# The Namur Principles: Criteria For The Evaluation of User Interface Notations *

Chris Johnson

Glasgow Interactive Systems Group (GIST), Department of Computing Science,
University of Glasgow, Glasgow, G12 8QQ.
Email: johnson@dcs.gla.ac.uk
Web: http://www.dcs.gla.ac.uk/∼johnson
Tel: +44 0141 330 6053

**Abstract.** Over the last decade, a wide variety of design notations have been proposed for the development of interactive systems. These range from textual formalisms, such as temporal logics, through tabular languages, such as the User Action Notation, to graphical approaches, including Petri Nets and Statecharts. These languages differ in the level of formality that they support. Some provide well-developed mathematical proof techniques. Others sacrifice formal reasoning in order to increase the 'accessibility' of the notation. Interface specification notations also differ in their descriptive qualities. Some can be used to represent cognitive requirements. Others focus more narrowly on user tasks or system behaviours. The rapid development of these new notations makes it difficult for designers to judge the usefulness of a particular formalism for their development tasks. This paper, therefore, identifies a set of principles or guidelines that can be used to evaluate the utility, or value, of interface notations.

## 1 Introduction

The last decade has seen the development of a dazzling variety of interface design notations. Textual formalisms, such as logics [16] and process algebras [21], have been proposed. Graphical notations, such as Petri Nets [1] and Statecharts [12], have also been applied to support the development of interactive systems. Tabular approaches, such as the User Action Notation (UAN) [14] and Sharratt's Memory-Cognition-Action formalism [22], have been applied to support interface design. These notations not only differ in their syntactic structure. They also vary in the amount of support that they offer for different aspects of development. For example, some languages focus upon the representation of operator requirements [6]. Others provide support for the software engineering of interactive applications [17].

* This document summarises the findings of a working group on the evaluation of interface notations. The members of the group are listed at the end of the paper.

### 1.1 The Limitations of Feature Accretion

The vigorous and rapid development of novel description techniques has not been mirrored by any widespread commercial exploitation. One of the reasons for this is that it is difficult to form any coherent view of the relative strengths and weaknesses of interface notations. For example, Petri Nets provide a powerful means of representing and reasoning about sequences of interaction (see Palanque, Paternó, Bastide and Mezzanotte, in this volume). These strengths must be balanced against a number of weaknesses. In contrast to the syndetic notations proposed by Faconti and Duke (this volume), Petri Nets do not represent the cognitive or perceptual requirements that interfaces place upon their users. Similarly, it can be argued that this graphical notation cannot easily be used to represent probabilistic or real-time properties. A commercial designer might, therefore, decide not to use Petri Nets during the development of a safety-critical interface. This decision would not be justified. Petri Nets can be extended to represent real-time requirements [4]. They can also be enhanced to include stochastic transitions and syndetic structures. The trouble is that in order for designers to make informed decisions about the utility, or value, of an interface notation they must know about the many different variants of the underlying formalism. This observation is not simply restricted to Petri Nets. There have been real-time extensions to XUAN [9] and probabilistic extensions to temporal logics [15].

### 1.2 The Utility of Interface Notations

A more fundamental question is; *are these extensions useful?* The ability to describe temporal and probabilistic behaviours is of little benefit if designers cannot learn how to use these techniques. It is relatively easy to introduce additional features to specification languages. The real issue here is what features are actually useful for the pragmatic development of interactive systems. A great limitation of research in this area is that there has been no systematic attempt to identify a coherent set of criteria that might be used to assess the overall value of a notation. This paper, therefore, presents a set of principles or guidelines that might be used to assess the strengths and weaknesses of user interface notations.

### 1.3 The Engineering Dimensions of Interface Notations

The criteria that are presented in the following sections were developed by a working group at the Namur conference. They are heuristic in the sense that they are the product of experience rather than empirical inquiry. This follows the model established by Green's cognitive dimensions of notations [11]. These dimensions provide a framework for assessing the 'usability' of notations whether or not they are used to support design. Our aims are less ambitious. The criteria presented in this paper focus more narrowly upon the engineering properties of languages that support the development of interactive systems.

### 1.4 Evaluating Criteria

Our approach differs from cognitive dimensions in a number of other respects. Green's work has mapped out a range of general concepts that can be used to discuss the cognitive attributes of notations. In contrast, we are keen to identify mechanisms and procedures that can be used to determine whether or not a notation satisfies our criteria. For instance, Green identifies the cognitive dimension of *viscosity*. This relates to the ease with which a designer might revise a structure or concept expressed in a language. Without some means of measuring this dimension, it remains a purely subjective concept. It is, therefore, open to a variety of different interpretations by different individuals. For example, some people view the Z notation as entirely viscous while others argue that the schema calculus is highly flexible [17]. Unless we have a clearer definition of viscosity then it will be difficult for designers to determine whether or not one notation is more flexible than another. One means of providing such a 'definition' is by identifying procedures that can be used to measure the viscosity of a notation. For example, designers might record the number of knock-on changes that must be made when a command is introduced into the specification of a well-known case study, such as an ATM (Automated Telling Machine or 'cash point').

### 1.5 Orthogonal Criteria

Another difference between our approach and the cognitive dimensions of notations is that we do not consider our criteria to be orthogonal. For example, Green argues that a notation may support many different *dependencies* in order to model complex situations. Constraint based notations exploit this dimension by modelling a system in terms of many different relations that must be preserved throughout interaction. The use of dependencies within a language can, however, only be achieved at the cost of *learnability*. It is difficult for novices to determine the impact of their actions if they do not understand the many different dependencies that may hold between interface components. In Green's view, therefore, designers must sacrifice one dimension in order to achieve another. In contrast, our criteria represent a 'wish list' where any trade-offs could only be made to the detriment of the notation. We would argue that a well designed notation ought to support high dependencies **and** be easy to learn. A weakness of many existing notations is that in order to gain benefits in one dimension, they simple ignore other attributes of the language. For instance, the development of non-monotonic logics has enabled designers to capture many different relationships between complex displays and input devices. These benefits have not been realised within commercial interface design because the developers of the notations have spent little or no time investigating the 'learnability' of the resulting formalism. In contrast, graphical notations provide many of the benefits of non-monotonic logic and may also ease the learning problems associated with textual formalisms (see Johnson, this volume).

## 2 The Namur Principles

The following section lists criteria that can be used to evaluate the utility of interface design notations. They are intended to map out the engineering dimensions against which designers might measure the success or failure of future generations of specification techniques for interactive systems.

### 2.1 Support for Abstraction

The first criteria to be applied to interface notations is whether or not they support the description and analysis of a design at many different levels of detail. The increasing complexity of many applications makes it vital that designers can abstract away from the implementation issues that often hide critical properties of interactive systems. This criteria can be applied in several different ways. For instance, the level of detail in a specification will change as development progresses. Additional information may gradually be introduced as the design nears implementation and testing. However, designers should not be forced to refine the level of detail in a specification in a uniform manner. Some areas of an interface may require a closer analysis than other areas. Safety-critical functions may be transformed to a lower level of abstraction than less important input or output sequences. Notations must enable designers to represent an interface at a level of detail that is appropriate to their development task.

As mentioned in the introduction, we are concerned to identify means of determining whether or not a notation satisfies our criteria. In order to assess a language's support for abstraction, we might develop a set of case studies that can be used to gauge the level of detail that can be introduced into a design. This is similar to the way in which researchers in the field of Information Retrieval have developed a set of standard documents that can be used to assess the precision and recall of new retrieval techniques. Informally, a number of case studies are already emerging as a means of bench-marking new specification languages. The ATM and light switch examples have appeared in many recent papers (Palanque, Paternó, Bastide and Mezzanotte, this volume). Unfortunately, these analyses often focus upon a single level of detail. They, typically, describe the case study at a single level of abstraction that is a very long way from implementation. We would, therefore, argue that these examples ought to show designers how abstract specifications can be brought closer to execution. A related point is that our case study material must be extended to include more complex examples than those cited above.

### 2.2 Support for Structuring

The second criteria that might be used to measure the utility of an interface notation is whether or not designers can use it to structure their description of an interface. This is important if formal and semi-formal languages are to support the development of 'real world' applications. The specification of a complex interactive system can be decomposed so that different design teams focus

upon different aspects of the interface. This approach will only be successful if the resulting components can be assembled to form a coherent description of the interactive system. Unfortunately, the problems of structuring interface descriptions have been neglected. The previous section has already argued that the benefits of interface notations are often illustrated using "toy examples". These examples provide well-structured but simple case studies. They ignore the more complex structuring problems that typify 'real-world' design. In consequence, there has been relatively little work into techniques that support the team-based specification of interactive systems [18].

We are unaware of any tests or benchmarks that have been designed to evaluate the structural power of specification techniques. This would seem to be a rich field for the integration of research in CSCW (Computer Supported Cooperative Work), software engineering and psychology. For example, tests might be devised to determine whether teams of interface designers can construct an interactive system from modular components. Each of these component might be produced by a semi-independent design process. This mirrors the way in which software engineers currently exploit libraries of code that have been written by other development groups [2]. Some steps have already been taken to exploit this approach. For example, Fischer, Lemke, McCall and Morch argue that it is possible to develop a large corpus of domain specific design rationale to guide the development of future systems [8]. Development teams might integrate the interface components that are described in these libraries with their own interface designs.

### 2.3 Visual Appeal

It might seem a trivial requirement that interface notations ought to be visually appealing. This criteria is, however, important if we are to persuade commercial interface designers to adopt new tools and techniques. This issue of visual appeal partly explains the success of tabular notations, such as UAN, and graphical formalisms, including Petri Nets. The problems faced by specification notations, in this respect, can be contrasted to those faced by the developers of programming languages. Many people are reluctant to exploit declarative approaches, such as functional or logic languages, simply because they look so different from more 'conventional' procedural implementations [20]. This does not imply that all specification notations ought to look like C++ code. However, the more difficult and alien a notation appears to a designer then the greater must be the perceived benefits of learning the formalism.

The visual appeal of a notation can be assessed through the use of qualitative questionnaires. Johnson (this volume) describes an evaluation that assesses subjective attitudes towards UAN, State Transition Networks and temporal logic. The findings indicate that graphical and tabular notations are more readily accepted than textual logics. However, comprehension questions also indicate that less errors are made with the textual notation. It is possible to argue that this reflects the orthogonality proposed in Green's cognitive dimensions. Increased visual appeal can be obtained at the cost of increased errors. In contrast, we

would argue that the low acceptance of temporal logic might inspire designers to search for alternative, more visually appealing means of expressing temporal constraints without sacrificing the precision obtained by the textual form.

### 2.4 Incremental Acquisition

Different notations offer different degrees of support to various stages of the learning process. For instance, programming languages such as Pascal may be easier for novices to understand than C or C++. Features such as automatic pointer de-referencing help people at the lower ends of the learning curve to focus upon key concepts rather than underlying mechanisms. Conversely the more advanced features of C or C++, such as unit linking, inheritance and pointer manipulation, help more experienced programmers to exploit the full power of these languages. Unfortunately, most interface notations have more in common with C than with Pascal [20]. There is little support for incremental acquisition where novices may gradually gain an understanding of the underlying formalism. There are relatively few manuals for interface specification notations.

Laboratory evaluations might be conducted to determine whether a notation supports incremental acquisition. These tests might be designed to determine how long it would take a designer to learn how to perform a series of basic tasks using a particular notation. Such evaluations are necessary if companys to assess the training costs associated with a particular specification technique. Carroll argues that these costs must be carefully balanced against the perceived benefit of using the formalism [3]. Designers must be enabled to perform basic tasks with only a cursory training. This may involve the provision of support tools or learning environments just as training wheels are provided when we first learn to ride a bike. More advanced tasks should involve proportionally more effort. At this stage the training wheels may be removed and the designer is freed to explore the full power of the approach.

### 2.5 Tool Support

The previous criteria mentioned the need for tool support when designers are learning to apply a notation. Without such assistance it is difficult for individuals to ensure that they are applying a formalism in a coherent fashion. There are many other ways in which tools can be recruited to support the exploitation of formal notations. Firstly, environments can be used to directly develop prototype implementations from formal specifications [16]. This is important because mathematical specification techniques provide an extremely poor impression of what it would be like to interact with a potential interface. Secondly, tool support can be recruited to perform proofs over specifications. This provides designers with a semi-automatic means of checking whether certain properties do or do not hold for a particular design [13]. Thirdly, model checking tools can be used to search for particular situations that may or may not arise during the course of interaction. These tools increase the level of automation provided by theorem

proving systems and provide direct means of searching for particular scenarios of interaction.

The means of establishing whether tool support exists is relatively straightforward. The proponents of a particular approach need only refer to the existence of the system. This approach is already adopted in the closing sections of many papers on interface specification notations. We would, however, suggest more rigorous criteria. In particular, the developers of new techniques should provide concrete examples of the ways in which these tools might be used to support interface development. This is extremely important when systems are being 'borrowed' from other disciplines, such concurrent programming. It is not always clear how theorem proving and simulation techniques can be transferred from systems engineering to interface development. For instance, the liveness and safety properties that are, typically, established using theorem provers do not necessarily have any impact in terms of the usability of an interactive system.

### 2.6 Support for Error Identification

A further criteria for the assessment of interface notations is *how easy is it to spot any errors in a specification?* Design is not an error-free process and no language can guarantee that designers will not make mistakes. Some notations provide better support for the identification of errors than others. For example, state transition networks may hide errors deep within sub-networks. Less hierarchical notations, such as UAN, are arguably easier to review. A related point here is that the provision of automated tools and type checking systems can help to identify mistakes in a formal specification. This is true only providing that the system is reliable and that designers can be trained to use the tool correctly.

One means of assessing whether designers can identify errors in a specification is to provide them with a description that contains a mistake. An evaluation may then be performed to determine how long it would take a group of designers to identify the error. Initial investigations into this approach have been conducted by Johnson and Love and are currently being written up for publication. The main problem with such tests is that people frequently identify non-existent errors. A further problem is that designers may completely fail to identify an existing error. Further questions can be asked about the benefits of these evaluation technique. It is difficult for laboratory evaluations to faithfully simulate the peer reviews and design meetings that support the commercial development of complex user interfaces. A further problem is that 'toy examples' are frequently used in experimental evaluations. This is necessary in order to complete the evaluation within a reasonable time-scale. Most commercial designers cannot afford to spend long periods in experimental trials.

### 2.7 Clear Semantics

Any notation that supports the development of interactive systems must have a clear semantics. For some notations this may involve the use of formal analysis based around operational or denotational semantics [23]. For all notations, it

is critical that designers understand the intended meaning of the symbols and syntactic constructs within the notation. If this is not the case then there is a high probability that errors will be made through ambiguity and misunderstanding. Designers teams must share a common understanding of the semantics that support a language. If this is not the case then each individual may have a different understanding or interpretation of a specification [18].

Comprehension tests can be used to determine whether designers understand the semantics of a notation. This raises a number of problems. For instance, how can we factor out the analyst's skill in producing a specification? A poor comprehension result might indicate a weakness in the way that a specification was drafted rather than a problem with the semantics of the notation. Other problems relate to the selection of comprehension questions. Clearly, if we used an a-temporal notation to describe a highly dynamic, real-time interface then we might expect relatively low comprehension results. Conversely, if the same comprehension test was conducted using a temporal logic or a Petri Net notation then the results might be much better. Given these limitations, it is important to consider alternative means of determining whether designers can understand formal descriptions of interactive systems. For example, individuals might be asked to translate a specification into natural language or another notation. Understanding might then be tested by their ability to re-express the details of a design rather than through the answers to a limited set of questions.

### 2.8 Case Studies

Previous sections have argued that the utility of notations ought to be tested against a number of well known examples, such as the ATM. These case studies provide a basis for making clear comparisons between different notations. Unfortunately the existing studies, mentioned above, cannot easily be used to convince commercial designers about the benefits of specification notations. 'Toy' examples need to be augmented by case studies that would cause problems for more conventional development techniques. Greater emphasis needs to be place upon dynamic, safety-critical interfaces [10].

The provision of case studies can be demonstrated by the publication of well known examples. These must be supported by detailed comparisons between the proposed language and previous notations. Additional materials must be provided to explain how the case studies were developed. In particular, we are anxious to provide manuals that describe the detailed application of the notation. It is important to support notations with at least a minimal methodology if real-world designers are to effectively exploit interface specification techniques.

### 2.9 Clear Scoping

In order to select an appropriate notation it is important that designers have some idea of the scope of a formalism. By this we mean that they must be able determine whether or not a notation can describe particular properties of an interactive system. For example, it would not be appropriate to use classical

logic to describe interaction with a distributed CSCW application. The lack of sequencing would make it difficult to reason about data locking and synchronisation. Similarly, it would not be appropriate to use Petri Nets to represent menu driven interfaces that provide Undo facilities. This would have a profound impact upon the tractability of the specification because it would introduce additional transitions into every place in the network. Unfortunately, most papers on interface description techniques focus too narrowly on the strengths of a particular approach rather than their limitations. This makes it difficult to judge whether a particular notation will be successful for a particular interface.

The criteria of 'clear scoping' might be satisfied if designers can identify the strengths *and* weaknesses of a notation. Many of these costs and benefits are well known within the research community, although few papers have been written about them. There is a natural reluctance to expose the flaws and weaknesses of particular techniques. It is, however, important that designers have some means of obtaining such information. One means of doing this would be for researchers to express the degree of confidence that they have in the application of their approach for particular categories of interfaces. For instance, the proponents of production rules might express a high degree of confidence in their ability to represent reactive systems. They might be less confident in their ability to describe the complex sequences of interaction that characterise interaction with document processing systems.

### 2.10 Task Fit

Some notations actually prevent designers from thinking clearly about certain problems. This is apparent when people have to perform a complex transformation between real-world entities and the restricted types that are supported by particular notations. For example, it is not easy to see how the polling of three-dimensional input from a data glove might be conveniently expressed within the Z schema calculus. Many recent advances in interface specification techniques, including the York work on interactors, address this problem of 'task fit' between formal notations and interface objects [7]. Similarly, epistemic and syndetic approaches are easing the mapping between user characteristics and abstract specifications, see Duke and Hyde in this volume. However, there has been almost no work to demonstrate that these techniques are actually successful for 'real-world' designers performing 'real-world' development tasks.

The problem of measuring the 'task fit' for a particular notation is complicated because it depends upon the context of development. The use of a formal notation might involve high costs in terms of the transformation between real-world entities and mathematical abstractions. These costs might not be justified if designers were considering interaction with a spreadsheet. On the other hand, the expenditure might be acceptable if the spreadsheet controlled a safety-critical or financial application. In other words, the 'task fit' of the notation is determined by the context in which the notation is used. It is, however, possible to identify some general means of assessing the support which notations provide for modelling 'real world' entitles. For example, experiments might measure the

additional workload that designers incur when they have to translate informal concepts into a formal or semi-formal notation. This can be done by assessing the interference effects that arise from secondary tasks [19].

### 2.11 Social Value

Interface notations can also be assessed in terms of their social utility. By this we mean whether or not they help groups of individuals to achieve common objectives. Previous work on specification notations has focussed upon small teams of highly motivated, well educated individuals. These are not typical conditions for the commercial development of user interfaces. Further problems stem from the fact that most notations have only been used during the *construction* of interactive systems. Research has focussed upon the early stages of design. In contrast, commercial interface development may take place over many years. This raises particular problems for the team-based application of interface notations. The members of development groups will change over time. It is important that these new recruits can understand the previous work of their colleagues. Similarly, they must be able to use a notation to support the maintenance as well as the implementation of an interactive system.

Recent research into the psychology of programming has provided a range of techniques that can be applied to assess the 'social value' of interface notations [5]. Empirical tests provide means of determining whether formal documents can be used to coordinate the team based development of interactive systems. For instance, groups of designers may be given the task of organising the introduction of new commands or displays into an existing specification. Such approaches have much in common with the modular structuring tests mentioned in previous sections. Rather than focusing on the ability to compose specifications, these evaluations focus upon the control aspects. What are the consequences of allowing more than one author to simultaneously work on a formal or semi-formal description of a user interface?

## 3 Conclusion

The commercial exploitation of interface specification techniques has been disappointing. One reason for this is that research has focussed too narrowly on the generation of novel notations. Relatively little attention has been paid to the evaluation of these techniques. This makes it difficult for commercial designers and engineers to make an informed judgement about the strengths and weaknesses of a particular approach. In contrast, this paper has presented a set of criteria or principles that can be used to assess the utility or value of interface notations. These criteria can be summarised as follows:

- support for abstraction;
- support for structuring;
- visual appeal;

– incremental acquisition;
– tool support;
– support for error identification;
– clear semantics;
– case studies;
– clear scoping;
– task fit;
– social value;

These criteria focus upon the support that notations provide for the design and implementation of interactive systems. They are, therefore, less general than Green's earlier work on the cognitive dimensions of notations [11]. A second difference is that our principles are not orthogonal. In other words, designers should not sacrifice one criteria in order to satisfy another. Ideally, future generations of interface notations will satisfy all of the objectives listed above.

Many of the principles that have been advocated in this paper are already being investigated by researchers in software engineering. The development of theorem provers and techniques for increasing the modularity of specifications are research areas in their own right. In one sense, therefore, the notational problems that motivate the development of our criteria are no different from those that guide other areas of software engineering. It is important to note, however, that many of the criteria reflect the particular concerns of human computer interaction. Our emphasis upon team work and 'task fit' within the context of complex development tasks is intended to reflect the wider significance of these problems within the field of interface design. Unless we address these issues, we will continue to develop notations that can only be used by the people that designed them.

## Acknowledgements

This report is the product of discussions within the group that addressed the evaluation of interface notations. The members of the group were Remí Bastide, David Carr, Joe Hyde, Chris Johnson, Tom Moher, Philippe Palanque, Steve Reeves, Janet Wesson.

## References

1. R. Bastide and P. Palanque. Petri net objects for the design, validation and prototyping of user-driven interfaces. In D. Diaper, D. Gilmore, G. Cockton, and B. Shackel, editors, *Human-Computer Interaction—INTERACT'90*, pages 625–631. Elsevier Science Publications, North Holland, Netherlands, 1990.
2. B.W. Boehm, T.E. Gray, and T. Seewaldt. Prototyping vs. specification: A multi-project experiment. In *IEEE - Seventh Conference On Software Engineering*, pages 473–484. Computer Society Press, Washington, United States of America, May 1984.

3. J.M. Carroll. *The Nurnberg Funnel: Designing Minimalist Instruction For Practical Computer Skill*. MIT Press, Boston, United States of America, 1992.
4. P. Chretienne. Timed Petri nets: A solution to the minimum-time-reachability problem between two states of a timed-event graph. *Journal of Systems and Software*, 6(1-2):95–101, 1986.
5. B. Curtis and D. Walz. The psychology of programming in the large: Team and organisational behaviour. In J.M. Hoc, T.R.G. Green, R. Samurcay, and D.J. Gilmore, editors, *The Psychology of Programming*, pages 253–270. Academic Press, London, United Kingdom, 1995.
6. D. Diaper and P. Johnson. Task analysis for knowledge description. In J. Long and A. Whitefield, editors, *Cognitive Ergonomics For Human Computer Interaction*, pages 191–224. Cambridge University Press, Cambridge, United Kingdom, 1989.
7. D.A. Duce and D.J. Duke. Interaction, cognition and visualisation. In P. Palanque and R. Bastide, editors, *Design, Specification and Verification of Interactive Systems '95*, pages 1–20. Springer Verlag, Berlin, 1995.
8. G. Fischer, A.C. Lemke, R. McCall, and A.HJ. Morch. Making argumentation serve design. *Human Computer Interaction*, 6(3 and 4):393–419, 1991.
9. P. Gray, D. England, and S. McGowan. XUAN: Enhancing UAN to capture temporal relationships among actions. In G. Cockton, S. Draper, and G. Weir, editors, *People And Computers IX*, pages 301–312. Cambridge University Press, Cambridge, United Kingdom, 1984.
10. P.D. Gray and C.W. Johnson. A critical analysis of interface specification notations. In *The Design, Specification and Verification of Interactive Systems*, pages 113–133. Springer Verlag, Berlin, Germany, 1995.
11. T.R. Green. Cognitive dimensions of notations. In A. Sutcliffe and L. Macaulay, editors, *People And Computers IV*, pages 443–460. Cambridge University Press, Cambridge, United Kingdom, 1989.
12. D. Harel. On visual formalisms. *Communications Of The ACM*, 31(5):514–530, 1988.
13. M.D. Harrison. The role of verification. In P. Palanque and R. Bastide, editors, *Design, Specification and Verification of Interactive Systems '95*, pages 342–344. Springer Verlag, Berlin, 1995.
14. D. Hix and H.R. Hartson. *Developing User Interfaces*. John Wiley and Sons, London, 1993.
15. C.W. Johnson. A probabilistic logic for the development of safety-critical interactive systems. *International Journal Of Man-Machine Studies*, 39(2):333–351, 1993.
16. C.W. Johnson. Specifying and prototyping dynamic human-computer interfaces for stochastic applications. In J.L. Alty, D. Diaper, and S. Guest, editors, *People And Computers VIII*, pages 233–248. Cambridge University Press, Cambridge, United Kingdom, 1993.
17. C.W. Johnson. Using Z to support the design of interactive, safety-critical systems. *IEE Software Engineering Journal*, 10(2):49–60, 1995.
18. C.W. Johnson. Literate specification: Using design rationale to support formal methods in the development of human-machine interfaces. *Human Computer Interaction Journal*, 1996. Acceped and to appear early in 1996.
19. B.H. Kantowitz and P.A. Casper. Human workload in aviation. In E.L. Wiener and D.C. Nagel, editors, *Human Factors In Aviation*, pages 157–187. Academic Press, London, United Kingdom, 1988.

20. P. Mendelsohn, T.R.G. Green, and P. Brna. Programming languages and education: The search for an easy start. In J.M. Hoc, T.R.G. Green, R. Samurcay, and D.J. Gilmore, editors, *The Psychology of Programming*, pages 175–200. Academic Press, London, United Kingdom, 1990.
21. F. Peterno, M.S. Sciacchitano, and J. Lowgren. A user interface evaluation mapping physical user actions to task driven formal specifications. In P. Palanque and R. Bastide, editors, *The Design, Specification and Verification of Interactive Systems'95*, pages 35–53. Springer Verlag, Berlin, Germany, 1995.
22. B. Sharratt. Memory-cognition-action tables: A pragmatic approach to analytical modelling. In *Interact T90*, pages 625–631. Elsevier Science, North Holland, 1990.
23. J.E. Stoy. *Denotational Semantics : The Scott - Stratchey Approach To Programming Language Theory.* The M.I.T. Press, Cambridge, United States of America, 1981.

This article was processed using the LaTeX macro package with LLNCS style