Notational Techniques for Accommodating User Intention Shifts

Antonio C. Siochi, H. Rex Hartson and Deborah Hix

TR 90-18

Submitted for publication

NOTATIONAL TECHNIQUES FOR ACCOMMODATING USER INTENTION SHIFTS

Antonio C. Siochi
H. Rex Hartson
Deborah Hix

Department of Computer Science
Virginia Tech
Blacksburg, VA 24061-0106, USA

ABSTRACT
Good user interface designs allow for user intention shifts. 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 at the same time accommodating *user intention shifts* is difficult for interface designers when both these aspects are represented at the same design level. The User Action Notation (UAN), a technique for representing asynchronous interfaces, contains a mechanism for specifying points in a task where user intention shifts may occur. A complementary technique, Task Transition Diagrams (TTDs), 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.

KEYWORDS AND PHRASES:
Notation, user actions, asynchronous user interfaces, direct manipulation, task interrupts, abandoning, interleaving, user interface representation, task description.

INTRODUCTION
Users can and do change their intentions during the performance of a task. In general, user interfaces that are rigid, and hence unsupportive of such *intention shifts*, typically have poor usability. Consequently, a substantial difficulty in designing interfaces is *accommodating a user's intention shifts* while at the same time *maintaining a focus on the primary function of the task.* This problem is very familiar to anyone who has ever had to write a program where error and condition-checking code interferes with readability of the primary function of the program. Incorporating such considerations into a task description can obscure the sequence of actions required for "normal" accomplishment of the task, yet it is precisely those considerations that help shape the usability of a system. This problem is especially acute for direct manipulation interfaces, where their asynchronous nature practically guarantees the occurrence of user intention shifts.

The *User Action Notation (UAN)* [3, 9], briefly described in a following section, is a task-oriented technique for designing asynchronous interfaces such as the Macintosh™ Finder. It readily supports the representation of user actions with associated feedback and state change information for each specific user task. The focus of the UAN on the specifics of a *single* task is both a strength and a weakness.

When a designer's concern turns to the accommodation of intention shifts, the designer must be able to show the set of tasks to which a user can shift, and how those tasks are related. The UAN provides a means of specifying explicit points in the task at which the

™ Macintosh, Finder, and Multifinder are trademarks of Apple Computer, Inc.

task may be interrupted by the user, i.e., intention shift points. However, when a task can be interrupted, there is a need to show *how* that task can be interrupted. This paper introduces *Task Transition Diagrams (TTDs)*, a notation that allows a designer to map out a set of related tasks and intentions of a user without having to be concerned with the minutiae of how a user accomplishes those tasks. TTDs serve as a technique to complement the single-task-specificity of the UAN.

RELATED WORK

Numerous notational schemes have been developed to represent user interface designs. Two of the earliest notations are the Keystroke-Level model of Card, Moran, and Newell [1] and the Action Language of Reisner [8]. These notational schemes were designed to measure some aspect of the interface. In the case of the Keystroke-Level model, predictions can be made about performance times for specific tasks. Tasks are described at the level of keystrokes, homing operations, and mental preparations. In the Action Language, interfaces are described by means of a formal grammar, with the purpose of analyzing the interface for possible problems such as inconsistencies. A more recent notation, Task-Action Grammar (TAG) [7], is similar to the Action Language both in form (it also describes interfaces in terms of a grammar) and purpose. TAG, however, is task-oriented.

These three notations are actually analytical, rather than synthetic, tools. None of them provides a means of representing user intention shifts. In fact, the Keystroke-Level model assumes error-free user performance. In addition, none of the three supports representation of asynchronous user interfaces.

Another major class of notation is graphical. Interfaces are represented as directed graphs in which nodes correspond to machine states and arcs indicate control flow among the states. Examples include [10], [6], and [11]. These techniques are based on state machines, and hence lack adequate support for asynchronous interfaces. Jacob [4] addresses this problem by specifying several asynchronous state machines instead of a single machine. These schemes, however, provide no separation between how a task is performed and what tasks can be done.

The other major class of notation is based on task analysis, which is "an empirical method which can produce a complete and explicit model of tasks in the domain, and of how people carry out those tasks" [5]. The GOMS - Keystroke Level model [1], and TAG [7] fit in this class as well. Interface design based on task analysis proceeds by building a model of the user's task using some task representation technique.

THE USER ACTION NOTATION: AN OVERVIEW

There are at least two domains of interface design and representation: *behavioral* and *constructional* [3]. Behavioral design and representation involve physical, cognitive, and perceptual user actions and interface feedback, i.e., 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. Any description that can be thought of as "running on the machine" is constructional. Two examples are state transition diagrams and event handlers.

In contrast, behavioral descriptions can be thought of as being "performed by the user." Behavioral descriptions are important because *it is in the behavioral domain that interface designers and evaluators do their work.* Thus, there is a need for behavioral representation techniques to keep a user-centered focus during the interface development process.

The UAN is a user- and task-oriented notation that describes the behavior of the user and the interface during their cooperative performance of a task. The primary abstraction of the UAN is a task. A user interface is represented as a quasi-hierarchical structure of asynchronous tasks, the sequencing within each task being independent of that in the others. User actions, corresponding interface feedback, and state change 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 [2]. The UAN can be used to supplement scenarios, indicating precisely how the user interacts with the screen layout shown in a scenario. Figure 1 shows an example of a simple UAN task description. The task is that of moving a file icon in the Macintosh Finder.

Task: move a file icon		
User Actions	**Feedback**	**State Changes**
~[fileIcon] ; Mv;	fileIcon!	currentObject = fileIcon, fileIcon is selected
~[x,y]* ;	outline(fileIcon) > ~	
M^	@(x,y) show(fileIcon)	update location of fileIcon

Figure 1. UAN task description for moving a file icon.

The first column in this UAN task description specifies the user actions required for this task. Reading this column, the first line denotes moving the cursor to the context of the file icon (~[fileIcon]), and depressing the mouse button (Mv). In the second line, ~[x,y]* indicates movement of the cursor to any (x,y) coordinate on the screen, zero or more times (*). In the third line, the mouse button is released (M^). The second column describes interface feedback corresponding to user actions in a precise line-by-line correspondence, e.g., depressing the mouse button causes fileIcon to be highlighted (fileIcon!); in the second line, the outline of fileIcon follows the cursor as it changes position. This level of precision can be lost in a prose description, where actions and feedback are intermingled. In the third column, state information can also be associated as appropriate with user actions, e.g., depressing the mouse button selects fileIcon as the current object. For a more detailed presentation of the UAN, see [3] or [9].

TASK INTERRUPTION

From a human-computer interface development view, a user task consists of a sequence of user actions or sub-tasks, the last one of which marks the end of the task, i.e., task *closure*. Thus in the task of moving a file icon shown in Figure 1, the action of releasing the mouse button (M^) signals task closure. Each task is associated with a user intention. A user performing all and only the user actions of that task achieves task closure without an intention shift. Should the user change intention in the middle of a task and perform some action other than the next in the sequence, we say that task is *interrupted*. In the task in Figure 1, if the user positions the cursor over the file icon, but then moves the cursor away without depressing the mouse button, the task of moving the file icon is interrupted; there is a user intention shift. This point of interruptibility is indicated in a task description by a semi-colon, as shown in the first line of the user action column in Figure 1.

The semi-colon in a UAN task description allows a designer to accommodate intention shifts by indicating where a task can be interrupted without having to specify details of how the user interrupts the current task. This has the benefit of preserving the clarity of a UAN task description.

Two Classes of Interruption

Users can interrupt tasks in one of two ways, *abandoning* or *interleaving*, reflecting different intention shifts. Abandoning requires identification of explicit user actions that bring about closure with the intent to abort the task. For example, the user can abandon the task of using a dialogue box by clicking on a cancel button. Other examples include abandoning the task of choosing an item in a pulldown menu by moving the cursor off the menu then releasing the mouse button and abandoning the task of deleting a file by moving the file back into its original window instead of releasing it into the trash can.

The second class, interleaving, occurs when a user interrupts a task with another one but with the intent of returning to the original task. For example, a user might move between two open dialogue boxes, shift from one application to another in Multifinder™, or respond to a system event such as incoming mail.

Both classes of task interruption are specified in the UAN with semi-colons (see Figure 1). In this paper we are concerned only with the case of abandoning a task. Interleaving is discussed in [2].

Abandoning a Task

The Semi-colon: A Local View. During the performance of a task the user may change intentions and decide to abandon the current task. As an example, consider the task of selecting an item from a pulldown menu. Task descriptions in Figure 2 show the sequence of actions a user performs to select an item from a pulldown menu, together with exact points in the sequence where this task can be interrupted (i.e., the semi-colons). The first semi-colon indicates a point where users can change their mind about pulling down this menu at all. The second and third semi-colons indicate points where the user may decide not to pick any of the items in this menu. Note that the actions which cause this interruption are not explicitly specified in the task description. This makes reading (and writing) the actions for this task straightforward while allowing for a user's intention shifts.

Task: select item from pulldown menu		
User Actions	**Feedback**	**State Changes**
~[menutitle] ; Mv;	menutitle! show (pulldownmenu)	
(~[menuitem] [menuitem]~)* ;	menuitem! menuitem-!	
~[menuitem'] M^	menuitem'! menuitem'!! hide (pulldownmenu)	menuchoice = menuitem

Figure 2. Selecting an item from a pulldown menu.

Task Transition Diagrams (TTDs): A Global View. Unfortunately, the representation of interruption as a semi-colon in a UAN task description is incomplete. Very often there is a need to show what the user can do to interrupt a task. For example, at the second or third semi-colon in Figure 2, the user could move the mouse to another menu title (indicating a change of which menu to pull down), or move the mouse off the menu completely (indicating the user's intent of not selecting any of the items). Such detail is more efficiently and effectively presented at a level higher than the actions in this task description — a level which shows the relationships among these tasks.

State Transition Diagrams (STDs) are a well-known method for showing a constructional view of the system. The system can be in one of a set of states (represented by nodes), and the transition (represented by arcs) from one state to another is governed by events, e.g., the sensing of a user input. In contrast, a Task Transition Diagram (TTD) shows a set of related tasks a user can perform in a system. A task is represented as a node. The relationship among tasks reflects the various possible intentions of a user, i.e., an arc from task A to task B indicates that a user performing task A may have an intent to perform task B. There is no labelling of arcs, or rather there is an implicit labelling of an arc with the intent to perform the task to which the arc leads. The resulting graph can be considered a non-deterministic state machine which shows *what a user can do* in a system, *not how to do* those things.

In the pulldown menu example, Figure 3 shows the relationships among the sub-tasks of selecting a menu item from a pulldown menu. Note that a path from the first sub-task through the last shows completion of the task (task closure being indicated with the triangle). Thus the user selects a menu title, selects a menu item, then closes the task. This simple figure corresponds to the UAN task description of Figure 2 without the semi-colons: the sub-task of selecting a menu title corresponds to the user actions in the first line of Figure 2, while the sub-task of selecting a menu item corresponds to the user actions in the second and third line of the same figure. A designer looking at the TTD in Figure 3 might ask, "What if the user wants to abort this task, or discovers that the wrong menu is pulled down?" Figure 4 answers these questions by showing *interruptor tasks* (e.g., Select Another Menu Title, Abandon Choosing Menu Item) which are available to the user at the points indicated in the UAN by semi-colons.

Yet it is not sufficient to maintain a *separate* notational scheme identifying these interruptor tasks. It is equally important to know the exact points *within* a task at which a user may shift to the interruptor tasks. A complete view of the principal task and its interruptor tasks thus requires both a UAN description of the principal task, as well as a TTD identifying the interruptor tasks and showing the relationship among all these tasks.

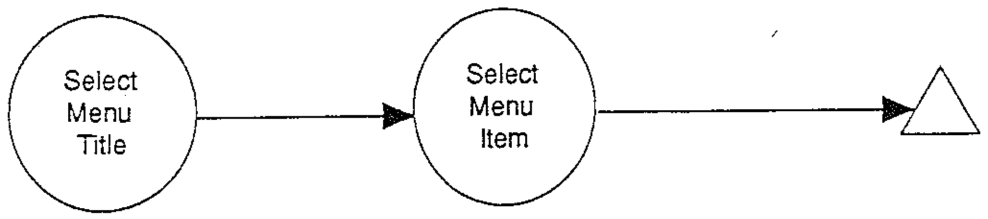

Figure 3. Initial TTD showing sequence of sub-tasks.

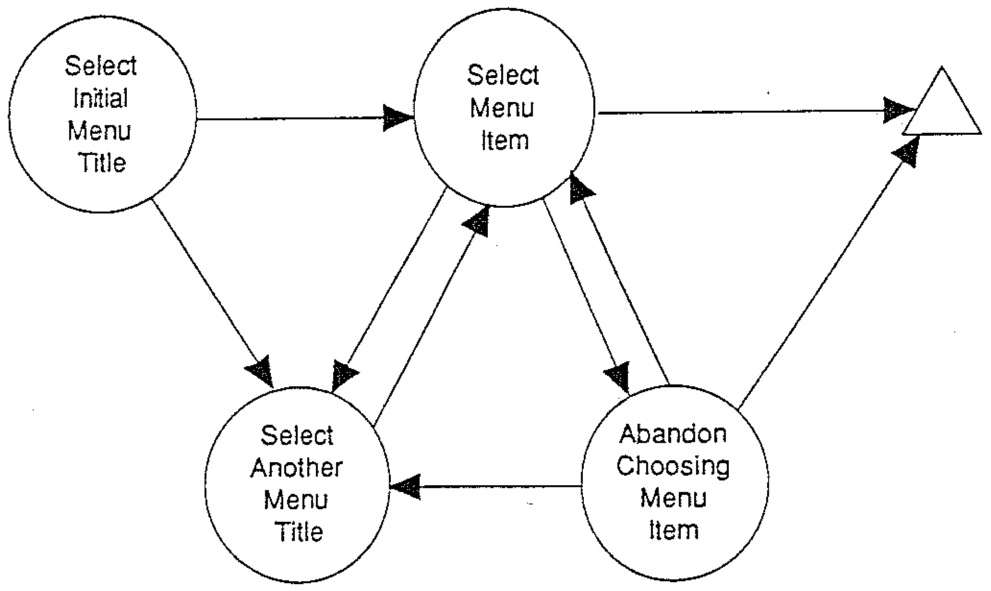
Select Initial Menu Title
Select Menu Item
Select Another Menu Title
Abandon Choosing Menu Item

Figure 4. How a user can interrupt the pulldown menu task.

THE COMMON PREFIX PROBLEM

From the constructional point of view, there is a distinction to be made between a user interrupting a task and the computer determining which of several tasks with a *common prefix* the user was performing. Consider the tasks of moving a file to another folder, copying the file to another disk, and deleting the file as shown in Figure 5. These tasks have a common prefix sequence of user actions:

~[file]Mv.

When the user performs the actions indicated in this prefix, the user knows which of the three tasks the user intended. A computer program, however, cannot "know" the user's intent, and hence must interpret the actions. In this case, not enough user input has been provided for the program to distinguish among the three possible intentions. Let us assume that the user intended to delete the file and now performs ~[trashIcon]M^. It is now possible for the program to determine that the task performed is deletion of the file. It cannot, however, be assumed that the user abandoned either the copy or move tasks, since neither was the intended task. This illustrates a difference between the behavioral view and the constructional view. The UAN shows the viewpoint of the user, and the user knows what task is intended. The computer, however, must interpret user actions to decide what task the user is performing.

In the behavioral domain, a designer needs to visualize the task structure supported by an interface. Given the asynchronous, task-based nature of the UAN, structures such as common prefixes are particularly difficult to discern *because they are distributed over the task structure*. In the current example, unless the move, copy, and delete tasks were listed together, it would have been difficult to realize that indeed a common prefix existed. In this case, the common prefix happens to reflect a design decision (select file first, then specify action to be performed on it). However, since the UAN forces a focus on individual tasks, it is easy to miss situations in which common prefixes might cause problems.

Consider the select and move tasks in MacDraw™. It is a common enough experience for users to move an object accidentally when they had intended only to select it. These tasks actually have a common prefix, ~[object]Mv. The select task would be closed by immediately releasing the mouse button, whereas the move task requires first moving the cursor to some other screen location. The degree of hand-eye coordination a user has will determine the amount of inadvertent movement that occurs during a select task, and since any movement results in a move task, the designer should incorporate some timing considerations or slack movement (free play) in order to reduce the chances of moving an object when the intent was only to select it.

Task: move (file)		
User Actions	**Feedback**	**State Changes**
~[file]Mv	file!	currentObject = file
~[newWindow]M^	show (file) @ newWindow	move file to subdirectory newWindow

Task: copy (file)		
User Actions	**Feedback**	**State Changes**
~[file]Mv	file!	currentObject = file
~[otherDisk]M^	show (file) @ otherDisk	copy file to disk otherDisk

Task: delete (file)		
User Actions	**Feedback**	**State Changes**
~[file]Mv	file!	currentObject = file
~[trashIcon]M^	erase (file)	mark file for deletion

Figure 5. UAN task descriptions for moving, copying and deleting a file.

TTDs can be used to show the relationships among tasks with a common prefix. Figure 6 shows a TTD for the three tasks of Figure 5. Note that this TTD shows a sequence of selecting a file then do something to it. A fundamental user question is thus answered: "What can I do to a file?" In fact, adding an arc connecting the select file task directly to task closure (the triangle) would indicate that in addition to copying, moving, or deleting a file, a user could just select the file. It should be possible to construct such a TTD automatically from a set of UAN task descriptions, thus enabling the designer to visualize the relationships of tasks with a common prefix and check the actual design against the planned design.

™ MacDraw is a trademark of Claris, Corp.

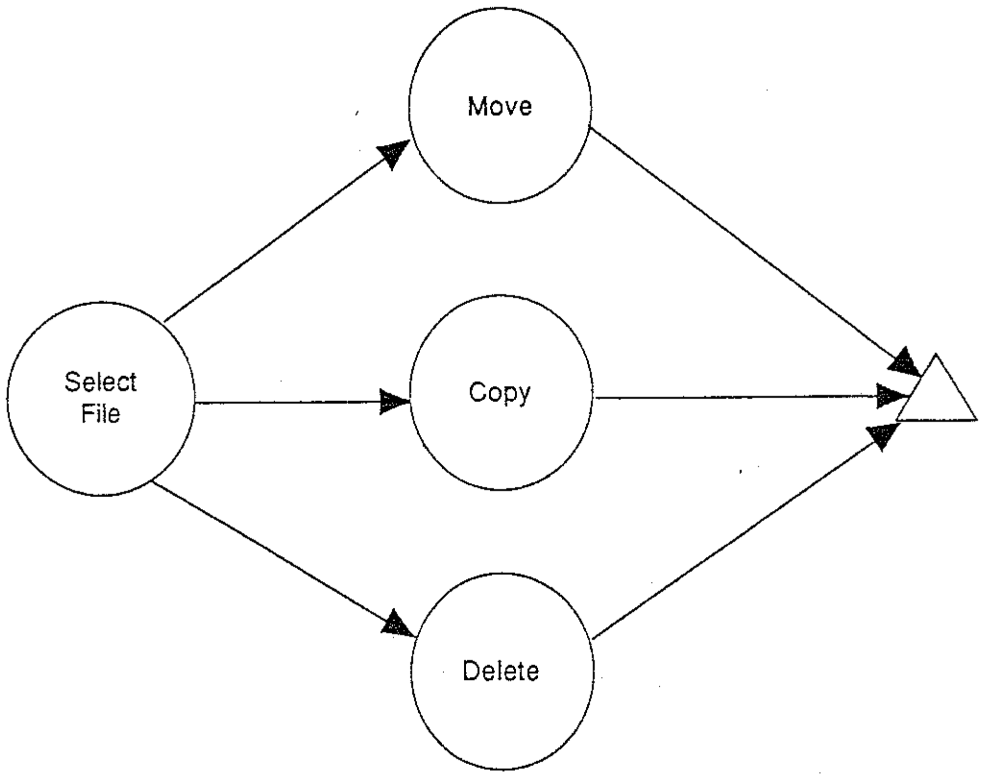

Figure 6. TTD for tasks with a common prefix.

SUMMARY AND CONCLUSION

Interfaces which fail to provide for user intention shifts often have poor usability. Accommodating such intention shifts when designing an interface is difficult if the description of the interruptor tasks must be made at the same design level as the principal task. A separation of these tasks can be achieved by using the User Action Notation (UAN) and Task Transition Diagrams (TTDs). The UAN provides a micro-behavioral view by describing details of how a task is performed, together with specifications of where in the task user intention shifts may occur. TTDs provide a macro-behavioral view by showing the interruptor tasks and the relationships between these tasks and the principal task. TTDs thus are a technique that complements the UAN. TTDs are also useful since they can be used to help a designer *visualize the task structure* present in the set of asynchronous UAN task descriptions.

FUTURE WORK

Research is underway on the translation of behavioral specifications of interfaces (e.g., UAN, TTD) to constructional implementations (e.g., Motif™). The major issue is that these domains have different structures. Effort is also being directed towards computer analysis of UAN task descriptions, in terms of common prefixes, inconsistency, and ambiguity. The automated production of TTDs from a set of UAN task descriptions for analyzing relationships within sequences of related tasks in a user interface design is also being studied.

™ Motif is a trademark of the Open Software Foundation

ACKNOWLEDGEMENTS
The authors acknowledge support for the Dialogue Management Project, of which this work is a part, from the Virginia Center for Innovative Technology, Contel Technology Center, the Software Productivity Consortium, IBM Corp., the National Science Foundation, and the Office of Naval Research.

REFERENCES

1. Card, S. K., Moran, T. P., and Newell, A. *The Psychology of Human-Computer Interaction.* Lawrence Erlbaum, Assoc., New Jersey, 1983.

2. Hartson, H. R., and Gray, P. D. *Temporal Aspects of Tasks in the User Action Notation.* Submitted for publication. Also available as TR 90-12, Department of Computer Science, Virginia Tech, Blacksburg, Va 24061-0106, 1990.

3. Hartson, H. R., Siochi, A. C., and Hix, D. *The UAN: A User-Oriented Representation for Direct Manipulation Interface Designs.* Submitted for publication. Also available as TR 90-16, Department of Computer Science, Virginia Tech, Blacksburg, Va 24061-0106, 1990.

4. Jacob, R. J. K. A Specification Language for Direct Manipulation User Interfaces. *ACM Trans. on Graph. 5*, 4 (1986), 283-317.

5. Johnson, P., and Johnson, H. "Knowledge Analysis of Tasks: Task Analysis and Specification for Human-Computer Systems." *Engineering the Human-Computer Interface.* McGraw-Hill, 1988.

6. Kieras, D., and Polson, P. G. A Generalized Transition Network Representation for Interactive Systems. In Proceedings of *CHI 1983, Conference on Human Factors in Computing Systems* , Boston, Mass., 1983, 103-108.

7. Payne, S. J., and Green, T. R. G. "Task-Action Grammars: A Model of the Mental Representation of Task Languages." *Human-Computer Interaction.* Lawrence Erlbaum Associates, Inc., 1986.

8. Reisner, P. Formal Grammar and Human Factors Design of an Interactive Graphics System. *IEEE Trans. Soft. Eng. SE-7* (1981), pp. 229-240.

9. Siochi, A. C., and Hartson, H. R. Task-oriented Representation of Asynchronous User Interfaces. In Proceedings of *CHI 1989 Conference on Human Factors in Computing Systems* , Austin, Texas, 1989, 183-188.

10. Wasserman, A. I., and Shewmake, D. T. "The Role of Prototypes in the User Software Engineering Methodology." *Advances in Human-Computer Interaction.* Hartson ed. Ablex, Norwood, New Jersey, 1985.

11. Yunten, T., and Hartson, H. R. "A SUPERvisory Methodology And Notation (SUPERMAN) for Human-Computer System Development." *Advances in Human-Computer Interaction.* Hartson ed. Ablex, New Jersey, 1985.