A Browse / Edit Model for User Interface Management

Dan R. Olsen Jr.
Computer Science Department
Brigham Young University
Provo, UT 84602

Abstract

The history of user interface management tools is discussed and the dominance of input event handling is pointed out. An alternative UIMS model based on browsing and editing is presented. A possible architecture for such a UIMS is presented and the implications of such a design are discussed. User interfaces are specified in this model by a selecting from a library of generic browse / edit facilities and then refining them with additional specifications or formats.

Keywords: user interface management systems, interactive data, browse / edit.

1. Introduction

For a number of years researchers have been developing techniques for improving the way that human-computer interfaces are created. This work has centered around toolboxes, user interface management systems (UIMS) and to some extent object-oriented languages. It is the purpose of this paper to explore those approaches and identify some underlying assumptions about user interface software that have driven this work. There are two basic assumptions that are pervasive in user interface support tools.

1. The primary function of a user interface support tool is to handle and interpret input events.
2. The interface between the UIMS and the application can have infinite variety.

This paper will first discuss how and why these two assumptions came to dominate user interface tool development and will then propose alternatives to these two assumptions. The ideas in this paper are not presented as finished or fully developed concepts. Many of the ideas presented are the result of conversations with colleagues or insights derived from the work of others. Many of the proposed techniques can be found in other research but these concepts have been peripheral rather than central to that research.This paper does not report a research result but rather proposes a shift of focus.

2. The Development of User Interface Tools

Interactive user interfaces have been part of computer graphics since Ivan Sutherland's SKETCHPAD [Sutherland 65]. Almost immediately after that first beginning proposals were made for tools that would ease the burden of developing interactive software. Newell's Reaction Handler [Newman 68] was the first and was based on state machines. Much later Kasik proposed a tree of menus with FORTRAN routines to be called at the leaves of the tree [Kasik 76]. Hanau and Lenorovitz modeled interactive user interfaces as grammars using YACC [Hanau 80] and Feldman developed the Abstract Interaction Handler [Feldman 82] which is also based on state machines.

In 1982 the Core and GKS discussions were a major topic of discussion when a workshop was held in Seattle on the subject of graphical interface and interaction techniques [Thomas 83]. The general tone of this workshop was that graphical output was relatively well understood (Core and GKS) but that graphical input was in a woefully undeveloped state. Although the folklore holds that the term user interface management system (UIMS) was in existence before this workshop its widespread usage dates from about this time. The perception that graphical input was not well understood along with the directions that researchers were already taking reinforced assumption number 1 (dominance of input events). Based on this assumption almost all UIMSs are characterized by their *dialog model* which is a somewhat inaccurate term for *how they handle input events.*

A second issue stemming from this first Seattle workshop is that of internal vs. external control. This issue is the primary dividing line between toolboxes and UIMSs. In an internal control system the application calls the toolbox to have interactive and graphical services performed but the application retains control of the dialog. In a UIMS (external control) it is the user interface software that is in control and makes calls on the application for services. With internal control the user interface/application interface is simply a set of routines to be called by the application. In an external control system the set of services that an application can provide to a user interface is potentially infinite which leads to assumption 2.

2.1. Event-based User Interface Systems

As mentioned earlier almost all user interface support tools have been dominated by their model for handling input events. These can be divided into three categories: grammars/transition networks, event dispatchers and toolboxes. Only a few example systems rather than an exhaustive set of references are included in this paper.

2.1.1. Grammar and transition network tools

These UIMSs are descendants of Newman's Reaction Handler and as Foley has pointed out have not progressed extensively since then. The Syngraph system [Olsen 83] was based on grammars with IPDA [Olsen 84] and Jacob's work [Jacob 85] being based on transition networks. The primary goal was to translate sequences of input events into calls on the services of

a particular application. These systems did recognize, however, that the user interface also needed to control the presentation of the dialogue including how messages, menus, buttons and help appeared on the screen. The input dialog nature of the systems, however, made the inclusion of presentation information somewhat awkward.

2.1.2. Event Dispatchers

A variety of systems have viewed the user interface problem as one of dispatching events. The model that objects appearing on the screen can be poked and prodded by mice (or other "pointy" devices) is the basis for a number of object oriented strategies [Anson 82]. The Menulay [Buxton 83] system attacked the presentation problem by drawing pictures and attaching C routines to them. The C routines are invoked when the corresponding picture is poked or prodded. Work by Rosenthal [Rosenthal 83] dispatches events to C routines based on the window where the event occurs. As with the grammar and transition network systems the model is to map events to application routines. The fundamental difference is that the event mapping is not based on event sequencing but rather on the visual object that the event is directed at.

2.1.3. Toolboxes

After UIMS research was well underway a fundamental shift occurred in computer graphics away from vector based displays with a few non-overlapping viewports to bit mapped displays with multiple overlapping windows. The output models of Core and GKS were not adequate and new graphics packages (or tool boxes) were developed. As such tool boxes developed, new attention was paid to the input model. At a basic level most such tool boxes are based on some GetEvent routine through which all inputs are passed along with some window-based dispatching of events. The quest to give toolboxes more sophisticated input facilities has, however, lead to some approaches which deviate from the two basic assumptions of event dominance and variety of application interfaces. These developments and their implications are discussed in a later section.

2.2. Task based UIMS models

Recognizing the limitations of event dispatching or event parsing dialog modes, a number of UIMSs have moved towards models based on the purpose the user interface is to serve rather than on what sequence of events is necessary. The idea is to describe the nature of the application and then let the UIMS decide the sequence of events necessary to implement the interaction.

2.2.1. EZWIN

EZWIN [Lieberman 85] is a Lisp-based object oriented system that models the user interface as a set of presentation objects and a set of command objects. Command objects consist of their menu name and a list of their desired arguments. Presentation objects are anything else displayed on the screen. EZWIN is responsible for automatically providing a dialog which maps presentation objects as arguments to commands.

2.2.2. MIKE

The MIKE UIMS [Olsen 86] models the application as a set of data types and a set of procedures and functions that can be applied to objects of those types. From this application description a default user interface is generated which allows the designers to interact with the application. The user interface is then refined using a profile editor which remaps the default menus, names, windows and messages into new visual forms that are more pleasing as part of the user interface. MIKE then decides what interactive events and actions are necessary to invoke various application commands and functions.

The focus of both of these systems an others like them is on what the purpose of the application is rather than the sequence of input events. This is the basis for the shift of focus proposed by this paper.

3. Browse/Edit Toolbox Facilities

In an effort to extend the input capabilities of graphical toolboxes a number of features have been added. The first feature that almost every tool box includes after windows is menus. These take the form of defining some data structure which specifies how the menu should appear and then calling a tool box routine to display the menu and possibly a different routine to get input from the menu. The result returned is an identifier of which item was selected. For purposes of this discussion the data structure that describes the menu will be termed its *format*, the choice returned will be its *data value* and the concept of a menu will be termed a *format class*.

Another facility (or format class) that is frequently included is a dialog box or property sheet. In this case the format consists of a description of the graphical objects displayed in the dialog box along with the locations of certain fields or controls. Fields typically allow data to be typed into them. Controls such as radio buttons control a set of choices for some data value; check boxes turn values on and off and scroll bars and thermometers change integer or real values. The application either passes in a format or creates one by calls on the toolbox routines. The application can then either pass in or set the values that the fields and controls are to represent and can get the changed values back. Tool boxes vary in how dialog box formats are created, how values are set and retrieved and how the application is notified of value changes but the concept is essentially the same.

Some tool boxes have become more ambitious and supply a full text editor as a toolbox facility. The format consists of the editing window, text style and font and other attributes such as word wrapping and tabs. The value is a string to be edited. The latest version of the Apple Macintosh toolbox includes a complete formatted text editor including varying font size, bolding etc. inside of the text. The format information is essentially the same but the value being edited is extended to include the formatting attributes.

A similar facility which is either included in or is available with most tool boxes is an icon or bit map editor. The format is simply a window to edit in and the data is a bit map.

Each of the toolbox facilities described above has the following characteristics:

a. The facility embodies a particular style for viewing and editing data
b. There is either an implicit or explicit format which can store the application specific choices that control how an instance of this facility is to behave.
c. There is an implicit or explicit model of the application data that is being viewed or edited.

Each facility is then an instance of a particular, widely used style of visual browser/editor. Apple's Hypercard system [Apple 87] is simply a combination of the bit map, dialog box and text edit facilities with a primitive data model. Hypercard, with all of its limitations, has demonstrated the ability to build

interactive applications by piecing together simple browser/editors.

4. A Browse/Edit UIMS Model

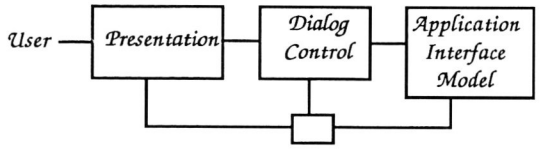
Most previous user interface work has centered around handling the inputs when in actuality no interactive user is concerned about how to handle input. The task of almost every interactive system is to allow a person to browse through and modify data which represents some problem domain. The power of interactive computer graphics is not in its device handling but rather in its ability to present information in a visual manner and to allow users to manipulate that information. When one focuses on information manipulation rather than on input event handling the architecture of user interface support software radically changes.

This information visualization and manipulation or browse/edit model is further reinforced by the successes of various user interface support products. With the exception of Apollo's Domain Dialogue system and to a more limited extent TIGER [Kasik 82] most UIMSs have remained in research laboratories even though implementations have been available for several years. On the other hand Hypercard [Apple 87] has had almost immediate acceptance. The most commercially successful user interface software systems are found associated with fourth generation languages in the data processing field. There are numerous screen design systems with wide commercial usage and success. They admittedly solve a simpler problem than most UIMSs have attempted but they are successful. They achieve this success because they directly attack the application's user interface goals rather than attacking the mechanisms and because they map user interface concepts directly to programming concepts. They provide simple, clean metaphors for both users and programmers.

One of the long standing deficiencies in UIMS research has been the lack of support for graphical output. Experience with the MIKE UIMS has shown that the input portion of the interface is completed in several days while the screen update code takes several months to design, integrate with the input definition and debug. The Editing Templates system imposed a simple browse/edit model in top of MIKE and achieved limited success in resolving this display update problem. Garret and Foley [Garret 82] have modeled user interfaces based on relational databases shared between the application and the user interface. Traditional relational data models, however, are inappropriate for many of the kinds of highly structured data that are graphically manipulated. A related approach is being taken on the Serpent project at the Carnegie-Mellon Software Engineering Institute[1].

4.1. Architecture

Many UIMSs have taken the Seeheim [Green 83] model as their guide. This model is shown in the figure below.

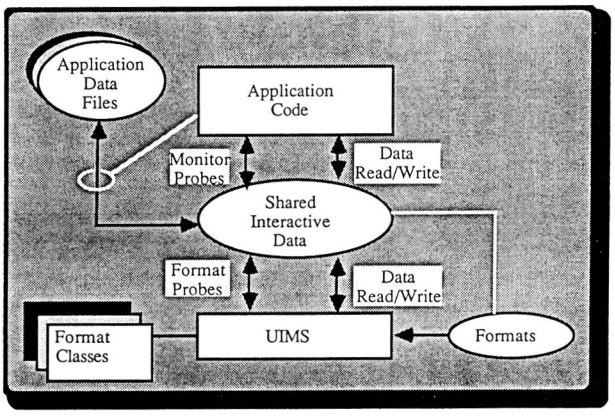


Seeheim Model

This model is dominated by input event handling with relatively weak support for interactive output.

[1] private conversations with Len Bass at Carnegie Mellon University, Software Engineering Institute.

An alternative model is shown below.



In this model the UIMS shares a data structure with the application. This data structure contains all of the information that the application wishes to share with the interactive user. A format class is a basic data browsing and editing technique. Text editing, dialog boxes, pop-up menu or bit map editing are examples of such format classes. A format is the piece of information that describes how data is to be edited or viewed using the format class. In the case of a menu the format describes how the menu is to be displayed. In the case of a dialog box the format contains how the fields should be laid out. Formats are closely analogous to resources on the Apple Macintosh.

4.1.1. Formats and Format Classes

A format class is a piece of code that must know how to draw images on the screen given a format and some data. It must be able to respond to interactive inputs from the user and update the shared interactive data values in response to such inputs. For example the dialog box format class would change the corresponding data field in response to characters typed into one of its fields on the screen. Formats must be represented in the same data model as the shared interactive data. A format class must also provide a format/format class pair which can be used to edit its own formats. Like many tool boxes a set of predefined format classes would come with the UIMS. It must be relatively easy, however, for programmers to add new format classes to the set. Dialog design for a new application then consists of editing new formats for each of the kinds of information that the application wants to share with the interactive user. Since each format class carries with it the information necessary to edit its own formats the system is used to create itself. By adding new format classes and designing formats to edit their formats the UIMS itself can readily grow and expand to meet new application needs.

At present tool boxes provide only a limited set of format classes which will be widely useful to a variety of applications. One can conceive, however, of format classes such a constructive solid geometry views of objects which are manipulated using Skitters and Jacks [Bier 86]. The shared data is the CSG tree and the edit/browse facility manipulates the objects and views of the objects. Another possible format class is B-spline patches. Control points are the shared data. Browsing consists of changing the 3D view while editing consists of moving, creating and deleting control points.

At first blush each of these new format classes seems like an application unto itself. In actuality some applications may consist only of a format class. By factoring out the browse / edit portion as a format class, CSG viewing and editing, for example, can be integrated readily into a variety of CAD/CAM applications which use 3D solids. The present state of the art is

to view and edit the solid geometry in a separate program from where other analysis takes place.

4.1.2. Application / UIMS Interface

An application interacts with the user by constructing the shared data structure and then by attaching *format probes* to various parts of this data structure. A format probe consists of a format and its corresponding format class. A format probe defines a single view of that piece of data and is the connection between the data and the user interface. A format class may create additional format probes of its own. For example the dialog box format class might create new format probes which connect each field data item with each field window displayed on the screen. The function of a format probe is to trap all changes that occur in the data that it is monitoring and notify the UIMS so that the format class can update the relevant portion of the screen. Multiple format probes can be placed on the same piece of data to create multiple views. If one view changes the data the probe structure will make sure that other views are notified and updated.

As the format class attached to a format probe responds to interactive input it may change some of the data that the probe is attached to. In many cases the application will want to know about such changes and will want to respond with additional processing or reject the change. For example if the CSG format class is being used to manipulate images of robot arms the application may want to verify that a particular movement is consistent with the abilities of the robot and reject the movement if it is illegal. The application can place monitor probes on pieces of data which will notify the application in the event of changes. Monitor probes then are the mechanism for application involvement in the user interface.

An application designer then has the following tasks in creating a new user interface:

1. Design the structure of the data to be shared with the interactive user.
2. Select format classes by which the user will manipulate this data and where necessary implement new format classes to be added to the library.
3. Edit formats for the various ways that the data is to be viewed and edited
4. Implement the application code to
   a. construct the shared data
   b. place format probes where the data is to be interactively accessed
   c. place monitor probes where application intervention is required
   d. process activation of monitor probes

4.1.3. Existing Approaches

Many aspects of this architecture are found in the Cousin system [Hayes 84]. In Cousin the interactive environment consists of a set of slots which can be modified by the user and monitored by the application. Invoking a command consists of modifying the contents of the command slot. Although the essential elements of the architecture are present there has been little done in terms of variety of interaction techniques beyond forms filling.

The Higgins system [Hudson 85] also constructs user interfaces as views of data among which dependencies have been defined. The resolution of dependencies and views is performed using incremental attribute evaluation.

4.2. Implications

There are a variety of implications of this model in terms of the kinds of facilities that can be supported by the UIMS. Each of these is uniquely facilitated by the Browse/Edit UIMS model described above.

4.2.1. Universal Cut/Paste

The data-based model provides a natural interpretation of Cut, Copy and Paste for a variety of data. Since every visible object is mapped directly to a particular data object Cut and Copy simply take the data object in its Shared Interactive Data format. The clipboard or scrap heap is simply another branch of the shared interactive data.

4.2.2. UNDO and Patching

Because all changes to shared interactive data are monitored, a real UNDO facility can be automatically provided. The UIMS simply logs all changes to interactive data as they occur. When UNDO is requested the data is restored to its original value. To the application program the restoring of the changes looks no different than if the user had manually reentered the old values. An extension of this facility is patch files. The changes are logged into a file as the interactive session progresses. Such a change file represents the difference between two versions of the shared interactive data. UIMS facilities can then be added which reapply changes to data to recreate the new version from the original or changes can be retracted to recreate the original from the new version. Again such a feature becomes part of the UIMS and the user's interactive environment rather than unique to the particular application.

4.2.3. Graphical Spreadsheets

Another feature that such a UIMS could support is the concept of dependent values. In addition to Cut and Copy the UIMS would support Refer. When one selects a data item from the screen and selects Refer a special reference to the item is constructed. When the reference item is pasted into some other view the values that the item represents are pasted in but the UIMS remembers the existence of the reference. Whenever the original item that was referred to is modified, the paste is reperformed. A reference is analogous to a UNIX pipe which transmits data from one part of the display to another without the active involvement of either the sender or the receiver. Making one displayed value on the screen dependent upon another is the basis for traditional spreadsheets. The UIMS now an active and programmable user interface environment that is more powerful than the basic application components themselves.

4.3. A Uniform Application Model

As was mentioned earlier, a basic assumption of most UIMSs is that there are a variety of functions that an application can and should perform. The UIMS must be structured so that it can adapt to this variety. This is typically done by generating large case statements or tables of function addresses. In the browse / edit model the application functionality is reduced to notifying the application of one of the following.

1. Some data item has been deleted
2. Some data item has been inserted
3. Some data item has changed
4. Some data item has been selected for attention by the user

As long as monitor probes can respond these four messages the application can handle the entire user interface. This greatly simplifies the UIMS/Application interface and fits very naturally into the object-oriented programming paradigm. To the application programmer the user interface problems are now posed in terms of changes to data items rather than in terms of sequences of input events.

5. Summary

An alternative focus for UIMS research has been presented which views user interface software as an information manipulation problem rather than as an input event handling problem. This greatly simplifies the UIMS / application interface. Most importantly it shifts attention of UIMS development towards providing a rich environment in which users manipulate information and away from the the issues of simple event handling.

There are a large number of issues still to be resolved in this model. Most important is the nature of the shared interactive data. There are a variety of candidate models from Lisp-style lists through relational tables. The model chosen will in many ways dictate how other pieces of the model interrelate. The mechanism for placing monitor, format and reference probes into the data without obscuring the application and format's normal "unprobed" view of the data. There are significant efficiency and space problems with a poorly conceived probe strategy. There are a number of possible race conditions as multiple formats based on references are constructed. These must be resolved so that the interactive environment remains robust. Lastly a large number of new format classes must be designed to move user interfaces beyond simple form filling dialog boxes.

This model should, however, provide a fresh approach to UIMS research that has grown somewhat stagnant of late.

6. References

[Anson 82] Anson, E. "The Device Model of Interaction." Computer Graphics 16, 3 (July 1982).

[Apple 87] "HyperCard User's Manual", Apple Computer Inc., (August 1987).

[Bier 86] Beir E. "Skitters and Jacks: Interactive 3D Positioning Tools." Proceedings of 1986 Workshop on 3D Graphics." ACM (Oct 1986).

[Buxton 83] Buxton, W., Lamb, M.R., Sherman,D., Smith, K.C. "Towards a Comprehensive User Interface Management System." Computer Graphics 17, 3 (July 1983).

[Feldman 82] Feldman, H.B. and Rogers, G.T. "Towards the Design and Development of Style-independent Interactive Systems." Human Factors in Computer Systems (March 1982).

[Garret 82] Garret, M.T. and Foley, J. D., "Graphics Programming Using a Database System with Dependency Declarations." ACM Transactions on Graphics 1, 2 (April 1982).

[Green 83] Green, M. "Report on Dialogue Specification Tools." User Interface Management Systems, Ed. Gunther Pfaff, Springer-Verlag (1985).

[Hanau 80] Hanau, P.R. and Lenorovitz, D.R. "Prototyping and Simulation Tools for User/Computer Dialogue." Computer Graphics 14, 3 (July 1980).

[Hayes 84] Hayes, P. "Executable Interface Definitions Using Form-Based Interface Abstractions," In Advances in Computer-Human Interaction, H.R.Hartson, E., Ablex, New Jersey, 1984.

[Hudson 85] Hudson, S.E. and King, R. "Efficient Recovery and Reversal in Graphical User Interfaces Generated by the Higgens System." Proceedings of Graphics Interface '85, (June 1985).

[Jacob 85] Jacob, R.J.K. "A State Transition Diagram Language for Visual Programming." IEEE Computer. 18, 8 (August 1985).

[Kasik 76] Kasik, D.J. "Controlling User Interaction." Computer Graphics 10, 2 (July 1976).

[Kasik 82] Kasik, D.J. "A User Interface Management System." Computer Graphics 16, 3 (July 1982).

[Lieberman 85] Lieberman, H. "There's More to Menu Systems than Meets the Screen." Computer Graphics 19,3 (July 1985).

[Newman 68] Newman, W.M. "A System for Interactive Graphical Programming." SJCC 1968, Thompson Books, (1986).

[Olsen 83] Olsen, D.R. and Dempsey, E.P. "SYNGRAPH: A Graphic User Interface Generator." Computer Graphics 17, 3 (July 1983).

[Olsen 84] Olsen, D.R. "Push-down Automata for User Interface Management." ACM Transactions on Graphics. 3, 4 (July 1984).

[Olsen 86] Olsen, D.R. "MIKE:The Menu Interaction Kontrol Environment." ACM Transactions on Graphics 5,4 (Oct 1986).

[Rosenthal 83] Rosenthal, D.S.H. "Managing Graphical Resources." Computer Graphics 17, 1 (January 1983).

[Sutherland 65] Sutherland, I.E. "SKETCHPAD: A Man-Machine Graphical Communication System." MIT Lincoln Lab. Tec. Rep. 296, May 1965.

[Thomas 83] Thomas, J.J. and Hamlin, G. "Graphical Input Interaction Techniques: Workshop Summary." Computer Graphics 17, 1 (January 1983).