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

Real-Time Systems

Database Systems

Education

Practice and Experience

Information Technology and Organizational Issues

Abstracts

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 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

Real-Time Systems

Database Systems

Education

Practice and Experience

Information Technology and Organizational Issues

Abstracts

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 formalisms 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

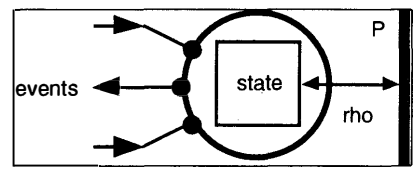

Figure 1: The Interactor.

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

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

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

be distinguished by giving them names, either individually, or by 'tagging' a collection of interactors with values drawn from a set.

We now consider an interactor which uses press-button. This specification manages an incoming messages queue. It also supports two buttons that can be pressed by the user. 'clear' clears the visible message from the queue. 'goto' has a more complicated role and is used when a message of priority 2 or 3 is used. In fact only the behaviour of the clear button is established at this point, since the goto button has an effect on the display which would involve a further interactor. The specification is actually based on a real system concerned with air traffic control, and the messages refer to incoming flights maintained in a stack. We can describe the specification more rigorously.

We assume the existence of a type 'msg' to represent messages, and a function $pr: msg \rightarrow N$ that takes each message to its priority level. At this point we are not interested in the contents of particular messages, simply in their existence and priority level. The msgs interactor appears below. It includes two copies of the button interactor to represent the *clear* and *goto* controls. The state of the msgs interactor is determined by three variables, representing the queue of messages, (queue) the displayed message (mesg), and the number of messages (nr-msgs). Of these, the latter two are derived from the first and together form the presentation of the interactor (via the visual modality). One new action is introduced by the theory - it can receive a message (recv). However note that the two actions clear.press and goto.press are also available within msgs on account of interactor inclusion.

interactor [*msgs*]

attributes

clear : *press-button*
goto : *press-button*
queue : seq *msg*
vis mesg : [*msg*]
vis nr-msgs : N

actions

recv : *msg*

axioms

1. $mesg = nil \Leftrightarrow queue = \langle\rangle$
2. $queue \neq \langle\rangle \Rightarrow mesg = hd\ queue$
3. $nr\text{-}msgs = len\ queue$
4. $\forall i,j \in queue.i \leq j \Rightarrow pr(queue(i)) \leq pr(queue(j))$
5. $queue = x \Rightarrow [recv.m]queue =$
$(x \upharpoonright \{m' \in \text{ran}\, x \mid pr(m') \leq pr(m)\}) \frown [m] \frown (x \upharpoonright \{m' \in \text{ran}\, x \mid pr(m') > pr(m)\})$
6. $queue = \langle mesg\rangle \frown s \Rightarrow [clear.press]queue = s$
7. $clear.enabled \Leftrightarrow queue \neq \langle\rangle$
8. $goto.enabled \Leftrightarrow pr(mesg) \in \{2,3\}$

Because the message queue can be empty, the perceivable message is represented by an optional value; for any type T, the type [T] is the set of values defined by T plus a distinguished 'nil' value. Axiom 1 establishes that the visible message is nil if and only if the queue is empty, and if the queue is non-empty axiom 2 requires that the message displayed is at the front of the queue. Axiom 2 connects the length of the queue to the 'number of messages' value displayed in the message area.

Each message in the queue has a priority level, pr(m). The queue is organized so that all priority 1 messages appear before priority 2 which in turn appear before priority level 3 messages (axiom 4). Axiom 5 states that as new messages arrive they are placed in the queue after all other messages of their priority but before any messages whose priority level is higher. This is expressed by splitting the message queue into two parts - those messages whose priority is the same as or lower than the priority of the received message, and those with a higher priority - and inserting the new message between these sections to construct the new queue. The remaining three axioms express the connection between the buttons and the message queue; axiom 6

says that pressing the clear button removes the first message from the queue (and consequently changes the visible message, through axioms 1 and 4). Further (axiom 7) the clear key is only enabled if the queue is non-empty. Finally the 'goto' key becomes enabled if the message refers to some level 2 or level 3 message.

The receipt of a message may require that the displayed message changes; for example, if the current message has priority 2 and a priority 1 message arrives:

$$pr(mesg) = 2 \wedge pr(m) = 1 \Rightarrow [recv.m]mesg = m$$

A formal proof of this property is not difficult and will be demonstrated later; we need to apply axioms 5, 4, and then 2.

Elsewhere we discuss a specific approach to relating formalised task descriptions to interactor style system models of interactive systems (Fields et al., 1995b).

Refinement and Prototyping

An important concern in the development of formal specifications is the means by which the specification is converted into an executing system. There are two aspects to this problem. The first is how the specification can be *refined* into a correct program. The second is whether and when the specification can be executed directly so that the developer may get the look and feel of the system at an early stage and possibly use it for evaluation and iteration.

Refinement

Rigorous software development is concerned with demonstrating that a program correctly implements a specification, either through a process of verification (see next section) or through the systematic derivation of program from specification by valid refinement transformations (Morgan, 1994). Refinement involves the construction of data structures and operations that are closer to the level of the machine than those in the original problem description. These transformations, when applied to the design state, assume that the specification of the system is primarily concerned with the functional behaviour of the system rather than its interface behaviour. In the case of data refinement, the data in the original specification must map to data in the refined specification, and the operations on the refined specification must mirror the behaviour of the original operations. In practice data refinement usually involves the addition of new operations. It is also necessary that the new specification is conservative in the sense that the properties of the operations defined in the original specification must be true also of the mirrored operations in the new one. Operational refinement, on the other hand, is concerned with the implementation of operations. It requires that a refined operation be defined on at least the states of the original but be more determined over these states, hence restricting the generality of the original operation.

In the development of most sorts of systems, the interaction between the user and the system should also be taken into account in an analogous process of refinement. How is the specification of the presentation and the actions or events involved in interactive behaviour affected by refinement transformations? At a gross level of specification we are concerned about whether the semantics interpreted by the user from the perceivable data correspond to what the system supports. This problem of interface refinement has been considered at varying levels of detail by Bramwell (who calls it *enhancement*) (Bramwell, 1995), Dix, Duke and Harrison (Harrison and Dix, 1990; Duke and Harrison, 1995). There are three corresponding aspects to interface refinement. We shall first describe these concepts and then typical examples:

- *data refinement*: (see for example: (Duke and Harrison, 1995)) as data is refined to more concrete representations is there a corresponding refinement of the presentation?
- *trace refinement*: (see for example: (Bramwell, 1995)) as the system is refined, the class of behaviours that can be engaged in is limited. Here the word *trace* is used to mean a sequential event

structure. This can happen in two ways: (1) more abstract specifications of behaviour may include non deterministic behaviour, for example the choice about how certain events occur may be delayed, and as the system is refined these behaviours may become more explicit; (2) the system will be designed to support particular tasks and these tasks may limit the possible behaviours of the system under design.

- *event structure refinement*: here a single event, or trace of events, may be replaced by a structure of events that is more detailed.

We can elaborate each in terms of an example.

Data Refinement

An abstract model of a file system may represent a file as an uninterpreted, atomic, value. A refinement of a file system, so that file concatenation may be described in terms of the explicit structure of the file, represents the file as a sequence of records. Further operations may be introduced that use this same structure. The problem now arises that a corresponding "refinement" of the *presentation* of the specification may lead to a representation that is inconsistent with, say, the desk top analogy that is being used in the *presentation*. The introduced operations may have no analogue in the desk top metaphor. It is therefore necessary to preserve appropriate presentation style characteristics as the specification of the data structures are enriched and refined. (Duke and Harrison, 1995) describes a mechanism for preserving the conformance of these specifications.

Trace Refinement

An example, of trace refinement may be illustrated by an Automatic Teller Machine (ATM). Suppose the specified system supports inserting a card, a Personal Identity Number (PIN), requesting a facility, receiving the result requested and returning the Card. Trace refinement of a specification may take place (1) so that the ATM may support more precisely the tasks for which it is intended. Hence the specification of the ATM will be designed to prevent the user from engaging in events that are not appropriate to a task once a particular option is chosen from a menu; and (2) to ensure that, for cash withdrawal, the card is withdrawn *after* the cash is taken. Even though either trace makes sense in the design of the ATM, the removal of the trace in which cash is withdrawn first will avoid premature closure, see (Fields et al., 1995b), where the customer takes the money and leaves without collecting the card. Hence trace refinement involves reducing the set of traces that represent the behaviour of the system, the events that may be engaged in remain the same.

Event Structure Refinement

In the case of event structure refinement, the same example of the cash dispenser may be used again. It is normal to consider actions at a high level in the initial stages of specification development. Hence entering a PIN can be considered as a single action. In subsequent stages of refinement, the PIN entry may be considered as four numeric actions and be considered in terms of the possibility of recovery. So entering a PIN will involve a behaviour that allows the four digits to be inserted in sequence, and in addition the behaviour associated with a cancel action that allows the user to start the PIN entry again at any point in the sequence. In both cases, trace refinement and event structure refinement, it is most usual that functional behaviour will not be affected by particular choices at the refinement stage. Here refinement may involve adding events that express the more detailed behaviour.

Prototyping

The top-down philosophy of system design is appropriate only in theory. In practice, because users and context are involved, it is important that the system be evaluated early, possibly experimentally. A number of systems exist which provide a basis for exploring a user interface, see for example Myers (Myers, 1988). The problem in the context of formal specification is whether it is possible to get an impression of an interactive system on the basis of executing the formal specification. Little research relates to this problem in the broader context of full specifications of interactive systems. A broader range of systems have been designed to be driven by dialogue specifications, see for example an early review by Mark Green (Green, 1986).

More relevant is work by Alexander (Alexander, 1987). This approach uses the MeToo system for executing VDM like specifications. Dialogue is described using CSP and events are linked to functions with pre- and post-conditions described in terms of VDM. Related research has been carried out at York with iterative evaluation of interactive systems in mind. Johnson (Johnson and Harrison, 1992) links temporal logic as a means of specifying interactive systems with a screen presentation system *Presenter* (Took, 1990). This provides a means of getting a rapid look and feel of a specification. The problem with this approach is that it is necessary to map models, such as interactors, into a logic based executable specification. Roast takes this work a small step further, though the link with a good quality presentation is not so well established. He links a model and properties by which the model is constrained into a working prototype (Roast, 1993). Here Roast uses a specially developed logic called *interaction logic.*

The link between specification and rapid prototyping is an important one. Much more research needs doing that supports sufficient input/output resolution to be used a "sketch" of the interface so that it may be valuable from the point of view of evaluation with users beyond existing paper techniques.

Checking Properties

As discussed earlier, interactor based system engineering is designed to support the specification of interactive systems. One objective of this process is to capture user centred properties and concepts in the specification. It is also an objective that there should be sufficient detail of the design to provide a basis for accurate implementation of the specified system. Hence it should be possible to use interactor notions to specify different aspects of the interactive system and to verify them against properties. In this context the interactor structuring notion that supports hybrid specification techniques (in a variety of forms) is designed to support all aspects of the specification. These interactors are used to describe both function and interface, whereas the LOTOS style approach is designed for the purpose of constructing a user interface system (UIS) that intervenes between the user and the functional core. In the case of the LOTOS style interactor the aim of the structure is to provide a mechanism for emphasizing perceivable and user accessible components of the state and system function.

The same techniques have also been used to model the whole domain of the system before commitment to whether components are to be expressed in software or are to be carried out by the users of the system (see Fields, Harrison and Wright, (Fields et al., 1995a)). Interactors have also been used to describe aspects of the users cognition and the interaction between the user and the system. This notion, called syndesis, is described in (Duke (Duke, 1995)). Here the proposed advantage is that syndesis enables both the system and the cognitive modeling to be provided in the same language using the same tools and supporting the same checking procedures.

This section of the paper is concerned with the question of how to *validate* or *verify* properties of the specification. If a system is to exhibit these properties then either there should be a method for ensuring that a particular system captures them (in other words, the properties are used generatively) or it should be possible to check that a specification, in some form of completeness, exhibits the properties. Here we shall discuss properties that relate more specifically to the system under consideration than the generic properties that we have been discussing so far.

Three types of argument are illustrated here to show how an interactor specification may be used: an in-

formal argument based on the formal specification; an informal argument based on a diagram that graphically represents event behaviour; a formal argument based on natural deduction. We shall focus on the 'msgs' interactor that deals with the receipt, queueing and display of incoming messages. They approaches are illustrated by asking three questions of the specification:

1. Can the user perceive all relevant parts of the system state possibly through the use of commands that change the perceivable state?

This question can be dealt with by inspecting the specification. Only one message ⟨*mesg*⟩ can be observed at a time. It is an invariant (axiom 2) that it is always the first message in the queue. If the operator is to perceive other messages then the queue itself must change. New messages of higher priority displace the first message; older messages can only be observed by removing the currently displayed message. These actions cannot be undone.

This level of discussion is adequate to assure the system developer of the scope of the possible implementation based on this specification. More rigorous argument would be unnecessary and perhaps error prone. Properties may also be checked more formally using the same specification.

2. Is it possible for a message to be lost, that is discarded, accidentally?

A first approach to this might be based on the idea of choosing a scenario which represents a situation in which a message may be lost. This can be done informally using a diagram to indicate event behaviour.

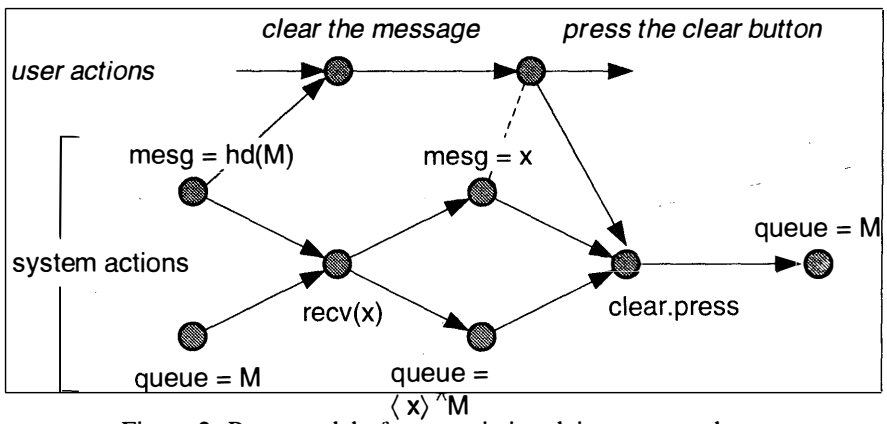

Figure 2: Poset model of a scenario involving message loss.

This graphic description of a scenario uses the action description provided in the interactor and visualizes a partially ordered set to describe one possible behaviour. The scenario suggests that a high priority message may be lost if it is received at 'about' the time that a user is clearing a displayed message. A user starts executing the clear action, either not noticing that a new message suddenly appears on the display, or not able to stop a process of having committed to an action.The dashed line between the events 'mesg = x' and 'press the clear button' indicates the course of the scenario. The result of the action is that the queue is unchanged, consequently the user may not even notice that a message has been lost. It is possible that the unchanged message line will be attributed either to a mis-hit of the clear button or to a fault in the system. Clearly such a scenario involves making assumptions about (a) user behaviour and (b) user capabilities. While these may need to be validated by user modellers or experimentation, for the purpose of reasoning about interaction, such assumptions can be seen as a form of best- (or worst-) case analysis.

3. Proving question 2 by natural deduction

A formal proof can be done by natural deduction. The property can be expressed as a hypothesis to the effect that if a message is received of higher priority than that on the display, and the user subsequently

presses the clear button, then the message will be the same as that before the new message.

$$H: pr(x) > pr(mesg) \wedge queue = M \Rightarrow [recv(x)][clear]queue = M$$

Proof of this property uses axiom 5 from the 'msgs' interactor. The axiom is used to prove a lemma that states that if the priority of a received message is greater than the currently displayed message then the incoming message is added to the queue. We demonstrate the rigorous proof of the property, using the lemma, by means of a tableau method for modal action logic (Atkinson and Cunningham, 1991).

$$Lemma: pr(x) > pr(mesg) \wedge queue = M \Rightarrow [recv(x)]queue = \langle x \rangle^\frown M$$

Proof:

From axiom 2 the value of mesg is the head of the queue, and from axiom 4 no other message in the queue can have a higher priority. Axiom 5 says that a new message is inserted between those of higher or equal priority and those of lower priority. Since the assumption is that there are no higher priority messages in the queue, the new message must be appended to the front.

The proof follows by negating the hypothesis (H) and then attempting to show that this contradicts the axioms of the msg interactor. At each step we can develop one of the axioms, expressing it either as a disjunction or conjunction; the former results in a branch in the tableau. Developing a modal formula involving action A opens up a new tableau which contains those axioms earlier in the tree that are prefixed with [A]. A branch is closed (shown by a box) when it contradicts axioms further up the tree.

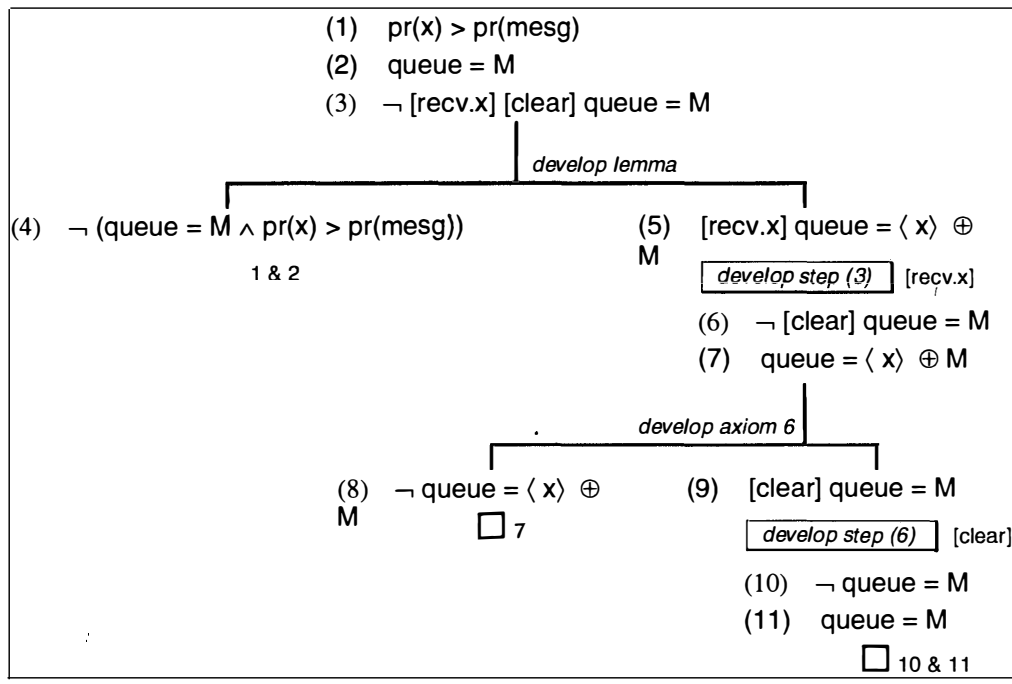

Figure 3: Tableau proof of simple property.

Scenarios may be investigated in terms of the paths generated in more detail and this is where other approaches may provide a better pay-off. In practice it may not be obvious that there are interesting scenarios that fail to satisfy given requirements. Paterno and others (Coutaz et al., 1995) take a LOTOS specification to describe the interface between application and user. The interface is described in terms of a set of processes also called interactors. The architectural structure reflected by these processes is constructed by means of a set of heuristics associating the process decomposition with the task structure of the system. The processes in the system therefore have a structure in relation to each other that reflects tasks and the resources that are required by the user in order to carry out the task.

The sorts of property that this work deals with are:

1. that a particular user action will always result in a particular application input (roughly reachability);
2. that all user actions have a corresponding interface appearance (roughly visibility);
3. that a user action is reflected in an interface appearance immediately before any further user action is permitted (continuous feedback);
4. that user actions are available to recover from an error (recoverability).

These properties are expressed in terms of ACTL and the LOTOS description is transformed into the underlying model of ACTL using a tool from the LITE toolset and checked using a model checker, for relevant work see (Coutaz et al., 1995).

It is clear that although these are general properties they can in practice be tailored to the requirements of the particular system being specified. In fact most of these properties presume some idea of state. In an interactor description as described above the appropriate property would say something about the relationship between user action and the interface appearance. For example, the interface appearance should reflect a property of the state of the system.

Conclusions

Within HCI in general, there is still a substantial gulf between the concerns of behavioural scientists and those of computer scientists. There have been some promising attempts to offer hands between communities but these offers are still little understood or accepted. We propose that putting user requirements on a more precise footing will help bridge the gulf between the scientists/engineers.

In the paper we have illustrated the sorts of characteristics of interactive systems (visibility, predictability, consistency) that may lead to easier use or less human error prone behaviour. Whether or not these properties actually lead to these characteristics involves a broader interdisciplinary concern; we have alluded to some of the work that is continuing to bring this broader human behavioural context. The notion of interactor can be seen as a mechanism that forces the system designer to take a more user centred view and eases the expression of some user related properties. Given this type of specification, it then becomes easier to check that these properties hold and to recognise interface constraints as the engineer moves to implement the system. A continuing concern is to develop a more systematic understanding of how specification might be scoped so that human behaviour can be addressed more adequately. An aim then in the context of specification is to produce system specifications that are more readily accessible to designers and are more easily connected with behavioural techniques. There is a need to show how interactor specifications scale up and to demonstrate convincing case studies of the verification of system specification against user relevant properties.

There are important gaps in the sorts of analyses that are being pursued here. For example, there is little attention paid to real-time properties (with the recent exception of an unpublished workshop on time at the University of Glasgow) although formal approaches to real-time systems abound. It is still unclear whether there are special characteristics of multi-modal systems such as virtual reality systems and what user requirements are relevant here. A recent paper from the Amodeus 2 group has made an initial step in this direction (Duke and Harrison, 1994; Duke, 1995). There is also little attention to group interaction. Recent work by Dix is promising in that it concerns a temporal logic in which liveness and safety properties may be expressed in terms of single users or groups of users (Dix, 1994).

Acknowledgments

Much of the work reported in this survey has been carried out within CEC funded projects (Esprit Basic Actions Amodeus (3066) and Amodeus 2 (7040)). We thank Praxis Systems plc for giving us access to system descriptions that have enabled us to check our techniques in a industrial setting.

References

Alexander, H. (1987). *Formally-Based Tools and Techniques for Human-Computer Dialogues.* Ellis Horwood Ltd.

Atkinson, W. and Cunningham, J. (1991). Proving properties of safety-critical systems. *Software Engineering Journal.*

Baber, C. and Stanton, N. (1994). Task analysis for error identification: a methodology for designing error-tolerant consumer products. *Ergonomics*, 37(11):1923–1941.

Blandford, A. and Young, R. (1993). Developing runnable user models: Separating the problem solving techniques from the domain knowledge. In Alty, J., Diaper, D., and Guest, S., editors, *People and Computers VIII*, pages 111–122. Cambridge University Press.

Bowen, J. (1992). X: Why Z? *Computer Graphics Forum*, 11(4):221–234.

Bramwell, C. (1995). *Formal aspects of the Design Rationale of Interactive Systems.* PhD thesis, Dept. Computer Science, University of York.

Coutaz, J., Duke, D., Faconti, G., Harrison, M., Mezzanotte, M., Nigay, L., Paterno', F., and Salber, D. (1995). Theoretical framework with reference model and multi-agent presentations. Technical Report D9, ESPRIT BRA 7040 Amodeus-2.

Diaper, D. (1989). *Task Analysis for Human Computer Interaction.* Ellis Horwood.

Dix, A. (1994). LADA - a logic for the analysis of distributed actions. In Paternò, F., editor, *Proc Eurographics Workshop on Design Specification and Verification of Interactive Systems, Italy*, pages 197–214. Eurographics.

Dix, A. J., Harrison, M. D., Runciman, C., and Thimbleby, H. W. (1987). Interaction models and the principled design of interactive systems. In Nichols, H. and Simpson, D. S., editors, *European Software Engineering Conference*, pages 127–135. Springer Lecture Notes.

Duke, D. (1995). Reasoning about gestural interaction. *Computer Graphics Forum*, 14(3).

Duke, D. and Duke, R. (1994). Towards a semantics for object-z. In Bjorner, D., Hoare, C., and Langmaack, H., editors, *Proceedings of VDM'90: VDM and Z!*, number 428, pages 242–262. Lecture Notes in Computer Science.

Duke, D. and Harrison, M. (1993). Abstract interaction objects. *Computer Graphics Forum*, 12(3):25–36.

Duke, D. and Harrison, M. (1994). A theory of presentations. In *FME'94: Industrial Benefit of Formal Methods*, volume 873 of *Lecture Notes in Computer Science*, pages 271–290. Springer-Verlag.

Duke, D. and Harrison, M. (1995). Mapping user requirements to implementations. *Software Engineering Journal*, 10(1):13–20.

Faconti, G. and Paternò, F. (1990). An approach to the formal specification of the components of an interaction. In Vandoni, C. and Duce, D., editors, *Eurographics 90*, pages 481–494. North-Holland.

Fields, B., Harrison, M., and Wright, P. (1995a). Applying formal methods to improve usability. In *IEEE Workshop on Software Engineering and Human Computer Interaction: Joint Research Issues, Sorrento*, number 896, pages 185–195. Springer-Verlag. Lecture Notes in Computer Science.

Fields, R., Wright, P., and Harrison, M. (1995b). A task centered approach to analysing human error tolerance requirements. In *Proceedings of IEEE Symposium RE'95*, pages 18–26.

Gaudel, M.-C. (1994). Formal specification techniques. In *16th IEEE International Conference on Software Engineering*, pages 223–232. IEEE.

Green, M. (1986). A survey of three dialogue models. *ACM Trans. on Graphics*, 5(3):244–275.

Green, M. (1987). Directions for user interface management systems research. *ACM Computer Graphics*, 21(2):113–116.

Green, T. R. G., Schiele, F., and Payne, S. J. (1988). Formalisable models of user knowledge in human-computer interaction. In van de Veer, G. C., Green, T. R. G., Hoc, J. M., and Murray, D., editors, *Working with Computers: Theory versus Outcome*, pages 3–46. Academic Press.

Grudin, J. (1989). The case against user interface consistency. *Communications of the ACM*, 4(3):245–264.

Hall, J. (1990). Seven myths of formal methods. *IEEE Software*, pages 65–68.

Harel, D. (1987). Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, 8:231–274.

Harrison, M. (1992). Engineering Human Error Tolerant Software. In Nichols, editor, *Proceedings of the Z User Conference*, pages 191–204. Springer-Verlag.

Harrison, M. D. and Dix, A. J. (1990). A state model of direct manipulation. In Harrison, M. D. and Thimbleby, H. W., editors, *Formal Methods in Human Computer Interaction*, pages 129–151. Cam-

bridge University Press.

Hutchins, E. (1994). *Cognition in the Wild.* MIT Press.

Johnson, C. and Harrison, M. (1992). Using temporal logic to support the specification and prototyping of interactive control systems. *International Journal of Man-Machine Studies*, 37:357–385.

Johnson, P., Johnson, H., Waddington, R., and Shouls, A. (1988). Task-related knowledge structures: analysis, modelling and application. In Jones, D. M. and Winder, R., editors, *People and Computers IV*. Cambridge University Press.

Morgan, C. C. (1994). *Programming from Specifications*. Prentice-Hall International. 2nd Edition.

Myers, B. (1988). *Creating user interfaces by demonstration*. Academic Press.

Myers, B. (1990). A new model for handling input. *ACM Transactions on Information Systems*, 8(3):289–320.

Nardi, B., editor (1996). *Context and cognition: a theory of human computer interaction*. MIT Press.

Palanque, P. and Bastide, R. (1994). Petri net based design of user-driven interfaces using the interactive co-operating objects formalism. In Paternò, F., editor, *Proc Eurographics Workshop on Design Specification and Verification of Interactive Systems, Italy*, pages 215–228. Eurographics.

Payne, S. J. and Green, T. R. G. (1986). Task-action grammars: a model of mental representation of task languages. *Human-Computer Interaction*, 2(2):93–133.

Roast, C. (1993). *Executing Models in Human Computer Interaction*. PhD thesis, Dept. Computer Science, University of York.

Suchman, L. (1987). *Plans and Situated Actions: The Problem of Human Machine Interaction*. Cambridge University Press.

Took, R. K. (1990). Surface interaction: A paradigm model for separating application and interface. In Chew, J. and Whiteside, J., editors, *Proceedings of CHI '90*, pages 35–42. ACM Press.

Vissers, C. A., Scollo, G., van Sinderen, M., and Brinksma, E. (1991). Specification styles in distributed systems design and verification. *Theoretical Computer Science*, (89):179–206.

Young, R. and Whittington, J. (1990). Using a knowledge analysis to predict conceptual errors in text-editor usage. In Chew, J. and Whiteside, J., editors, *CHI'90 Conference Proceedings*, pages 91–97. Addison Wesley.