A DESCRIPTIVE/PRESCRIPTIVE MODEL FOR MENU-BASED INTERACTION

by

James D. Arthur

January 1986

TR-86-3

A Descriptive/Prescriptive Model for Menu-Based Interaction

James D. Arthur

Department of Computer Sciences
Virginia Polytechnic Institute
Blacksburg, VA 24061
(703) 961-7538

ABSTRACT

As software systems continue to increase in sophistication and complexity, so do the interface requirements that support user interaction. To select the proper blend of ingredients that constitutes an adequate user interface, it is essential that the system designer have a firm understanding of the interaction process, i.e, how the selected dialogue format interacts with the user and with the underlying task software. One major approach to understanding the software design process and improving the quality of a product is through the use of *models*. The application of models to user/system interaction can provide the crucial feedback and innovative insights for designing and developing exemplary interactive systems. In this paper, we present one such model that *describes* as well as *prescribes* the critical elements for menu-based interaction and their interface dependencies. The model structure provides the flexibility for characterizing menu-based interactions that vary in levels of sophistication, and include 1) computational and decision capabilities based on task oriented actions, 2) user response reversal for error recovery, and 3) user directed movement. Finally, to illustrate the intrinsic power of our model, we present a "descriptive" narrative of two prominent menu-driven systems, Smalltalk and Zog, followed by a discussion of the model's "prescriptive" influence on the design and development of a third menu-based system, Omni.

A Descriptive/Prescriptive Model for Menu-Based Interaction

Categories and Subject Descriptors: C.0 [**Computer System Organizations**]: General - *systems specification methodologies*; D.2.2 [**Software Engineering**]: Tools and Techniques - *user interfaces, modules and interfaces*; H.1.2 [**Information Systems**]: Models and Principles - *user/machine systems, human information processing*

General Terms: Menu-Driven Systems, Man/Machine Interaction, User Interfaces

Additional Keywords: Frames, Item selection, User response reversal, History, Frame memory

1. Introduction

The software development life cycle model [11] defines a systematic approach for constructing software systems. One major step in this cycle addresses the system design process, that is, translating system requirements into an integrated set of task specifications that defines the functional (or logical) capabilities of a software system and its physical components. Throughout the design process, system engineers often exploit the power of *models* to guide them in selecting design alternatives that achieve desired goals.

Models have not always played such an important role in system development. As a general rule, they have been used for *ex post facto* characterizations of physical systems that cannot be altered, e.g., chemical reactions and planetary motion. In addition to describing system behavior, models can also *prescribe* design alternatives. Model elements that characterize physical properties can serve as a blueprint for constructing systems that possess such properties. Because of the traditional emphasis on the descriptive power of models, their prescriptive capabilities are often overlooked. Today, however, the complexity of software systems demands that we fully exploit the potential of every available tool to develop a better understanding of software system intricacies,

and subsequently, the software design process. In particular, software system models must be designed and used for describing and predicting system behavior as well as *prescribing* system attributes and components.

A very real need exists for such models that address human/computer interaction. As software systems continue to increase in sophistication and complexity, so do the interface requirements that support user interaction. To select the proper blend of ingredients that constitutes an adequate user interface, it is essential that the system designer have a firm understanding of the interaction process, i.e., how the selected dialogue format interacts with the user and with the underlying task software. To promote such an understanding, this paper presents a model that characterizes one prevalent dialogue format: *menu-based* interaction. The adaptability of menu-based systems to many diverse applications and their simplistic approach to user interaction has contributed significantly to the widespread acceptance of menu-driven systems. The continued integration of menu-based interaction with increasingly sophisticated software systems necessitates a comprehensive understanding of their capabilities, as well as their limitations. The descriptive/prescriptive model presented in this paper provides a basis for achieving this understanding. It not only provides a scheme for classifying and characterizing menu systems but serves as a tool for prescribing necessary components and interface requirements.

To provide a foundation for presenting our model, we first give an intuitive view of menu-based interaction including the concepts of frames, frame networks, and frame item selection. Next, we formalize a descriptive/prescriptive model for menu based interaction and discuss its characterization capabilities. The last two sections illustrate how the model can be used to describe and prescribe menu-based systems.

2. Menu-Driven Systems

Intuitively, a system is *menu-driven* if each user response is predicated on a set of choices provided by the system. The system presents the user with a sequence of *frames* (often called *menus*), each containing some descriptive text and a list of *items*. The text provides a description

of the frame, and the items present a set of choices to the user. The user responds by selecting one of the items, causing the system to perform an action associated with that item selection. Usually, this action includes displaying frames. A typical selection process cycle is illustrated in Figure 1.

(0) Display initial frame
(1) Get item selection indicator
(2) If not valid, go to (1)
(3) Execute selected item action, if any
(4) Display next frame, if any
(5) Execute frame action, if frame changed
(6) Go to (1)

Figure 1
Menu Selection Process Cycle

By successively selecting a sequence of items, the user traverses a network of frames that, at times, can be very large [17] and quite complex [5, 3]. A simplified frame network is illustrated in Figure 2. To help the user navigate such networks, many menu systems provide additional facilities beyond those of simply accepting a response and displaying frames. For example, the user may need to revisit a frame, review previous responses, or negate an item selection. Such facilities include commands like:

history : display a history of the items selected,

undo : remove all effects of the last command, and

redo : execute a sequence of previously specified responses as if the user had entered them.

These commands and others like them are independent of the system application, and help provide a standard interface for the user.

Menu-based systems host a variety of operational characteristics. Before presenting the model, we would like to distinguish between a system's features and operations that are intrinsic to the underlying menu system. System *features* are operational characteristics that work in *tandem* with

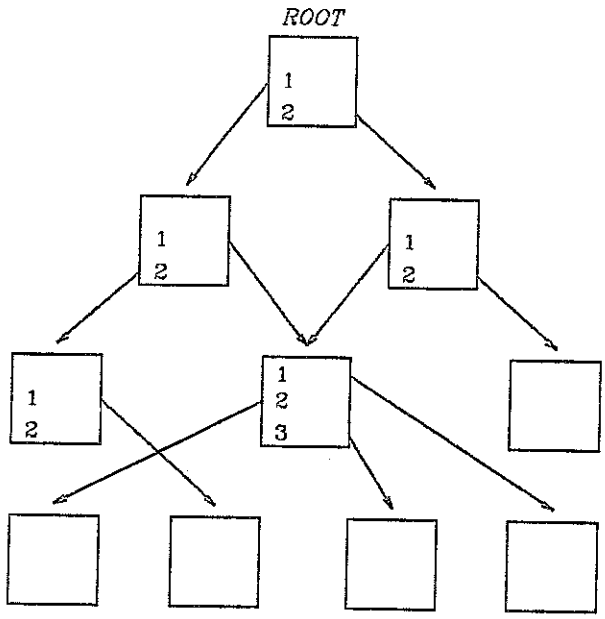

Figure 2
Frame Network Accessible From: *Root*

operations that affect frame network traversal. For example, multiple windows are integral parts of Interlisp [18] and Smalltalk [10]. They are system features, however, and not characteristics of the underlying menu system; essentially the same results can be achieved without them. On the other hand, operations supported by commands like the *undo* and *goto* directly affect the sequence of frames presented to the user. They are not system features because their removal changes the nature of the system. The model presented below characterizes operational characteristics inherent to menu-driven systems, while ignoring details of particular "features".

3. The Descriptive/Prescriptive Model for Menu-based Interaction

Menu systems can be classified according to their characteristic behavior [4]. For example, menu-based systems like Browse [6] and the information subsystems found in Emacs [8] are primarily information dissemination systems; Menunix [14], Quickchart [16], and Promis [20], on the other hand, are task oriented systems. The discriminating characteristics that provide a basis for classifying menu-based systems also serve as a foundation for defining our model.

3.1 The Discriminating Elements

The minimal set of elements inherent to any menu system are: a finite set of *frames*, a set of *user responses*, and a *mapping* from each frame/response pair to another frame. Systems that can be modelled by these elements alone are said to be information systems because their response to any valid item selection is simply the displaying of another frame. By extending this minimal set of elements to include a set of *actions* associated with frame item selection, we can characterize basic task oriented systems. Such menu-based systems can support computational as well as decision operations. The fifth discriminating element, an *incremental history sequence*, is needed to model menu systems that provide user support facilities such as response reversal, item selection histories, and passive response recognition. Finally, to characterize user directed movement within a frame network, we associate with each frame a modicum of *memory* that permits both the naming and marking of frames. Frame names are a prerequisite for operations like the "goto" and "find", while frame marking provides a basis for gathering statistics like frame visit and frequency counts.

In terms of these discriminating elements, basic model components can be intuitively described as follows:

F: Frames. Let F represent a set of frames, each of which, possesses a small, finite amount of user-accessible memory. This set of frames and their respective items define a rigid menu network topology. In general the user traverses the network by selecting frame items. By using frame-associated memory to attach a unique designator (or name) to each frame, the user can locate and specify the next frame to be visited. This property embodies the salient characteristics of a broad class of operations, namely those that support *textual search* and *user directed movement.*

R: User Responses. Let R represent the set of user responses consisting of frame item selection, user response reversal, and user directed movement. These three classes of responses are particularly significant because they alone represent all user responses that can alter the

effective state of a menu-based system. There are, of course, other "responses" such as *show_history* and *find_frame*, but we designate them as ***user commands***, and not user responses, because they do *not* alter the current system state; they only provide information.

A: Actions. Let A be the finite set of actions that are initiated each time the user issues a response to the currently displayed frame. These actions can be partitioned into two distinct groups. We classify the first group to be *task* actions because they are directly related to the application of a given menu system. Task actions include operations that make decisions, calculate, evaluate, and store results for later use. In general, task actions provide an incremental progression toward the user's task solution. The second group of actions are associated with *system* facilities that support user functions, e.g., maintaining history sequences and providing user-directed movement. There is a third group of actions that support basic frame management operations and user commands, but they only display frames and provide information. They are not directly associated with specific frame/response pairs, and hence, are not considered elements of A.

$\mathbb{H}$ *: The Set of History Sequences.* Let $\mathbb{H}$ be a set of all possible history sequences where each individual sequence, $H \in \mathbb{H}$, is composed of zero or more 3-tuples from $F \times R \times A$. Intuitively, a history sequence is an ordered list that describes the interaction between the system and the user in a stepwise, progressive manner. By including the appropriate elements in this sequence, we can characterize operations such as viewing previous responses, revisiting frames, and most importantly, user response reversal.

T: The Transition Function. The transition function T defines the relationship between elements of $F \times R \times \mathbb{H}$ and $F \times A \times \mathbb{H}$. In effect, given a current frame and history sequence, for any valid user response, the transition function defines the next frame to be displayed, actions to be initiated, and the new history sequence.

3.2 The Model

Menu-based interaction can be described and prescribed by the 5-tuple

$$\mathsf{M} = (F, R, A, \mathbb{H}, T),$$

where

F is a finite set of *frames*, each of which possesses a modicum of memory,

R is a finite set of *user responses*,

A is a finite set of actions that support system and task oriented operations,

$\mathbb{H}$ is the set of all sequences over $F \times R \times A$, i.e., the set of all possible history sequences, and

T is a transition function that maps $F \times R \times \mathbb{H}$ into $A \times F \times \mathbb{H}$ as follows:

Let $H \in \mathbb{H}$ and define

$app: \mathbb{H} \times (F \times R \times A) \mapsto \mathbb{H}$ such that $app(H, x)$ is the sequence obtained when the 3-tuple x is appended to H.

$\forall f, f' \in F, r \in R,\ a \in A,$ and $H,\ H' \in \mathbb{H}$ then $T(f, r, H) = (a, f', H')$ where $H' = app(H, (f, r, a))$.

Within the basic framework of our model, we can now describe a basic set of concepts that play important roles in menu-based interaction, i.e., user movement, incremental history sequences, and individual system states.

Let $\{t_0, t_1, t_2, \ldots, t_c\}$ be the set that represents discrete times at which frames are displayed. We use t_0 to denote the time at which the initial frame is displayed and t_c to denote the time at which the current frame is displayed. Let f_i denote the frame displayed at time t_i and r_i denote the user response to frame f_i. Some distinguished frame, f_0, is always the first frame displayed.

User Movement Within a Menu-driven System

When a frame is displayed on the terminal it is said to be *visited* and is called the *current frame*, denoted as f_c. Initially, $f_c \equiv f_0$. The user *moves* from the current frame f_c to frame f_{c+1} by issuing response r_c and is denoted by $f_c \overset{r_c}{\underset{a_c}{\vdash}} f_{c+1}$, where a_c is the associated action. We define a move by the relation

$$L = \{(f_c, r_c, H_c, a_c, f_{c+1}, H_{c+1}) \mid \ T(f_c, r_c, H_c) = (a_c, f_{c+1}, H_{c+1})\},$$

where H_c , $H_{c+1} \in \mathcal{H}$ and denote history sequences at times t_c and t_{c+1}, respectively.

The Incremental History Sequence

A *response path* from time t_m to t_n, denoted by P_m^n, is defined by the sequence of moves

$$\{f_m \overset{r_m}{\underset{a_m}{\vdash}} f_{m+1},\ f_{m+1} \overset{r_{m+1}}{\underset{a_{m+1}}{\vdash}} f_{m+2}, \ldots, f_{m+n-1} \overset{r_{m+n-1}}{\underset{a_{m+n-1}}{\vdash}} f_{m+n}\}.$$

A response path describes a sequence of frames visited, the user response to each frame, the resulting action, and the next frame displayed.

The *history sequence* H is an ordered list of triples

$$\{(f_0, r_0, a_0), (f_1, r_1, a_1), \ldots, (f_{c-1}, r_{c-1}, a_{c-1})\}$$

that provides a system accessible representation of the complete response path P_0^c. H is said to be an *incremental history sequence* because the first element, f_i, in each triple is defined by $T(f_{i-1}, r_{i-1}, H_{i-1})$.

The Current State of the System

As the user traverses a given network topology, new system states are induced. We define a *system state* to be the minimal set of information sufficient to uniquely determine the next system response to a given user response. Intuitively, this means that if we are given the current history

sequence, the current frame, and user response to that frame, the resulting new state of the system can be uniquely determined. At any given time t_i, the corresponding system state, denoted as S_i, can be defined by $S_i = (f_i, G_i, H_i)$, where G_i represents the compound effects of all actions, $a \in A$, resulting from valid user responses. Intuitively, S_i is a *snapshot* of the system at time t_i.

The rationale for including f_i and H_i in S_i is straightforward, they denote the current frame and set of responses leading to the current frame. In addition to these, the combined effect of all task actions resulting from frame item selection and the possible *negation* of item selections must also be included in S_i. G_i embodies this information, and is characterized next.

Let G represent the set of global objects subject to modification by any action $a \in A$. We define G_i to be the set G at time t_i; hence, G_0 represents the initial state of G. Define "Δ" to be the binary operator that describes the result of applying a task action to elements of G. Let Φ be a function defined as follows

$$\Phi(G_i, a_i) = G_i \Delta a_i \equiv G_{i+1},$$

where a_i is the task action at time t_i. In effect,

$$\Phi(G_i, a_i) = G_0 \Delta a_0 \Delta a_1 \Delta \ldots \Delta a_i,$$

and describes the element values of G_{i+1} after $i+1$ moves.

Next, define "∇" to be the binary operator that describes the result of *negating* or *nullifying* the application of an action to elements of G. Let Φ^{-1} be a relation defined by

$$\Phi^{-1}(G_i, a_{i-1}) = G_i \nabla a_{i-1} \equiv G_{i-1}.$$

Effectively, Φ^{-1} defines an inverse operation for Φ. In reality, a program that implements such an inverse operation may be difficult, or impossible, to construct. There exists, however, computational methods that effectively negates an item without requiring a program for Φ^{-1} (see Algorithm 2 in the Appendix). Hence, the *current* state of a system modelled by M can be described by $S_c = (f_c, G_c, H_c)$, where f_c is the current frame, G_c is constructed by c successive invocations of Φ

and Φ^{-1}, and H_c is the current history sequence. Intuitively, if we let "$\odot$" represent the "$\triangle$" and the "∇" operators, then the current system state can be equivalently represented as

$$S_c = (f_c, G_0 \odot a_0 \odot a_1 \odot \ldots \odot a_{c-1}, H_c).$$

3.3 Comments on the Characterization Capabilities of Model M

In general, menu-based interaction is initiated and controlled by the menu system. The system presents a frame to the user and the user provides a response to that frame. In characterizing menu-based systems, one must be able to describe all user/system interactions that induce changes to the current state of the system. Because system states change as a direct response to user interaction, characterizing the effects of elements in R on menu-based systems provide a characterization of menu-based interaction. The following paragraphs discuss those elements.

Item Selections: Active Versus Passive Responses

There are actually two types of item selection: *active* selections, and *passive* selections. Intuitively, an active item selection is a user response that selects a displayed item; a passive selection is a *null* response that causes frame f_{c-1} to be displayed at time t_{c+1}. For example , a simple *return* or *newline* is usually considered to be a passive response.

Active item selections move the user "forward" in the defined network topology. Given the current system state, $S_c = (f_c, G_c, H_c)$, and an active item selection, denoted as r_c^a, then $T : (f_c, r_c^a, H_c) \mapsto (a_c, f_{c+1}, H_{c+1})$ where f_{c+1} is the new frame displayed at time t_{c+1}, a_c is the action applied to G_c (as defined by Φ), and $H_{c+1} \equiv H_c \cup \{(f_c, r_c^a, a_c)\}$. The new system state is $S_{c+1} = (f_{c+1}, G_{c+1}, H_{c+1})$.

The passive response "selects" an item that causes a move to the previously displayed frame. The implication of a passive response is a *nondestructive* transition to the last frame visited. That is, the operations associated with the transition do *not* alter any actions initiated by previous item selections. We emphasize *nondestructive* because a similar move can be initiated via the *undo*

response, but destructively so. The advantages of the passive response are twofold. First, the frame network designer can omit explicit paths back to a frame's immediate predecessor; hence, a reduction in network complexity. Second, by nondestructively returning to predecessor frames, the user can add to the information acquired on previous visits. This is particularly attractive for constructing variable length information like path names.

Given the current system state, S_c, and a passive item selection, r_c^p, then $T : (f_c, r_c^p, H_c) \mapsto (a_c^\phi, f_{c+1}, H_{c+1})$ where a_c^ϕ is a null action, $f_{c+1} \equiv f_{c-1}$, and $H_{c+1} \equiv H_c \cup \{(f_c, r_c^p, a_c^\phi)\}$. Effectively, the new system state $S_{c+1} \equiv (f_{c-1}, G_c, H_{c+1})$. (See Algorithm 1 in the Appendix for further details on item selection).

User Response Reversal

User response reversal is indispensable to any system that supports an interactive user interface. It not only provides a method for correcting user errors and encouraging experimentation, but also reduces the user's *anxiety factor* [9]. In menu-based systems user response reversal is exemplified by the *undo* operation, and has three primary variations: it can function as 1) a meta-operation that is not subject to negation [2], 2) an operation subject to negation by another distinct operation, e.g., *redo* [1], or 3) a parameterized operation that allows the user to select and negate any previous sequence of responses, including other *parameterized undos* [18].

In the model presenter above, user response reversal is based on the ***parameterized undo*** with one restriction: the negation sequence specified by the user must be a sequence of immediately preceding responses. In menu-based systems this restriction is desirable because the negation of internal subsequences may produce system states that are difficult to anticipate or even understand. Because the ***parameterized undo*** effectively subsumes the ***meta-undo*** and the ***undo/redo*** mechanisms, we include its capabilities in model M and assume that the ***parameterized undo*** is the method used for user response reversal. Using M, we now characterize the effects of the ***parameterized undo*** on user response sequences.

Let $r_i \in R$ denote a non-undo user response at time t_i and $u_i(x, y)$ denote the invocation of a parameterized undo at time t_i, where x and y delimit the response sequence to be negated, $0 \leq x \leq y$ and $y = i - 1$. Suppose that we are given the following sequence of responses:

$$V = \{r_0, r_1, r_2, r_3, r_4, u_5(4,4), u_6(3,5), r_7, u_8(7,7), u_9(8,8)\}.$$

In V, $u_5(4,4)$ negates response r_4 and $u_6(3,5)$ negates responses r_3 through $u_5(4,4)$ producing S_6, where $f_6, G_6 \in S_6$ are the same as $f_3, G_3 \in S_3$. Response r_7 is then given but immediately negated by $u_8(7,7)$, hence, $f_8, G_8 \in S_8$ are also identical to $f_3, G_3 \in S_3$. When response $u_9(8,8)$ is given r_7 is reinstated. The final state of the system is $S_{10} = (f_{10}, G_{10}, h_{10})$ where $G_{10} \equiv (G_0 \Delta a_0 \Delta a_1 \Delta a_2 \Delta a_7)$; frame f_{10} is effectively produced by response r_7 to frame f_3.

Because V has nested and overlapping *undo* ranges, a more detailed characterization is difficult. The sequence can, however, be translated into an equivalent form that is easier to analyze, produces the same effective response sequence, but has no nested *undo's*, i.e.,

$$\overline{V} = \{r_0, r_1, r_2, r_3, r_4, \overline{u_5}(3,4), \overline{r_6}, \overline{u_7}(6,6), \overline{r_8}\}$$

where responses $\overline{r_6} = \overline{r_8} \equiv r_7 \in V$. An analysis of sequence $\overline{V}$ will yield a final state S_9, where f_9 and G_9 are equivalent to f_{10} and G_{10} in the original sequence V.

By using an alternate, yet equivalent, user response sequence, we can more easily characterize the *parameterized undo* within the framework of **M**. We note that even though the response sequences are equivalent, their elements differ, and hence, produce different final history sequences. This does not, however, affect the effective sequence of frames visited, actions initiated, nor the end result of menu-based interaction.

The GOTO Response

A basic trait of menu-based interaction is that the *system* controls the dialogue and forces the user to traverse predefined frame network paths. Menu systems that include frames with user and system accessible information objects (i.e., *frame-associated memory*), however, can support

functions that allow the *user* to select the next frame. In such systems, the user can move to any frame regardless of the network topology. By using frame-associated memory to attach a unique designator (or name) to each frame, the user can locate and specify the next frame to be visited. Two user support functions, characterized by the *find* and *goto* operations, implement these capabilities.

The *find* takes as its argument a string of characters and systematically searches the frame network topology for a frame that contains a matching string. If such a frame is found, *find* returns the frame's designated name. The *find* operation, however, is considered to be a user *command* and has no effect on the current state of the system or on the current history sequence. It simply provides the user with the name of the frame that contains the specified string.

The *goto* takes as its argument a user supplied frame name, and initiates a *direct* move to the given frame. Unlike the *find*, the *goto* induces a change in both the current state of the system and the history sequence. The set of user responses that correspond to a *goto* can be formally described as

$$R_{goto} = \{r^j \mid \forall f^j \in F,\ 0 \leq j \leq |F|,\ \exists\ r^j \equiv goto(f^j)\}.$$

That is, for each frame $f^j \in F$ where j denotes a frame's unique designator, there exists a valid user response, r^j, that corresponds to the command "$goto(f^j)$". It is assumed that $R_{goto} \subset R$.

The transition function T induces the mapping $F \times R_{goto} \times I\!H \mapsto A \times F \times I\!H$. In effect, $T(f_c, r_c^j, H_c) = (a_c, f_{c+1}^j, H_{c+1})$ for all possible current frames $f_c \in F$ and $r^j \in R$.

4. Using Model M as a Descriptive and Prescriptive Tool

It is crucial that modelling tools like M exist because they play important roles in categorizing, characterizing and prescribing many facets of physical systems. The following two subsections illustrate the intrinsic power of model M as a descriptive as well as a prescriptive tool.

4.1 Two Descriptive Applications

SMALLTALK

Smalltalk [7, 19] is an environment that supports a style of software development termed "exploratory". In Smalltalk, windows reflect the result of frame item selections and represent global objects that are modified by actions associated with item selections. For clarity, all windows will be denoted as objects. Smalltalk provides two distinct methods for representing frames and items, depending on the type of object being manipulated.

The first method of representing frames assumes a standard format where one frame is presented at a time, the object is viewed as textual data, and frame items describe actions that manipulate that text. The selection of any item within the frame implies a direct or indirect action directed at the associated object. Because there are numerous items within the frame, we have selected only a few representative ones for discussion. The user can select an *editor* item that enables direct manipulation of the object, including defining a domain (text marking) for the next action. If the user has appropriately marked the object, frame items can be selected whose actions initiate *copy*, *delete*, and *insert* operations. If an action is needed for which no item exists, the user types an appropriate Smalltalk command and then selects the *doit* item. The corresponding action interprets the constructed command and applies it to the object. Modelling these operations require only a global object (i.e., G_c), and items whose implied actions are task oriented (i.e., elements of A). Smalltalk also provides an *undo* and *cancel* operations. The *undo* selection negates the effects of the last item selection, and the *cancel* negates all selections back to the last edit session. The Smalltalk *undo* is precisely modelled by the *parameterized undo:* $u(i,i)$ (ref. section 3.3). The *cancel* action can be modelled by $u(i,j)$ where i is time of the first user response that followed the last edit session, and j represents the time at which the last user response was given.

The second method of representing frames is exemplified by "browse windows". Unlike a text object, the object associated with a browse window is manipulated by a set of five frames,

all of which can be viewed simultaneously (see Figure 5). By selecting elements in the first four frames, the user defines a "path" to external, user-defined objects. Frame 4 displays the objects available for viewing and modification. Selection of an item (external object) in frame 4 results in the creation of a browse object whose contents are the selected external object.

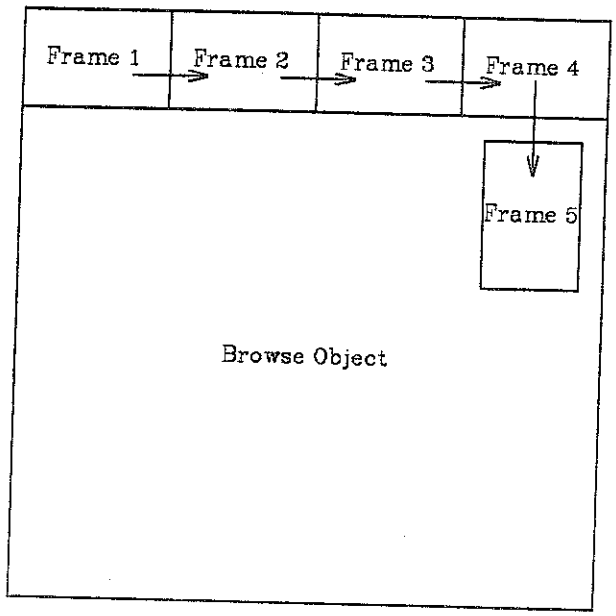

Figure 5

A Browse Window

When the user requests a browse window, frame 1 contains a list of categories; the user selects a category, at which time, frame 2 presents sub categories within the selected category of frame 1. A similar relationship exists between frames 2 and 3, and frames 3 and 4. Although the method of presenting frames has changed somewhat, their characterization is straightforward. There exists a distinguished frame (frame 1), from which, a selected item initiates the display of frame 2, and so forth. The fifth frame duplicates the single frame discussed in the previous method and is characterized accordingly.

ZOG

Zog [12, 13, 15] is a rapid response, menu selection system developed at Carnegie-Mellon University. The user traverses the network of frames by selecting the appropriate frame item. Each

time a new frame is visited, the previous frame is placed on a "backup list", i.e., a *history* list. The backup list provides a history of previously visited frames, responses to those frames, as well as a basis for response reversal (*undo*).

The user can also "mark" a frame before it goes on the backup list; such frames are considered "anchor points". The user support function *return* negates the effects of all selections back to the last marked frame on the backup list.

Zog also supports the user functions *goto* and *find*. *Find* searches for a specified text string within frame specific memory and returns the appropriate frame name. The *goto* permits the user to select the next frame to be visited. If these two functions are used in tandem with the *return* function, e.g., *return* followed immediately by *find* or *goto*, the set of applicable frames is restricted to those on the backup list or the frames pointing to the last marked frame on the list.

The backup list, history facility, and simple response reversal are modelled by $\mathcal{H}$, a *history* command, and $u(i,i)$, respectively; the *return* is precisely modelled by the *parameterized undo*. Frame marking, however, requires frame-associated memory; the supporting operations are actions $a \in A$. The *goto* and *find* functions assume frame memory and are precisely characterized by the like named functions described in section 3.3. Their restrictive invocations can be modelled by a global test variable (in G_c) that is set by *return* and checked by both *find* and *goto*.

4.2 A Prescriptive Application

Model elements that characterize physical properties also serve as a blueprint for constructing systems that possess such properties. The model presented in section 3 was instrumental in the design and implementation of Omni [2], a menu-based, interactive environment for tool selection, specification and composition.

OMNI

In Omni, the primary user interface is based on menu interaction. Model M dictates that any system supporting menu interaction must include frames of items and be sensitive to user

responses. Moreover, M precisely defines the relationship between frames and responses, as well as items and succeeding frames. That is, each frame is associated with a set of user responses that correspond to item selection, and items may point to additional frames. Omni follows this prescribed approach exactly.

If a menu system is to construct information that is not know *a priori*, as is the case with Omni, model M dictates the necessity of operations (actions) that perform the constructions. M further prescribes a direct association between these operations and items selected. That is, each selected item initiates an associated operation that assists in constructing the desired results. Once again, Omni incorporates such an approach.

A third (and desirable) element for menu-based systems is user response reversal. In characterizing such a facility, model M introduces the notion of history sequences as well as operations that support it. Omni bases its *undo* operation on the approach prescribed by model M.

The History Sequence: A Prescribed Omni Component

In designing the Omni environment, it was deemed sufficient to implement user response reversal as a *meta-undo* operation rather that the more powerful *parameterized undo*. As suggested by model M, however, we used an incremental history list as a basis for the *undo* command. Elements of the Omni history list are triples of the form (f_i, r_i, g_i), where g_i represents a pre-modified copy of the elements of G_i that are subject to change by action a_i. (Storing g_i rather than a_i in the triple was a pragmatic decision.) To negate the effects of a_i, g_i is used to restore G_{i+1} to its previous configuration.

User response reversal is only one example of how model M served as an implementation guideline. There are numerous others, e.g., item and frame marking. Although models are primarily used as descriptive tools, their potential as prescriptive tools should not be overlooked. If model M had not served a dual purpose, the implementation of Omni would have been considerably more difficult.

5. Conclusion

One major approach to understanding the software process and its improvement is through the use of models. The application of models to user/system interaction can provide the crucial feedback and innovative insights for designing and developing interactive systems. In this paper we have presented one such model that *describes* as well as *prescribes* the critical components and their interface dependencies for menu-based interaction. The model structure provides the flexibility for characterizing menu-based interaction, varying in levels of sophistication, that includes 1) task oriented actions which support computational and decision capabilities, 2) user response reversal for error recovery, and 3) user directed movement. Finally, to illustrate the intrinsic power of our model, we have presented a brief "descriptive" narrative of two prominent menu-driven systems, followed a by discussion of the model's "prescriptive" influence on the development of a third menu-based system.

Appendix

Algorithm 1 describes a frame item selection. When the user selects an item, T maps the current frame f_c, the user response r_c, and H_c into action a_c, the next frame to be displayed f_{c+1}, and a new history sequence H_{c+1} [(1)]. The history element (f_c, r_c, a_c) is appended to the history sequence H_c [(2)]. The action a_c is performed and G_c is updated [(3)]. A move from frame f_c to f_{c+1} is initiated, resulting in a new current frame [(4)].

Algorithm 1: Frame Item Selection

Select: /* $S_c \equiv (f_c, G_c, H_c)$ */

(1) $(f_{c+1}, a_c, H_{c+1}) = T_3(f_c, r_c, H_c)$
(2) where $H_{c+1} = Append(H_c, (f_c, r_c, a_c))$
(3) $G_{c+1} = \Phi(G_c, a_c)$
(4) Move (f_c, f_{c+1})

/* $S_{c+1} \equiv (f_{c+1}, G_{c+1}, H_{c+1})$ */

Algorithm 2 describes the operations associated with undoing a sequence of previous responses. The transition function, T, defines the next frame and new history sequence. Statements (3) - (16) use the new history sequence to construct an *effective action list* [(3) - (10)] and then rebuild G_i from a global copy of G_0 [(11) - (16)]. History elements are retrieved, starting with those associated with time t_c, by the *Get* command [(5)]. If the associated response at time t_j was an ***undo***, the variable j is set to ignore all history elements affected by that ***undo*** [(6)]. If an item selection was given at time t_j, its associated action is pushed onto an *effective action list* [(8)]. When $j < 0$, an action list has been constructed; when its elements are applied to a copy of G_0 [(11) - (15)], G_i is produced [(16)].

In presenting model M, Φ and Φ^{-1}, are introduced. The function Φ describes the result of *applying* an action to elements of the set G, while Φ^{-1} represents the inverse of Φ, i.e., *negating* the effects of an action. A closer look at Φ^{-1} reveals that it simply transforms G_c into G_{c-1}, the representation of G before action a_{c-1} was applied. In describing the *parameterized undo* in

Algorithm 2: Undo an Item Selection

Undo(i,c): /* Assume a global copy of G_0 exists */
/* $S_c \equiv (f_c, G_c, H_c)$ */

(1) $(f_i, a_c, H_{c+1}) = T(f_c, u(i,c), H_c)$
(2) where $H_{c+1} = Append(H_c, (f_c, u_c(i,c), a_c))$ and
(3) $j = c$
(4) While $j \geq 0$ do
(5) $(f_j, r_j, a_j) = Get(H_{c+1}, j)$
(6) if $(r_j = u(x,y))$ then $(j = x - 1)$
(7) else
(8) *Push* (a_j) onto Eff_Action_List
(9) $j = j - 1$
(10) Endwhile
(11) $G_{temp} = G_0$
(12) While Eff_Action_List not empty
(13) $a = Pop$ (Eff_Action_list)
(14) $G_{tmp} = \Phi(G_{tmp}, a)$
(15) Endwhile
(16) $G_{c+1} = G_{temp}$
(17) Display (f_i)

/* $S_{c+1} \equiv (f_i, G_i, H_{c+1})$ */

Algorithm 2, we have effectively given a method for computing Φ^{-1}. Negating action a_{c-1} is tantamount to issuing a $u(c-1, c-1)$ response that reconstructs G_{i-1}.

LIST OF REFERENCES

1. J. Archer, R. Conway, F. Schnider, "User Recovery And Reversal In Interactive Systems," Technical Report TR 81-478, Department of Computer Sciences, Cornell University, October, 1981.

2. J. Arthur and D. Comer, "Omni: An Interactive Programming Environment Based on Tool Composition," *Proceedings of the IEEE Computer Software and Applications Conference*, Chicago, IL, November, 1984, pp. 28-36.

3. J. Arthur, "Partitioned Menu Networks for Multi-Level, Menu-Based Interaction," to appear in *The 4th Annual Phoenix Conference on Computers and Communications*, Phoenix, AZ, March 1985.

4. J. Arthur,"An Interactive Environment for Tool Selection, Specification, and Composition," Purdue University Ph.D. Thesis, University Microfilms, Ann Arbor, Michigan, August 1983.

5. J. Brown, "Controlling the Complexity of Menu Networks," *Communications of the ACM*, Vol. 25, No. 7, July, 1982, pp. 412-418.

6. M. Fox and A. Palay, "The BROWSE System: An Introduction," *Proceedings of the Annual Conference of the American Society of Information Science*, Minneapolis, MN, October, 1979.

7. A. Goldberg and D. Robson, "A Metaphor for the User Interface Design," *Proceedings of the 12th Hawaii International Conference on Systems Sciences*, Vol.1, 1979, pp. 148-157.

8. J. Gosling, "Unix EMACS," EMACS User's Manual, Carnegie-Mellon University, December, 1981.

9. M. Hammer, *et al*, "Etude: An Integrated Document Processing System," *Proceedings of the 1981 Office Automation Conference*, AFIPS, March, 1981.

10. D. Ingals, "The SMALLTALK-76 Programming System Design And Implementation," *The Fifth Annual Symposium On The Principles Of Programming Languages*, A.C.M., January, 1978.

11. R. Jensen, and C. Tonis, *Software Engineering*, Prentice-Hall, Inc.,Englewood Cliffs, N.J., 1979.

12. D. McCraken and G. Robertson, "Editing Tools for ZOG, a Highly Interactive Man-machine Interface," *International Conference on Communications*, Vol. 1, Boston, MA, 1979, pp. 22.7.1-22.7.5.

13. A. Newell, D. McCracken, G. Robertson, "ZOG and the USS CARL VINSON," *CMU Computer Science Review*, 1980-81, pp. 95-117.

14. G. Perlman, "The Design Of An Interface To A Programming System," University Of California, San Diego Technical Report 8105, November, 1981.

15. G. Robertson, D. McCracken, and A. Newell, "The ZOG approach to man-machine communication," *International Journal on Man-Machine Studies,* Vol. 14, 1981, pp. 461-488.

16. W. Schenker, "Physician-Generated Clinical Records Using A Menu-Driven, Touch-Panel Microcomputer," *Proceedings of the 4th Annual Symposium on Computer Applications In Medical Care,* Vol. 1, November, 1980, Washington, D.C., pp. 1405-1411.

17. J. Schultz and L. Davis, "The Technology of PROMIS," *Proceedings Of The IEEE,* Vol. 67, No.9, September, 1979, pp. 1237-1244.

18. W. Teitelman and L. Masinter, "The Interlisp Programming Environment," *IEEE Computer,* April, 1981, pp. 25-33.

19. L. Tesler, "The Smalltalk Environment," *BYTE,* August, 1981, pp. 90-147.

20. P. Walton, R. Holland, and L. Wolf, "Medical Guidance and PROMIS," *IEEE Computer,* Vol. 12, No. 11, November, 1979, pp. 19-27.