TOWARD A FORMAL SPECIFICATION OF MENU-BASED SYSTEMS

BY

JAMES D. ARTHUR

JANUARY 1986

TR-86-4

Toward A Formal Specification of Menu-Based Systems*

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 the corresponding 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 particular dialogue format: *menu-based* interaction. This model is actually a sequence of models, *hierarchically* structured, where each successive model builds on its predecessor by introducing additional characterization elements. The first model describes the minimal set of elements inherent to any menu-based interface; successive models extend this minimal set by introducing task actions, incremental history sequences, and frame-associated memory. These principal model elements enable the characterization of fundamental, menu-based operations like computational and decision processes, user response reversal, and user directed movement. Moreover, because the principal model elements correspond directly to "real world" objects, an intuitive as well as formal understanding of menu-based interaction can be achieved. Effectively, the model elements and the hierarchical structure imposed by these elements provide and ideal environment for *characterizing* and *classifying* menu-based systems at various levels of sophistication.

* This work was supported in part by R.R. Donnelley & Sons, Chicago, IL

Toward A Formal Specification of Menu-Based Systems[*]

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

There are many techniques commonly used for communication between humans and computer systems. Among the more prevalent ones are question/answer, command-driven, and menu-based interaction; representative systems include Mycin [1], the Unix[†] shell [2, 3], and Browse [4], respectively. Moreover, the complexity of a dialogue session often requires a *collection* of communication techniques. For example, Omni [5] integrates all three of the above interaction formats into a unified system that supports interactive tool selection, specification, and composition.

To construct an effective interface, it is imperative that one understands the ensuing relationship between that dialogue format and the user, as well as the supporting software system. This paper presents a sequence of *hierarchically* structured models that characterizes one prominent communication format: menu-driven interaction. The first model in the hierarchy introduces the basic elements inherent to any menu-driven system and characterizes their role in user/system interaction. Subsequent models introduce additional enhancements that provide characterization capabilities for a wider variety of menu systems, in particular, those systems that support history

* This work was supported in part by R. R. Donnelley & Sons, Chicago, IL

† Unix is a trademark of Bell Laboratories.

facilities and user directed movement. Finally, several existing menu-based systems are classified and characterized within the framework of the hierarchical model.

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. The system may also perform operations that are transparent to the user. These operations consist of validating the user's response, recording it, and in general, providing the continuity among 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 [6] and quite complex [7] (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. These operations and others like them are independent of the system application, and help provide a standard interface for the user.

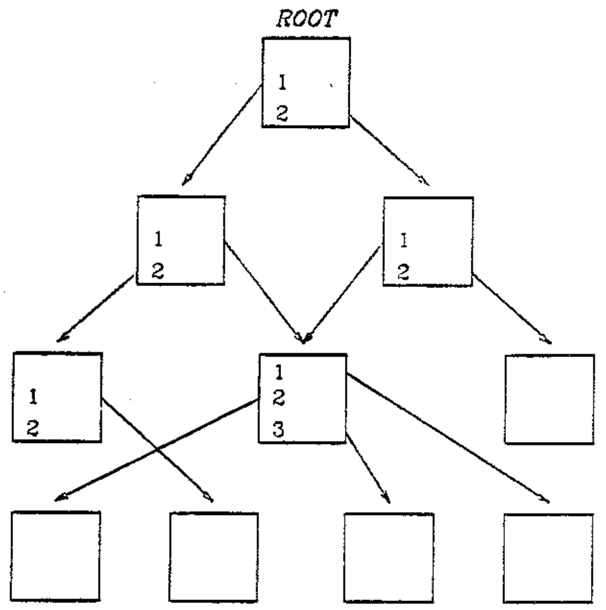

Figure 2
Frame Network Accessible From: *Root*

Menu-based systems host a variety of operational functions. Before presenting the first model, however, a clarification between system features and intrinsic operations must be presented. System *features* are operational characteristics that work in *tandem* with operations that affect frame network traversal. For example, multiple windows are integral parts of Interlisp [8] and Smalltalk [9]. They are system features, however, and not intrinsic to 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 models presented below characterize operational characteristics inherent to menu-driven systems, while ignoring details of particular "features".

3. The Hierarchical Model

As mentioned earlier, the model presented in this paper is actually a sequence of models, hierarchically structured, where each successive model builds on its immediate predecessor. This hierarchy, illustrated by Figure 3, not only provides a basis for *characterizing* menu-driven systems, but also presents a framework for *classifying* them. Moreover, because models often serve as a

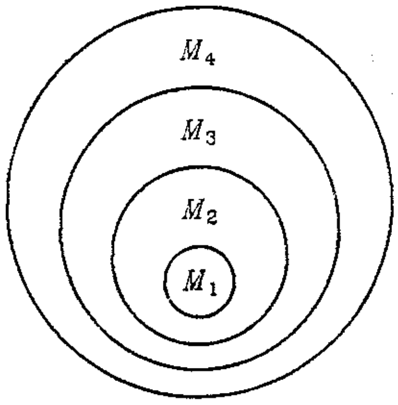

Figure 3
A Hierarchical View of Menu System Models

"blueprint" for system development, a conscious effort has been made to present in "real world" terms the discriminating characteristics that induce the hierarchical structure. Abstractions are used primarily to facilitate a succinct description of these characteristics and to elide unnecessary details.

Because a *sequence* of models is presented, a brief description of each model, its discriminating characteristic(s), and the class of menu systems it can characteristize are given below.

Model $\mathbf{M_1}$ characterizes the minimal set of elements inherent to any menu system. They are: a finite set of *frames*, a set of *user responses*, and a mapping from each frame/response pair to another frame. Systems modelled by M_1 are said to be *information systems* because their response to any valid item selection is simply the displaying of another frame.

Model $\mathbf{M_2}$ extends model M_1 by associating both a frame *and action* with each frame/response pair. These actions are task oriented and support computation as well as decision operations.

Model $\mathbf{M_3}$ extends M_2 by including an *incremental history sequence*. With this extension, M_3 can model menu systems that support response reversal, item selection histories, and passive response recognition.

Model M_4 incorporates the characterization elements of the previous models and additionally, associates a small amount of *memory* with each frame. This frame memory 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 frame specific information, e.g., frame visit counts.

3.1 Menu-Driven Information Systems

One basic function of all menu systems is to provide information to the user. The user obtains information about a particular subject by selecting frame items that successively refine the available subject catalogue. Depending on the design of the frame network, the user may obtain information gradually as frames are displayed, or all at once when the subject catalogue has been sufficiently pruned. The library information system Browse [4] and the information subsystem in Emacs [10] are two menu-based systems that provide such functions. Basic information systems are modelled first because they represent minimal menu-driven systems and illustrate the framework around which all menu systems are constructed.

3.1.1 M_1: A Model for Information Systems

Let $\{t_0, t_1, t_2, \ldots, t_c\}$ be the set that represents discrete times at which frames are displayed. Let t_0 denote the time at which the initial frame is displayed and t_c denote the time at which the current frame is displayed. A minimal menu system can be modelled by the triple $M_1 = (F, R, T_1)$ where

F is a finite set of frames,

R is a set of user responses that select items, and

T_1 is a transition function that maps elements of $F \times R$ into F, i.e., $T_1 : F \times R \mapsto F$.

Using these model elements, a precise description user movement within menu-based systems is now possible, as well as a characterization of individual system states resulting from such movement.

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. 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. A move is defined by the relation

$$L_1 = \{(f_c, r_c, f_{c+1}) \mid T_1(f_c, r_c) = f_{c+1}\},$$

and is denoted by $f_c \overset{r_c}{\vdash} f_{c+1}$. If a response is invalid, denoted as r_{inv}, then the transition function T_1 maps the current frame into itself; in practice, an error message is displayed as well.

Movement is initiated by a user response. The next frame to be displayed, however, is a function of both the response and the current state of the system. In general, a ***system state*** is defined to be the minimal set of information sufficient to uniquely determine the next system response to a given user response. Let S_i denote the system state at time t_i. For menu systems modelled by M_1, the *current* state of the system is defined by $S_c = f_c$. Intuitively, this means that if one is given the current frame and the user response to that frame, the resulting new state of the system can be uniquely determined. Subsequent models will characterize more powerful menu-driven systems, but at the expense of information required to describe a system state.

3.1.2 Comments on Model M_1

Model M_1 characterizes a minimal menu system. Because the behavior of such systems is limited exclusively to moves that result in new frame displays, information dissemination is the only practical function. Moreover, each move must be initiated by an *active* user response, that is, the user must select an item presented by the current frame (this restriction is relaxed in later models). Additionally, the user cannot choose to deviate from the predefined, forward progression of frames. All movement is precisely defined by the transition function T_1, and is based on the current state of the system S_c which is defined to be only f_c (no previous frame or response history exists).

3.2 Task-oriented Menu Systems

The functional capabilities of most menu systems surpass simple information dissemination. In task-oriented systems, there exist actions that define operations for assimilating information that may not be known *a priori*. Such actions are defined to be ***task*** actions because they are directly related to the application of a given menu system. Task actions include routines that make decisions, calculate, evaluate, and store results for later use. In task-oriented systems, frames also display information, but in a manner that facilitates the specification of a user's task. For example, a user may select a sequence of frame items whose corresponding task actions retrieve, modify, and print a specified file. As the user interacts with such menu-based systems, operations defined by task actions provide an incremental progression toward a task solution. Menunix [11], a menu-driven system that assists in selecting and executing Unix programs, is a task-oriented system where the impetus behind the selection and execution operations is supplied by task actions.

3.2.1 M₂: A Model for Task-Oriented Systems

The model presented in this section extends model M_1 by including a set of task actions, elements of which are executed each time the user selects a frame item. In addition to information systems, the resulting model provides a classification category and sufficient characterization capabilities to precisely describe basic, task oriented menu-driven systems.

Task-oriented menu systems are modelled by the quadruple $M_2 = (F, R, A, T_2)$, where

A is a finite set of task actions,

T_2 is a transition function that maps elements of $F \times R$ into elements of $A \times F$, i.e., $T_2 : F \times R \mapsto A \times F$, and

F and R are defined as in model M_1.

The principal difference between menu systems that are modelled by M_1 and those modelled by M_2 is that members of the latter set execute actions associated with each item selection. Elements of A are now used to characterize such activity as well as describe resulting states of the system.

Let a_i denote the task action resulting from response r_i. A move from frame f_c to f_{c+1} now implies the initiation of task action a_c, and can be defined by the relation

$$L_2 = \{(f_c, r_c, a_c, f_{c+1}) \mid T_1(f_c, r_c) = (a_c, f_{c+1})\}.$$

The notation $f_c \overset{r_c}{\underset{a_c}{\vdash}} f_{c+1}$ indicates a move from the current frame f_c to frame f_{c+1}, where r_c is the response to frame f_c, and a_c is the associated task action.

The system response to an item selection must now reflect a move to frame f_{c+1} that includes the effects of the task action a_c. Because task actions can produce compound effects, it is no longer trivial to characterize individual system states. To facilitate a precise characterization of system states that capture those effects, the following terminology is presented.

Let G represent the set of global objects subject to modification by any action $a \in A$. Define G_i to be the set G at time t_i; hence, G_0 represents the initial state of G. Define "$\triangle$" 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

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

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

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

and describes the element values of G_{i+1} after $i+1$ moves. The *current* system state can now be defined by $S_c = (f_c, G_c)$, or equivalently,

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

Intuitively, S_c is a *snapshot* of the system at time t_c.

3.2.2 Comments on Model M$_2$

The transition function T_2 defines an additional relation between elements of $F \times R$ and A. It is in fact, the pairing function

$$T_2(f_i, r_i) = (\beta(f_i, r_i), \alpha(f_i, r_i)),$$

where $\alpha \equiv T_1$ and $\beta : F \times R \mapsto A$. Hence, if the user is visiting frame f_c and provides response r_c, then T_2 maps (f_c, r_c) into (a_c, f_{c+1}). The task action a_c is initiated, and upon completion, frame f_{c+1} is displayed, thus completing a move from frame f_c to f_{c+1}.

How does model M_2 compare to model M_1? Clearly any information capability ascribed to systems modelled by M_1 can be characterized by M_2, $(M_1 \subset M_2)$. In fact, with the inclusion of task actions, the power and application scope of menu systems modelled by M_2 increases significantly. The additional cost of this increase, however, is reflected in the complexity of S_c. Each system state description must now include the current frame as well as the compound effect of all task actions from time t_0 to t_{c-1}.

It should be mentioned that most applications addressed by existing menu systems can be implemented by systems modelled by M_2. This statement does not imply, however, an ideal implementation because many common facilities such as history and user error recovery are lacking. Nevertheless, M_2 does illustrate the intrinsic power of task-oriented menu systems as well as provides a classification category and characterization mechanism for such systems.

3.3 Menu Systems with History Facilities

The previous two models have characterized elements pertinent to the *application* scope of menu systems. Menu-based interaction is often accompanied, however, by far more complex functions than those described above. In particular, system *support* facilities that provide response histories and user response reversal pervade many menu-driven systems.

The model presented next extends model M_2 by incorporating an ***incremental history sequence*** accessible by both the user and the system. As described below, this extension provides the basis for a third classification category, history oriented menu-based interaction. Intuitively, an incremental 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 history sequence, the resulting model can easily characterize operations like viewing previous responses, revisiting frames, and most importantly, user response reversal.

3.3.1 M$_3$: A Model for Menu Systems with History Facilities

Before presenting a formal definition of model M_3, the following notation is introduced to aid in the subsequent discussion of history sequences. Let $\mathbb{H}$ be the set of all sequences over $F \times R^+ \times A$. Let $H \in \mathbb{H}$ and H_i denote the history sequence at time t_i. 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, and

del: $\mathbb{H} \mapsto \mathbb{H}$ such that $del(H)$ is the sequence obtained when the most recently appended 3-tuple is deleted from H.

A menu system with history facilities can now be modelled by the 5-tuple
$M_3 = (F, R^+, A, \mathbb{H}, T_3)$, where

$R^+ = R \cup N \cup O$ such that

R is the set of user responses that select an item,

N is the set of user responses that negate an item selection, and

$O \equiv R^+ - \{R \cup N\}$,

A is a finite set of actions,

T_3 is a transition function that maps $F \times R^+ \times \mathbb{H}$ into $A \times F \times \mathbb{H}$ as follows:

$\forall f, f' \in F$, $a \in A$, and H, $H' \in \mathbb{H}$ then

$\forall r \in R$, $T_3(f,r,H) = (a, f', H')$ where $H' = app(H, (f,r,a))$, and the first two elements of $T_3(f,r,H)$ are independent of H,

$\forall n \in N$, $T_3(f,n,H) = (a, f', H')$ where $H' = del(H)$, and

$\forall o \in O$, $T_3(f,o,H) = (a, f', H)$, and

F is defined as in M_2.

It should be noted that elements of A now include task actions as well as those actions associated with *system* facilities that *support* user functions, e.g., the displaying and maintaining of a history sequence. By letting A^t denote all task actions, A^s denote all system facility support actions, and by including an empty (or null) action in each set, A can be precisely describe by

$$A = \{(x,y) \mid x \in A^t, \text{ and } y \in A^s\}.$$

Because user movement now dictates a change in the current history sequence, a corresponding modification to our definition of user movement is required. A move is now defined by the relation

$$L_3 = \{(f_c, r_c, H_c, a_c, f_{c+1}, H_{c+1}) \mid T_3(f_c, r_c, H_c) = (a_c, f_{c+1}, H_{c+1})\}.$$

A move from the current frame f_c to frame f_{c+1} is denoted by $f_c \overset{r_c}{\underset{a_c}{\vdash}} f_{c+1}$, where r_c is the response to frame f_c and a_c is the associated action.

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. A response path is said to be *complete* if it describes all moves during the time period t_0 to t_c. A complete response path is denoted by P_0^c. Intuitively P_0^c represents the complete interaction history of the current user session; the cumulative effect can be conveniently described and maintained by the following ordered list.

A *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 the function $T_3(f_{i-1}, r_{i-1}, H_{i-1})$. The incremental history sequence is instrumental in characterizing history oriented menu interaction because it provides the basis for reviewing previous item selections as well as reversing previous user responses. Before presenting a detailed analysis of the ensuing ramifications, however, the effects of user response reversal on the current state of the system must be described.

In M_3, the *current* system state is an extension of the one presented in model M_2 and is defined as follows. Let G and "Δ" be defined as in model M_2. 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 described in [12] that effectively negates an item selection without requiring a program for Φ^{-1}. Hence, Φ^{-1} will continue to be used because of its notational preciseness and simplicity, with the understanding that the implementation may not contain a program for Φ^{-1}. The *current* state of a system modelled by M_3 can now be defined 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.

3.3.2 Comments on Model M$_3$

The existence of an incremental history sequence enables model M_3 to characterize menu systems that support *bi-directional* movement. Normal item selection moves the user forward in the frame network; representative commands like the *undo* and *passive* or *null* responses provide backward movement.

A Characterization of User Response Reversal

As previously stated, a history sequence is an ordered list of triples

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

where $f_i \in F$, $r_i \in R$, and $a_i \in A$, $0 \leq i < c$. Let h_i denote (f_i, r_i, a_i), that is, the history element appended to H_i at time t_i.

The initial history sequence H_0 is empty. Depending on the user response, each successive history sequence is constructed by either appending a history element to the previous history sequence or deleting an element from it. For example, if the user selects a frame item at time t_i, $H_{i+1} = app(H_i, (f_i, r_i, a_i))$. On the other hand, an *undo* command deletes the last element of H_i, that is, $H_{i+1} = del(H_i) \equiv H_{i-1}$. Hence,

$$H_{i+j} = \{\ldots, (f_i, r_i, a_i), (f_{i+1}, r_{i+1}, a_{i+1}), \ldots, (f_{i+j-1}, r_{i+j-1}, a_{i+j-1})\},$$

where $0 \leq |H_{i+j}| \leq i + j$.

In model M_3, history elements are appended to the history sequence only when a valid item is selected; the undo operation, itself, is not added. This restriction does not limit the characterization of negating the effects of an undo (i.e., *undoing* an *undo*) because such an operation is accomplished and modelled by simply (re)selecting the item that was initially negated.

Active Versus Passive User Responses

In general, most user responses are considered *active* responses; that is, the user actively selects a frame items or initiates a support function activity. Menu based systems that include history facilities, however, often support an additional type of user response that is said to be a *passive* (or *null*) response. A passive user response is one in which the user neither selects an item nor invokes a support function. For example, a simple *return* or *newline* is usually considered to be a passive response. A passive response at time t_i causes frame f_{i-1} to be redisplayed as the current frame at time t_{i+1}. Information regarding the specific frame displayed at time t_{i-1} (i.e., f_{i-1}) is provided by $h_{i-1} \in H_c$. Effectively, the passive response selects the ubiquitous item that causes a move to the previously displayed frame.

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, a passive response causes a *nondestructive* transition to the last frame visited. That is, the operations associated with the transition do *not* negate any actions initiated by previous item selections. This is particularly attractive because the user can add to the information acquired on previous visits. *Nondestructive* is emphasized because a similar move can also be accomplished via the *undo* command, but destructively so.

Menu systems that support the passive response must also maintain some type of history mechanism. Correspondingly, model M_3, provides a basis for precisely characterizing such a response (via the incremental history sequence) *and* suggests data structures as well as operations for implementing the passive response. Note, however, that the only possible moves from the current frame are still dictated by the design of the frame network. That is, if the user is currently at frame f_c, the only permissible transitions are to its immediate predecessor or successor as defined by the frame network. The forth model, M_4, addresses this restriction.

3.4 Menu Systems with Frame Memory

The menu systems characterized thus far have all shared one common trait: *they* determine the set of frames from which the *next* frame is selected. There exist, however, menu systems that allow the *user* to select the next frame to visit based on information stored in *frame-associated memory* [13, 14]. In such systems, the user is no longer forced to follow a rigid, predefined frame network, but can instead, move to any frame regardless of the network topology. In general, the user can locate and select the next frame to visit via frame-associated memory that contains a unique designator (or name) for each frame. This approach embodies the salient characteristics of a broad class of operations, namely those that support *textual search* and *user directed movement.* Two user support functions, characterized by the *find* and *goto* operations, implement these capabilities.

The *find* operation provides the user with the name of the frame that contains the specified string. Although it has no effect on the current state of the system or on the current history sequence, the *find* does take advantage of frame-associated memory, and in particular, the memory that contains the frame name.

The *goto* takes as its argument a user supplied frame name, and initiates a *direct* move to the specified frame. Unlike *find*, however, the *goto* induces a change in both the current state of the system and the history sequence. Accordingly, the *goto* represents a class of user responses that necessitates an expansion of the set R and the transition function T characterized by model M_3.

3.4.1 M_4: A Model for Menu Systems with Frame Memory

Currently, the set of user responses that initiates constructive user movement is characterized in the hierarchical model by R: item selections and passive responses. That is, they always result in an expanded history sequence and an application of Φ to G_c. Because the *goto* response also initiates such movement, it too must be included in R. Additionally, the appropriate transition mappings must be defined by T. In lieu of this observation, the following notation is introduced to assist in defining the transition function T_4, redefining R, and hence, provide the capability for characterizing user directed movement.

Let F^* be a finite set of frames with memory. Define a new set of responses

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

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 "$goto(f^j)$". Now, define R' to be the set $\{R \cup R_{goto}\}$, where $R \subset R^+$ in model M_3. In essence, valid elements of R' include frame item selections, passive responses, *and* response elements r^j.

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

Model M_4 can now be defined by the 5-tuple $(F^*, R^*, A, I\!H, T_4)$, where

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

R^* $= R' \cup N \cup O$ such that

R' is defined as above, and

N and O are defined as in model M_3,

A is a finite set of actions that includes those described in M_3, as well as system actions that support the *goto* and *find.*

T_4 is a transition function that maps all elements of $F^* \times R^* \times I\!H$ into $A \times F^* \times I\!H$; subsequently, it includes all mappings defined by T_{goto}.

$I\!H$ is defined as in model M_3.

3.4.2 Comments on Model M_4

The frame memory characterized above is said to possess *static* information. That is, the memory contents are immutable during a user session. There also exists frame memory whose contents are *dynamic*, and hence subject to modification by the system. Mutable frame memory provides a basis for supporting data collection statistics like frame visit counts, and human engineering features such as item selection indicators. Because actions that modify frame memory can be classified as task or system actions, both of which are elements of $A \in M_4$, menu systems that exploit the properties of mutable frame memory can be easily characterized within the framework of model M_4.

4. A Characterization of Several Menu Systems

The hierarchical model presented above provides a well defined mechanism for succinctly classifying and characterizing a wide spectrum of menu-based systems. It should be emphasized that the model elements presented, and in particular each sub-model's discriminating characteristics, are defined and described as "real world" objects. This approach not only facilitates the classification and characterization of menu systems, but also provides a natural framework for translating model characteristics into concrete implementation details. In developing and validating the hierarchical model, several menu-driven systems were studied. Part of this procedure included the characterization of four extant systems. To fully illustrate the flexibility and intrinsic power of the hierarchical menu model, the next section presents a brief description of the menu-based systems, Quickchart [15], Smalltalk [16, 17], Promis[6, 18], and Zog [13, 14, 19] within the framework of the appropriate sub-models.

QUICKCHART

The Quickchart system [15] was developed in response to the problems originating from the traditional methods physicians use to generate clinical records. Quickchart is modelled first because it is relatively simple to characterize and exemplifies a "classical" M_3 application.

When the physician initiates a Quickchart session, a global object is created that represents a template for a clinical record. Traversing the frame network corresponds to selecting the appropriate sections within the template; selecting a frame item not bound to another frame causes information to be placed in the currently active template section. Hence, within the Quickchart system there exists a set of frames characterized by $F \in M_3$, R^+ represents the physicians responses (i.e., item selection), and T_3 defines the next frame, action, and history sequence as a function of the current frame, physician's response, and current history sequence. Because the clinical record is a global object, subject to modification, it is characterized by G_c.

Quickchart also provides the following user support functions:

cancel : negates the effect of the last response,

next : causes an activation of the next template section, and

prev : allows the physician to revisit the preceding frame.

The *next* function is effectively a *move*, where the system selects the response; *cancel* and *prev*, however, have a more substantial impact because they require some type of history mechanism. *Prev* requires knowledge of the previous frame (an element of h_c) and is modelled by the *passive* response. *Cancel* can be characterized by an *undo*.

In characterizing Quickchart, model M_2 is not sufficient because Quickchart requires a history mechanism. The capabilities of model M_4 are not necessary because only global memory, represented by G_c, is required.

SMALLTALK

Smalltalk [16, 17] is an environment that supports a style of software development termed "exploratory". With respect to the systems presented in this section, Smalltalk is the most difficult to model because it provides many "features" that operate in conjunction with the menu system.

Some of these features are mentioned during the characterization of Smalltalk, but are noted as such.

In Smalltalk windows reflect the result of a frame item selection. They represent global objects that are modified by actions resulting from an item selection. For clarity, all windows will be denoted as objects. Smalltalk also 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 (actually only one frame exists), the object is viewed as textual data, and the 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, only a few representative ones have been selected 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 resulting action interprets the SMALLTALK command and applies it to the object. These operations can be modelled by M_2, and require only a global object (G_c), and items whose implied actions are task oriented (elements of A). Because Smalltalk also provides *undo* and *cancel* operations, however, a more powerful characterization model is necessary, i.e., M_3. The *undo* selection negates the effects of the last item selection, and the *cancel* negates all selections back to the last edit session. A characterization of the *undo* is straightforward (ref. section 3.3.2), and the *cancel* action can be modelled by successive *undo* operations.

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. Figure 4 illustrates a browse object and its associated frames. By selecting elements in the first four frames, the user defines a "path" to external, user-defined objects. Such objects are typically definitions of abstract data types and their associated operations. An analogous structure exists within Unix, where the items in frame 1 represent directories in the user's home directory; frame 2 represents the directories within a selected directory of frame 1, and so forth. Frame 4 displays the objects available for viewing and modification. Selection of an item (external object) in frame 4 causes the creation of a browse object whose contents are the selected external object.

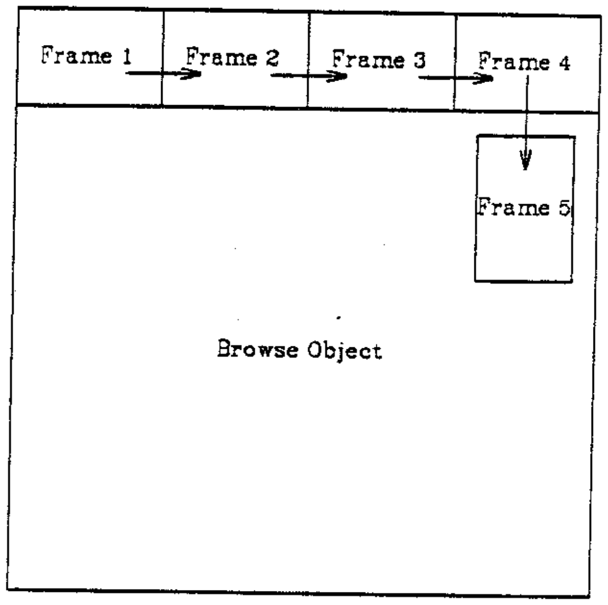

Figure 4
A Browse Window

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.

PROMIS

Promis [6, 18], developed at the University of Vermont, is a problem-oriented medical information system containing branching displays that depict a patient's complete medical history. The user is first prompted for the patient's account number, at which time a frame is displayed and a global object is created that represents the patient's personal information. The information is retrieved from a database entry defined by a dynamic, frame specific, pointer element. This frame is denoted as the "home base" and is the point from which all subsequent frames are visited. The user can select items that modify the current global object, as well as items that select new frames, and subsequently, additional patient information.

When the user selects an item, the given response and current state of the system, S_c, are forwarded to a "Frame Engine". The Frame Engine determines the appropriate application action and the next frame to be displayed. This process is precisely modelled by $T_4 : F^* \times R^* \times \mathbb{H} \mapsto A \times F^* \times \mathbb{H}$. Because each frame represents a different facet of the patient's medical history, the frames are considered to possess mutable memory: a characteristic of model M_4. This information is retrieved and updated by selecting appropriate frame items. PROMIS includes a pattern matching facility (*find*) that searches frame specific information, but instead of returning a frame name, each *find* is followed by an implicit *goto*.

There is no facility for negating the effects of a user selection; however, confirmation is required for all actions that update a patient's medical history. Additionally, the user can revisit frames, thus necessitating a history sequence.

ZOG

Zog [13, 14, 19] is a rapid response, menu selection system developed at Carnegie-Mellon University. The system is actually a skeletal menu system whose application is defined by the given set of frames and corresponding task actions. 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*). Zog promotes the "No Sudden Death" concept, that is, no selection is irreversible.

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 and actions back to the last marked frame on the backup list.

Zog also provides the user functions *goto* and *find*. *Find* searches for a specified text string within frame specific memory and returns the 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. In some sense, this restriction preserves the integrity of the frame network because the user can only move along predefined transition paths.

Model M_4 is essential for the characterization of Zog. The backup list, history facility, and response reversal are modelled by $I\!H$, *history*, and *undo*, respectively. *Return* can be modelled by successive *undo* commands. Frame marking, however, requires frame-associated memory; the actions that support marking are considered to be elements of A^*. The *goto* and *find* functions assume frame memory and are precisely characterized by the like named functions described in section 3.4.1. 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*.

5. Conclusion

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, however, necessitates a comprehensive understanding of their capabilities, as well as their limitations. The hierarchical 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 can serve as a blueprint for implementing them. The hierarchical structure of the model allows one to characterize menu systems, varying in degrees of sophistication, without being encumbered by unnecessary details. Although there may exist menu systems that elude a *complete* characterization by the hierarchical model, the extensibility of the model to include such systems is enhanced by its modular composition and well defined properties.

List of References

1. R. Davis, "A DDS for Diagnosis and Therapy," *Data Base*, Vol. 8, No. 3, 1977, pp. 58-72.

2. S. Bourne, "The UNIX Shell," *Bell System Technical Journal*, No. 6 (Part 2), July-August, 1978, pp. 1971-1990.

3. W. Joy, "An Introduction to the C Shell," *The Unix Programmers Manual*, Vol. 2, Fourth Berkeley Distribution, 1978.

4. 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.

5. 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.

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

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

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

9. A. Goldberg, *Smalltalk-80 - The Interactive Programming Environment*, Addison Wesley, 1984.

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

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

12. J. Arthur, *An Interactive Environment for Tool Selection, Specification, and Composition*, Purdue University Ph.D. Thesis, 1983.

13. G. Robertson, D. McCraken and A. Newell, "Experience with the ZOG Human-Computer Interface System", *International Journal on Man-Machine Studies*, Vol. 21, 1984, pp. 293-310.

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

15. 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.

16. 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.

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

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

19. 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.