relating to a specific types of interactive system. The result in the case of Sufrin and He's framework is undefined but can be developed to fit the application at hand. The examples they explore are, however, all non-interactive as well.

When considering specification frameworks, the interactors framework developed by Duke and Harrison [4], allows for the specification of global states subdivided into groups of components, each of which has interactive behaviour, to render possible the specification of interactive aspects. Each of these components is then modelled by means of an interactor that describes it state and behaviour. Attributes in an interactor are annotated by means of the symbols '{-}' and '@' to indicate whether the attribute will be visible or audible to the user, respectively.

This approach could directly be used to describe the relationships among the internal state and the display state using an ASE, in that a CBI environment can be subdivided into groups of components, each with attributes that are either hidden from, or perceivable by the author in the form of a visible display or in an audible format.

To indicate that an attribute of the internal state will appear in the resulting program of the result state, or which attributes will be perceivable by the author in the display state, but not by the student in the delivered product or result-display state, can be accomplished by adding more

Figure 5: The CBI display state for the example in Figure 2
annotation symbols, each indicating whether or not an attribute would occur in any of the state subsets, besides the fact that they would be audible, visible, or neither, in these different states. This could, however, result in a specification polluted with annotations without associated semantics.

Therefore, unless a totally new framework for specifying interactive systems such as ASEs is derived, modifications must be made to all of the existing frameworks in order use them to model interactive ASEs. One such a modification is presented here, based on a modified version of Sufrin and He's framework, and incorporating the idea of a global state subdivided into components as proposed by Duke and Harrison. This allows for the definition of states and behaviour for all the essential objects manipulated by ASEs. The decision to combine these approaches maintains as broad a reusability range as possible for the ASE framework and seeks to maximize its expressiveness.

4.4 General process framework

Authoring support environments are used to create and modify information structures, called 'objects'. The internal state of an object at any particular time is defined by the values of its components or attributes — that is the cumulative result of the defined node, link, network, lesson, and course information in the CBI system under development. In order to change the internal state of an object, the user issues commands to the system which are legal and applicable to the object under consideration. The general process framework for ASEs should therefore be able to reflect system states, state changes, and the effect of such changes.

In line with the Sufrin and He framework, systems operations are defined by means of state transformations. An operation can be seen as a process involving a particular state set and events in which the state can participate. Operations on a state set are triggered by valid events that form part of the alphabet of the system. The alphabet of the system is formed from the set of all events allowed in the system.

Two types of events exist. Firstly, an interactive process is driven or initiated by a user. The user is required to initiate an operation and must convey information to the computer concerning the actions he wants taken. Secondly, the interactive system must convey information to the user regarding required actions or events needed for the successful completion of a particular operation. The set of valid user-initiated events is called commands and the set of valid system-initiated events prompts. The set of traces of a process is the set of all possible sequences of events in which a process can legally participate.

The interpretation of an event sequence is denoted by a function interpret from alphabet to the set of state transformations. State transformations are defined as functions on a state set. ASEs are required to be deterministic systems and therefore require the behaviour function to be deterministic in mapping each event to a function between states, rather than a relation as defined by Sufrin and He.

The relation render represents the mapping between the internal system state and the external display state of the ASE, while the relation result represents the mapping between the internal state and the resulting CBI system. We also have to represent the mapping between this resulting CBI system and what the student perceives in the delivered product. We have added a relation result render to the original Sufrin and He framework representing this mapping between the resulting CBI system and the CBI display. The CBI display represents what the student perceives at the interface of the CBI system, at a given point, based on a particular set of performance and control data.

To summarize:

\[
\begin{align*}
\text{ASE Process} & : \text{STATE, EVENTS, RESULT,} \\
& \quad \text{ASE DISPLAY, CBI DISPLAY} \\
\text{initial} & : \mathbb{P} \text{STATE} \\
\text{alphabet} & : \mathbb{P} \text{EVENTS} \\
\text{interpret} : \text{EVENTS} & \rightarrow (\text{STATE} \rightarrow \text{STATE}) \\
\text{commands} & : \mathbb{P} \text{EVENTS} \\
\text{prompts} & : \mathbb{P} \text{EVENTS} \\
\text{traces} & : \mathbb{P} (\text{seq} \text{EVENTS}) \\
\text{render} : \text{STATE} & \rightarrow \text{ASE DISPLAY} \\
\text{result} & : \text{STATE} \rightarrow \text{RESULT} \\
\text{result render} & : \text{RESULT} \rightarrow \text{CBI DISPLAY} \\
\end{align*}
\]

Given this core model of interactive behavior, we are able to define the state sets applicable to the use of ASEs, define operations on these state sets, and discuss different principles relating to the interactive behavior specific to ASEs, which can be used to form the basis for considering the options in design. We will first define the four state sets \text{STATE}, \text{ASE DISPLAY}, \text{RESULT}, and \text{CBI DISPLAY}. This will be followed by showing how typical ASE operations and usability principles directly related to CBI authoring, can be defined using these four state sets.

4.5 The internal state space

We have defined the content and structure of the typical CBI components (objects) in section 3. An ASE creates and manipulates these object structures and translates their attributes and associated values into a computer-executable program.

The internal state set (\text{STATE}) of an ASE should represent all the attributes relating to the CBI objects under consideration. It should also contain attributes essential to the proper functioning and control of the ASE, for example, attributes identifying the current object under consid-
eration, the cursor/pointer position, the content of dialogue boxes, etc.

For example, when nodes are being compiled for a specific application, the ASE should be aware of the type of node involved, the typical characteristic attributes of this node type, and the media and monitor objects of which the node content is composed. The ASE should also keep a record of which node content object is currently being manipulated. The state of a node can therefore be defined as a tuple consisting of the information related specifically to the Nodes object type (see section 3.1), together with the attributes essential to the central functionality of the system (such as the current object being edited):

\[
\text{Node\_State} = \langle \text{Nodes}, \\text{Nodes\_that\_exist}, \\text{current\_cursor} : \mathbb{N} \rangle \\
\text{current\_cursor} \in \text{dom(\text{Node\_Content}\_node\_content)}
\]

The complete state set, STATE, should be able to handle all of the components applicable to typical CBI environments, as identified in section 3. This set STATE is mapped at the ASE interface to the set ASE\_DISPLAY representing the external renderings of the ASE, as well as to the set RESULT representing the resulting CBI program under development.

4.6 The result and the result display state spaces

The result state space represents the CBI program the ASE would produce as a result of the author's interaction with the ASE. The format this resulting program can take, differs considerably between different ASEs. Some deliver the program in a high-level programming or authoring language, which can be edited afterwards using a standard editor, while others produce a program in encoded format specific to the ASE involved. The result state includes the whole of a completed CBI system, i.e. all the possible options available for traversal. The result state space therefore represents what the student could receive (but not necessarily perceive) in the delivered CBI product.

The result state of an ASE is composed of a set of states (RESULT) representing the resulting CBI system. The objects we have identified in section 3 and on which we based our definition of STATE above, effectively represent the objects and attributes that determine the variables and values, as well as the control structures, present in the result state space. The RESULT state related to node objects is, for example, defined as:

\[
\text{Node\_Result} \equiv \text{Nodes}
\]

The result display state space (CBI\_DISPLAY) has to do with what the student will actually perceive at the interface at any point during the traversal of the CBI system. It effectively portrays only a partial, differentiated view of both the node content (hiding the node monitors for example) as well as the content and structure of the networks making up the CBI system. Since what the student will perceive is completely dependent on the actual application, and the student's performance in this application, the result display space can only be defined on a highly abstract level, without any detail about which attributes can be perceived and which cannot.

A number of attributes that are generally perceivable by a student can, however, be identified. These include the media content part of reachable nodes, referential links attached to these nodes, lesson descriptions, and course layout and descriptions. The structure of nodes, networks, and lessons, is not usually revealed to the student, as is the case with the node monitors and link conditions that determine the traversal route through the CBI system.

The result display state is therefore characterized by a set of observable attributes and their values. It is composed by restricting the attributes in the RESULT set to those that can be observed by the student. For example, in the case of nodes:

\[
\text{Node\_Result\_Display} \equiv \text{Nodes\_Result} \setminus (\text{node\_monitors})
\]

4.7 The result and result\_render functions

The state of the resulting CBI system at any stage is defined by mapping the internal state to the result state set by means of the function result. The result function effectively strips the internal state of all the attributes relevant only to the central functionality of the ASE, and maps the values of the remaining attributes to their counterparts in the result state:

\[
\begin{align*}
\text{result} &: \text{STATE} \rightarrow \text{RESULT} \\
\text{result} &= \lambda \text{STATE} \cdot \theta \text{RESULT}
\end{align*}
\]

RESULT represents the whole CBI system and not only what is rendered to the student at the CBI interface, or what a specific student will receive during the instructional process. What is made accessible to the student is represented by the state set CBI\_DISPLAY. The function result\_render maps the perceivable attributes of RESULT state set to CBI\_DISPLAY:

\[
\begin{align*}
\text{result\_render} &: \text{STATE} \rightarrow \text{CBI\_DISPLAY} \\
\text{result\_render} &= \lambda \text{RESULT} \cdot \theta \text{CBI\_DISPLAY}
\end{align*}
\]

4.8 The ASE display state space

While the result state space relates to what the student can receive in the delivered CBI system, the display state space relates to what the author (teacher) will perceive of the CBI material under development, and in which format it will be presented to him. In designing interfaces to any interactive system, including ASEs, the concern should be with representing task-relevant data in a way which is well matched...
to the user's cognitive needs and task expectations. The concept of external representation implies a distinction between what is represented and its representation. As stated before we are concerned with the former — the semantics involved in the relationship between the attributes of the display state set and the internal state set.

As is the case with the internal state and the result state, the display state of an ASE is composed of a set of states, represented by ASE_DISPLAY. Generally, not all state attributes are rendered at the ASE interface at any particular instance, but only those pertaining to certain object attributes. The set of attributes of the ASE_DISPLAY state set is composed of a filtered set representative of the specific object type under consideration, as well the attributes essential to the central functionality of ASEs. How and which of the internal state attributes are rendered, is determined by the ASE specific mapping principles applied, as discussed in section 5 below.

The display state, reflecting the internal state of the ASE at the interface, is defined by mapping the internal state to the display state set by means of the render function:

\[
\text{ render : } \text{STATE} \rightarrow \text{ASE_DISPLAY} \\
\text{ render } = \lambda \text{STATE} \bullet \text{ASE_DISPLAY}
\]

### 4.9 Operations on state sets

Defining the behaviour function interpret for a particular process requires that the state transitions associated with all the possible events are specified as constraints on the process schema. This can be achieved by describing the individual operations allowed on each CBI object type, and then using Z's schema binding facility to generate a set of functions of the type \(\text{STATE} \rightarrow \text{STATE}\) that correspond to each possible operation. To complete the behaviour function, the possible events in alphabet are mapped onto the state transitions. Each possible event-input pair would generate an additional constraint on the function interpret. The process of defining operations as state changes, is described in full in Sufrin and He [13].

As an example we develop the framework for the Node_Process by considering the operation of creating a new node. To create a new node we need to specify the node's identity and type, and initialize the cursor indicator for identifying media and monitor objects under consideration. The operation can be defined in Z as:

\[
\text{Create_Node} \\
\Delta \text{Node_State} \\
\text{node_type_info? : NODE_ID} \rightarrow \text{Node_Type}
\]

\[
\forall \text{nid} : \text{dom}(\text{node_type_info}) \mid \\
\text{nid} \notin \text{dom node_instances} \Rightarrow \\
(\text{dom node_instances'}) = \\
(\text{dom_node_instances} \cup \{ \text{nid} \}) \land \\
(\text{node_instances'(nid).node_id}) = \text{nid} \land \\
(\text{node_instances'(nid).Node_Type}) = \\
(\text{node_type_info?(nid)}) \land \\
(\text{node_instances'(nid).current_cursor}) = 0
\]

To add this operation to the behaviour function interpret of the Node_Process it must be expressed as a function from Node_State to Node_State. The Create_Node operation on the state of a node is equivalent to the set of all pairs of bindings to Node_State and Node_State' that satisfies the constraints of the Create_Node schema. This set of pairs of bindings can be used to define a function create_node:

\[
\text{create_node : } (\text{NODE_ID} \rightarrow \text{Node_Type}) \rightarrow \\
(\text{Node_Statuses} \rightarrow \text{Node_Statuses})
\]

\[
\forall \text{node_type_info? : NODE_ID} \rightarrow \text{Node_Type} \bullet \\
\text{create_node(node_type_info?) =} \\
\{ \text{Create_Node(node_type_info?)} = \\
\text{nullNode_Statuses} \}
\]

To complete the definition of the behaviour function of the Node_Process, we have to describe the alphabet of Node_Process and the correspondence between its elements and these functions. First we define a set of events corresponding to creating a new node:

\[
\text{Node_Events ::= CREATE_NODE((NODE_ID} \\
\rightarrow \text{Node_Type})) \mid ...
\]

Finally, a constraint must be added to the Node_Process schema to ensure that, given one of these events, the behaviour function interpret will correspond to the state transformation defined by Create_Node. This constraint is represented by:

\[
\forall \text{node_type_info? : NODE_ID} \rightarrow \text{Node_Type} \bullet \\
\text{interpret(CREATE_NODE(node_type_info?)) =} \\
\text{create_node(node_type_info?)}
\]

The same procedure has to be followed for each operation applicable to the relevant object state.

For a framework to describe a node in terms of a process, the definition of Node_State can be used to provide a state parameter to the general process schema and the definition of Node_Events as the event parameter.

The Node_Process is therefore modelled at a level of granularity that treats the creation, deletion or modification of a node as a single event. In practice such state
changes may require a sequence of interactions by the author. When refining a design towards a concrete application of an interactive ASE, the internal structure of these composite events should be defined in full. Such internal structures might differ considerably in different ASEs, and including such structures in a general framework would therefore limit the range of its applicability and increase its complexity.

5 ASE-specific interface principles

The key to designing user interfaces for applications in a specific domain is to understand the important aspects of how work activities are performed in that domain, and how one activity might influence the behaviour of other activities in the domain. The interface principles related to specific activities are important in so far as they help or hinder the user in achieving his goals. Two examples, object observability and task migratability, are used to illustrate how the frameworks can be used to define the interface principles that would specifically apply to the class of interactive ASEs.

5.1 Observability of objects

Although the general notion of state-display observability is also important when using an ASE, a slightly different concept of observability is more significant when authoring CBI material. Throughout the authoring process the author should be able to observe exactly what effect his actions would have on the delivered product.

This notion of observability allows the author to evaluate the result state of the CBI system under development by means of its perceivable representation at the ASE interface. It specifically involves only those attributes of the internal state set (STATE) that will be present in the resulting state set (RESULT), and the way in which these attributes will be rendered to the author at the ASE interface, via the ASE_DISPLAY state. The emphasis is therefore on the relationship between the ASE_DISPLAY, and the RESULT and CBI_DISPLAY state sets.

Careful examination of the CBI objects reveals that all objects manipulated by an ASE consist of a set of sub-objects connected in some or other way. Each object thus has a content and a structure. How these content and structure components are rendered at the interface determines the specific principles to which the render function adheres. For example, the whole of the node content of a particular node should be observable to the author, and in the case of the instructional content, in the same format as it would be presented to the student in the delivered product.

This principle is useful in assessing the overall effect of a node. As far as instructional content is concerned, the author would, for example, be able to observe the effect of colour combinations, whether all text and diagrams are legible, etc. When considering node monitors, this principle would allow the author to observe all node monitors associated with a particular node, as well as the way in which they might affect each other’s values. Display-based ASEs such as QUEST (a version of which is illustrated in Figure 2) adheres to this principle, while most map-based ASEs, such as IconAuthor, as well as code-based ASEs, such as ACT III, violate this principle.

Furthermore, since CBI structures are not merely connections between media objects, but also determine the traversal and progress of a student through such material, the availability and manipulability of such structures become even more important than for ordinary multimedia presentations. An author should be able to observe the effects of his actions on the structure of a CBI system in order to ensure the deterministic nature of CBI networks.

Node structure observability refers to the way in which the order of the media object instances presented to the student, and the monitor instances manipulated by a specific node, are made perceivable to the author. The principle of node_structure_observable holds if the rendering of an ASE includes the structural build-up of a node:

node_content_observable : Node_State → Node_Display

∀ is : Node_State •
∃ rs : Node_Result | (result(is) = rs ∧
(is, render(is)) ∈ node_content_observable) •
∀ nid : NODE_ID;
nc : bag Node_Content_Type |
(nid ∈ dom(is.node_instances) ∧
nc = ran((is.node_instances(nid)).
node_content)) •
(render(nc) ≠ ∅ ∧
render(nc) ⊆ render(is) ∧
(∀ sic : bag Media | sic = nc> med •
render(nc) =
result.render(result(nc))))
The principle of `node_structure_observable` would assist the author in performing tasks such as deleting a particular media object from the instructional content sequence, changing the order in which the media objects are displayed to the student, adding a new media item to the sequence, etc. The display-based ASE QUEST, for example, adheres to this principle, while the map-based ASE ICON author treats media and monitor objects as disjoint entities and therefore adheres to a different structuring principle.

### 5.2 Task migratability

A principle more related to the dialogue initiative an author is allowed to take, and not directly related to the objects being manipulated, is related to the locus of authoring control.

An ASE adheres to the principle of `proactive control`, if all events that the ASE accepts are in the form of commands, *i.e.* the author is in total control of the authoring process (for example authoring languages such as ACT III, QAL, and PILOT). If the author only provides values to attributes forming part of a fixed shell, for example, for creating a test bank of text-based multiple choice questions, authoring is based on the system control principle. An ASE adheres to the principle of `system control` if all events that the ASE will accept are in the form of prompts. An ASE adheres to the principle of `coactive control` if the events that the ASE accepts are in the form of a combination of commands and prompts, *i.e.* the author is in control of the authoring process but uses the fixed data structures provided by the ASE.

If an author is allowed to switch between options at any stage during the authoring process, we say the ASE allows for task migration. The principle of `task_migratable` holds if an author can create equivalent CBI systems by using either system prompts or user commands, or combinations of system prompts and user commands:

\[
\forall is : Node\_State \implies Node\_Display
\]

\[
\exists rs : Node\_Result \mid (\text{result}(is) = rs \land (is, \text{render}(is)) \in \text{media\_structure\_observable})
\]

\[
\langle \forall nid : NODE\_ID; \ni : \text{seq Node\_Content\_Type} \mid \text{sic} \in \text{dom}(\text{is} \cdot \text{node\_instances}(\text{nid})) \land \ni = (\text{is} \cdot \text{node\_instances}(\text{nid}))\cdot \text{content} \cdot \text{nctype} = (\text{ran} \cdot \text{med}) \cdot \text{media\_type} \land \forall nt : \text{Monitor\_monitor\_type} \mid \text{nctype} = (\text{ran} \cdot \text{mon}) \cdot \text{monitor\_type} \cdot (\text{render}(\text{nctype}) \neq \emptyset \land \text{render}(\text{nctype}) \subseteq \text{render}(\text{is}))
\]

The frameworks presented contribute to several aspects in the field of mathematical models for interactive systems and aim to fill a current shortcoming in the range of systems that can be modelled by existing modelling frameworks.

An interactive framework was presented that can express both the general functionality of a system as well as the interactive principles related to the use of the system. It was further demonstrated how an abstract mathematical framework could be used to model an interactive system that has another interactive system as its product. We showed how existing abstract models can be, and need to be, extended to incorporate the different state sets applicable to interaction with such systems. In the process it was demonstrated how the modified Sufrin and He model combined with the state set approach of the Duke and Harrison interactors framework, can be used to argue about general usability principles relating to a class of interactive systems, as opposed to a single system instance of such a class.

Although we have limited our example to interactive ASEs, the same approach could be applied to other modern visual programming environments and multimedia development tools.

### References


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 notes. However, non-refereed review and exploratory articles of interest to the journal’s readers will be considered for publication under sections marked as Communications of 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 1½ or double spacing.
- The first page should include:
  - the title (as brief as possible)
  - the author’s initials and surname
  - the 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.
  - 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 according to the Harvard method.

Manuscripts accepted for publication should comply with guidelines as set out on the SACJ web page, http://www.cs.up.ac.za/sacj, which gives a number of examples.

SACJ is produced using the \TeX document preparation system, in particular \TeX 2e. Previous versions were produced using a style file for a much older version of \TeX, which is no longer supported.

Please see the web site for further information on how to produce manuscripts which have been accepted for publication.

Authors of accepted publications will be required to sign a copyright transfer form.

Charges
Charges per final page will be levied on papers accepted for publication. They will be scaled to reflect typesetting, reproduction and other costs. Currently, the minimum rate is R30.00 per final page for contributions which require no further attention. The maximum is R120.00, prices inclusive of VAT.

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

Proofs
Proofs of accepted papers may 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.

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 specialised production costs, which will be borne by the advertiser. Enquiries should be directed to the editor.