Requirements For The Next Generation Of User Interface Specification Languages

Phil Gray and Chris Johnson

Department of Computing Science, University of Glasgow,
Glasgow, G12 8QQ.
{pdg,johnson}@dcs.glasgow.ac.uk

Abstract Many existing interface specification notations cannot capture the temporal properties that characterise interaction with distributed systems, such as the Internet browsers Mosaic and Netscape. This paper presents a range of formalisms that avoid this limitation. We present a spectrum of approaches that range from the purely textual constructs of branching time logics, through the mixed graphical and textual notation of Extended User Action Notation (XUAN) to the fully graphical representations of Petri Nets. It is concluded that each notation highlights different aspects of the interaction problems that arise when using distributed systems. It is also argued that these notations all suffer from significant usability problems. The paper ends by presenting an agenda for future research into the development of temporal notations for the design of human computer interfaces.

1 Introduction

The increasing popularity of Internet browsers, such as Mosaic and Netscape, has created a new user population who have little knowledge of the communications mechanisms that support distributed information retrieval [2, 17]. Issues of propagation delay, network bottlenecks and data fusion have little relevance to their tasks. The physical and engineering constraints of international communications networks prevent designers from eliminating these problems [4]. This has created a range of usability problems that, in many instances, frustrate attempts to exploit the Internet to its greatest potential. Unfortunately, many existing notations that might be used to represent and reason about these usability problems cannot easily be applied to analyse temporal properties of interaction.

1.1 The Limitations Of Existing Notations

Many notations that support the specification of user interfaces are poorly equipped to represent and reason about temporal properties of interaction. For example, a number of authors have been forced to develop complex extensions to the Z schema calculus in order to represent the flow of interaction over time [23]. Such enhancements create hybrid notations that require considerable skill and expertise. Designers must learn about temporal logics or Hoare's CSP in addition to the Z formalism [14]. There are further problems. Most existing notations provide non- formalists with little impression of what it would be like to interact with complex distributed systems [6, 10, 11]. This

paper explores these limitations in more detail. A series of case studies is presented. Existing interface specification notations are applied to represent and reason about interaction with dynamic, distributed systems. Our experience with these case studies is then used to identify requirements for future interface specification notations.

1.2 Outline Of The Paper

This section has argued that the recent growth of dynamic, distributed systems poses considerable challenges for existing interface specification languages. Section 2 builds upon this argument by describing the temporal properties that frustrate the use of Internet browsers, such as Mosaic and Netscape. Section 3 argues that first order logic must be extended with some notion of time in order to represent and reason about such problems. Unfortunately, it can be extremely difficult to trace the properties described by clauses of textual notations, such as branching time temporal logics. Section 4 introduces a formalism that avoids this limitation. User Action Notation provides tabular structuring mechanisms that focus upon system and operator requirements during complex tasks. This formalism is extended to represent complex temporal properties of interaction. The resulting XUAN notation still relies heavily upon the use of textual annotations as a means of representing temporal properties of interaction. Section 5 argues that these annotations can be reduced to a minimum by exploiting Petri Nets. The spatial cues provided by this notation can help non-formalists to interpret the requirements that are embodied within formal specifications. Section 6 performs a brief comparative analysis of the three notations: branching time temporal logics; XUAN and Petri Nets. A set of requirements are developed for future generations of interface design notations. Section 7 presents the conclusions that can be drawn from this work.

2 The World Wide Web

Figure 1 characterises the interaction architecture with distributed information retrieval systems over the Internet. Users exploit browser programs to request and retrieve information from remote sites. We are interested in these applications because they typify a growing range of systems that are introducing users to usability problems that rarely occur on stand-alone machines. For example, Internet browsers hide the details of the underlying networks from the user. This leads to unpredictability because without this information it is difficult to determine whether retrieval commands will be successful. Remote site failure prevents information from being returned to the browser. Communications bottlenecks can delay the transmission of data between the browser and the remote servers.

Figure 2 shows the user interface to the National Centre for Super-Computer Applications' Mosaic browser. This system and its commercial successor, Netscape, are used to illustrate this paper. Users typically request information by selecting labelled buttons, such as `Playing Backgammon On The Internet`. The browser then assembles the requested information by sending messages to local or remote servers. Graphical images, sounds, video clips and pages of text are referenced by what are known as Uniform Resource Locators (URLs). The URL of the page in Figure 2 is:

`http://www.statslab.cam.ac.uk/~sret1/backgammon/main.html`

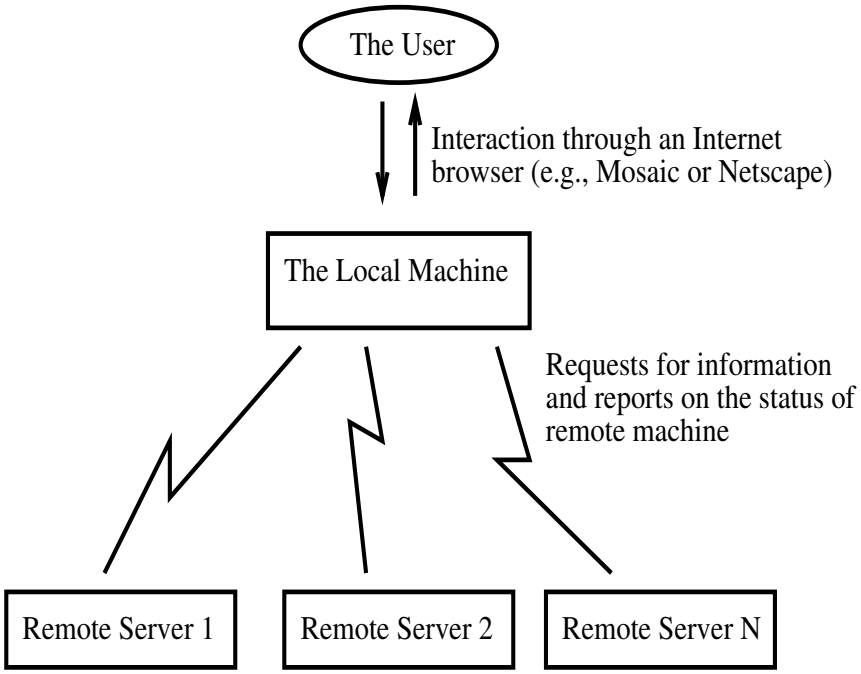

Fig. 1. Information Retrieval Over A Distributed Network

The http refers to the transfer protocol that is used to convey the information from remote sites. The machine that the information is retrieved from is specified after the protocol. In our example, it is the Internet site at `statslab.cam.ac.uk`. The location of the information on that machine is given after the address of the site; `~sret1/backgammon/`. The important point here is that users can access vast resources of information by the selection of labelled buttons. Users are not required to understand the underlying architecture that supports the various protocols and URL retrieval mechanisms.

3 A Textual Notation: Branching Time Temporal Logic

Unpredictable delays can occur when users attempt to access information over distributed networks. Bottlenecks can form in the passage of information back to the browser. These problems can increase or can be resolved as the loading on the network changes over time. A number of studies have shown that users find it difficult to complete retrieval tasks under such circumstances, especially if they only have a fixed amount of time at their disposal [20]. Other studies have shown that variable delays in system responses lead to frustration and error [16]. It is important, therefore, that designers can represent and reason about the impact of these problems. For instance, logic can be used to specify that a page will be displayed if the user selects an area of text that is linked to that page. The following clause helps to clarify the point that delays can occur during the retrieval process, $fetch(Http_address, Page)$, or during presentation, $present(Page)$, if the resource requires considerable processing:

$$display_page(Page) \Leftarrow$$

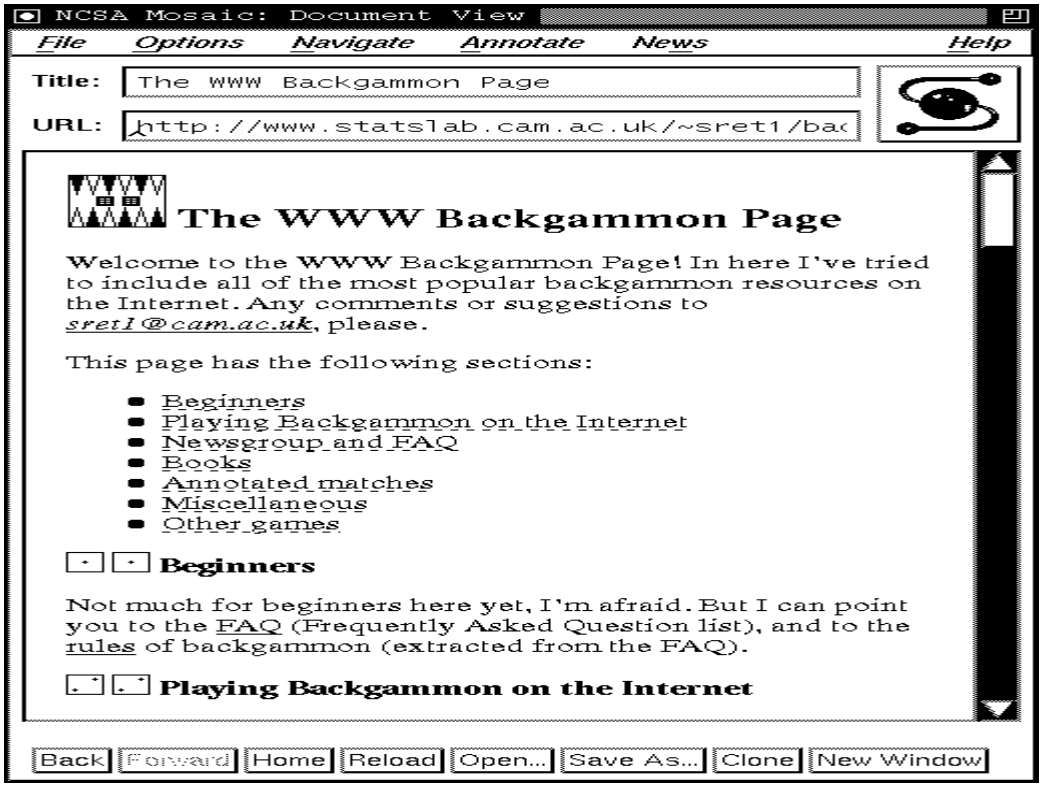

Fig. 2 The User Interface To The Mosaic Browser

$$user_selects(Text) \wedge linked(Text, Http_address) \wedge fetch(Http_address, Page) \wedge present(Page) \quad (1)$$

There are a number of limitations with such requirements. For example, there is no notion of sequence in classical logic; formula (1) would be true if the user requested the page after it had already been presented. This is clearly not what the designer intended. A further limitation for our purposes is that it does not capture the potential delay between the user selecting the text and the page being returned from a remote machine. Such problems can be avoided by introducing models of time into the notation. Clarke and Emerson's [3] Computation Tree Logic is an example of this approach. CTL is based upon a branching model of time; it can be used to describe alternative future traces of interaction. Johnson [13] discusses the formal underpinnings of the notation. Intuitive readings for CTL clauses will be given throughout the paper. The intention is to focus upon the application of the formalism to support interface development. Figure 3 provides an illustration of the way in which designers might view CTL's model of time. In this diagram two possible futures lead from the user selecting the text. In one path, the page is successfully retrieved from a remote site. In another path it cannot be retrieved because the remote site is unavailable. The following clause illustrates the way in which designers can exploit CTL's textual notation to introduce the sequential

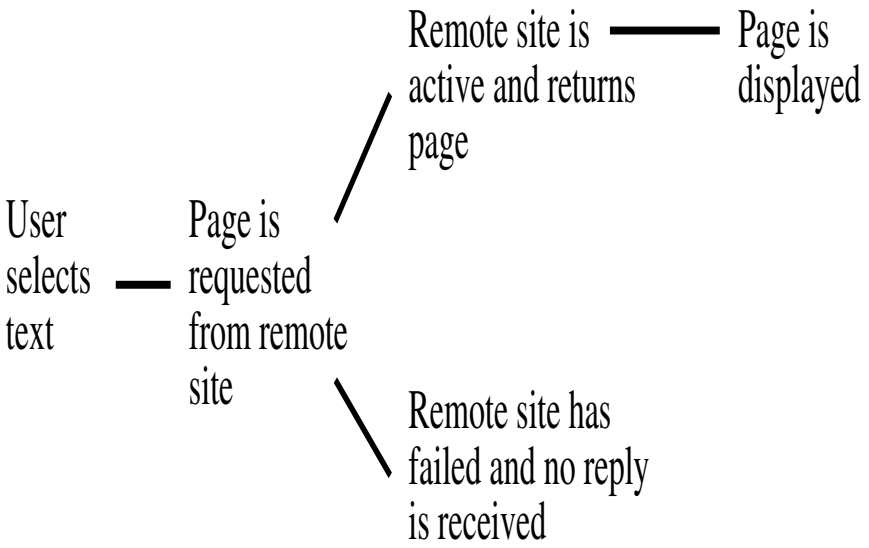

Fig. 3. A Branching View Of Possible Futures

information that was missing from clause (1). The AX operator specifies that the page is retrieved in all possible paths after the user selects the text. If this clause could be satisfied then the user would be able to predict that all requests would be immediately successful. This describes the situation on stand-alone machines:

$$\begin{aligned} &display_page(Page) \Leftarrow \\ &\quad user_selects(Text) \wedge linked(Text, Http_address) \wedge \\ &\quad AX(fetch(Http_address, Page) \wedge present(Page)) \end{aligned} \tag{2}$$

This formula does not, however, represent the potential delay that can lead to unpredictability in Web browsers. Given existing communications technology it is not possible to guarantee that all pages will be retrieved immediately after they have been requested. Site failure and heavy network loading can delay the transmission of information. It is only possible to specify that there are some traces in which the page is immediately presented after it has been requested. This requirement can be represented using the EX operator (read as 'there exists some next state'):

$$\begin{aligned} &display_page(Page) \Leftarrow \\ &\quad user_selects(Text) \wedge linked(Text, Http_address) \wedge \\ &\quad EX(fetch(Http_address, Page) \wedge AXpresent(Page)) \end{aligned} \tag{3}$$

Such clauses provide a framework for subsequent development. They can be used to identify critical phases during interaction with a complex system. In this case, designers must develop techniques that provide users with sufficient context to enable them to predict the behaviour of their application. For instance, it might be specified that a warning should be displayed as soon as the failure of a request is detected:

$$\begin{aligned} &failure_warning(Page) \Leftarrow \\ &\quad user_selects(Text) \wedge linked(Text, Http_address) \wedge \\ &\quad EX(notfetch(Http_address, Page) \wedge AXpresent(request_failure)) \end{aligned} \tag{4}$$

The way in which warnings, such as $request_failure$, are presented can have a profound impact upon the usability of an interface. For instance, Mosaic displays 'Unable to connect to remote host' when a request has not been acknowledged by the server. The warning is presented in a small bar approximately two thirds of the way up the window. This is easily over-looked. Designers might, therefore, seek alternative solutions. For instance, logic abstractions can be used to specify that the user must provide input to confirm that they have read about the failure of their request. This approach is exploited by Netscape. In the following clause, the CTL $\mathcal{U}$ operator (read as 'until') is used to describe the duration of the warning:

$$\begin{aligned} &acknowledge_failure_warning(Page) \Leftarrow \\ &\quad user_selects(Text) \wedge linked(Text, Http_address) \wedge \\ &\quad EX(notfetch(Http_address, Page) \wedge \\ &\quad AX[present(request_failure)\mathcal{U}user_selects(request_failure)]) \end{aligned} \tag{5}$$

Unfortunately, a number of usability problems restrict the utility of branching time temporal logics. It can be difficult for designers to exploit such notations; a training in discrete mathematics is required in order to understand the formal underpinnings of the language. It can also can be extremely difficult to construct the complex chains of interaction that arise between a system and its user. Individual clauses, such as $present(request_failure)$, may themselves contain further temporal operators. Each trace must, therefore, be painstakingly built from many individual clauses. The following section introduces a formalism that avoids these limitations.

4 A Semi-Textual Notation: XUAN

The Extended User Action Notation, XUAN [7] is a semi-formal notation for describing the actions of human and computer participants in interactive tasks. It is based on the User Action Notation [8].

4.1 The Tabular Layout of Task Descriptions in XUAN

Both UAN and XUAN organise the actions comprising a task into categories based on the agent which executes them and their function in the task. These categories define the columns of a tabular layout of the actions making up the task. For example, Figure 4 presents the UAN specification of clicking a typical screen button widget. A XUAN task is, like a UAN task, a set of the actions comprising the task, separated out into columns (or types) for some communicative purpose. However, the categories have been systematised in order to make clearer their roles and their relationship to one another[1]. The left/right-top/down ordering of the table elements may be viewed as a partial representation of temporal ordering on actions and the columnar organisation as modelling the task agents.

One consequence of treating user and system function similarly in the task description is that the somewhat ad hoc nature of system annotations of user actions is replaced by a symmetrical treatment of each task agent. Both perform visible actions and

[1]The UAN tabular layout is used as one way of writing a XUAN description. The description can also be expressed entirely textually; with only a 1D rather than a 2D ordering of the actions [7].

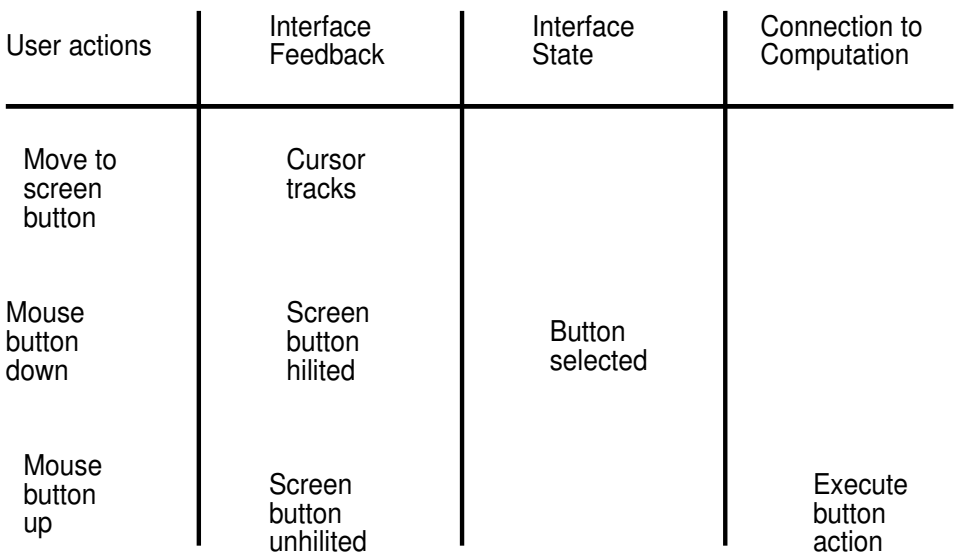

Fig. 4. UAN Task Description

(hypothesised) invisible internal actions. These internal actions for the system have already been considered (dialogue state, application action); those for the user are similar (changes to short term memory, long term memory, planning and other cognitive actions). From this standpoint XUAN resembles Memory-Cognition-Action tables [22].

Within a column, the temporal ordering among elements in the action set is specified using the *UAN action language*. For example, if task T contains the user action set a1,a2, the relationship of strict sequence would be expressed by:

```
a1, a2 (usually separated by a newline in UAN)
```

Order independent execution of the set (i.e., all must be executed, but in any order) is shown with the operator '&' :

```
a1 & a2
```

A full description of the action language can be found in Hix and Hartson [8] but briefly the main temporal relations are:

```
,               strict sequence
&               Order independence
||              Concurrent with
->              Interruptible by (as indicated by direction of arrow)
<->             Mutually interruptible
<|>             Interleavable
```

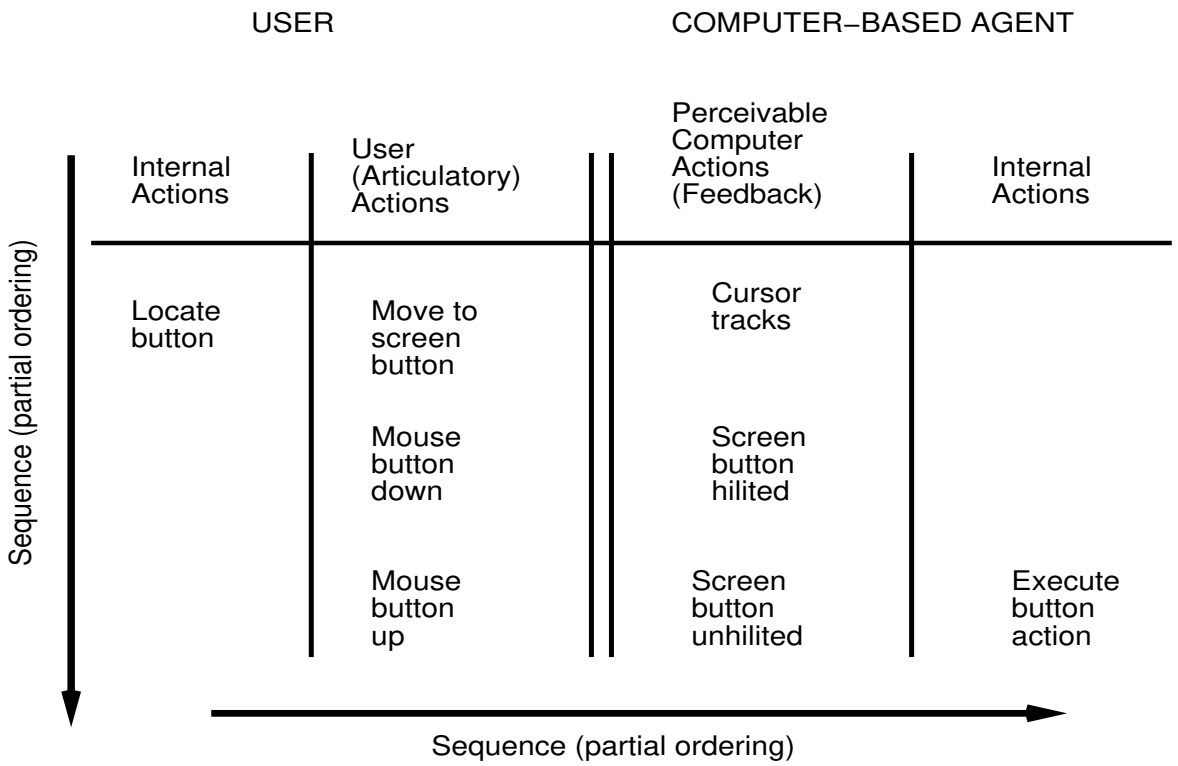

Fig. 5. XUAN Task Description

User actions are either primitive actions, typically manipulations of physical input devices (pressing a key, moving a mouse), or tasks. Additionally, and optionally, a UAN action specification may be annotated with information about system feedback (viz., perceivable changes in system state), non-perceivable changes to user interface state and application-significant operations. This approach, although possessing the virtues of simplicity and readability, is overly restrictive when attempting to deal with complex temporal relationships between actions in different categories. In particular, the tabular layout assumes that each feedback element (immediately) follows the action to which it is attached. Consider specifying type-ahead or mouse-ahead using this UAN table:

```
k1              echo k1
k2              echo k2
```

If we follow strict left-right/top-bottom sequencing, then type ahead would not be possible, since k2 must occur after echo k1. Instead, we require partial orderings of the actions:

```
k2 follows k1
echo k1 follows k1
echo k2 follows k2
```

This set of temporal constraints can be satisfied by an interaction in which type-ahead may occur; that is; the constraints may be true even though

```
k2 follows echo k1
```

is false. In order to accommodate these features systematically in XUAN, we treat each assertion about the temporal relationships among actions as a temporal constraint partially definitive of the task. That is, a task may be defined as a tuple of (1) a set of agent actions and (2) a set of temporal constraints on that action set:

```
task = (agent actions, temporal constraints)
```

Some of the temporal constraints may be shown in the conventional UAN tabular format, using UAN operators and syntax. However, the set of constraints may also be represented as a list, each constraint self-contained. We can express the type- ahead example more succinctly with the following textual format:

```
TASK: keypress
actions = press key :  USER, echo key:  SYSTEM FEEDBACK
constraints = press key , (echo key || keypress)
```

Notice that in this specification it is only guaranteed that the echo for a particular key will follow that key pressing action. The temporal ordering of the press-echo pairs is left undetermined, thus allowing for any degree of type-ahead.

4.2 Dealing With System Delays in the World Wide Web

Type-ahead is a desirable feature in cases of unpredictable feedback delays when the user knows what is wanted and the system can accommodate the input regardless of the state of the interaction. However, when the system interpretation of the user action depends on the information feedback or on the result of previous user actions, problems can arise if the user is allowed to generate input too soon. This is especially a problem in distributed systems, like the World Wide Web.

Mosaic for the Macintosh, for example, allows a user to select embedded buttons while the system is still engaged in fetching the page in which the buttons are located. A user may see part of the new page, locate a button and attempt to select it before the entire page has been downloaded. Such premature user input is ignored, often resulting in disorientation. A XUAN description can help to elucidate the source of the problem. The actual sequence from the user's point of view (i.e., what the user thinks is happening) can be described as shown in Figure 6. However, a more accurate description of the interaction (i.e., what should happen) is shown in Figure 7. We can see that the problem has arisen because the user's second select button action has taken place too early. Typically, this is because the display of the new page begins, and hence button are visible, before the download is complete. The message which indicates that the download has been completed is quite small and may be overlooked by the user. Two solutions to this problem are immediately apparent from the task description, as illustrated in Figure 8. First, one could lock out user input until the download is complete, viz., disallow a select button action before the download complete occurs. The X version of Mosaic enforces this option. This does not provide any information

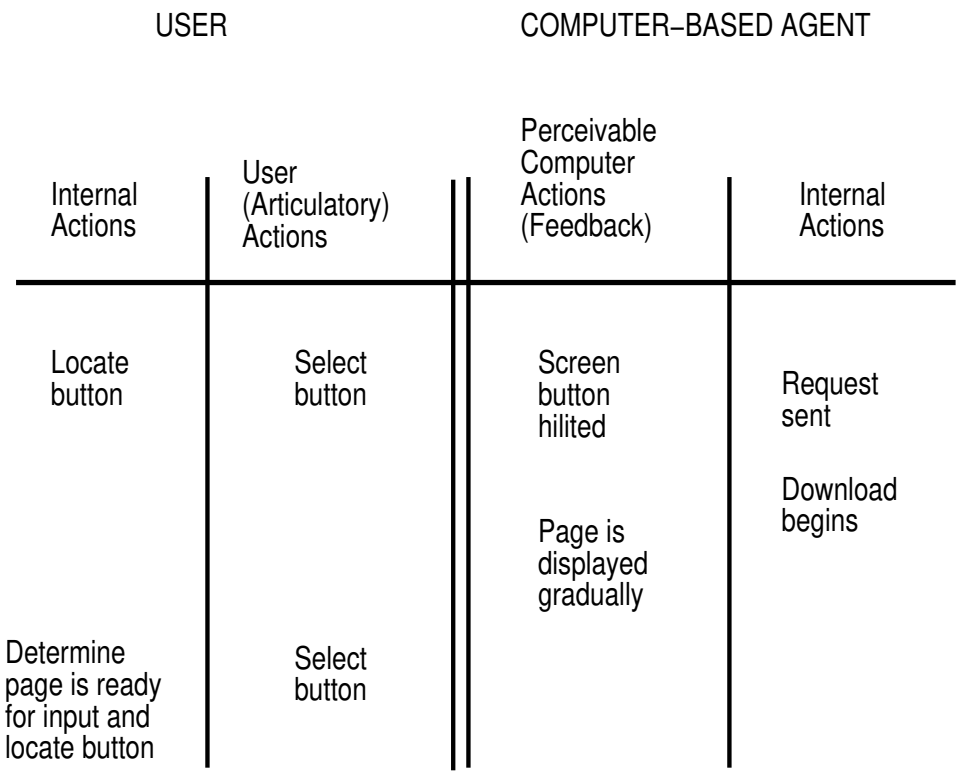

Fig. 6 Selecting an Embedded Button: The User's Mistaken View

from which the user may determine that the page has been completely downloaded. However, Mosaic also uses the cursor shape to provide feedback; during downloading the cursor has a watch shape and when the download is complete, it returns to an arrow. Since the cursor is usually in the focus of the user's attention during this task, it serves as a more effective indication of system state than the textual 'Download complete' message. This example demonstrates the way XUAN can be used to identify and reason about interaction problems arising from delays introduced by distributed systems like the World Wide Web. We have used task descriptions to specify what the user thinks is happening (a hypothetical but false description) and to compare that with what should happen, shown as alternative descriptions. An important element of the use of XUAN for such analyses is the ability to annotate actions and their relationship with design-significant assertions.

The tabular layout of UAN and XUAN is more accessible to a non-formalist than the textual clauses of branching time temporal logic. It is important to emphasise, however, that a number of usability problems still restrict the application of the XUAN notation. In particular, it can be difficult to trace prolonged periods of interaction through the columns of many different tables. Graphical notations, such as Petri Nets, provide means of reducing such problems [1].

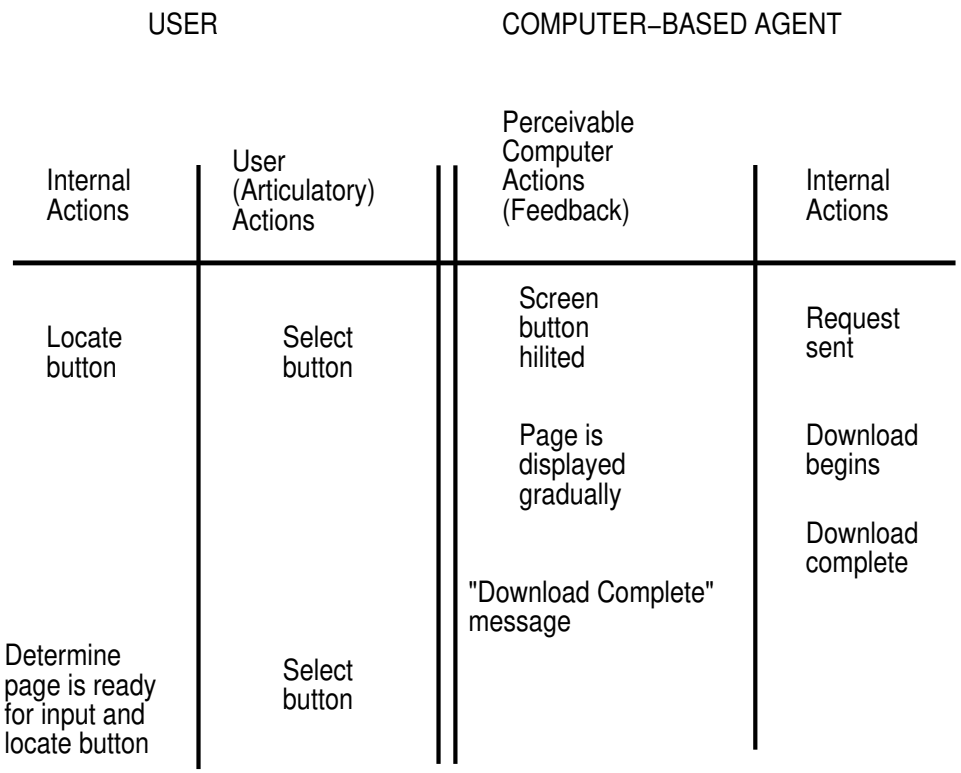

Fig. 7 Selecting an Embedded Button: What Should Happen

5 A Graphical Notation: Petri Nets

Petri Nets have been used to support the engineering of concurrent systems [15, 5]. They have also been used to represent human factors requirements. Bastide and Palanque [1] exploit Petri Nets to derive formal specifications of interactive systems at a very high level of abstraction. Johnson, McCarthy and Wright [9] have shown that Petri Nets can be used to represent the operator-system interaction which can lead to accidents. This section builds upon this work and shows how Petri Nets might be used to represent and reason about interaction with distributed systems.

Petri Nets represent the conditions that are satisfied during intervals of interaction [21]. When particular conditions hold, certain events can occur. If an event takes place then it can alter the state of the human-machine system. Changes in state are represented by the new conditions that hold after an event has occurred. These new conditions enable further events to take place. More formally, Petri Nets are directed graphs; $PN = (P, T, E, M)$. They consist of a set of places, P, transitions, T, edges, E and markings, M.

Places can be used to describe the properties that hold during particular intervals of interaction. For instance, a place might represent the fact that the system is presenting the "download complete" message. Alternatively, places can also be used to represent facts about the system operator. For instance, a place might be used to represent the fact that a user is reading the information displayed by the system.

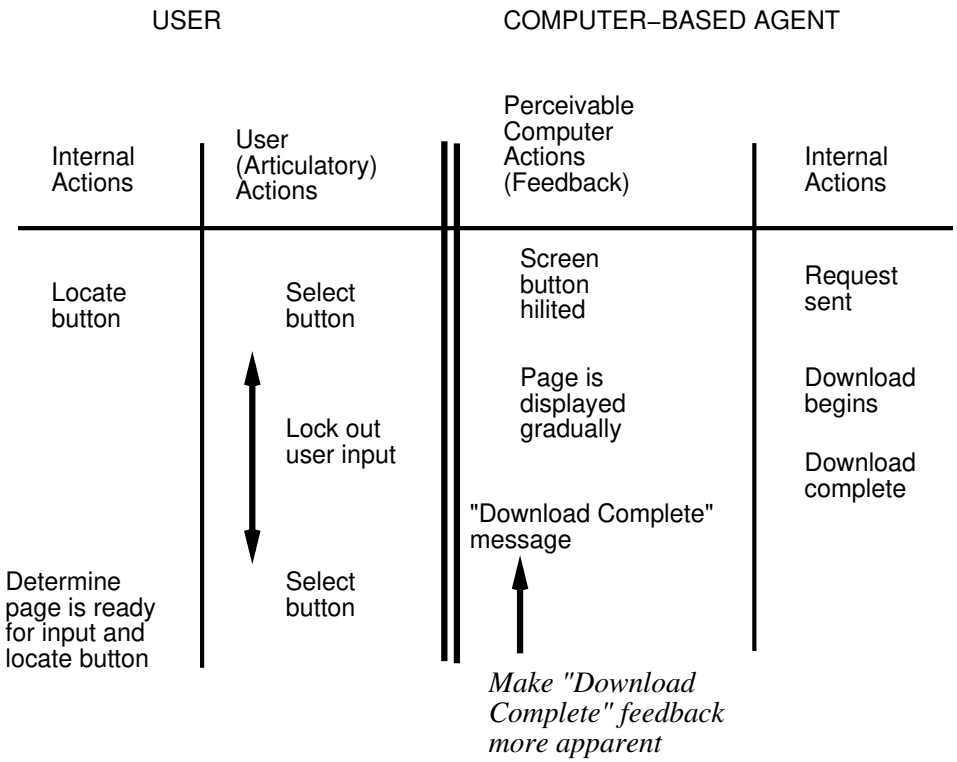

Fig. 8. Selecting an Embedded Button: Design Annotations

Transitions can be used to represent the events that alter the state of interaction. They can represent events initiated by both systems and their operators. For instance, a transition might be used to represent input events; such as the user selecting a button. They can also be used to indicate the events that occur when the system receives information from remote sites. Isolating these critical transitions provides a focus for subsequent human factors and systems analysis. Previous sections have argued that delays between a request and the presentation of information can have a profound impact upon usability.

Edges connect places to transitions: $E \subseteq \{P \times T\} \cup \{T \times P\}$. They can be used to form the chains of events and conditions that characterise the course of interaction with a distributed system. They can be described in terms of two functions. The function Op maps from each transition to its set of output places. The output places of a transition represent the conditions that hold after an event has occurred. For example, an output place might be used to represent the observation that the "download complete" message is displayed after information has been retrieved. An input place function, Ip, maps from each transition to the set of input places for that transition. The input places of a transitions specify the conditions which must hold for an event to occur. The input place of a transition might be used to represent the human factors observation that the presentation of the message led to the user requesting another page of information .

Petri Nets can be represented graphically. Events, or transitions, are shown as bars. Conditions, or places, are denoted by unfilled circles. Edges are shown as arrows linking places and transitions. Figure 9 shows a simplified graphical Petri Net representation for

interaction with a web browser. It is derived from the XUAN description in Figure 8. It illustrates the way that the spatial layout of Petri Nets can be used to represent temporal properties of interactive systems. In order to model the dynamic behaviour

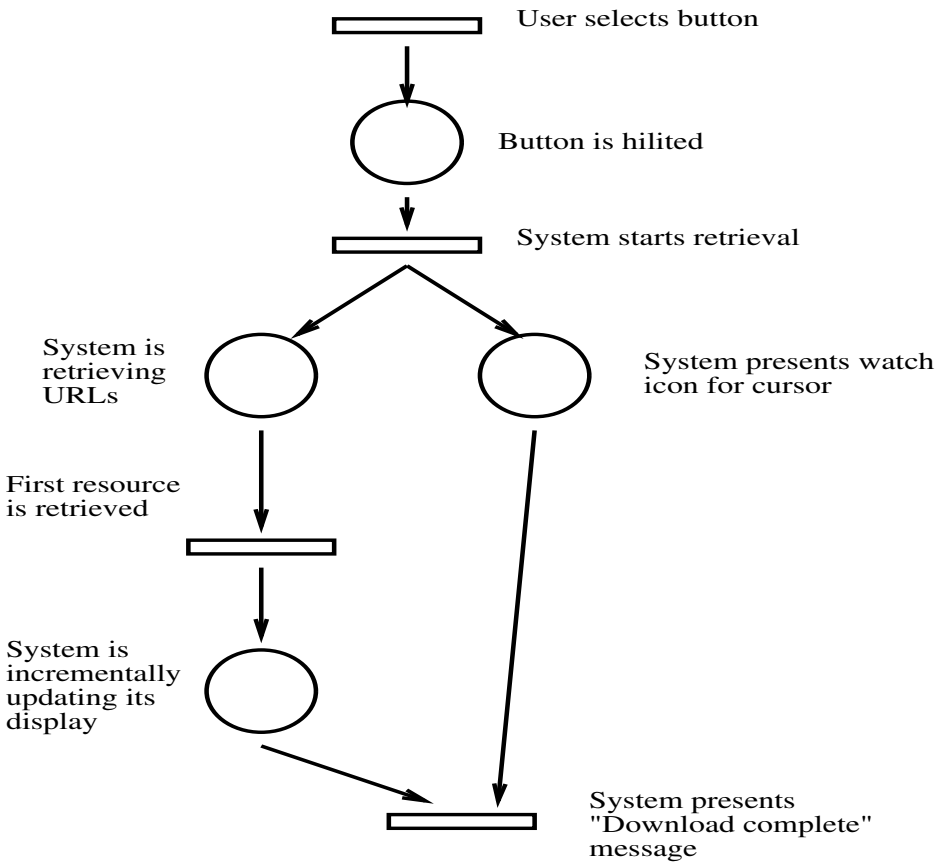

Fig. 9 A High Level Petri Net

of a human-machine system, tokens are used to mark those places in a Petri Net that are enabled. A place is enabled if its conditions hold. The tokens in a net are said to characterise a marking state and are denoted graphically by filled dots. For instance, Figure 10 is marked to show that the system is retrieving URLs and that the watch icon is being presented instead of the arrow for the cursor. Designers might alter the marking of a Petri Net to indicate the different conditions that hold for users and their systems. These walk-throughs can be used to simulate the sequences of events and states that are identified during critical traces of interaction. A transition can fire if all of its input places contain at least one token. After firing, a token is deposited in each of the output places of a transition. In Figure 10 it is possible for the transition indicating that the first resource is retrieved to fire. If this transition fired then the place indicating that the system is incrementally updating its display would be marked with a token. This, in turn, would enable the transition labelled 'System presents "download complete" message' to fire as both of its input places would now be marked.

The markings in a network can also be used to represent concurrent properties. For instance, users might make another selection in the interval between an initial request

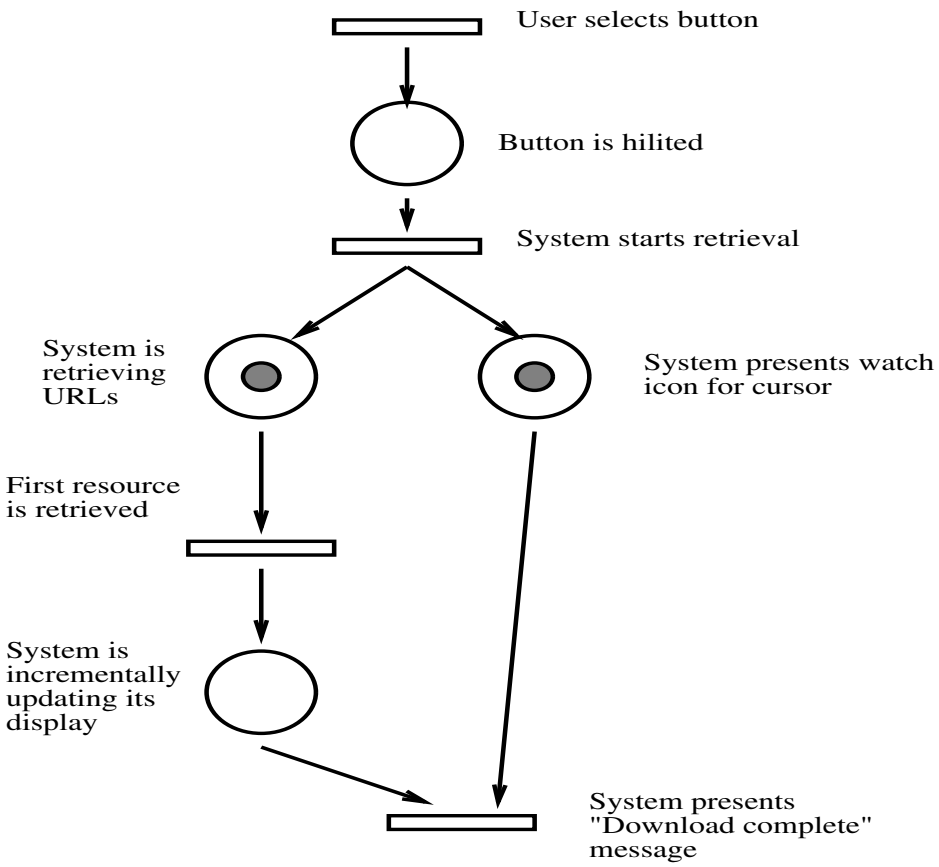

Fig. 10. A Marked Petri Net

and the retrieval of the resources associated with that request. In the XUAN of Figure 8 this was not possible because the user was locked out of the system. Figure 11 annotates the Petri Net to show an alternative approach. Here, the user has selected a second button before the tokens produced by the first selection have "cleared" the network. In other words, the user has ignored the watch icon and has typed ahead to make further retrieval requests. Experience with Computer Supported Co-operative Work systems suggests that arbitrary locking mechanisms can lead to frustration and error [18]. Expert users benefit from the potential offered by type ahead mechanisms. Our point here is not that one solution or another provides a panacea for usability problems in distributed systems. Rather it is that new generations of design notations must be used to represent and reason about such applications. Petri Nets address some of the problems that arise when tracing temporal properties in XUAN tables and temporal logic clauses. There are, however, limitations with this graphical formalism. Extensive tool support is required in order to construct and maintain complex networks. A further limitation is that it is almost impossible to represent and reason about undo facilities using Petri Nets. Designers must provide highly elaborate networks that reconstruct previous markings after the undo has been selected. This is a significant limitation if the graphical formalism is to represent and reason about changes to distributed data rather than simple browsing mechanisms.

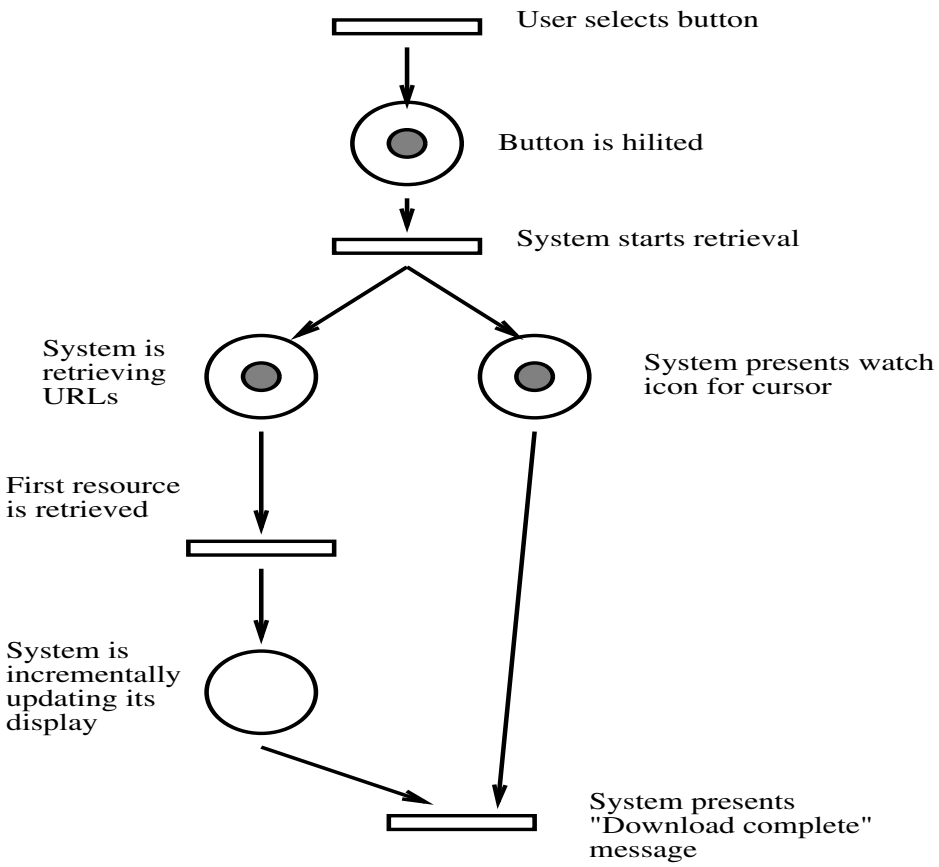

Fig. 11. A Marked Petri Net Showing Type ahead

6 An Agenda For Future Research

Previous sections have used a number of formalisms to represent temporal properties of interaction with distributed systems. These notations form a continuum from purely textual specification languages, through table based approaches to graphical formalisms. This section builds upon our analysis and identifies requirements for future generations of interface specification languages.

6.1 Requirements for Temporal Notations To Support Interface Design

An adequate notation must capture a variety of types of information about interaction:

- it must offer sufficient expressiveness so that any temporal property of the interaction that is relevant to designers may be described. Problems in representing and reasoning about undo clearly limit the application of Petri Nets;

- it must capture the salience of temporal properties during interaction; it must be relatively easy to identify usability problems from their temporal representation. This can be difficult in purely textual notations such as temporal logic;
- it must have, at least for the temporal aspects of the description, a well-found semantics. Designers must be able to determine that particular sequences of interaction are or are not logically possible. The lack of any complete, formal under-pinning for XUAN created significant problems in constructing the analysis presented in this paper;
- it must support the integration of temporal representations with other relevant properties of interaction, such as the state of the display and the physical properties of input/output media.

Figure 12 summarises the properties of the three notations examined in this paper against the requirements listed above: It should be stressed that the evaluations in

	Expressiveness	Salience	Semantics	Integration
Temporal Logic	High	Low	High	Medium
XUAN	High	Medium	Low	High
Petri Nets	High	Medium	High	Low

Fig. 12 Subjective Comparisons Between Notations

this table are subjective estimates based upon our experience in applying the notations. Some of the findings may have been biased by the particular applications under investigation. Of course, the list of desirable properties is incomplete as well.

6.2 Potential Solutions

From Figure 12, it can be seen that none of these notations satisfies all of our requirements. There are at least two potential means of addressing this problem:

- create an entirely notation avoiding the weaknesses of existing formalisms;
- develop a hybrid notation from the best features of several existing formalisms.

The first approach seems to be fraught with difficulty. Many years of research are required in order to identify and develop appropriate language constructs. Once this has been done, it remains a non-trivial task to built a consistent and complete semantics for those constructs.

In anticipation of this work, the second alternative would seem to offer a more cost-effective solution. For instance, the task allocations in XUAN tables might be used to identify critical incidents during interaction between a system and its user. The graphical structures of the Petri Net notation might then be used to analyse concurrency and synchronisation requirements. The additional semantic support of textual formalisms, such as temporal logics, might then be recruited if formal reasoning were required in order to establish safety properties. This integration might be achieved in a number of ways. One might annotate XUAN descriptions using Petri Nets. Places and transitions might be annotated with logic clauses. In fact, several of these approaches may turn out to be appropriate in different design contexts. Some work towards these hybrid approaches is currently underway at Glasgow and in Toulouse [19]. Many questions remain to be addressed by this research. In particular, tools and techniques must be developed so that designers can establish the consistency of multiple design notations for complex and dynamic user interfaces.

7 Conclusion

This paper argues that many existing approaches to interface design cannot easily be used to capture the temporal properties that characterise interaction with distributed systems. This paper introduces a range of specification notations that avoid this limitation. We present a spectrum of approaches that range from the purely textual constructs of branching time logics, through the mixed graphical and textual notation of Extended User Action Notation (XUAN) to the graphical representations of Petri Nets. It is concluded that each notation highlights different aspects of the interaction problems that can arise when using distributed, dynamic systems. It is also argued that these notations suffer from significant usability problems. The paper ends by presenting an agenda for future research into the development of temporal notations for human computer interface design.

Acknowledgements

NCSA Mosaic is a trademark of the National Centre for Computer Applications. Netscape is a trademark of the Netscape Communications Corporation. This work was conducted as part of the Temporal Aspects of Usability project SPG9201233, funded by the Joint Council Initiative in Cognitive Science and Human Computer Interaction.

References

1. Bastide, R. and Palanque, P. (1990) Petri Net Objects For The Design, Validation And Prototyping Of User-Driven Interfaces. In D. Diaper, D. Gilmore, G. Cockton and B. Shackel, editors, *Interact'90*, pages 625-631, Elsevier Science Publishers, North Holland.

2. Berners-Lee, T.J., Cailliau, R. and Groff, J.-F. (1992) *The World Wide Web*, Computer Networks And ISDN Systems, (25):454-459.

3. Clarke, E.M. and Emerson, E.A. (1982) The Design And Synthesis Of Synchronisation Skeletons Using Branching Time Temporal Logic. In D. Kozen (ed.) *Logic Of Programs 1981*, 52-71, LNCS 131, Springer Verlag, Berlin, Germany.

4. Coulouris, G., Dollimore, J. and Kindberg, T. (1994) *Distributed Systems: Concepts And Design*, Addison Wesley, Wokingham, United Kingdom.

5. Deng, Y., Perkusich, A., Figueired, J. and Chang, S.K. (1993) Integrating Software Engineering Methods And Petri Nets For The Specification And Prototyping Of Complex Information Systems. In *Proceedings of the 14th International Conference On The Application And Theory Of Petri Nets*, Chicago, USA, June 1993, pp. 206-223.

6. Dix, A. (1991) *Formal Methods For Interactive Systems*, Academic Press, London, United Kingdom.

7. Gray, P.D., England, D. and McGowan, S. (1994) XUAN: Enhancing UAN To Capture Temporal Relations Among Actions. In G. Cockton, S.W. Draper and G.R.S. Weir, editors, *People And Computers IX*, pages 301-312, Cambridge University Press, Cambridge, United Kingdom.

8. Hix, D. and Hartson, H.R. (1993), *Developing User Interfaces*, John Wiley and Sons, London, 1993.

9. Johnson, C.W., McCarthy, J. and Wright, P.C. (1995) *Using Petri Nets To Support Natural Language In Accident Reports.* To appear in Ergonomics.

10. Johnson, C.W. (1991) Declarative Graphics And Dynamic Interaction. In F.H. Post and W. Barth, editors, *Eurographics '91*, 195-207. Elsevier, North Holland.

11. Johnson, C.W. (1992) A Formal Approach To The Integration Of Human Factors And Systems Engineering, DPhil thesis, University Of York, York, United Kingdom. Available as technical report YCST 95/05.

12. Johnson, C.W. (1993) A Formal Approach To The Presentation of CSCW Systems. In J.L. Alty, D. Diaper and S. Guest, editors, *People And Computers VIII*, 335-352, Cambridge University Press, Cambridge, United Kingdom.

13. Johnson, C.W. (1993a) *A Probabilistic Logic For The Development Of Safety-Critical Interactive Systems*, International Journal of Man-Machine Studies, (38)2:333-351.

14. Johnson, C.W. (1995) *Using Z To Support The Design Of Interactive, Safety-Critical Systems.* The Software Engineering Journal, (10)2:49-60.

15. Kramer, B. (1991) *Introducing The GRASPIN Specification Language SEGRAS*, Journal Of Systems And Software, 15(1):17-31.

16. Kuhmann, W. (1989) *The Stress Inducing Properties Of System Response Times*, Ergonomics, (32)3:271-280.

17. Martin, J.L. (1993) *Travels With Gopher*, IEEE Computer, (26)5:84-7.

18. Miles, V., Johnson, C.W., McCarthy, J. and Harrison, M. (1991) Supporting Prediction In Complex Dynamic Systems. In D. Diaper and N. Hammond (editors), *People And Computers VI*, 133-144, Cambridge University Press, Cambridge, United Kingdom.

19. Palanque, P. (1995), Integrating UAN and Object Petri Nets, Personal Correspondence.

20. Pejtersen, A.M., (1989) *A Library System For Information Retrieval Based On A Cognitive Task Analysis And Supported By An Icon Based Interface.* In ACM SIGIR Proceedings, Boston. ACM Press, New York, United States of America.

21. Peterson, J.L. (1977), *Petri Nets*, Computing Surveys, 9(3):223-252.

22. Sharratt, B. (1990), Memory-Cognition-Action Tables: A Pragmatic Approach To Analytical Modelling. In D. Diaper, D. Gilmore, G. Cockton and B. Shackel, editors, *Interact'90*, pages 271-275, Elsevier Science Publishers, North Holland.

23. Sufrin, B. and He, J. (1990), Specification, Refinement And Analysis Of Interactive Processes. In M. D. Harrison and H. W. Thimbleby, editors, *Formal Methods in Human Computer Interaction*, pages 153-200, Cambridge University Press, Cambridge, United Kingdom, 1990.