Interactive Tools: Making UIMS Usable

Roger W. Ehrich
Deborah Hix
H. Rex Hartson

TR 86-30

INTERACTIVE TOOLS: Making UIMS Usable

Roger W. Ehrich, Deborah Hix, and H. Rex Hartson*

Department of Computer Science
Virginia Polytechnic Institute and State University
Blacksburg, Virginia 24061

Abstract

The earliest UIMS provided primarily run-time facilities for interface management and a set of programming tools for the development of application interfaces. Aside from the implementation requirements with which many tool designers have approached UIMS design, there are also methodological requirements that have been seriously neglected. One reason is that interface design methodology is poorly understood and rarely axiomatic. Nevertheless, it is important that we formulate methodological theories and provide UIMS with tools that support them. This paper proposes a storyboard metaphor for the conceptual design of human-computer interfaces.

Introduction

> There are two major metaphors for the nature of human-computer interaction, a conversation metaphor and a model world metaphor. In a system built on the conversation metaphor, the interface is a language medium in which the user and system have a conversation about an assumed, but not explicitly represented world. In this case, the interface is an implied intermediary between the user and the world about which things are said. In a system built on the model world metaphor, the interface is itself a world where the user can act, and that changes state in response to user actions. The world of interest is explicitly represented and there is no intermediary between user and world (Hutchins, Hollan, and Norman, pp. 93–94, 1986).

The Seeheim model of user interface management systems (Pfaff, 1985) is a well known structural model that describes the relationship of the end-user to the application system that the interface is to support. Of the three components of this model — presentation, dialogue control, and application interface model — the dialogue control component has been most thoroughly investigated. Green (1985) and others have recognized the importance of the role of human factors in the design of these three components and in the evaluation of descriptions of these components, but despite much debate, there have been no significant advances toward *making UIMS usable* in the conceptual design phase. Indeed, the implementation issues for wide classes of interfaces are themselves extremely complicated, and discussion of these has dominated over the discussion of methodological and representational issues.

Many approaches to interface design and specification have borrowed from related technologies elsewhere in the computer science field. For example, there have been numerous attempts to use BNF, state transition diagrams, and ATNs for specification of dialogue control, and their respective strengths and weaknesses have been well documented (Jacob, 1986). Many tools have been constructed to assist in generating interface specifications and in generating working interfaces from those specifications.

---

*The authors gratefully acknowledge the support of IBM, the Virginia Center for Innovative Technology, and the National Science Foundation under Grant IST 8310414.

These have largely been textual, as in the case of BNF specifications, and graphical, as in the case of state transition diagrams or ATNs. However, most of the notations themselves tend to describe the interfaces rather than to show directly how they would actually behave. Consequently even interactive graphical tools for constructing ATNs have the characteristic that the interface designer (whom we will call the dialogue author) is rather isolated or detached from the actual behavior or appearance of the interface being designed, and the best alternative is frequently to wait to try out the interface implementation and see what happens. But how, then, does the dialogue author actually compare interface behavior with requirements and make the necessary changes? And what cognitive support is available to help the dialogue author relate the notation to actual performance as the interface is being designed?

Let us further consider the nature of the tools used by the dialogue author to design interfaces. Hutchins, Hollan, and Norman, (1986) have used the term *directness*, to characterize an impression or feeling that the dialogue author would have about the designer's interface under the assumption that "directness results from the commitment of fewer cognitive resources" in the use of the interface. *Direct manipulation* (Shneiderman, 1983) is one of the most important techniques for providing directness since it has the characteristic that one is working not with descriptions of the target interface but rather with the interface objects themselves. For this same reason we have assumed for years that direct manipulation tools would be the most effective in building interface objects and have vigorously pursued their construction in AIDE (Hartson, Johnson, and Ehrich, 1984, and Ehrich and Williges, 1986). Unfortunately there are many situations in which direct manipulation is inadequate. For example, in cases where interface objects have parameters or contain information that cannot be bound until run-time, it is possible to deal directly only with examples of the interface objects at design-time. However, these are still tremendously useful to the dialogue author. In the following, interface objects that can be bound at design-time are called *static objects*, and those whose binding must be deferred until run-time are referred to as *dynamic objects*.

Cognitive Directness for the Dialogue Author

The dialogue author has at least four cognitive requirements during the design process that can be supported by UIMS tools. These are visualization of time sequencing, visual design, levels of abstraction, and cognitive directness of these three.

Time Sequence:. Dealing with time sequence in the design of interfaces is a fundamental aspect of interface design because end-user actions are time-ordered. This is, for the most part, the responsibility of the dialogue control component of the Seeheim model. Jacob (1986) points out that BNF is inadequate for visualizing time sequencing since that information is only implicitly encoded, but state transition diagrams, ATNs, and supervised flow diagrams (or SFDs) (Yunten and Hartson, 1985) do a very credible job in helping the dialogue author visualize time sequencing of interface events.

There are several resolution levels at which time sequencing needs to be visualized, and these are seen in the transaction model of Hartson, Johnson, and Ehrich (1984). The *transaction model* describes a hierarchy of components that every exchange between human and computer must have, and it forms the basis of the *Dialogue Management System*. The largest identifiable dialogue entity is called a *transaction*. This

transmits either a complete unit of validated information such as an entire command from the end-user to the application system, or possibly a unit of information from the application system to the end-user. Dialogue transactions must be sequenced with one another and with the computational procedures of an application system. Each input transaction consists of a sequence of ***interactions*** whose purpose is to obtain from the user one [possibly empty] token of information. Each interaction consists of three components — a prompt, an input, and a confirmation, any part of which can be null. Finally, each input consists of a sequence of ***lexemes***, whether generated by keystrokes or by more general user actions.

At the global control level, the dialogue author is concerned about the way in which the dialogue exchanges relate to the computational algorithms of an application. At an intermediate level, the dialogue author is concerned about how a time sequence of tokens forms a sentence of the end-user's interaction language and about how the arrival of those tokens is related to the prompts, echos, and confirmations seen by the end-user. Finally, at the lowest level the dialogue author is concerned about the user actions and their time sequence as the language tokens are produced from those actions.

Visual Design:. Descriptions of visual designs such as screens and prompts are usually part of the presentational component. Green has pointed out that since visual designs are graphic, it makes little sense to describe them textually with a programming language (Green, 1985), and that is a fundamental point. Wherever possible, direct manipulation tools must be provided by means of which graphical objects can be generated. We will return to this point shortly.

Levels of Abstraction:. One of the major problems faced by the dialogue author attempting to create the interface for a large application system is that of being able to understand the behavior of the system at all resolution levels. For example, the dialogue author might need to achieve a global understanding of the relationships at the highest levels between computational and dialogue entities. At a lower level, the dialogue author might need to know the names of the commands available in a system or the qualifiers permissible for a particular command. State transition diagrams and ATNs permit hierarchical structuring, but unlike supervised flow diagrams there is no associated methodology that provides organizational rules for forming the hierarchy. In the case of SFDs, the dialogue transaction model is the basis for the hierarchical decomposition of the interface. While there is much variability in SFDs across application systems, the dialogue SFDs all contain an identifiable hierarchy of levels that correspond to the structure levels within the transaction model.

Cognitive Directness:. Last but far from least is the requirement that the amount of cognitive processing on the part of the dialogue author be minimized. This is important whether dialogue objects are generated by direct manipulation or through the use of an intermediate notational stage. This is not necessarily a new concept; although he has omitted SFDs from his discussion, Jacob (1986) has done an excellent job of comparing various BNF and transition system notations from the point of view of the directness of their descriptive power. What is new is the recognition of the importance of directness and new attempts to take the issue further into account in the design of

UIMS and their design interfaces. This requirement has a number of special manifestations. For example, it allows interfaces to be visualized first in specific detail and then generalized from examples or storyboards into notations that describe the full range of possibilities for an interface, since that is a natural way for humans to approach design. We see no reason why adaptations of storyboards cannot also be found for describing languages in much the same way as they are used to describe graphic displays and screen layouts. Cognitive directness also implies that direct manipulation should be employed wherever possible in the design process. One of the obvious places to employ direct manipulation is in the preparation of the storyboards.

An Interface Example

In order to clarify the issues discussed above, an interface example is given next and it will be shown how ATNs and supervised flow diagrams (through AIDE) might handle the specification of the interface example. The example is a fictitious Employee Leave System with a menu that is used to select one of two forms, vacation or sick leave. Selections are made from the menu either by typing an item number, typing a prefix of an item name, or picking an item with a mouse click. On the vacation form, both the current date and the employee name are supplied automatically by the system. However, the date *is not* editable, whereas the employee name *is*, and the blank fields are shown in reverse video. The **exit** and **file and exit** control functions are activated from this form whenever there is a corresponding mouse click.

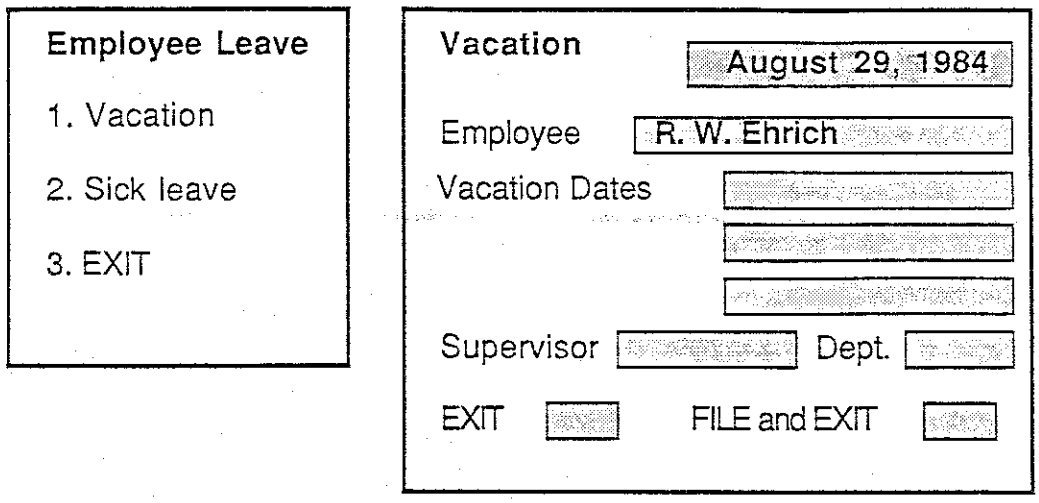
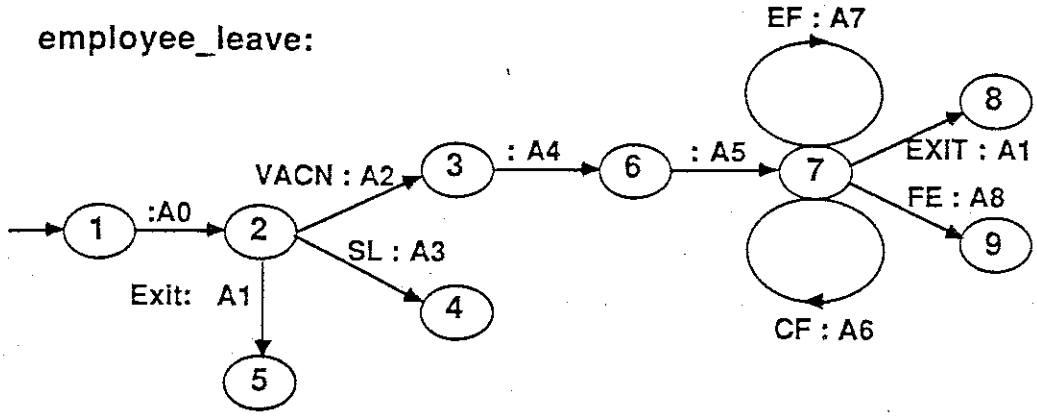


Figure 1. *Employee leave menu and vacation form.*

A possible ATN for a high level description of the Employee Leave System is given in Figure 2. The transition arcs all have the form, **input : action**, where the input may be a simple lexeme or a complicated input recognized by sub-ATNs. The semantic actions to the right of the colons may be primitives such as coded procedures, or they may refer to transition networks or other descriptions of the actions that are to take place. These actions are labeled A0 through A8 and are specified below the ATN itself.

The ATN in Figure 2 has the following interpretation. The arc into node 2 calls a procedure to display the employee leave menu, and three sub-ATNs are initiated to detect and identify a selection from that menu. If the vacation form is selected, the form



A0: **Act:** menu ("VACATION","SICK LEAVE","EXIT")
A1: **Act:** exit ()
A2: **Act:** form (vacation)
A3: **Act:** form (sick_leave)
A4: **Act:** date (date_field)
A5: **Act:** name (employee_field)
A6: **Act:** change_field ()
A7: **Act:** edit_field ()
A8: **Act:** file_and_exit ()

Figure 2. *ATN for Employee Leave System.*

is displayed. On this particular form, the date and name fields are treated differently from the others, and it was decided to show the invocation of semantic actions explicitly on the main ATN to bring these actions to the designer's attention. However, the time sequencing implied by doing so is irrelevant. Finally, the arcs from node 7 show form navigation inputs, field editing inputs, and form termination inputs.

The dialogue author might have a graphical tool for generating the ATNs and another for automatically generating a run-time system from the graphical description. This would then be linked with the appropriate semantic action routines and executed by the end-user. Typically these semantic action routines would be coded with the aid of programming tools such as language-sensitive environments and graphics packages. In terms of the Seeheim model, the ATN describes the dialogue control component, the application interface model consists of the description of the **exit** and **file and exit** procedures, and the presentation component is described by the graphics system code and perhaps by the user's manual that describes what it does.

Figure 3 shows the supervised flow diagram for the Employee Leave System. In SFDs the dialogue components are described by circles, computational components are described by squares, and further decomposable nodes are shown by squares with inscribed circles. Each circle corresponds to a dialogue transaction, and the arcs between nodes are labeled with predicates that are evaluated when the predecessor node is exited. The supervisory call to the file system supervisor is both a square and a circle because, although the node is primarily computational, there are dialogue subcomponents that describe the handling of file system error messages. Expansion of the **file and exit** node would reveal this structure. This upper level SFD reveals the sequencing of the transactions relative to one another and to the **file and exit** supervisor.

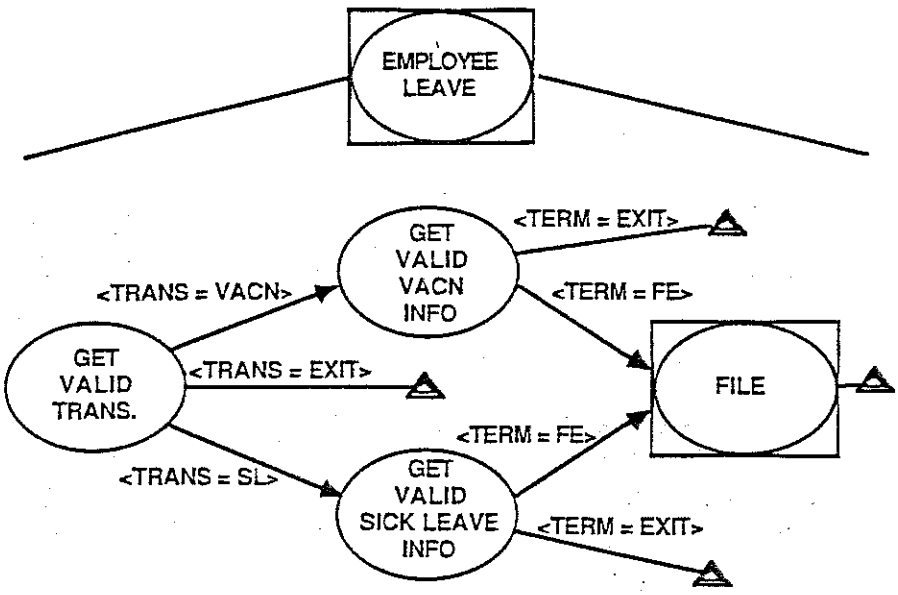


Figure 3. *SFD for Employee Leave System.*

Aside from major organizational differences, there are more subtle differences between the ATN notation and the SFD notation. For example, the predicates state the explicit conditions for arc transitions in the SFDs, whereas the same conditions are implicit in the ATNs. Furthermore, the nodes of SFDs correspond to dialogue or computational procedures that return a value to be used in the evaluation of the arc predicates, whereas in ATNs the states are associated only with a particular set of semantics, with no clear methodological delineation between dialogue and computation.

Tools and Storyboards

To provide cognitive support at design-time it is necessary to provide tools for producing dialogue objects and systems that far surpass the programming and notational systems that have been in common use. We have attempted to provide some of these for DMS, although there seems to be no reason why they cannot be provided for other systems as well. The following summarizes the direction of DMS development.

Using a graphical programming language (GPL), the dialogue author can help with top down development to the dialogue transaction level (circles in the SFDs). When the dialogue author picks a circle, AIDE provides an interactive environment for the development of dialogue objects such as displays, forms, languages, and prompts. AIDE contains specific tools for the development of dialogue objects, and these generate descriptions in *base programming language*, or just BPL, which is later compiled or interpreted by the run-time system. This BPL is actually a graphical language for producing the dialogue SFDs, which are the expansions of the circle nodes corresponding to dialogue transactions. These tool outputs can be edited by the GPL editor in the same way as the upper level SFDs.

As dialogue objects are selected for expansion using the AIDE tools, the first task of the dialogue author is to create storyboards by using corresponding AIDE tools. Figure 1 shows two of the storyboards for the Employee Leave System; it is striking to realize how much more understandable they are than the representations in Figures 2 or 3, no matter how much detail is given. That is the reason for using storyboards. Thus,

while the global system structure often undergoes top down development, much of the dialogue can be generated by bottom up development, allowing the dialogue author to begin in a natural way with concrete examples and end with the most general system specification. In fact, the entire system, including control and functional software, can be built upon sequences of storyboards. An operational information system has been constructed in just this manner by the DMS group.

If the dialogue objects are static, that is, completely definable at design-time, then the tools generate the final SFDs, written in BPL, for those objects. In the case of dynamic objects, there are so many different manifestations of dynamics that tools will frequently be inadequate. In such situations the interface can be writted directly in BPL, or, if the dialogue object is slightly different from that which an existing tool is capable of producing, the BPL SFDs generated by that tool can be edited to provide the desired features. This provides DMS with extensibility. Finally, the tools themselves are constructed in BPL, and the entire system can be used to generate new tools if the objects they produce are needed with sufficient frequency.

Several issues raised here are consequences of the concept of using interactive tools to define dialogue objects. For example, the distinction between static and dynamic objects first becomes important when using non-programming interactive tools because there will never be enough tools to cover the full spectrum of dynamics. In the same way, extensibility becomes an important issue so that tools can be constructed easily by tool designers to produce specific classes of interface objects. It is hoped that the approaches presented in this paper can lead to making UIMS more usable and more effective in an expanded role of facilitating the design process.

References

Ehrich, R. W. and Williges, R. C. (Eds.). (1986). *Advances in human factors/ergonomics, volume 2: Human-computer dialogue design.* New York: Elsevier.

Green, M. (1985). Report on dialogue specifications. In G. E. Pfaff (Ed.), *User interface management systems* (pp. 9–20). New York: Springer.

Hartson, H. R., Johnson, D. Hix, and Ehrich, R. W. (1984). A human-computer Dialogue Management System. In *Proceedings of INTERACT '84, 1* (pp. 57–61). London, England: IFIP.

Hutchins, E. L., Hollan, J. D., and Norman, D. A. (1986). Direct manipulation interfaces. In Donald A. Norman and Stephen W. Draper (Eds.), *User centered system design* (pp. 87–124). Hillsdale, NJ: Lawrence Erlbaum Associates.

Jacob, R. J. K. (1986). *Survey and examples of specification techniques for user-computer interfaces* (Tech. Report 8948). Washington, DC: Naval Research Laboratory, Computer Science and Systems Branch, Information Technology Division.

Pfaff, G. E. (1985). *User interface management systems.* New York: Springer.

Shneiderman, B. (1983). Direct manipulation: a step beyond programming languages. *Computer, 16* (8), 57-69.

Yunten, T., and Hartson, H. R. (1985). A SUPERvisory Methodology And Notation (SUPERMAN) for human-computer system development. In H. R. Hartson (Ed.), *Advances in human-computer interaction* (pp. 243–281). Norwood, NJ: Ablex.