The UAN: A User-Oriented Representation for Direct Manipulation Interface Designs

H. REX HARTSON, ANTONIO C. SIOCHI, and DEBORAH HIX
Virginia Tech

Many existing interface representation techniques, especially those associated with UIMS, are constructional and focused on interface implementation, and therefore do not adequately support a user-centered focus. *But it is in the behavioral domain of the user that interface designers and evaluators do their work.* We are seeking to complement constructional methods by providing a tool-supported technique capable of specifying the *behavioral aspects* of an interactive system—the tasks and the actions a user performs to accomplish those tasks. In particular, this paper is a practical introduction to use of the User Action Notation (UAN), a task- and user-oriented notation for behavioral representation of asynchronous, direct manipulation interface designs. Interfaces are specified in UAN as a quasihierarchy of asynchronous tasks. At the lower levels, user actions are associated with feedback and system state changes. The notation makes use of visually onomatopoeic symbols and is simple enough to read with little instruction. UAN is being used by growing numbers of interface developers and researchers. In addition to its design role, current research is investigating how UAN can support production and maintenance of code and documentation.

Categories and Subject Descriptors: D.2.1 [**Software Engineering**]: Requirements/Specifications—*languages*; D.2.2 [**Software Engineering**]: Tools and Techniques—*user interfaces*; D.2.10 [**Software Engineering**]: Design—*representation*

General Terms: Design, Human Factors, Languages

Additional Keywords and Phrases: Behavioral design, constructional design, human-computer interface, representation of interfaces, task analysis, user interface

1. INTRODUCTION

The past few years have seen an increase in the variety of software tools to support development of interactive computer systems. One common theme that has emerged is applying software engineering to the production of user interfaces. However, it has been realized that software engineering methods still do not necessarily produce user interfaces with high usability. Because of the difficulty of specifying and building user interfaces, the view of the user has been difficult to maintain. Developers know better how to construct a system than how to specify what it is to accomplish and how it is to interact with the user. The

Authors' addresses: H. Rex Hartson and Deborah Hix, Department of Computer Science, Virginia Tech, Blacksburg, VA 24061; Antonio C. Siochi, Department of Physics and Computer Science, Christopher Newport College, Newport News, VA 23606.

development community has excellent tools for the construction of software, but needs behavioral tools for design of user interfaces. We are seeking to complement constructional methods by providing a tool-supported technique capable of specifying the *behavioral aspects* of an interactive system—the tasks and the actions a user performs to accomplish those tasks. Integration of behavioral techniques into the constructional development process can lead to a more effective method of creating user interfaces that are at once useful and usable.

In particular, this paper is a practical, rather than theoretical, introduction to the *User Action Notation* (*UAN*) [28], a task- and user-oriented notation for behavioral representation of asynchronous, direct manipulation interface designs. UAN was initially intended as a communication mechanism between interface designers and implementers. Our goal was to be precise enough in representing the design of an interface so that details were not left to the imagination or best guess of its implementers. UAN is being used by growing numbers of interface developers and researchers, as discussed in Section 9.

In teaching use of UAN to interface designers, we have found that "by example" is the most effective approach. Therefore, we begin this paper with a simple example.

2. A SIMPLE EXAMPLE

Imagine a hypothetical description in a user manual for the task of selecting a file icon. (For purposes of introduction, only the user action portion of task descriptions are described in this section. The full task description notation are described in Section 5.) This task might be described in prose as

(1) move the cursor to the file icon;
(2) depress and immediately release the mouse button.

The user action portion of the UAN description for this task is as follows:

(1) ~[file_icon]
(2) M$\vee\wedge$

The ~ denotes moving the cursor, in this case into the context of the file icon. The second line represents depressing ($\vee$) and releasing ($\wedge$) the mouse button (M). Even this simple example illustrates the brevity and readability of UAN.

Let us describe another task, moving a file icon, which can be stated in prose as

(1) move the cursor to the file icon. Depress and hold down the mouse button. Depressing the mouse button selects the file, indicated by the highlighting of its icon.
(2) with the button held down, move the cursor. An outline of the icon follows the cursor as you move it around.
(3) release the mouse button. The display of the icon is now moved to where you released the button.

The user action portion of the corresponding UAN description is shown below:

(1) ~[file_icon] M$^{\vee}$
(2) ~[x, y]* ~[x', y']
(3) M$^{\wedge}$

Reading this task description, we again note moving the cursor into the context of the icon and depressing the mouse button. In the second line, ~[x, y] indicates movement of the cursor to an arbitrary point x, y on the screen. The * (Kleene star for expressing iterative closure in regular expressions) means to perform, zero or more times, the task to which it is attached. Thus, ~[x, y]* ~[x', y'] means to move the cursor to a succession of zero or more arbitrary points about the screen, ending at the point x', y'. Finally, in the third line the mouse button is released.

3. MOTIVATION FOR BEHAVIORAL DESIGN REPRESENTATION

Historically, and as a practical matter, many user interfaces have been designed by software engineers and programmers as part of the software of an interactive system. The result has been interfaces of varying quality and usability. Much work in the field of human-computer interaction has been directed toward new approaches to user interface development in hopes of improving quality and usability. Among these new concepts is the notion that design of software to construct a user interface is different from design of the interface itself, and that interface design has special requirements not shared by software design. A major distinction is that, while software design is properly system-centered, good interface design must be user-centered. Being user-centered means focusing on the behavior of the user and what the user perceives while performing tasks with the computer. To underscore this distinction we use the terms *behavioral domain* and *constructional domain* to refer, respectively, to the working worlds of the people who design and develop user interfaces and the people who design and develop the software to implement those interfaces.

In the behavioral domain one gets away from the software issues of interface design into the processes that precede, and are inputs to, software design. These processes include task analysis, functional analysis, task allocation, and user modeling. The people who are in various development roles must have a representation of the interface design to do their work. Thus, the concept of design *representation* itself is very important. High usability of an interface stems from a good design. Good designs depend on the ability of each person in a developer role, for example, designer, implementer, evaluator, customer, bidder, to understand and evaluate (and thereby improve) interface designs in the development process. Understanding and evaluating designs depends, in part, on the methods used to *represent* those designs. Design and representation are very closely related; design is a creative, mental, problem-solving process and representation is the physical process of capturing or recording the design.

It follows that each domain ought to have representation techniques tailored to its perspective and needs. As Richards, Boies, and Gould [25] state about tools for mocking up user interface prototypes, "few of these provide an interface

specification language directly usable by behavioral specialists." Many existing interface representation techniques, especially those associated with UIMS, are constructional. *But it is in the behavioral domain of the user that interface designers and evaluators do their work.* Thus, there is a need for behavioral representation techniques coupled with supporting interactive tools to give a user-centered focus to the interface development process.

Behavioral descriptions can be thought of as *procedures executed by the user.* Behavioral design and representation involve physical and cognitive user actions and interface feedback, that is, the behavior both of the user and of the interface as they interact with each other. Each behavioral design must be translated into a constructional design that is the computer system view of how the behavior is to be supported. Because UAN supports task description, which is important in many of the early interface development activities, it is suitable for use by behavioral specialists. UAN is used to describe how a user performs a task, but not how the system is implemented to interpret user behavior. Because UAN is in the behavioral domain, it should not be confused with, for example, specification languages for program behavior. Interface designs represented in UAN must still be translated, manually or automatically (see Section 11.3), into the constructional domain. Therefore, UAN is not a replacement for constructional representation techniques; it serves in a different domain.

One behavioral technique that has long been used both formally and intuitively is scenarios (or story-boarding) of interface designs. While this technique is effective for revealing a very early picture of interface appearance, because a scenario is an example (extension) of the interface, it cannot represent the complete design (intension). Scenarios can show much about screen layout, but do not adequately or efficiently show the user's behavior while interacting with the computer.

UAN is a task-oriented notation that describes behavior of the user and the interface during their cooperative performance of a task. The primary abstraction of UAN is a task. A user interface is represented as a quasihierarchical structure of tasks that are asynchronous, that is, sequencing within each task is independent of that in the others. User actions, corresponding interface feedback, and state information are represented at the lowest level. Levels of abstraction are used to hide these details and represent the entire interface. At all levels, user actions and tasks are combined with temporal relations such as sequencing, interleaving, and concurrency to describe allowable temporal user behavior. UAN is used to supplement scenarios, indicating precisely how the user interacts with screen objects shown in a scenario. The need for detailed scenarios and task descriptions is articulated by Gould and Lewis [9]: "Another method is to construct detailed scenarios showing exactly how key tasks would be performed with the new system. It is extremely difficult for anybody, even its own designers, to understand an interface proposal, without this level of description."

4. RELATED WORK

Techniques for representing user interface designs can generally be divided into the categories of behavioral or constructional, as described in Section 3 above.

The behavioral techniques describe interaction from the user's view and are generally task-oriented. These include the GOMS model [3], the Command Language Grammar (CLG) [18], the keystroke-level model [2], the Task Action Grammar (TAG) [23], and the work by Reisner [24] and Kieras and Polson [17]. Design of interactive systems, as with most kinds of design, involves an alternation of analysis and synthesis activities [13]. Most of the models just mentioned were originally oriented toward analysis; they were not intended to capture a design as it was being created but to build a detailed representation of an existing design with the purpose of predicting user performance for evaluating usability. Synthesis includes the activities that support the creative mental act of problem solving (creating new interface designs) and the physical act of capturing a representation of (documenting) the design. It is this kind of design representation that is referred to in the title of this paper. UAN shares the task orientation of these other behavioral models, but is presently more synthesis-oriented, because it was created specifically to communicate interface designs to implementers. In practice, most techniques mentioned above can be used to support synthesis as well, but typically do not represent the direct association of feedback and state with user actions. Also, many of these models, GOMS, CLG, and keystroke in particular, are models of expert error-free task performance in contiguous time (without interruption, interleaving of tasks, and without considering the interrelationships of concurrent tasks), not suitable assumptions for the synthesis-oriented aspects of interface design.

The GOMS model is very important to task analysis for interface design. The amount of detail generated in a GOMS description of an interface allows for thorough analysis but can be an enormous undertaking to produce. GOMS and UAN have similarities, especially at higher levels of abstraction where tasks are described in terms of sequences of subtasks. The keystroke-level model includes actions other than keystrokes, but at the same level of time granularity (i.e., single simple physical user actions). CLG formalism offers a thorough and broad framework for describing many aspects of a user interface. Description at each level (task, semantic, syntactic, and interaction) contains procedures, written in a language much like a high-level programming language. The work of Reisner with the ROBART graphics system interface uses an action language grammar and applies metrics to predict user performance to make comparisons of alternative designs and to identify design choices that could cause users to make mistakes. TAG is a formal, production rule-based description technique for representing mental models of users in task performance. Similarly, the work by Kieras and Polson is used to model user tasks and apply metrics to obtain measures of complexity of user knowledge required in performing specific tasks.

Among the earliest representation techniques for dialogue control flow (sequencing) are those based on formal, machine-processable production rule grammars represented in, for example, Backus–Naur Form (BNF) (e.g., Syngraph [22]). Grammatical representations tend to be behavioral because they describe expressions that come from the user, but they are difficult to write and read and are not used much now. Multiparty grammars [27] are an interesting extension to the production rule-based techniques. By representing the computer system as one of the interacting parties, the multiparty grammar allows direct association

of interface feedback to user inputs. The multiparty grammar, however, is not easily adapted to the variety of user actions found in a direct manipulation interface.

State transition diagrams (STDs) and their variations [15, 30, 31] are similar in expressive power to BNF, but show control flow explicitly in a graphical form. STDs are constructional because they are a representation that executes directly on the system (e.g., the system is in state X; if user input A is sensed, then the system makes a transition to state B). BNF and STDs are used to represent mainly state change information, but not interface feedback or screen appearance.

Event handlers [10, 14] are used to represent events that result from user actions. Because event handlers represent the system view of an interface (e.g., cause computational procedures to be invoked in response to an event), they are constructional. Event handlers offer an object orientation and have more expressive power than BNF or STDs [11]. Concurrent programming concepts have also been used to specify or implement the interface [4, 7].

Other work has involved specifying interfaces by demonstration (e.g., Peridot [19]). This is a novel and creative approach, but it produces only program code, with no other representation of the interface that conveys its design or behavior, or that can be analyzed. On the other hand, this approach is very suitable for producing rapid prototypes. An interface can also be generated from a set of application functions [21]. This is a quick constructional method of producing a default interface and is also useful for prototyping. Another technique combines two constructional techniques, state diagrams and object orientation [16]. In this case a mutually asynchronous set of state diagrams represents the interface, avoiding the complexity of a single large diagram. Another approach (UIDE [8]) involves building a knowledge base consisting of objects, attributes, actions, and pre- and post-conditions on actions that form a declarative description of an interface, from which interfaces are generated.

5. MORE ON THE UAN

5.1 Interface Feedback

Section 2 showed how to describe the *user actions* necessary for a task. Comparing the UAN description to the prose in the simple example of Section 2, we find that the prose contains feedback and semantic information as well as user actions. Such *interface feedback* information—interface responses to user actions—allows a more complete description of user and interface behavior, as seen in Figure 1. This task description is read left to right, top to bottom, and indicates that when the user moves the cursor to the file icon and depresses the mouse button, the icon is highlighted (file_icon!). As the user moves the cursor around the screen, an outline of the file icon follows the cursor, and upon release of the mouse button, the file icon is displayed at the new position. *Note the line-by-line association of feedback with the corresponding user action;* this level of precision is lost in the prose description, where actions and feedback are intermingled. For example, consider this description of selecting an icon:

(1) move the cursor to the icon.

(2) Click the mouse button and the icon will be highlighted.

TASK: move a file icon	
USER ACTIONS	INTERFACE FEEDBACK
~[file_icon] Mv	file_icon!
~[x,y]* ~[x',y']	outline of file_icon follows cursor
M^	display file_icon at x',y'

Fig. 1. UAN description of the task "move a file icon" with interface feedback in response to user actions.

TASK: select an icon	
USER ACTIONS	INTERFACE FEEDBACK
~[icon] Mv^	icon!

Fig. 2. UAN description of the task "select an icon."

TASK: select an icon	
USER ACTIONS	INTERFACE FEEDBACK
~[icon] Mv	icon!
M^	

Fig. 3. UAN description of the task "select an icon" showing, more precisely, relationship of feedback to user actions.

The corresponding UAN task description is shown in Figure 2. In the Macintosh® interface[1], however, highlighting occurs when the mouse button is depressed (rather than when it is clicked—depressed and released). Figure 3 shows how UAN can be used to represent, more precisely than in Figure 2, this correspondence between feedback and separate user actions in the sequence.

The feedback in Figure 3 is still not complete, however. In this case, selection (and, therefore, highlighting) of icons is mutually exclusive. This means that this task is technically the task to "select one icon and deselect all others." A highlighting action is applied to the icon (icon!) only if the icon is not already highlighted; highlighting depends upon the condition icon-!, which means the icon is not highlighted. The feedback in Figure 4 has been extended to include these two notions, where ∀ means "for all" and a colon is used between the condition and corresponding feedback.

If the designer feels that added information about dehighlighting other icons clutters the feedback description for an icon, abstraction can be used to hide those details. For example, the definition of highlighting (!) can contain the unhighlighting (-!) behavior for all other icons in the same mutually exclusive set.

® Macintosh is a registered trademark of Macintosh Laboratories.

[1] UAN is not limited to the Macintosh nor is it oriented toward any one specific graphical direct manipulation interface style. However, we have taken advantage of the popularity of the Macintosh desk top to illustrate use of the UAN.

TASK: select an icon	
USER ACTIONS	INTERFACE FEEDBACK
~[icon] Mv	icon-!: icon!, ∀icon'!: icon'-!
M^	

Fig. 4. UAN description of the task "select an icon" showing, more precisely, complete feedback.

TASK: move a file icon	
USER ACTIONS	INTERFACE FEEDBACK
~[file_icon] Mv	file_icon-!: file_icon!, ∀file_icon'!: file_icon'-!
~[x,y]* ~[x',y']	outline(file_icon) > ~
M^	@x',y' display(file_icon)

Fig. 5. UAN description of the task "move a file icon" with a more precise feedback description.

TASK: move a file icon		
USER ACTIONS	INTERFACE FEEDBACK	INTERFACE STATE
~[file_icon] Mv	file_icon-!: file_icon!, ∀file_icon'!: file_icon'-!	selected = file

Fig. 6. UAN description of the task "move a file icon" with interface state information.

While the feedback column of the example to move a file icon in Figure 1 is easy to read, it can be more precise. In particular, the symbology X > ~ is used to denote object X following the cursor. The exact behavior of the outline as it follows the cursor in the second line and displaying the file icon in the third line can be encapsulated as feedback functions, defined precisely in a single place. The task description then appears as shown in Figure 5.

Making the feedback description more formal detracts little from readability but improves precision and consistency of the resulting interface by not leaving these design details to the discretion of the implementer.

5.2 State Information

In addition to feedback, it may be necessary to include some *state information*, both for the *interface* and for its *connections to the computation*, associated with user actions. For example, if the interface design includes the concept of selectable objects, details about how this concept is applied to objects must be communicated to the implementer. Figure 6 shows just the first line of the task description from Figure 5 for moving a file icon. This shows that depressing the mouse button when the cursor is over a file icon causes the object represented by the file icon (i.e., the file) to be selected. This approach to semantics allows a connection to be made between an object being selected and its icon being highlighted.

TASK: move a file icon			
USER ACTIONS	INTERFACE FEEDBACK	INTERFACE STATE	CONNECTION TO COMPUTATION
~[file_icon] Mv	file_icon-!: file_icon!, ∀file_icon'!: file_icon'-!	selected = file	
~[x,y]* ~[x',y']	outline(file_icon) > ~		
M^	@x',y' display(file_icon)		location(file_icon) = x',y'

Fig. 7. UAN description of the task "move a file icon" with connection to computational semantics.

TASK: select a file icon		
USER ACTIONS	INTERFACE FEEDBACK	INTERFACE STATE
~[file_icon] Mv	file_icon-!: file_icon!, ∀file_icon'!: file_icon'-!	selected = file
M^		

Fig. 8. UAN description of the task "select a file icon."

TASK: select a file icon		
USER ACTIONS	INTERFACE FEEDBACK	INTERFACE STATE
file_icon-!: (~[file_icon] Mv	file_icon!, ∀file_icon'!: file_icon'-!	selected = file
M^)		

Fig. 9. UAN description of the task "select a file icon" with condition of viability (i.e., file icon is not already highlighted).

If the location of the icon is significant to the computational (semantic or noninterface) component of the application, the computational component must be informed, as shown in the lower right hand cell of Figure 7.

5.3 Conditions of Viability

Consider the description of the task of selecting a file icon given in Figure 8.

If we wish to show that this task applies only to a file icon that is not already selected, we make use of a *condition of viability*, which is similar to the condition applied earlier to the feedback, except here it is a condition applied to a user action or possibly an entire task. In Figure 9 the condition of viability is file_icon-!. The scope of the condition of viability is indicated with parentheses. A condition of viability acts as a precondition, or guard condition, that must be true in order for user actions within its scope to be performed as part of this task. A condition of viability with a false value does not mean that a user cannot perform the corresponding action(s); it just means that the action(s) will not be part of this particular task. The same action, however, might be part of another task in the overall set of asynchronous tasks that comprise an interface. Note

TASK: select a file icon		
USER ACTIONS	INTERFACE FEEDBACK	INTERFACE STATE
~[file_icon-!] Mv	file_icon!, ∀file_icon'!: file_icon'-!	selected = file
M^		

Fig. 10. UAN description of the task "select a file icon" with alternative (built-in binding) form for condition of viability.

TASK: delete file			
USER ACTIONS	INTERFACE FEEDBACK	INTERFACE STATE	CONNECTION TO COMPUTATION
~[file_icon] Mv	file_icon-!: file_icon!, ∀file_icon'!: file_icon'-!	selected = file	
~[x,y]*	outline(file_icon) > ~		
~[trash_icon]	outline(file_icon) > ~, trash_icon!		
M^	erase(file_icon), trash_icon!!	selected = null	mark file for deletion

Fig. 11. UAN description of the task "delete a file."

that the use of this condition as a condition of viability for the user action removes the need for its use with the feedback.

The term file_icon in the condition of viability is bound to the same term in user actions within its scope, like a bound variable in first order predicate logic. In Figure 10 the condition of viability (file_icon-!) from Figure 9 is written as a built-in binding (~[file_icon-!]), which is more concise and easier to read. In this form, conditions quite naturally provide specific instructions for user behavior, that is, move the cursor to an unhighlighted file icon.

5.4 Another Example

The task description in Figure 11 ties together many of the previous concepts; it represents one version of the task of deleting a file from the Macintosh desk top by dragging its icon to the trash can icon.

6. FURTHER DISCUSSION OF THE UAN

6.1 Actions Applied to Devices

At a detailed design level, UAN describes physical user actions that are applied to physical devices. Determination of the symbols to be used in representing an interface design requires identification of the devices and the operations that can be performed on them. For example, consider the class we call switch-like devices. These are devices that can be depressed and released, thereby causing transmission of a single character or signal. Switch-like devices include, for example, the mouse button (M), the control key (CNTL), the shift key (S), all special and function keys, and possibly devices such as knee switches, foot pedals, and "puff and sip" tubes. Pressing and releasing this kind of device is represented in UAN

with ᵛ and ^, respectively, as shown in the above examples. An idiomatic shorthand for clicking (depressing and immediately releasing) is ᵛ^, as in Mᵛ^, for example. Clicking on the left button of a three-button mouse can be indicated by M_Lᵛ^.

Another class contains devices from which user actions result in character strings. Examples are keyboards and voice recognition devices. Although keyboards are comprised of keys, individual keys are abstracted out of the description because the significant feature is the character string. An example of UAN for such devices is K"*abc*," the description of the user action of typing the literal string abc, and K (user_id), the description of the user action of typing a value for a string variable named user_id. In addition, a regular expression can be used inside the parentheses to specify the lexical definition of the variable to be entered by the user, for example, K (user_id = [A–Z][A–Z 0–9]+).

6.2 Cursor Movement

The mouse is composed of two or more devices, a cursor position controller and one or more buttons. The buttons are switch-like devices, described above. Unless it is important to address the details of how the user physically and cognitively interacts with the cursor controlling device, UAN represents user actions that cause *cursor movement* in terms of where the cursor is moved. At this level of abstraction (the level addressed in this paper), the cursor controlling devices have the same behavior in the sense that the notation ~[X] specifies, in a somewhat device-independent manner, that the user moves the cursor to the context of X, without an indication of how this is done or whether it is done by mouse, joy stick, track ball, arrow keys, touch panel, or by other means such as an eye tracker.

At a lower level of abstraction, pragmatic differences between devices cannot as easily be represented. It may be possible to produce detailed UAN descriptions of the task of using each device to move the cursor, but it would require perceptual, cognitive, and decision making actions, and perhaps even kinesthetics of the physical actions, all in a very tight feedback loop. While acknowledging the importance of carefully deciding the issues of device pragmatics, this paper assumes these issues are settled and that the implementer and the user know the device pragmatics.

6.3 Context of Objects

The UAN symbology ~[X] describes the user task of moving the cursor into the *context* of the interface object X. Moving out of the same context is denoted by [X]~. The context of an object is that "handle" by which the object is manipulated, such as the object itself. Other handles may include a rectangle circumscribed about the object or a small grab handle (e.g., of the kind used to manipulate lines and corners in MacDraw®). The context is modal in that it remains until explicitly changed. For example, in the expression ~[X] Mᵛ, it is assumed that the pressing of the mouse button occurs within the context of X.

® MacDraw is a registered trademark of Claus, Inc.

6.4 Feedback

UAN symbology describing *feedback* includes X! for describing the highlighting of object X and X-! for its dehighlighting. X!! is used to indicate a different type of highlighting. X!-! means to blink the highlight; $(X!-!)^3$ means to blink three times. The effect of X! (or X-!) is null if it is already the case that X! (or X-!). Also, of course, there are functions for displaying and erasing objects in the feedback. Dragging an object is indicated by $X > \sim$, and rubberbanding an object as it follows the cursor is shown by $X \gg \sim$. The difference between these last two is illustrated by the difference in moving a box on the screen and resizing that box by rubberbanding one of its handles.

6.5 Temporal Relations

In addition to the need for a behavioral view, another problem arises from new styles of interaction involving direct manipulation of graphical objects and icons. These interaction styles are more difficult to represent than the older styles of command languages and menus. User actions in these interfaces are asynchronous, having rather more complex temporal behavior than those of earlier interfaces that were largely constrained to predefined sequences. A brief introduction to these concepts is given in this section; Hartson and Gray [12] give a more detailed discussion of *temporal aspects* of UAN.

The most basic temporal relationships we have identified are

—sequenced with,
—are order independent,
—interruptible by,
—interleavable with,
—can be concurrent with,

and are listed in decreasing order of temporal constraint. *Sequencing* is the most constrained temporal relation; the first action must be performed completely, then the next, and so on, until all actions are completed. In many sequential interface designs, this constraint is arbitrary and even opposed to the cognitive and task needs of the user. For example, initiation of a second task in the middle of a first task may be required to get information necessary to the completion of the first task. It is very desirable that the second task can be interleaved with the first, so it will not destroy the context of the first task.

With *order independence*, all actions must be performed and each one completed before another is begun. But the constraint on specific ordering among actions is removed. An example of order independence at a very low task level is seen in the task of entering a "command-X" in a Macintosh application, a combination of the ⌘ and X keys. Since the ⌘ key must be depressed before the X key, but the order of their release does not matter, the task is defined in UAN as

Task: command-X
⌘v Xv (⌘^ & X^)

While order independence relaxes the sequentiality constraint, *interleaving* removes the constraint of a task or action having to be performed to completion before beginning another action, that is, allowing an action to be interrupted.

User actions are mutually interleavable if and only if they can *interrupt* each other; it is possible that a period of activity of either action can interrupt a period of activity of the other. Consider the case of a help facility available during some other complex user action such as editing a document. Suppose that help information, when invoked, appears in a window separate from the document being edited. The editing and help actions are interleavable since the user may alternate the focus of attention, and therefore task performance, from one window to the other.

With interleavability, user actions can be alternated among tasks; with *concurrency*, user actions for two or more tasks can occur simultaneously. Concurrency is a temporal relation that has not been greatly exploited in user interfaces. Nevertheless, it has been shown [1] that there are cases in which it is possible and, indeed, preferable, to carry out more than one task at the same time, for example, via input techniques that rely on the simultaneous use of both hands.

Time intervals are also important in task descriptions. For example, the prose description of a double click with the mouse button might tell the user to click the mouse button and immediately click it again. This task can be represented precisely in UAN. Note that waiting a certain amount of time, that is, not doing any action for that interval, is itself a user action. The UAN task description for double clicking is

$$M^{\vee\wedge}\ (t < n)\ M^{\vee\wedge}$$

where the value of n can be controlled by the user via a control panel setting and an appropriate default value can be empirically determined by developers.

Analysis of the various temporal relations gives an interface designer the ability to distinguish task types that are significantly different, but which, without these relations, would be difficult to identify. Furthermore, adding operators to UAN to express these relations gives a designer a powerful means of specifying such interfaces.

6.6 Design Aspects of UAN Symbols

The UAN symbols were chosen with specific requirements in mind:

—usage separate from definition,
—typable from a standard keyboard, and
—mnemonically meaningful.

The first requirement provides a locality of definition similar to that in programming. It allows the designer, for example, to use highlighting before the method of highlighting is defined. It also preserves consistency by requiring a single modification of the definition of highlighting for a class of objects.

Although computer-based tools can provide specialized, even graphical, symbols for UAN, it is desirable that UAN design representations be producible with regular word processing (possibly with an extended font for some mathematical

Table I. Summary of Some Useful UAN Symbols

Action	Meaning
~	move the cursor
[X]	the context of object X, the "handle" by which X is manipulated
~[X]	move cursor into context of object X
~[*x*, *y*]	move the cursor to (arbitrary) point *x*, *y* outside any object
~[*x*, *y* in A]	move the cursor to (arbitrary) point within object A
~[X in Y]	move to object X within object Y (e.g., [OK_icon in dialogue_box])
[X]~	move cursor out of context of object X
v	depress
^	release
Xv	depress button, key, or switch called X
X^	release button, key, or switch X
X^v	idiom for clicking button, key, or switch X
X"*abc*"	enter literal string, *abc*, via device X
X (*xyz*)	enter value for variable *xyz* via device X
( )	grouping mechanism
*	iterative closure, task is performed zero or more times
+	task is performed one or more times
{ }	enclosed task is optional (performed zero or one time)
A B	sequence; perform A, then B (same if A and B are on separate, but adjacent, lines)
OR	disjunction, choice of tasks (used to show alternative ways to perform a task)
&	order independence; connected tasks must all be performed, but relative order is immaterial
⇔	interleavability; performance of connected tasks can be interleaved in time
‖	concurrency; connected tasks can be performed simultaneously
;	task interrupt symbol; used to indicate that user may interrupt the current task at this point (the effect of this interrupt is specified as well, otherwise it is undefined, i.e., as though the user never performed the previous actions)
∀	for all
:	separator between condition and action or feedback
Feedback	**Meaning**
!	highlight object
-!	dehighlight object
!!	same as !, but use an alternative highlight
!-!	blink highlight
(!-!)^n	blink highlight *n* times
@ *x*, *y*	at point *x*, *y*
@ X	at object X
display (X)	display object X
erase (X)	erase object X
X > ~	object X follows (is dragged by) cursor
X ≫ ~	object X is rubber-banded as its follows cursor
outline (X)	outline of object X

symbols) and that specific parts of designs be usable within other textual documents. This arrangement facilitates the use of standard keyboards for writing UAN task descriptions.

For mnemonic purposes the symbols were chosen to be visually onomatopoetic. For example, ~ carries the impression of movement and [X] conveys the idea of a box around X. Similarly, ! attracts attention as highlighting and > reflects the notion of following, while ≫ is following but stretching out (rubberbanding).

6.7 Summary of UAN Symbols

The definition of UAN has deliberately been kept open in the sense that interface developers can add and/or modify symbols or columns as needed for their particular design situation. Many of the symbols and idioms we have found useful are summarized in Table I. The reader is referred to [12] for a more formal definition of UAN, especially its temporal aspects.

7. USING UAN AT HIGHER LEVELS OF ABSTRACTION

So far we have discussed UAN as it is used to represent low level physical user actions directly associated with devices. It is possible to build, on these physical actions, levels of abstraction to represent the complete task structure for an entire application. We shall use the terms task and action interchangeably here to underscore the fact that the higher level tasks and the lower level physical actions have many of the same properties when combined into a task structure. Following are some definitions that show how tasks can be combined in UAN:

—The physical actions on devices described so far are tasks. Examples include all actions such as ~[X], M$^{\vee\wedge}$, and so on.

—If A is a task, so are (A), A*, and {A}.

—If A and B are tasks, so are A B, A OR B, A & B, A $\Leftrightarrow$ B, and A $\|$ B.

Thus, a task description written in UAN is a set of actions interspersed with logical and temporal operators. Tasks built up as combinations using these rules can be named and the name used as a reference to the task. A task is invoked by using its name as an action within another task, in the same manner as a procedure call in a computer program. Also in the same manner as computer procedures, this leads to higher levels of abstraction, necessary for understanding by readers and writers of the notation, and a quasihierarchical "calling" structure of tasks. Use of a task name as a user action corresponds at run-time to the invocation of a user-performed procedure.

Task descriptions are eventually written at a detailed level of abstraction in terms of user actions. This level is the *articulation point* between two major activities within the interface development life cycle: task analysis and design. Because these task descriptions are at once the terminal nodes of the task analysis hierarchy and the beginnings of a user interface design, *it is a case where task analysis quite naturally drives the design process.*

As one example of the way symbols are used together in higher level task descriptions, consider the symbol OR used to indicate a disjunction of choices. For example, A OR B OR C denotes a three-way choice among user actions or tasks. A common high level construct in UAN is seen in this example of a repeating disjunction:

(A OR B OR C)*

This notation means that tasks A, B, and C are initially concurrently available. Once a task is begun it is performed to completion, at which time the three tasks are concurrently available again. The cycle continues arbitrarily, each time any one of the three tasks is initiated and performed to completion. Note that this notation is a compact high level description of the use of a menu.

TASK: delete multiple files			
USER ACTIONS	INTERFACE FEEDBACK	INTERFACE STATE	CONNECTION TO COMPUTATION
(Sv (~[file_icon] Mv M^)+ S^)+	 file_icon-!: file_icon!, file_icon!: file_icon-!,	 selected = selected ∪ file selected = selected - file	
~[file_icon!] Mv ~[x,y]*	 outline(icons!) > ~		
~[trash_icon]	outline(icons!) > ~, trash_icon!		
M^	erase(icons!), trash_icon!!	selected = null	mark selected files for deletion

Fig. 12. UAN description of the task "delete multiple files."

TASK: delete multiple files			
USER ACTIONS	INTERFACE FEEDBACK	INTERFACE STATE	CONNECTION TO COMPUTATION
select_multiple_files			
delete_selected_files			

Fig. 13. UAN description of the task "delete multiple files" at a higher level of abstraction.

TASK: select multiple files			
USER ACTIONS	INTERFACE FEEDBACK	INTERFACE STATE	CONNECTION TO COMPUTATION
shift_multiple_select **OR**			
drag_box_multiple_select			

Fig. 14. UAN description of the task "select multiple files" showing alternative methods.

The use of task names as abstractions—for modularity, consistency, and reusability—is illustrated in the following example, using the task of deleting multiple files from the Macintosh desk top. To begin, Figure 12 shows the task description without use of abstraction. Note that S denotes the shift key.

This task of deleting multiple files can be decomposed into two tasks:

(1) select files (the top block in Figure 12)

(2) delete selected files (the other three blocks in Figure 12).

Both these tasks are performed often and will appear as part of other tasks as well. Figure 13 shows how the overall task description of Figure 12 is stated in terms of names for these lower level tasks, which are then defined elsewhere.

TASK: drag box multiple select			
USER ACTIONS	INTERFACE FEEDBACK	INTERFACE STATE	CONNECTION TO COMPUTATION
~[x,y] Mv		x,y is fixed corner of rectangle	
~[x',y']*	dotted rectangle >> ~ see figure "selection rectangle"		
~[x",y"] M ^	items intersected by rectangle are !	selected = all intersected items	

Fig. 15. UAN description of the task "drag box multiple select," with reference to a scenario figure.

The task of selecting multiple files can be done in (at least) two ways: using the shift key, as described in the first block of Figure 12, or by dragging out a selection rectangle with the mouse, as described in Figure 15. Figure 14 is a higher level task description, stated as a disjunction of the names of these two versions of the task.

8. COMPLEMENTARY REPRESENTATION TECHNIQUES

UAN is intended for describing user actions in the context of interface objects, along with feedback and state information. It does not, however, describe screen layouts. State transition information is often distributed among task descriptions. Therefore, designers have found it useful to augment UAN by other representation techniques.

8.1 Scenarios

Figure 15 shows how a task description can be augmented with a *scenario* figure (shown in Figure 16), referenced in the feedback column of the second row. Note that the screen picture of Figure 16, which shows the layout of the screen objects, is also annotated with UAN descriptions.

8.2 Task Transition Diagrams

The asynchronous nature of direct manipulation interfaces inherently demands consideration of user intention shifts during the performance of a task. Maintaining a focus on the primary function of a task while accommodating user intention shifts is difficult for interface designers when both these aspects are represented at the same design level. UAN contains a mechanism for specifying points in a task where user intention shifts may occur. A complementary technique, task transition diagrams, is used to specify tasks that users can perform to interrupt their current task. The task transition diagram is a notation that allows a designer to map out the set of tasks and intentions of a user without having to be concerned with the minutiae of how a user accomplishes those tasks. Further details about task transition diagrams are given in [29].

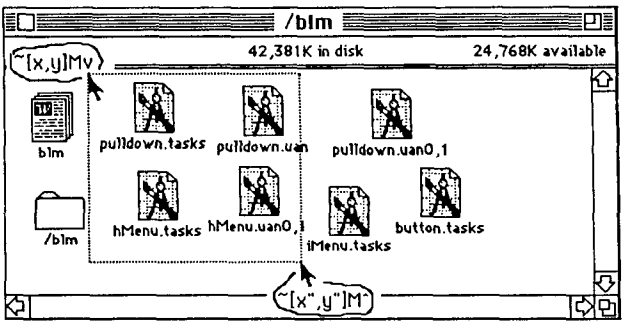

Fig. 16. Scenario figure called "selection rectangle."

8.3 Discussion Sheets

Because the design document is a working document as well as the means for communication among developer roles, designers are encouraged to include, as part of the design representation, their comments about tradeoffs faced, and reasons behind design decisions. These comments appear on *discussion sheets* that augment UAN task descriptions, scenarios, and state diagrams as a more complete representation of interface designs. At a later time the maintenance process especially benefits from information on design decisions, often preventing a repetition of an unsuccessful design previously tried and rejected by designers. Implementers and evaluators also use discussion sheets to comment as early as possible on the design. Implementers are in the best position to estimate the cost of implementing certain features; evaluators must document their reasons for suggested design changes.

9. EXPERIENCE WITH UAN

UAN was created within the Dialogue Management Project at Virginia Tech to represent the design of a user interface management system called DMS 3.0. The interface designers simply grew tired of struggling with the imprecision and verbosity of prose descriptions of how the DMS interface should behave, and the DMS implementers grew tired of reading and trying to understand them. Out of this need, the DMS interface designers produced UAN as a notation for helping to alleviate this problem. Thus, UAN was originally devised to communicate behavioral descriptions of interface designs to implementers for construction and to evaluators for a pre-prototype view of the design. It was useful in this capacity. We used UAN to conduct walk-throughs of the DMS interface design and to check implementation against the design. We estimated that approximately 80 percent of the design was implemented exactly as specified. Of the 20 percent that did not conform to design, 10 percent was due to misinterpretation of the UAN by the implementers, and 10 percent was due to their simply not wanting to implement as represented.

To determine more about the usefulness of UAN, we are promoting its use in commercial, government, and academic development environments. In particular, UAN is being used experimentally in the Human Interface Lab at

Texas Instruments in Dallas, Texas for design of a telephone interface. The Bureau of Land Management in Denver, Colorado is procuring the development of a Geographical Information System for which UAN was used to represent some prototype designs. An interface designer at the Naval Surface Warfare Center in Dahlgren, Virginia, struggling through a several inch-thick stack of human-computer interface guidelines for the Navy, is using UAN to document these guidelines precisely and to prevent repeated rereading of many lines of prose. UAN is being used to represent several multimedia interfaces involving full motion video and audio for a digital video interactive (DVI) application being developed for NCR Corporation. Interface designs for a UIMS and some applications systems in Project DRUID at the University of Glasgow (Scotland) have been represented with UAN. A DEC U.K. interface designer used it to precisely describe what happens during a complex scrolling-in-a-window task for a new design. In the United States, designers at DEC are using UAN for the interface of a graphical editor. A data flow configuration system has been designed with UAN at the Jet Propulsion Laboratory. This diversity of uses of UAN indicates the broad range of interface styles it can represent.

In general, users of UAN report that it is easy to learn to read and to write. They find its symbols and idioms to be mnemonic and intuitive. Perhaps more importantly, they like the thoroughness of the descriptions and feel it facilitates communication between interface designer and implementer. They find it to be precise, concise, and easily extensible as needed for their particular environment. Negative comments contend that UAN descriptions may be too detailed to be used early in the design process, when too much specificity about the interface may limit the designer's creativity. One or two developers have commented that the symbols should be more expressive (e.g., they wanted to change the ~ to move_to), or that the symbols or columns did not allow them to fully represent their design. For this reason, we encourage extension of UAN symbols, columns, or any other feature, so that the UAN technique more completely supports interface representation.

10. SOFTWARE TOOLS

Work reported in this paper is intended to support the development process through software tools, and the evaluation of designs through analytical processing. We are exploring various other uses of UAN, and particularly its support by software tools, in the interface development process.

The primary UAN support tool, on which we are currently working, is a UAN editor that developers can use to design the interface of an interactive system. The tool supports textual entry of UAN task descriptions. Task description "by demonstration" [19] is being considered as well.

The tool also supports the addition of other columns. For example, there are system events that occur asynchronously with respect to user actions (e.g., a clock updating itself) that cannot therefore be described in terms of user actions. Other uses for extra columns include:

—memory actions, memory loading, closure [26];
—cognitive and perceptual actions—seeing feedback and acting on it in a closed feedback loop mode, decision making [5, 26];

—semantic connections (to invoke computational functions, for semantic feedback, especially for interreferential I/O);
—user goals and intentions, that is, Norman's theory of action [20]; and
—task numbering for cross referencing of task invocations in large interface design structures.

The UAN editor tool will also eventually be used to support other development activities such as implementation and interface evaluation. Other tools being researched include tools for analytic evaluation of interface usability, rapid prototyping, code generation and translation, and generation of end-user documentation.

11. FUTURE WORK

In conjunction with the tools mentioned above, there are many other rich research issues involving UAN, some of which involve difficult problems. A few examples are discussed briefly in this section.

11.1 Analytic Evaluation

Analytic evaluation is an interface development activity that is applied to interface designs to predict and assess usability before even a prototype is built and tested. The model on which each analytic evaluation technique is based must be validated against user testing to determine its ability to predict accurately usability. Some types of analytic evaluation use direct user performance prediction metrics that predict elapsed times to perform user actions such as mouse movement (e.g., Fitts' Law [6]) and keystrokes (e.g., keystroke model [2]). Cognitive, perceptual, and memory actions can also have associated performance times. Other types of analytic evaluation predict user performance indirectly by identifying inconsistencies and ambiguities in an interface, determining equivalency of tasks, and analyzing information flow among tasks [24]. We plan to apply analytic evaluation to UAN interface descriptions. A prerequisite is the inclusion of columns in UAN task descriptions for cognitive, memory, perceptual, and decision-making user actions, in addition to physical actions. Also, because this kind of analysis requires machine processing of task descriptions, descriptions must be even more formal, precise, and complete than those currently produced. Because UAN expressions are parsed, UAN must itself have a more rigorous grammatical definition (e.g., BNF-style production rules).

11.2 Rapid Prototyping

A rapid prototype is an early version of an interactive system with which users can interact in order to evaluate the system design. Constructional representation techniques are well-suited for prototyping; their view of the interface design is the system's view. Constructional representation techniques allow direct execution, often by interpretation, of interface design representations or specifications. The system's part is played by the prototype as it is executing, allowing evaluation of the user's part as played by a human user. However, a behavioral representation of a design is the user's part. Running a behavioral description directly on the system would make the system appear to behave as a user. (There is some interest

in doing just this, if the object is to study complexity of user behavior [17], but this is not the goal in prototyping.) In other words, the representational domain that one chooses determines which roles play opposite in the game of prototyping. Thus, for prototyping, behavioral representations must be translated to a constructional equivalent either before or during the prototyping process.

11.3 Code Generation and Translation to Constructional Domain

Both prototyping and generation of executable code from UAN task descriptions require translation of design representations from the behavioral domain to the constructional. This kind of translation is not a trivial problem, because it is more than a line-by-line translation between languages. It is translation from one "coordinate system" to another, requiring transformations in structure that can reach deeply into a design. Solutions to the translation problem will require formal understanding of structural, semantic, and syntactic connections between the two domains. The object-oriented paradigm seems to be the most suitable constructional model; user actions in the behavioral domain have, as their counterparts, events in the constructional domain. The translation process will involve identification of objects, classes, and methods from UAN descriptions.

11.4 End-user Documentation Generator

Because UAN describes how a user performs each task, the basic ingredients for user documentation are present. We are investigating how to automatically derive a minimal user's manual from UAN descriptions. In addition, maintaining integrity between the user manual and the implemented system is a common problem. Using the UAN tool to maintain the system definition, and providing other tools to translate these definitions into code and user manuals, can assure a consistent match between manuals and system.

12. SUMMARY

User-centered interface design necessitates a *behavioral view of the user* performing tasks to interact with the computer. Thus a tool-supported behavioral technique for representing the user interface is needed to complement the constructional techniques needed for implementation. User Action Notation (UAN) is a behavioral representation because it describes the actions a user performs to accomplish tasks, rather than the events that a computer interprets.

UAN provides an articulation point between task analysis in the behavioral domain and design and implementation in the constructional domain: an interface is specified as a set of quasihierarchical tasks, with the lower level tasks comprised of user actions. Precision and clarity are obtained because associated feedback and system changes are written in separate columns and in line-wise correspondence to actions. Because UAN uses a text-based representation, analytic evaluation of interfaces is possible.

Real-world users of UAN report it to be highly readable and writable with little training because of its simplicity and natural mnemonicity. Use within interface design and implementation projects has shown UAN to be thorough, concise, and precise in conveying large complex user interface designs from designers to implementers and evaluators. Current work involves software tools to support code and documentation generation, in addition to evaluation of interface designs.

ACKNOWLEDGMENTS

The Dialogue Management Project has received funding from the Office of Naval Research, the National Science Foundation, the Virginia Center for Innovative Technology, IBM Corporation, the Software Productivity Consortium, and Contel Technology Center. We also thank the UAN users who have given us valuable feedback on their experiences with its use, and the anonymous reviewers for their thoughtful and helpful comments.

REFERENCES

1. BUXTON, W. There's more to interaction than meets the eye: Some issues in manual input. In *User Centered System Design.* D. A. Norman and S. Draper, Eds. Lawrence Erlbaum, Hillsdale, N. J., 1986.
2. CARD, S. K., AND MORAN, T. P. The keystroke-level model for user performance time with interactive systems. *Commun. ACM 23* (1980), 396–410.
3. CARD, S. K., MORAN, T. P., AND NEWELL, A. *The Psychology of Human-Computer Interaction.* Lawrence Erlbaum, Hillsdale, N.J., 1983.
4. CARDELLI, L., AND PIKE, R. Squeak: A language for communicating with mice. *Comput. Graph. 19,* 3 (1985), 199–204.
5. DRAPER, S. Personal communication, 1989.
6. FITTS, P. M. The information capacity of the human motor system in controlling the amplitude of movement. *J. Exper. Psych. 47* (1954), 381–391.
7. FLECCHIA, M., AND BERGERON, R. D. Specifying complex dialogs in ALGAE. In *Proceedings of CHI + GI Conference on Human Factors in Computing Systems* (Toronto, Apr. 5–9, 1987). ACM, New York, 1987, 229–234.
8. FOLEY, J., GIBBS, C., KIM, W., AND KOVACEVIC, S. A. Knowledge-based user interface management system. In *Proceedings of CHI Conference on Human Factors in Computing Systems* (Washington, D.C, May 15–19, 1988). ACM, New York, 1988, pp. 67–72.
9. GOULD, J. D., AND LEWIS, C. Designing for usability: Key principles and what designers think. *Commun. ACM 28,* 3 (1985), 300–311.
10. GREEN, M. The University of Alberta user interface management system. *Comput. Graph. 19,* 3 (1985), 205–213.
11. GREEN, M. A survey of three dialog models. *ACM Trans. Graph. 5,* 3 (July 1986), 244–275.
12. HARTSON, H. R., AND GRAY, P. Temporal aspects of tasks in the user action notation. To appear in *Human-Computer Interaction,* 1990.
13. HARTSON, H. R., AND HIX, D. Toward empirically derived methodologies and tools for human-computer interface development. *Int. J. Man-Mach. Stud. 31* (1989), 477–494.
14. HILL, R. Event-response systems—A technique for specifying multi-threaded dialogues. In *Proceedings of CHI + GI Conference on Human Factors in Computing Systems* (Toronto, Apr. 5–9, 1987). ACM, New York, 1987, 241–248.
15. JACOB, R. J. K. An executable specification technique for describing human-computer interaction. In *Advances in Human-Computer Interaction.* H. R. Hartson, Ed. Ablex, Norwood, N.J., 1985.
16. JACOB, R. J. K. A specification language for direct manipulation user interfaces. *ACM Trans. Graph. 5,* 4 (1986), 283–317.
17. KIERAS, D., AND POLSON, P. G. An approach to the formal analysis of user complexity. *Int. J. Man-Mach. Stud. 22* (1985), 365–394.
18. MORAN, T. P. The command language grammar: A representation for the user interface of interactive computer systems. *Int. J. Man-Mach. Stud. 15* (1981), 3–51.
19. MYERS, B. Creating dynamic interaction techniques by demonstration. In *Proceedings of CHI + GI Conference on Human Factors in Computing Systems* (Toronto, Apr. 5–9, 1987). ACM, New York, 1987, 271–278.
20. NORMAN, D. A. Cognitive engineering. In *User Centered System Design.* D. A. Norman and S. Draper, Eds. Lawrence Erlbaum, Hillsdale, N.J., 1986.
21. OLSEN, D. R. MIKE: The menu interaction kontrol environment. *ACM Trans. Graph. 5,* 4 (1986), 318–344.

22. OLSEN, D. R., JR., AND DEMPSEY, E. P. Syngraph: A graphical user interface generator. *Comput. Graph. 17,* 3 (1983), 43–50.
23. PAYNE, S. J., AND GREEN, T. R. G. Task-action grammars: A model of the mental representation of task languages. In *Human-Computer Interaction.* Lawrence Erlbaum, Hillsdale, N.J., 1986.
24. REISNER, P. Formal grammar and human factors design of an interactive graphics system. *IEEE Trans. Softw. Eng. SE-7* (1981), 229–240.
25. RICHARDS, J. T., BOIES, S. J., AND GOULD, J. D. Rapid prototyping and system development: Examination of an interface toolkit for voice and telephony applications. In *Proceedings of CHI Conference on Human Factors in Computing Systems* (Boston, April 13–17, 1986). ACM, New York, 1986, 216–220.
26. SHARRATT, B. Personal communication, 1989.
27. SHNEIDERMAN, B. Multi-party grammars and related features for designing interactive systems. *IEEE Trans. Syst. Man Cybern. 12,* 2 (Mar.–Apr. 1982), 148–154.
28. SIOCHI, A. C., AND HARTSON, H. R. Task-oriented representation of asynchronous user interfaces. In *Proceedings of CHI'89 Conference on Human Factors in Computing Systems* (Austin, Tex, April 30–May 4, 1989). ACM, New York, 1989, 183–188.
29. SIOCHI, A. C., HARTSON, H. R., AND HIX, D. Notational techniques for accommodating user intention shifts. TR 90–18, Dept. of Computer Science, Virginia Polytechnic Institute and State Univ., 1990.
30. WASSERMAN, A. I., AND SHEWMAKE, D. T. The role of prototypes in the user software engineering methodology. In *Advances in Human-Computer Interaction.* H. R. Hartson, Ed. Ablex, Norwood, N.J., 1985.
31. YUNTEN, T., AND HARTSON, H. R. A SUPERvisory methodology and notation (SUPERMAN) for human-computer system development. In *Advances in Human-Computer Interaction.* H. R. Hartson, Ed. Ablex, Norwood, N.J., 1985.