Partitioned Frame Networks for Multi-Level, Menu-Based Interaction

James D. Arthur

TR 87-24

Partitioned Frame Networks for Multi-level, Menu-based Interaction

James D. Arthur

Virginia Tech

Abstract

Menu-based systems have continued to flourish because they present a simple interaction format that is adaptable to many diverse applications. The continued integration of menu-based interaction with increasingly sophisticated software systems, however, is resulting in complex, monolithic frame networks with several undesirable characteristics. This paper presents a novel approach to frame network construction and menu-based interaction for application systems that support user task specifications. The approach is based on *partitioning* the conventional, monolithic frame network into a set of hierarchically structured, disjoint networks that preserves the original network topology while *reducing* its overall complexity and size. Moreover, by providing a menu-based interaction scheme that exploits this hierarchical structure, one can realize a system that supports a "top-down" approach to user task specification and user interaction at varying levels of sophistication.

CR Categories and Subject Descriptors: D.2.2 [**Software Engineering**]: User Interfaces; E.2 [**Data Storage Representations**]: Composite Structures, Linked Representations; H.1.2 [**User/Machine Systems**]: Human Information Processing

General Terms: Menu-based Interaction, Partitioned Frame Networks, Top-down Task Specification

Additional Keywords: Menus, Frames, Interfaces

Partitioned Frame Networks for Multi-level, Menu-based Interaction

James D. Arthur

Virginia Tech

Index Terms: Menu-Based Interaction, User Task Specification, Partitioned Frame Networks, Human/Machine Interaction.

1.0 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 [5], the Unix* shell [2, 8], and Smalltalk [6, 7], respectively. These techniques vary widely in their ease of use and their applicability to various problem domains. Menu-based systems are primarily used to present information and to control the actions of computer systems. They are designed to *guide* the user through a sequence of frames that provides information and/or results in a task specification acceptable to the user and consistent with the capabilities of the underlying support system. Because menu-based systems assume control of the dialogue process and solicit simple responses, they are ideal for the novice user, as well as the user who is familiar with an approach to solving a given problem, but needs assistance in selecting or initiating the appropriate sequence of operations that implements a corresponding solution [4].

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 intergration of menu-based interaction with increasingly sophisticated

* Unix is a trademark of Bell Laboratories

software systems, however, is resulting in complex, monolithic networks [3, 10] with undesirable characteristics. For example, a simple extension of a problem domain defined by one of these networks often causes an exponential growth in the size and complexity of that network. This paper presents an alternative approach to frame network definition and menu-based interaction that can significantly reduce the complexity of frame networks and provide capabilities not found in conventional menu-driven systems, e.g., interactive dialogue for varying degrees of user sophistication and true "top-down" task specification. In the next section we present an intuitive description of menu-based interaction and discuss the relationship among frames, frame items, and frame network topologies. Section 3 discusses *network partitioning* as a method for achieving reduced network complexity. It also describes a menu-driven interface that exploits the hierarchical structure of partitioned networks, and as a result, can easily provide the above mentioned non-conventional, menu-based capabilities. Finally, we discuss Omni, an interactive, menu-based environment that uses partitioned frame networks and a *two-level* user interface to support user task specification.

2.0 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. By successively selecting a sequence of items, the user traverses a network of frames. A simplified frame network is illustrated in Figure 2.

Although Figure 2 illustrates a very simple frame network, it is easy to envision a topology consisting of many frames, non-planar edges, and cyclic subnetworks [3]. As the complexity of

```
(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

such networks increase, understanding or modifying them becomes virtually impossible. In the next section, we address this problem by presenting an alternative method for constructing and traversing "would-be" complex networks.

3.0 A Multi-Level, Menu-based System

For clarity, we choose to restrict the discussion of partitioned networks and the multi-level user interface to a menu-based interaction format that supports user task specification. Nonetheless, the concepts presented in the remainder of this paper are applicable to most general menu-driven systems and their corresponding application domains.

We define a task to be a sequence of high-level operations that, when executed, provides a solution to a given problem. The user specifies a task by selecting the appropriate sequence of frame items that defines each operation and associated refinements. For example, suppose that a user has access to a menu-driven, file transformation system and wants to select certain records from a specified file, sort them, and then save them for later processing. First, the user selects the sequence of frame items that defines the file to be *retrieved* and reflects its associated physical attributes. Next the user chooses a sequence of frame items that defines the *record-select* operation and the criteria for selecting the appropriate records. The user then selects a sequence of items that leads to a description of the *sort* operation and all refinements that specify the desired sort sequence. Finally, frame items are selected that denote the *file-save* operation and define all characteristics relating to the destination file. The task operations, retrieve, select, sort, and save, can be initiated at the time each has been fully specified, or when the complete task has been specified.

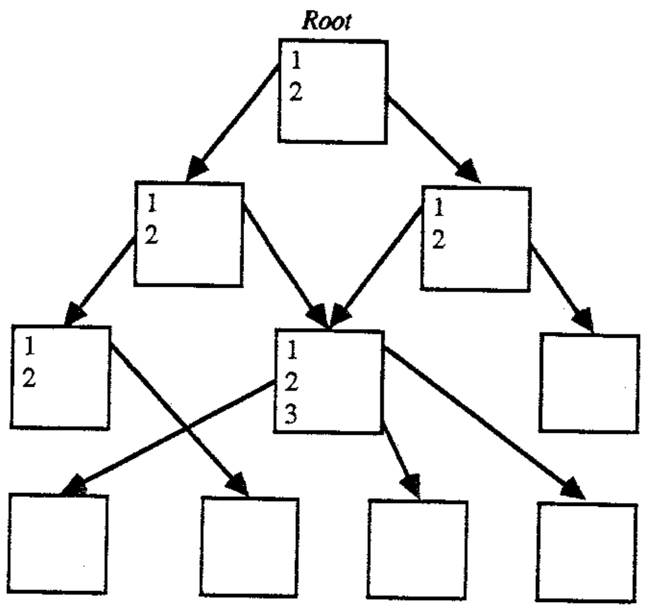

Figure 2
Frame Network Accessible From: *Root*

Because there exists transformation operations that are file-, record-, field-, and character-oriented, all with their own set of refinement possibilities, the supporting frame network will most likely be very large and considerably complex. Moreover, the user usually initiates the specification of each operation from the same frame; there is no way to bypass the predefined network traversal scheme. Not only do such monolithic frame networks form complex structures and impose rigid constraints on user movement, they also require that the user *completely* specify each operation *before* continuing to the next one. This requirement can have an adverse, if not devastating, effect on the user who has to specify a lengthy sequence of operations, each with several refinement attributes.

In the next two subsections, we address these problems within the framework of a multi-level, user interface based on partitioned networks.

3.1 Partitioned Frame Networks

For the example above, a simplified version of a single frame network is illustrated in Figure 3. The *root* frame defines the operation classes, and frames B and C define generic operations on

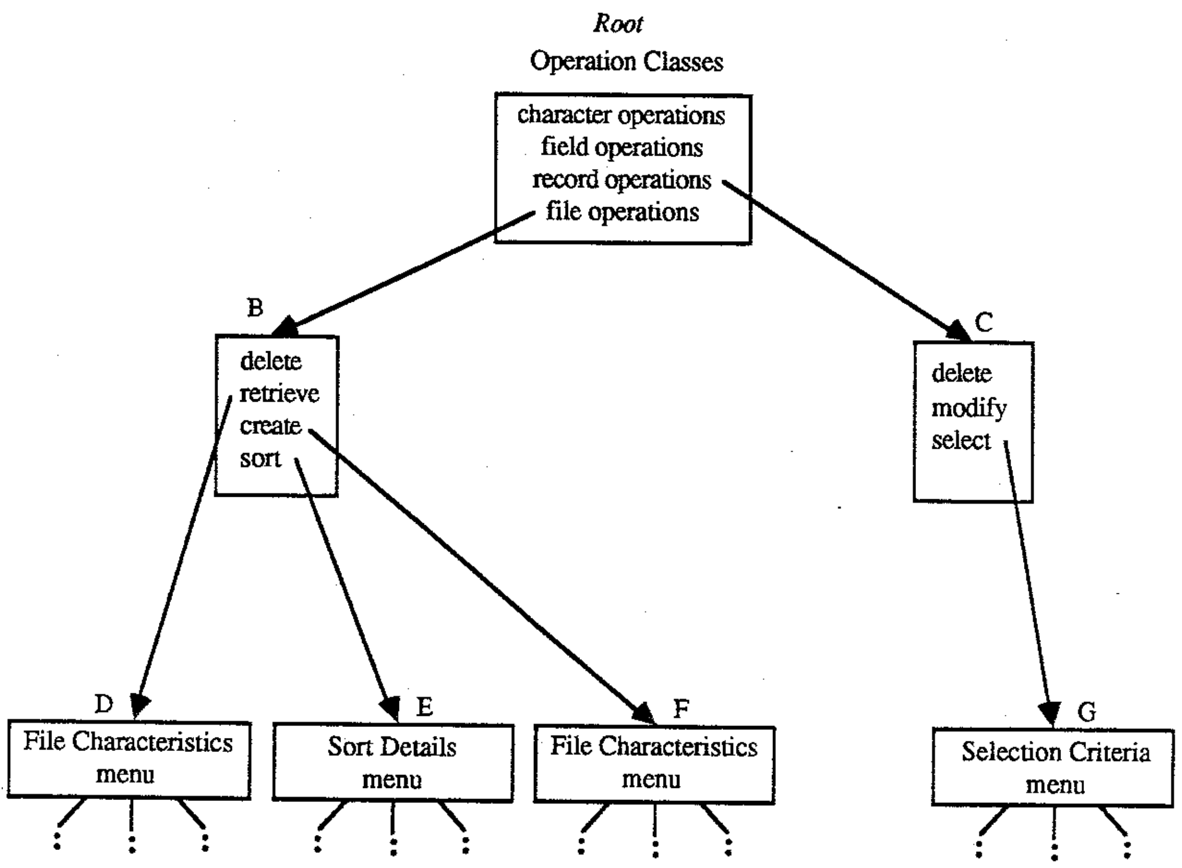

Figure 3
A Simplified Frame Network for File Transformations

two classes; the remaining frames define refinement possibilities. We note that the file-retrieve and file-create operations have similar subnetworks (starting at frames D and F, respectively); these can be combined to reduce the overall size of the network, but at the expense of network complexity.

In partitioning frame networks, and particularly those networks that define task specification sequences (e.g., the network shown in Figure 3), the criteria for determining the hierarchical structure of the resulting disjoint networks are based on the successive grouping (or partitioning) of similar specification properties. Each group induces an *interface layer* that delimits entry points for user interaction. (This concept is discussed in detail in the next section.) Partitioning the frame network serves two purposes: 1) it reduces the complexity of the original network topology, and 2) it provides a hierarchical network structure tailored for "top-down" task specification and interaction levels for varying degrees of user sophistication. Using the generic operations and refinement

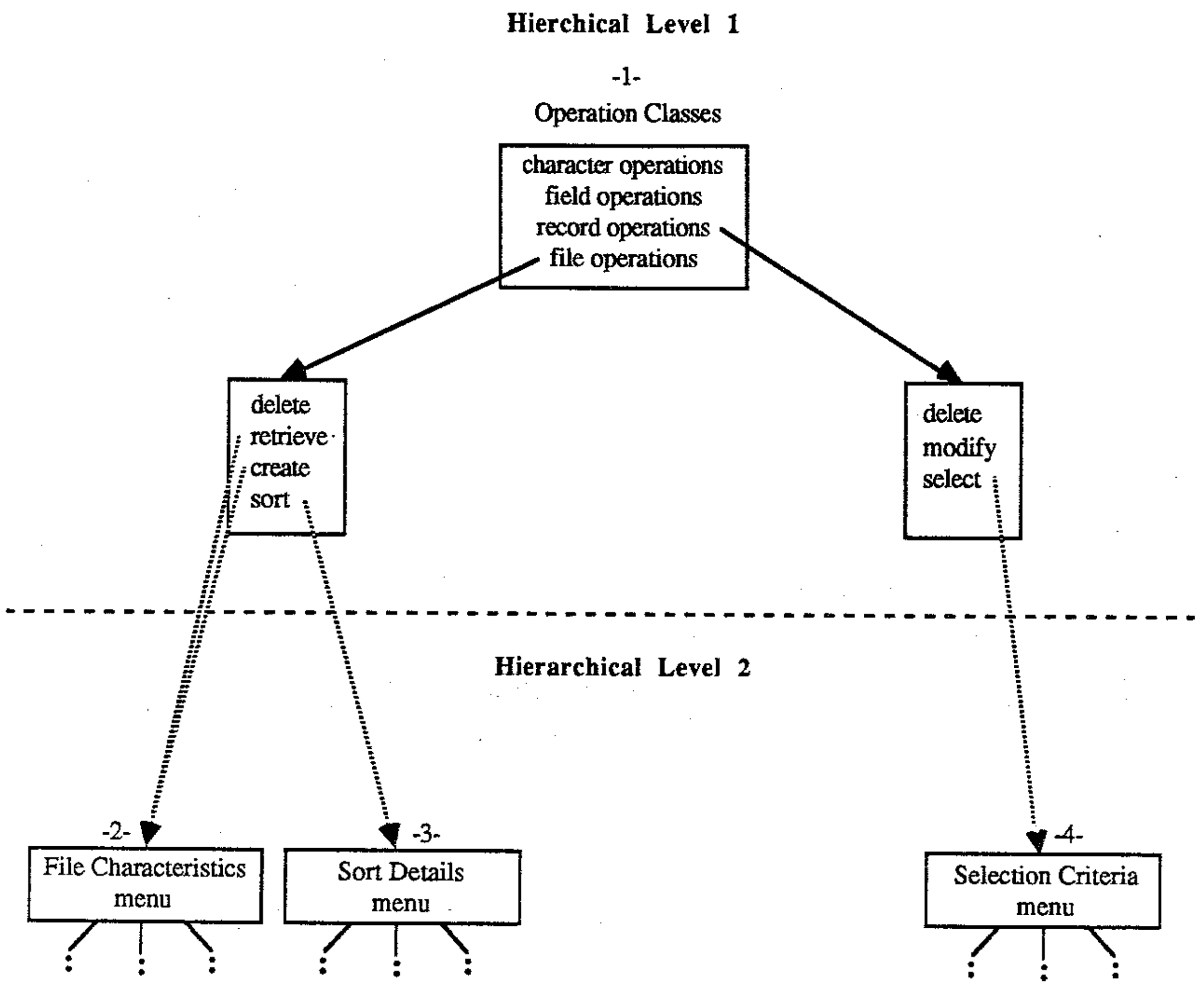

Figure 4
A Partitioned Frame Network for File Transformation

characteristics as grouping criteria for the network shown in Figure 3, the resulting partitioned network, shown in Figure 4, contains four disjoint subnetworks networks and two hierarchical levels. Note that the overall size of the network has been reduced by the number of frames in subnetwork 2 because it now serves as the attribute specification network for both the file-create and file-retrieve operations. Combining these subnetworks in Figure 3 was not desirable because of the added network complexity. Now, however, we have four disjoint networks, each with less complexity than the parent network, and a hierarchical structure that preserves the original network topology. In essence, the initial frame network has been partitioned into a set of hierarchically structured, disjoint networks, where each "new leaf" frame corresponds to a root frame at the next lower level.

How does the user specify a task using the partitioned network – in a *truely* "top-down" fashion. The user enters the network at level 1 and first specifies *all* generic operations in a task sequence. This task overview is constructed (and saved) *without* the user being encumbered with refinement details that could obscure the overall solution sequence. In turn, for each generic operation specified in the task overview, the corresponding frame network from the next lower level is presented to the user for defining refinements to the respective generic operation. If additional hierarchical levels existed, intermediate task specification overviews, each more refined that its predecessor, would serve as gateways to the lower-level networks.

3.2 The Multi-Level, Menu-Driven Interface

The primary function of any menu-driven interface is to guide the user through a sequence of frames that imparts information and/or assists in specifying and solving a given task. In addition to this basic operation, the multi-level, menu-driven interface exploits the hierarchical structure of partitioned networks to provide a truely "top-down", task specification dialogue sequence and a mechanism for allowing the user to enter the partitioned network structure at any level.

As previously mentioned, the successive grouping of similar specification properties induces interactive layers that define levels of user interaction. Because each layer is comprised of a set of distinct, yet complete, frame networks, it is not only possible, but often advantageous for the experienced user to initiate an interactive session and request *direct* access to a lower-level frame network. For example, in Menunix [9] a user is guided through a sequence of frame item selections that constructs a parameterized, tool invocation command which is executed by the Unix shell. By assuming an appropriately partitioned frame network, if the user already knows the general syntax for a tool invocation command, but needs assistance in constructing a particular refinement parameter, the user can request direct access to the hierarchical level that corresponds parameter specifications, traverse the appropriate subnetwork, and collect the desired information. Granting direct access to lower-level networks without first going through the higher level ones is a function of the multi-level user interface. It assumes, however, that the user possess fundamental knowledge

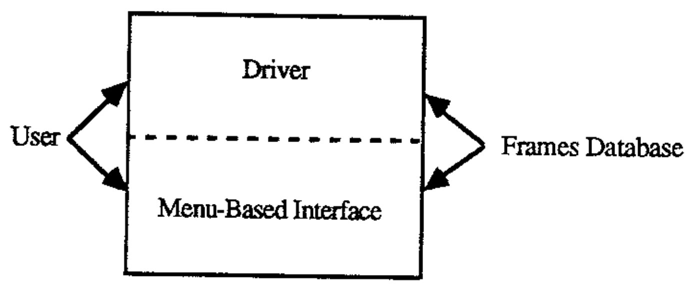

Figure 5
The Multi-Level, Menu-Driven User Interface

of the information defined by the upper-level networks. Hence, a novice user would initiate an interactive session that begins task specification at level 1; a more experienced user can enter at a level commensurate with his expertise.

Figure 5 illustrates a simplified design for the multi-level, menu-driven interface. The menu-based interface provides the standard functions of a basic menu system, but executes under the auspices of the *driver*. It is the driver that first interacts with the user and requests the desired level of interaction. This function can be performed by a question/answer dialogue format or via a frame network designed specifically for this purpose. All frame networks reside in a *frames database* and are accessible to both the driver and the menu-based interface. It is the responsibility of the driver to maintain all intermediate task specification sequences and to insure that each specified operation therein, be completely refined with respect to the current hierarchical level before directing the menu-based interface to start retrieving frame networks from the next lower level. Because each task operation is refined as much as possible at each level of interaction, a truely 'top-down" task specification is guaranteed.

4.0 Omni: An Environment Based on Partitioned Networks

Partitioned networks and a multi-level, menu-based interface are reflected in the design and implementation of the Omni environment [1]. Omni is an interactive, menu-based environment that supports problem solving via tool selection and tool composition. Each tool is a powerful parameterized program that performs a single high-level operation (e.g., *sort* a file). To solve a given problem, the user interacts with the system to select an appropriate set of tools, and then composes them into a sequence. Such sequences are called *compositions*; when a composition has been fully expanded to include punctuation and parameterization details it is called a *script*. The script is then passed to the underlying operating system, Unix, and is executed. Omni is a general purpose, menu-based environment whose application domain is defined by the partitioned frame network.

4.1 The Two-Level Interface

The initial prototype for Omni consisted of a single, menu-driven interface. The supporting frame network was structured in the conventional manner. In this environment, each time a high-level operation is detected in the problem specification, a corresponding tool is selected and *immediately* expanded into a tool call. The primary disadvantage of such an approach is that the user must contend with tool particulars (arguments) while maintaining a global view of the sequence of high-level operations that effects a problem solution.

The current prototype takes a *two-level* approach to problem specification and exploits the properties of partitioned frame networks. It allows the user to first specify all of the necessary operations for solving a given problem, and then concentrate on their individual details. Figure 6 illustrates the logical design of the two-level interface. The responsibility of the upper-level interface is to acquire a sequence of high-level operations that outlines a problem solution, and select a corresponding sequence of generic functions. In turn, the lower-level interface solicits the necessary details for constructing a corresponding sequence of tool calls.

As one might conclude from Figure 6, both interface levels share common software; in fact, they are the same mechanism. Their differing functions are respectively defined by the hierarchical levels of the partitioned frame network. When the user initiates an Omni session, the menu-based interface assumes the upper-level configuration. The *composition database* supplies the menu network that defines the primitive operations supported by the Omni environment. Once a problem

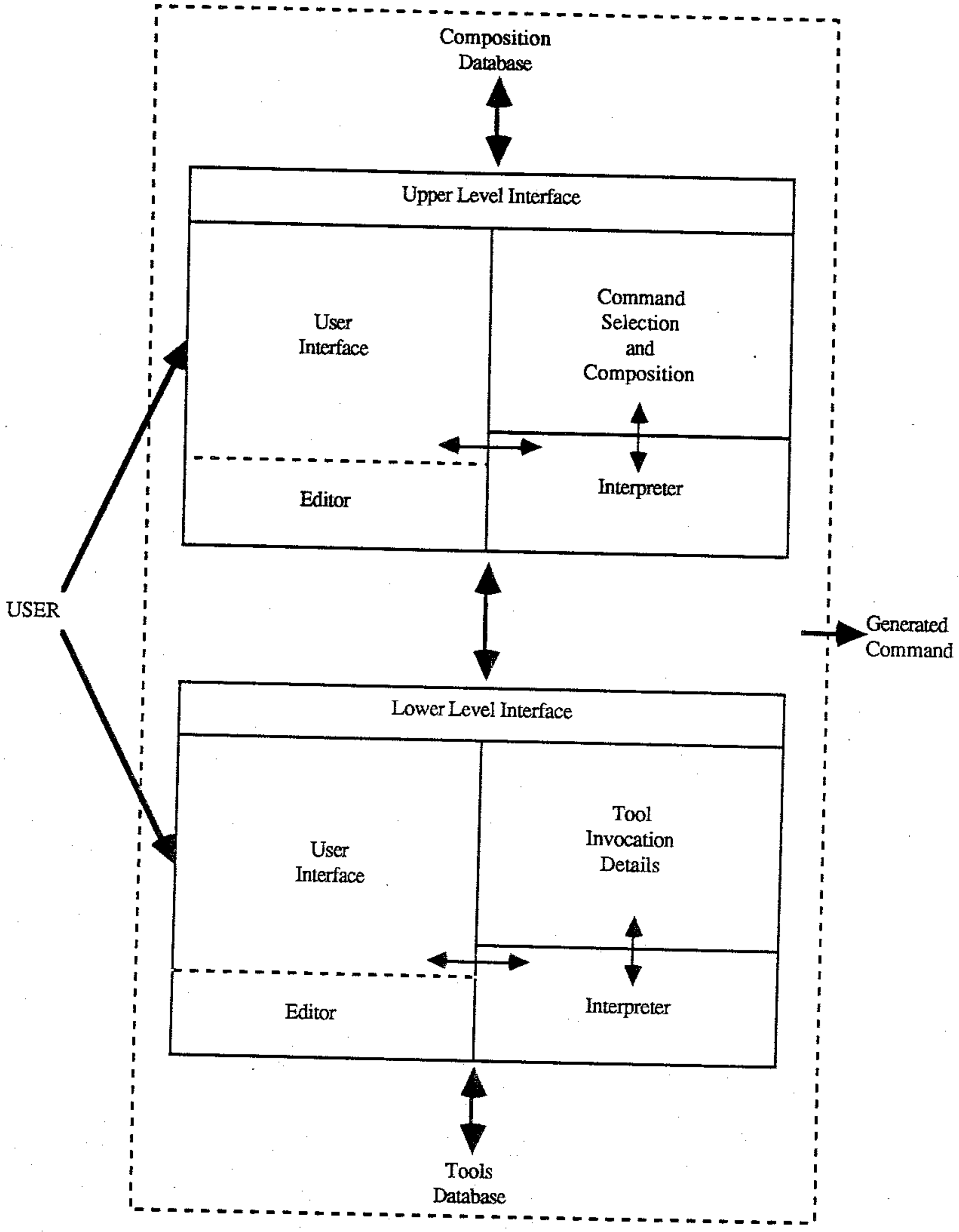

Figure 6
The Logical Structure of the Two-Level Interface

overview is established by the upper-level interface, the menu-based system assumes the posture of a lower-level interface and sequentially processes the list of generic functions. For each generic function, the menu-driven system is dynamically reconfigured according to the structure of the corresponding frame network obtained from the *tools database*. Interaction at this level enables the user to describe details peculiar of each generic operation. For example, the upper-level interface may generate the generic operation *sort-file*. The lower-level interface may expand this operation into the tool call *sort 5-10*, i.e., sort on columns 5 through 10.

From a user's perspective, one advantage of the two-level approach is that a problem overview can be specified before problem details, i.e., "top-down" specification. The user initiates an Omni session, describes the given problem as a natural sequence of simple, high-level operations, and then supplies the details for each operation. From an implementation perspective, the menu networks are considerably smaller and more easily constructed. Moreover, the semantic actions that construct the tool calls can be logically partitioned; this promotes modular software construction.

As the user becomes more proficient in the use of tools and tools composition, the need to describe an entire problem specification to Omni decreases. In an effort to provide services to both the novice user as well as the experienced user, Omni supports two distinct levels of interaction. In the two-level system, the first level concentrates on tool selection and composition; the second level provides tool details. This multi-level approach allows the novice user to enter the first level, select tools, and then explore the details through the second level. A more experienced user can enter directly at the lower level to determine details about a specific tool.

4.2 A File Transformation Example

Omni is a general purpose environment whose application domain is defined by partitioned frame networks. One current Omni environment addresses file transformation problems and supports character, string, record, and file oriented operations. The following paragraphs provide a brief overview of such an environment by describing a "tools approach" to file transformation and discussing the related operations.

The Tools Approach to File Transformations. Suppose we are given a file F and want to construct file F' by applying transformations t_1, t_2 , and t_3. One approach is to successively apply all three transformations to *each* record $r \in F$, $1 \leq i \leq |F|$. Such an approach requires one pass through the file F, but many invocations of the transformation routines. An alternate approach is to apply transformation t_1 to *all* records in F, apply t_2 to the results of the first transformation, and then apply t_3 to the results of the t_2 transformation. This approach requires three passes through the records but only one invocation of each transformation routine.

The second approach is consistent with the underlying principles of tools and tool composition. If tools T_1, T_2, and T_3 implement the transformations t_1, t_2, and t_3 respectively, and "$\odot$" denotes composition, then the script "T_1 *filename* $\odot$ T_2 $\odot$ T_3" describes the complete transformation process. From an Omni perspective, the transformations are high-level operations supported by the tools T_j. The script is a sequence of tool calls that specifies a solution to the user's problem.

File transformations involve several classes of operations. The following paragraphs describe these classes and discuss their respective elements.

File Primitives. File primitives are transformation operations that view files as a single entity. That is, the smallest manipulative element is the file itself. These operations include:

retrieve:	retrieve a file for subsequent operations,
move:	move a file from its current location to a specified destination,
copy:	copy a file to a specified location,
create:	create an empty file,
delete:	delete a file, and
sort:	sort a file.

As these operations are expanded into actual tool calls, file names and other pertinent data are obtained from the user via prompts.

Record Primitives. Record primitives are transformation operations that consider records as the atomic unit. Such operations include:

select: select a record for further processing,

delete: eliminate specified records from further consideration, and

change: change the contents of a record.

Although these operations may seem rather simple, they possess extensive capabilities. They understand both the distinction between variable and fixed length record formats, and can choose records based on their contents. This implies an additional understanding of *fields*, field *delimiters*, and comparison operators.

Character and String Primitives. The third class of transformation operations addresses character and string manipulation. These primitives consider each file as simple text without any perceived structure. Operations in this class include:

delete_text: delete all occurrences of a specified string,

replace_text: replace all occurrences of a specified string with another string, and

insert_text: insert a specified string according to given specifications.

By considering a character as a string of length one, the above operations generalize to character manipulation.

Hence, a top-level (generic) script that selects all student records with a graduation date of 1983 (record positions 52-55) and sorts them by last name (record positions 10-19) might appear as

$$\textit{retrieve}\ \text{student\_file}\ \odot\ \textit{select}\ \odot\ \textit{sort}.$$

The corresponding low-level (expanded) script is

$$\textit{retrieve}\ \text{student\_file}\ \odot\ \textit{select}\ \text{p52-55='1983'}\ \odot\ \textit{sort}\ \text{p10-19}.$$

5.0 Conclusions

As menu-based systems continue to proliferate and are applied to larger problem domains, the complexity of the supporting frame networks and ensuing user interaction will increase proportionally. This paper presents one method for reducing conventional frame network complexity and improving user interaction. This method employs a multi-level, menu-based interface driven by partitioned frame networks. Advantages of a multi-level, user interface are threefold. First, conventional, monolithic frame networks can be partitioned into a set of hierarchically structured, disjoint networks that preserves the original network topology while reducing its overall complexity and size. Second, because the networks are hierarchically structured, we can support user sophistication levels that range from that of the novice to that of the expert. Finally, the multi-level interface promotes a true "top-down" approach to task specification because the hierarchical nature of the supporting frame network can be constructed to reflect layers of successive task refinement.

LIST OF REFERENCES

1. J. Arthur and D. Comer, "Omni: An Interactive Programming Environment Based on Tool Composition," to appear in *Proceedings of the IEEE Computer Software and Applications Conference,* Chicago, IL, November, 1984.

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

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

4. E. Carlson, "Developing The User Interface For Decision Support Systems," IBM Research Report RJ3112, IBM Research Laboratory, San Jose, CA, April, 1981.

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

6. 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.

7. 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.

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

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

10. 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.