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* : $\mathbb{N}$

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. $queue = x \Rightarrow [recv.m]queue =$
$(x \upharpoonright \{m' \in \mathrm{ran}\, x \mid pr(m') \leq pr(m)\}) \frown [m] \frown (x \upharpoonright \{m' \in \mathrm{ran}\, x \mid pr(m') > pr(m)\})$
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 $\langle mesg \rangle$ 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.

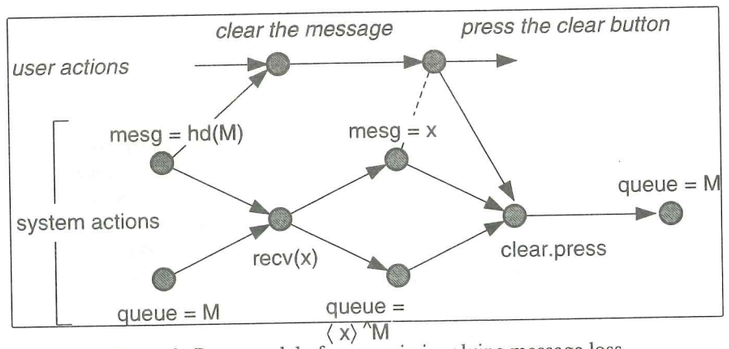

Figure 2: Poset model of a scenario involving message loss.

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.

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

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).

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

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.

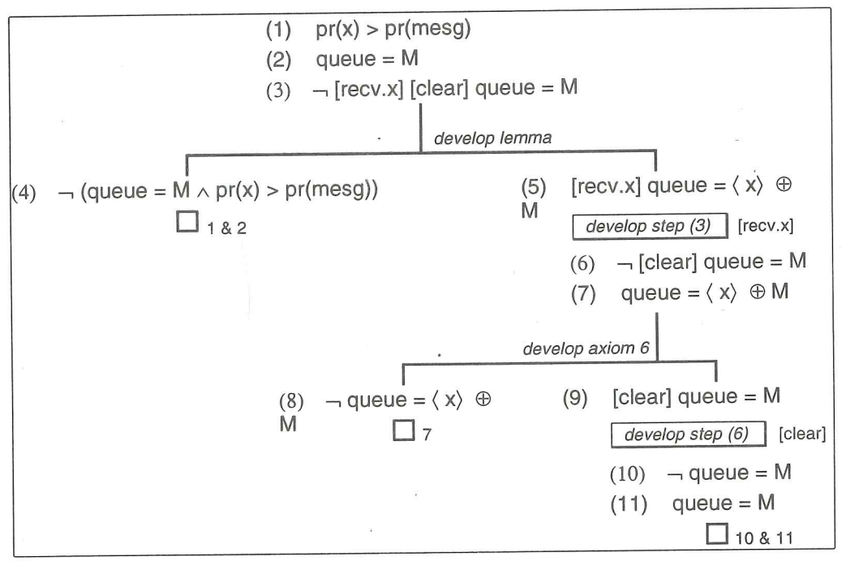

Figure 3: Tableau proof of simple property.

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.

INDUSTRY-ACADEMIC-GOVERNMENT COOPERATION TO BOOST TECHNOLOGICAL INNOVATION AND PEOPLE DEVELOPMENT IN SOUTH AFRICA

Tjaart J van der Walt
Executive Director: Industry
Foundation for Research Development
P.O. Box 2600, Pretoria, 0001

Abstract

Cooperation between industry, academia and government is increasingly viewed to be of key importance for economies to be globally competitive. In a world where research and technology development, and the diffusion and commercial exploitation of their outputs, form an important backbone to economic and social advancement, South Africa is challenged to realign its research, educational and industrial sectors to join hands in developing an appropriately skilled human resource base that will enable South African Industry to compete effectively in the world.

As an integral part of the paper, emphasis is placed on the unique challenges facing the "New South Africa" in terms of developing an innovative technological human resources base, within the framework of the Government of National Unity's Reconstruction and Development Programme (RDP). Special attention is given to capacity building among the disadvantaged through corrective actions, and the promotion of technology within the small, micro and medium enterprise (SMME) sector.

Furthermore, the paper provides an overview of the vision for a National System of Innovation (NSI) in South Africa in its role to boost technological innovation within an increasingly competitive global economy. Knowing that close cooperation between the business, Government and Educational sectors is key to the development of a vibrant and competitive workforce, hence a competitive economy, the paper also contains a perspective on the status of cooperative research in the natural sciences, engineering and technology (SET) in the development of leading and forefront expertise.

The two existing national technological innovation programmes are described, *viz*. The Support Programme for Industrial Innovation (SPII) and the Technology and Human Resources for Industry Programme (THRIP). Special attention is given to the latter programme, being a Joint Venture initiative between all key stakeholders in technology promotion in South Africa and managed for the Joint Venture by the Foundation for Research Development (FRD) together with the Department of Trade and Industry (DTI). THRIP is critically reviewed and its expanded vision contextualised within the NSI. Some factors and mechanisms are discussed to increase the gross national investment in research development, by both the public and private sectors. Such an effort should contribute to advancing market focus on the research and technology supply side, as well as promoting the involvement, ownership and long term vision of the market for technology and human resources in research.