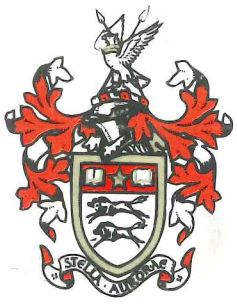

The South African Institute of Computer Scientists
and
Information Technologists

# The 1996 National Research and Development Conference

Interaction Conference Centre,
University of Natal, Durban

26 & 27 September

Hosted by




The Department of Computer Science and InformationSystems
University of Natal, Pietermaritzburg

SAICSIT 96

Sponsored by

IBM

PROCEEDINGS

Edited by Vevek Ram


The South African Institute of Computer Scientists
and
Information Technologists

# Proceedings

## of the

## 1996 National Research and Development Conference

## Industry meets Academia

Interaction Conference Centre, University of Natal,
Durban.
26 & 27 September

**Edited by**
**Vevek Ram**

ISBN 0-620-20568-7

Cover printed by Natal Printers (Pty) Ltd, Pietermaritzburg
Copying by the Multicopy Centre, University of Natal, Pietermaritzburg
Binding by Library Technical Services, University of Natal, Pietermaritzburg



## FOREWORD

This book is a collection of papers presented at the National Research and Development Conference of the Institute of Computer Scientists and Information Technologists, held on 26 & 27 September, at the Interaction Conference Centre, University of Natal, Durban. The Conference was organised by the Department of Computer Science and Information Systems of The University of Natal, Pietermaritzburg.

The papers contained herein range from serious technical research to work-in-progress reports of current research to industry and commercial practice and experience. It has been a difficult task maintaining an adequate and representative spread of interests and a high standard of scholarship at the same time. Nevertheless, the conference boasts a wide range of high quality papers. The program committee decided not only to accept papers that are publishable in their present form, but also papers which reflect this potential in order to encourage young researchers and to involve practitioners from commerce and industry.

The organisers would like to thank IBM South Africa for their generous sponsorship and all the members of the organising and program committees, and the referees for making the conference a success. The organisers are indebted to the Computer Society of South Africa (Natal Chapter) for promoting the conference among its members and also to the staff and management of the Interaction Conference Centre for their contribution to the success of the conference.

On behalf of the Organising Committee
Vevek Ram
Editor and Program Chair
Pietermaritzburg, September 1996

## Organising Committee

**Conference General Chairs**
Mr Rob Dempster and Prof Peter Warren (UNP)

**Organising Chair**
Dr Don Petkov (UNP)

**Secretariat**
Mrs Jenny Wilson

**Program Chair**
Prof Vevek Ram (UNP)

**Program Committee**

Prof Peter Wentworth, Rhodes
Dr Milan Hajek, UDW
Prof Derek Smith, UCT
Prof Anthony Krzesinski, Stellenbosch
Dr Don Petkov, UNP
Mr Rob Dempster, UNP
Prof Peter Warren, UNP



## Table of Contents

## List of Contributors


**Vladimir B Bajic**
Centre for Engineering Research,
Technikon Natal,
P O Box 953
Durban 4000

**C N Blewett**
Department of Accounting
University of Natal
King George V Avenue
Durban 4001

**Justin Cansfield**
Department of Accounting
University of Natal
King George V Avenue
Durban 4001

**Tom Considine**
Apron Services (Pty) Ltd
P O Johannesburg
International Airport
1600

**Eric Cloete**
School of Electrical Engineering
Cape Technikon
Box 652
Cape Town

**I Cloete**
Computer Science Department
University of Stellenbosch
Stellenbosch
7600

**O A Daini**
Department of Computer Science
University of Botswana
Gaborone
Botswana

**Nirvani Devcharan**
Umgeni Water
Box 9
Pietermaritzburg
3200

**P J A de Villiers**
Department of Computer Science
University of Stellenbosch
Stellenbosch
7700

**Ruth de Villiers**
Department of Computer Science and
Information Systems
UNISA
Box 392, Pretoria, 0001

**G J Erwin**
Business Information Systems
University of Durban-Westville
Private Bag X54001
Durban 4000

**G Ganchev**
Computer Science Department
University of Botswana
PBag 0022
Gaberone, Botswana

**J Geldenhuys**
Department of Computer Science
University of Stellenbosch
Stellenbosch
7700

**Louise Gibson**
BIS, Dept Accounting & Finance
University of Durban
Pvt Bag X10
Dalbridge 4014

**Mike Hart**
Department of Information Systems
University of Cape Town
Rondebosch
7700

**M. Hajek**
Department of Computer Science
University of Durban-Westville
Pvt Bag X54001
Durban 4000

**A C Hansen**
Dept of Agricultural Engineering
University of Natal
Private Bag X01
Scottsville 3209

**J M Hattingh**
Department of Computer Science
Potchefstroom University for CHE
Potchefstroom 2520





**Boris Jankovic**
Centre for Engineering Research
Technikon Natal
P O Box 953,
Durban 4000

**Paula Kotze**
Department of Computer Science and
Information Systems
UNISA
Box 392
Pretoria, 0001

**J W Kruger**
Vista University
Soweto Campus
Box 359
Westhoven 2124

**A C Leonard**
Dept of Informatics
University of Pretoria
Pretoria
2000

**Ben Laauwen**
Laauwen and Associates
P O Box 13773
Sinoville
0129

**Mari Le Roux**
Information technology, development: project leader
Telkom IT 1015
Box 2753
Pretoria 0001

**P W L Lyne**
Dept of Agricultural Engineering
University of Natal
Private Bag X01
Scottsville 3209

**Rose Mazhindu-Shumba**
Computer Science Department
University of Zimbabwe
Box MP167
Harare, Zimbabwe

**Meredith McLeod**
Centre for Engineering Research,
Technikon Natal,
P O Box 953
Durban 4000

**D Moodley**
Computer Management Systems
Box 451
Umhlanga Rocks
4320

**Andrew Morris**
P O Box 34200
Rhodes Gift
7707

**R C Nienaber**
Technikon Pretoria
Dept of Information Technology
Private Bag X680
Pretoria 0001

**E Parkinson**
Department of Computer Science
University of Port Elizabeth
Box 1600
Port Elizabeth 6000

**Don Petkov**
Department of Computer Science and
Information Systems
University of Natal
PBag x01
Scottsville 3209

**Olga Petkov**
Technikon Natal
Box 11078
Dorpspruit 3206
Pietermaritzburg

**N Pillay**
Technikon Natal
Box 11078
Dorpspruit 3206
Pietermaritzburg

**V Ram**
Department of Computer Science and
Information Systems
University of Natal
PBag x01
Scottsville 3209

**Melinda Redelinghuys**
Department of Computer Science and
Information Systems
UNISA
Box 392
Pretoria, 0001

**Karen Renaud**
Computer Science and Information Systems
UNISA
Box 392
Pretoria, 0001

**H N Roux**
Department of Computer Science
University of Stellenbosch
Stellenbosch
7700

**T G Scott**
Department of Computer Science
Potchefstroom University for CHE
Potchefstroom
2520

**T Seipone**
Department of Computer Science
University of Botswana
Gaborone
Botswana

**Derek Smith**
Department of Information Systems
University of Cape Town
Rondebosch
7700

**Anette L Steenkamp**
Department of Computer Science and
Information Systems
UNISA
Box 392
Pretoria, 0001

**T Steyn**
Department of Computer Science
Potchefstroom University for CHE
Potchefstroom 2520

**H. Suleman**
Department of Computer Science
University of Durban-Westville
Pvt Bag X54001
Durban 4000

**A Vahed**
Department of Computer Science
University of Western Cape
Private Bag X17
Bellville 7530

**A Van der Merwe**
Computer science and Informations Systems
UNISA
P O Box 392
Pretoria,0001

**Tjaart J Van Der Walt**
Foundation for Research and Development
Box 2600
Pretoria, 0001

**K Vavatzandis**
Department of Information Systems
University of Cape Town
Rondebosch
7700

**Y Velinov**
Dept Computer Science
University of Natal
Private Bag X01
Scottsville 3209

**H Venter**
Department of Computer Science
University of Port Elizabeth
Box 1600
Port Elizabeth 6000

**H L Viktor**
Computer Science Department
University of Stellenbosch
Stellenbosch
7600

**R Von Solms**
Department of Information Technology
Port Elizabeth Technikon
Private Bag X6011
Port Elizabeth 6000



**A J Walker**
Software Engineering Applications
Laboratory
Electrical Engineering
University of Witwatersrand
Johannesburg

**P Warren**
Computer Science Department
University of Natal
P/Bag X01
Scottsville 3209

**Max Watzenboeck**
University of Botswana
Private Bag 0022
Gaberone
Botswana

# THE ROLE OF FORMALISM IN ENGINEERING INTERACTIVE SYSTEMS

M.D. Harrison and D.J. Duke
Department of Computer Science
University of York
Heslington, York, YO1 5DD, U.K.

## Abstract

This paper is concerned with the role of formal notations and methods in engineering interactive systems. It begins by briefly reviewing the role of formal methods in Human Computer Interaction. The objective of capturing requirements for interactive systems, particularly those requirements that are concerned with folding a user orientated perspective into the design, is then discussed. An object oriented specification technique is introduced to emphasise human interaction with the system and to provide a first step towards specifying user requirements. The paper concludes by discussing the use of this approach to support design refinement and to check that specifications satisfy interaction requirements.

## Introduction

Formalism is commonplace in Human Computer Interaction.

- *Domain modellers* or *task analysts* use it to describe the work system in which a computer based artifact, or network of artifacts, resides. Here the purpose of the notation is precise description of work objectives, procedures for achieving these objectives, and general organizational and communication characteristics associated with the system. The role of the formalism is to aid the capture of the important concepts succinctly. The formalism also plays a role in checking consistency and accessibility of knowledge structures, see for example the TAKD notation (Diaper, 1989), or the TAG notation (Green et al., 1988). The task structure incorporating plans of how tasks should be carried out, may also be linked to a system model. For example Baber and Stanton use a state-transition diagram (Baber and Stanton, 1994)) in order to assess potential failures, and their impact, that might occur in execution of these plans.

- *Cognitive modellers* use formalism to assess what cognitive resources are required to understand and use the system. Here the formalism is required to provide conceptual clarity as well as to represent scenarios for simulation. Task Action Grammar was designed to capture the competence of a user (Green et al., 1988)). From another angle, Young and his colleagues use formalism to describe the domain and device characteristics of a system prior to using a planning system (the SOAR system) which simulates some aspects of cognition, to emulate what the planner would do to achieve certain objectives, and to compare a designer's idealised description of the behaviour of the system with what the simulator in fact produces (Young and Whittington, 1990).

- *Specifiers or modellers of dialogue* use a formal notation to describe the dialogue and to create the characteristic ("look and feel") of a particular application. Here the role of the formalism is to provide a basis for interpretation of the dialogue description which can be prototyped accurately and quickly (Green, 1987).

- *Software (or more generally systems) engineers* use formalism to describe the characteristics of an interactive system in order to facilitate its accurate construction and maintenance. Here there are a number of roles for the formalism, and it is these roles that will form the basis for this paper.

The advantage of a formal notation is that it is associated with a clearly defined meaning, often expressed mathematically, and may also be connected with rules for proving properties (about timing or consistency for example) of expressions of the language. The theme of this paper is the role of formal notations in engineering interactive systems. Here we are particularly concerned with the use of formal notations to represent interactive systems and what properties may be conveniently represented within them. There

are two reasons for representing interactive systems. The first is to provide a means of analyzing an existing system so that it becomes possible to check it for properties such as completeness or consistency. The second is to provide a representation that supports the conceptualization and refinement of interactive systems. Preoccupation with the analysis of specifications leads to an emphasis on design techniques that are rigorous rather than exploratory.

To support the special requirements of interactive systems, extensions and styles of specification have been specially developed. In particular any specification technique must take account at some level of the whole system: human, software and hardware. Formal notations are required that can express an "interactive view" of many agents to an interactive system, as we are interested in expressing user requirements of specifications as well as refining and checking specifications.

In the next section we identify briefly the role of formal notations in software engineering. We then discuss HCI specification and the role of formal notations in expressing interactive systems. In this context the problem of *folding* user or task issues as requirements into specifications will be articulated. We introduce a number of properties that we might want to prove true of interactive systems.

In the following section we present a specification structuring notion, that of interactor, that can be used to capture essential characteristics of interactive systems and use it to specify a simple system employing a hybrid of two existing systems engineering notations. Issues concerned with the refinement of formal specification of interactive systems are then introduced, discussing in particular the relationship of top-down and bottom-up techniques and refinement. We also discuss mechanisms for producing prototypes from specifications. Finally, work in progress demonstrating the validity of properties of interactive systems is presented.

## Engineering Interactive Systems

The argument for the use of formal notations in the engineering of interactive systems is that informal techniques often lack precision, and this can lead to ambiguity, and therefore to systems that fail to meet requirements. This failure can be expensive to deal with downstream during the implementation and validation phases of the design and implementation lifecycle. The use of formal notations in some safety critical systems has been justified on this basis (Hall, 1990) despite reasonable concerns about relative cost of the specification phase. Formal notations are regarded (possibly mistakenly, according to (Hall, 1990)) as difficult to understand and are often used in an obscure style. In the main, however, where these notations are used in practice, their practical role has been to assist the designer and implementer in understanding the system.

It is also recognized that two further goals may be achievable if formal notations are used. The first possibility is that system specification may be progressively transformed, preserving correctness, into an executable program. Refinement rules and properties are difficult to apply and prove. Their use and application could be much improved through the development of appropriate formal methods and the use of software tools that are currently unavailable. The second goal is that properties or requirements of a specification may be proved to be discharged by the program. It is clear that though both goals are desirable, extensive automatic tool support would be required to make them feasible.

A variety of formal approaches are being developed. There are a number of distinctions (see also (Gaudel, 1994; Vissers et al., 1991)) that can be made between them:

1. between model based specifications, in which established theories are incorporated, and algebraic specifications which permit the introduction of new theories;
2. where there is good support for conceptualization versus an adequate proof theory (supporting verification);
3. where the specification describes the internal behaviour such as state of the system versus where the specification describes external behaviour such as communication between processes;



4. whether the specification notation is textual or diagrammatic as is the case with approaches such as statecharts (Harel, 1987) or Petri nets (Palanque and Bastide, 1994).

The means of breaking the specification down into components in order to support abstraction and modularization in large scale specifications is also a key and somewhat neglected aspect of their design. The formal specification notation to be described in this paper, uses an object structure with the aim of dealing with problems of scale and providing a structure that corresponds to the way in which a presentation (display for example) is constructed.

## Folding the user into the system

### Role of formalism

The problem of Human Computer Interaction is to take the view of the user or user team in relation to the design of a computer system, in order to make the system more "natural", "usable", "human-error tolerant" etc. The concern of much applied psychology within HCI has been the individual behaviour of human users of computer systems, producing methods for experimenting with systems and theories for addressing the needs and resources of these users. More recently this study has been broadened, recognizing the limitations of a simple individual cognitive view, and incorporating a broader understanding of external considerations (Suchman, 1987; Hutchins, 1994; Nardi, 1996). Ethnography, organizational psychology and other organisational theories have had a role here. The difficulty with much of this work is that the connection between insights into human behaviour and the design of computer systems is difficult to forge. Much of the work that is done is at the level of post-hoc "holistic" evaluation. The problem we are concerned with is how this user view of a computer system may be incorporated into the representation of the system.

As has already been noted in the Introduction, there are a variety of formàlisms available for describing aspects of the HCI problem. In many cases, the primary purpose of the formalism is to act as a check for the cognitive modeller or work modeller. Hence Task Action Grammar (Payne and Green, 1986) is used to represent the competence of a user and can be used by the psychologist to analyze informally the consistency of the interface. A task analysis notation such as TKS (Johnson et al., 1988) may be used to express what is required in order to perform the set of tasks of the system. Notations also play an implementational role. So for example Young and Blandford's (Blandford and Young, 1993) Instruction Language is used to help the cognitive modeller conceptualize the problem but is also the representation that will be used by the SOAR system in simulation.This paper is concerned with software engineering notations with an emphasis on their ability to provide the possibility of more automatic checking of the artifact and thereby to assist the design process.

The role of formal specification is to make precise the behaviour of an artifact so that an implementer may construct a system appropriately. In the case of specification, where details of the state of the system and operations on the state of the system are expressed explicitly (as an abstract data type, for example), emphasis is on the ability to demonstrate that refinement to implementation preserves the requirements of the specification. However, it is also concerned with properties of specifications including general properties of consistency and completeness, as well as more specific properties of a particular specification. In the case of specifications where the concern is with external behaviour, the purpose of the specification is to show that certain properties are true of the system, for example it is deadlock free.

It will be necessary to structure a specification so that those perceivable aspects of the state (display, for example) may be reasoned about as well as those actions that the user carries out in order to invoke the functionality of the system. In practice, existing methods of specification are adequate for the purpose of reasoning. We shall adopt a particular approach to illustrate the technique. This approach is based on a structuring mechanism (*interactor*) which makes interactive behaviour explicit at an object level without compromising the use of existing and well-founded formal specification techniques.

### Interaction Requirements

Given a specification of an interactive system, requirements may be expressed that concern the resources and capacities of the user. We list some typical generic requirements.

- Information presented by the system should be relevant to the performance of the tasks that the system is designed to support.
- Immediately relevant commands should be directly accessible in the current mode.
- It should be possible to recover to a previous state when a mistake is made.

System support for the prevention of slips of action may be achieved by ensuring that the effects of actions are visible to the operator of the system. As shall be seen, the mechanism of interactors is designed to support this requirement by providing a structure that will encourage systems designers to ensure that the internal operations of the system are made visible to the operator. In practice the visibility of actions is often related to the context of the task that is being carried out. Mistakes may be protected against by providing a clearly visible model of how the system works.

Actions that are taking place in the system should be clearly visible in the "rendering" of the system. This idea is made explicit in notions of:

- *visibility* that requires that attributes of the state are perceivable in the presentation;
- *predictability* (Harrison, 1992), that takes into account the fact that the state of the system may affect the consequence of operations without the operator being aware of the state that has these effects.

A system may be more tolerant to mistakes if it is consistent. *Consistency* is a system property that supports appropriate model generalization and thus reduces the likelihood of error. It is a notion that should be used carefully because inappropriate or partial consistencies may have the effect of leading to inappropriate generalization (see (Grudin, 1989)).

### Mechanisms for incorporating requirements

The problem is to develop a model of the system that will make it possible to demonstrate that user requirements are satisfied. At one level, the concern is to express the interactive behaviour of the system in more detail than is the convention within the formal specification of systems, see for example (Bowen, 1992). It is also necessary to capture properties in the specification that may only have significance in understanding how the system is perceived. Hence, in the notion of interactor introduced next, a rendering defines those elements of the state that are perceivable (perhaps audible), and a theory of presentations (Duke and Harrison, 1994) defines specific characteristics of perception of different modalities, for example the way those modalities are "chunked".

Further, we might wish to take into account other external aspects of a system. For example, we may wish to define those aspects of the system that are relevant in the performance of particular tasks. For example, Roast (Roast, 1993) describes a notion of *template* to capture those aspects of the display and state of a system that are relevant to a particular task.

## Interactors

The question we consider now is how formal specification notations may be used to describe interactive behaviour appropriately. Interactors provide a means of bridging between the requirements of the user and the specification of interactive system that is used for implementation. The usability or human error properties, considered above, become more specific and can be expressed in terms of particular applications.



The term interactor has also been used to describe a class of low level generic objects that are instantiated to an implementation (Myers, 1990) (here the term *widget* is sometimes used). Hence an interaction object might include a generic menu widget for example that is instantiated to the particular menu when constructing the system. The notion of interaction object represents a useful structure for thinking and reasoning about the behaviour of interactive systems in general.

A number of other approaches have been taken to specifying interactor like objects (see for example (Faconti and Paternò, 1990)). We use a hybrid style of specification, linking state information and behavioural information. The two models each emphasizes different aspects of interaction, and the formalisms used to express the models afford different approaches to the construction and analysis of specifications.

### Specifying Interactors

This interactor model is developed in order to express useful properties of interactive behaviour (Dix et al., 1987). The model (Duke and Harrison, 1993) is based on states, commands, events and renderings. These ideas have been used to expose the properties expressed above as predictability and visibility. It is also based on the structuring of model based specification around object oriented concepts, in particular the Object Z notion of Duke and others (Duke and Duke, 1994). In outline, an interactor consists of an internal

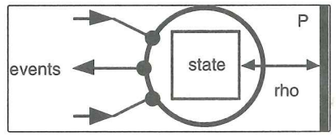



Figure 1: The Interactor.

state which is reflected through a rendering relation onto some perceivable representation. The interface between an interactor and its environment consists of a set of events. There are two kinds of events: *stimuli* are caused by agents within the environment and bring about state changes, while *responses* are events generated by the interactor.

**interactor** [*press-button*]

**attributes**

*vis* *enabled* : $\mathbb{B}$

**actions**

*press*

**axioms**

1. $enabled = X \Rightarrow [press]enabled = \tilde{X}$

The *state* of an interactor is modelled by a set of typed attributes (variables) such as 'selected'. In the example, this variable takes on a boolean value (true or false) to represent when the button has been selected by the user. This property of a button (that is, whether it is currently selected or not) can be perceived visually, hence the the boxed *vis* annotation. Such perceivable variables are called percepts, and make up the presentation component of the interactor. One action, press, is available in the interface of the interactor. Its effect is to toggle the button between being enabled or not enabled. This behaviour is described precisely by axiom 1, which uses a modal predicate that reads: if the value of the variable 'enabled' is given by X, then in the state that arises after the action 'press' has been performed, the value of 'enabled' will be the negation of X. In general, predicates of the form $P \Rightarrow [A]Q$ mean that in any situation where 'P' is true, performing the action 'A' will bring about a situation where 'Q' is true.

The button-press interactor can be inherited by other system components that might respond to the 'enabled' state in application-specific ways. In this approach different copies of an inherited interactor can