From:
Dan R. Olsen
User Interface Management Systems: Models & Algorithms
Morgan Kaufman Publishers
1992

1. Introduction

The history of computing has been one of harnessing the power of a machine to meet the needs of man. This process is seen most powerfully in the development of graphical human-computer interfaces. A major impediment to this harnessing process, however, has been the cost of developing software that tames the technology for human use and interacts in human rather than in machine terms.

In order to build quality user interfaces for a variety of graphical applications the notion of a User Interface Management System (UIMS) has been developed. This field of research has not reached maturity and only a handful of UIMSs have become commercial products. Most UIMSs have not left the laboratories where they were born. There is, however, a widening interest in applying this technology.

This book presents a wide range of research that has been done on UIMSs. The focus of the book will be on UIMS architectures and implementations. UIMSs are primarily aimed at graphical user interfaces and there will be little or no discussion of traditional textual user interfaces.

Computer Graphics and the User Interface

Historically, user interface issues have developed in tandem with the growth of computer graphics. With the advent of Sketchpad[1] the potential for computer graphics to provide an interactive medium was recognized. The user interface to Sketchpad, a bank of toggle switches, seems rather primitive today. Even in those early days researchers were building tools to aid in developing user interfaces. The earliest system which might be called a UIMS was Newman's Reaction Handler[2] which used the basic state machine as its model.

For many years the handling of graphical input was virtually ignored,

as researchers struggled with questions of geometry, optics, lighting, and modeling of virtual worlds. Simultaneously, computer graphics became steadily cheaper and more widely available. Early storage-tube display technology (which the majority of users had available to them) was not amenable to highly interactive programs. Textual interfaces with some informational pictures were the rule. The vector refresh technology was highly interactive but very expensive and thus not widely available. The vector refresh applications were all carefully handcrafted to take advantage of the special graphics hardware and display devices of each machine.

The major discussions concerning input during this period could be summarized by the differentiation between sampled and event devices, and the notion that inputs should not be directly linked to their output echoes. The separation of input and output allowed a variety of echoing and interactive techniques to be programmed.[3] These two concepts carried user input research in the graphics community up to the point of the CORE[4] and GKS[5] graphics standards.

With the advent of microprocessors and semiconductor memory the nature of computer graphics changed entirely. Raster displays came to dominate the marketplace; almost every personal computer has some graphics capability. With the processor intimately linked with the graphics display in low cost workstations the notion of a quality user interface became economically feasible and, soon, very important. It was during this period that UIMS research began to flourish. The computer graphics community was undergoing a radical transformation but the basic tenets of separation of input and output would remain, and influence UIMS development for many years to come.

The Nature and Evolution of UIMSs

UIMS research focused around three workshops sponsored by the ACM (Association for Computing Machinery), EUROGRAPHICS, and IFIPS. These workshops brought together a wide range of researchers, set much of the conceptual terminology, and defined the problems and challenges facing UIMS researchers.

Seattle

The first workshop was held in Seattle in 1982.[6] At this point only a handful of UIMSs had been built or published and the term User Interface Management System had not come into wide usage. The major conclusions of this workshop were:

1. The user interface implementation should be separated from the application code and be implemented using specialized programming tools.
2. Interactive applications should have external rather than internal control. The user interface and its supporting software would control the flow of the application (external control) rather than the application code itself (internal control).
3. Tools should be developed to assist user interface developers who are not necessarily programmers.
4. User interfaces should be specified using a separate dialog description tailored specifically for user interface design rather than programmed in a general purpose language.

These points represented a radical departure from normal development of graphical user interfaces, where interactive input was handled by the application calling subroutines from a library.

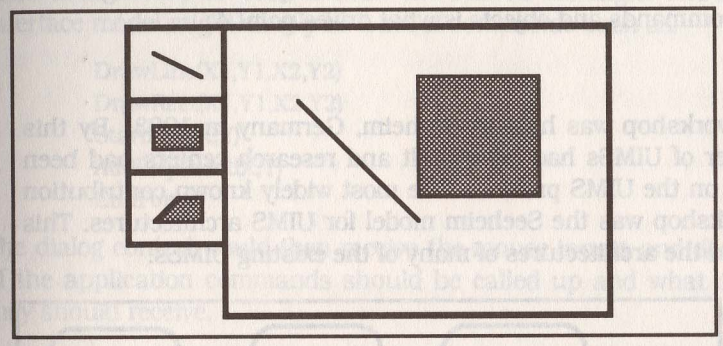

Fig. 1:1
Sample Drawing Application

To illustrate these points take the sample drawing application shown above. Traditional user interface techniques would consist of the following kinds of activities.

- Write code to draw each of the icons on the screen
- Write code to draw each of the objects in the draw window
- Write dialog handling code which:

1. gets a mouse input and decides which icon was selected
2. based on the icon selected, gets inputs for the object to be drawn
3. draws the object and adds it to the list of objects maintained by the application.

There are several problems with this strategy. The first is that icons, and other pictorial information, must be specified in code rather than by drawing. This is painful to do, even for programmers, and eliminates the

possibility of a graphics artist participating in the icon design. An additional problem is that the dialog is too rigid. If, in the middle of drawing a rectangle, the user wants to draw a polygon instead, the rectangle must be completed and deleted, and only then can the polygon be drawn. Allowing a more flexible dialog requires a lot of special code from the programmer.

If the icon images are not embedded in the code, but are placed in a separate dialog specification, then tools can be written that allow either programmers or graphic artists to draw rather than code the appearance of the icons. This is the essence of points 1 and 3 from the Seattle workshop. If the dialog specification is also pulled out of the code, more powerful control models can be created that allow users to freely exit dialog fragments and start something new. This is the essence of point 2, about external and internal control. The dialog required for this draw program is rather simple and does not require a great deal of understanding. The specification of more complicated dialogs with hundreds of commands and objects is what drives point 4.

Seeheim

The second workshop was held in Seeheim, Germany in 1983.[7] By this time a number of UIMSs had been built and research centers had been working hard on the UIMS problem. The most widely known contribution from this workshop was the Seeheim model for UIMS architectures. This model reflected the architectures of many of the existing UIMSs.

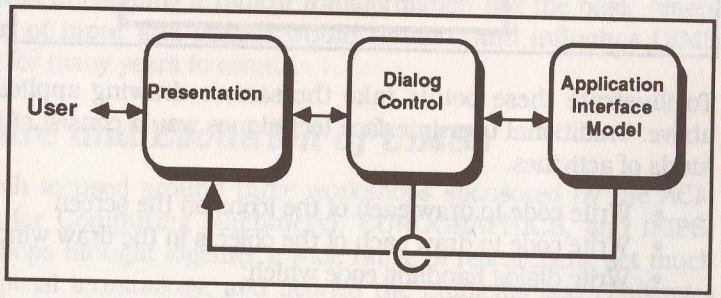

Fig. 1:2
Seeheim Model for UIMS Architecture

The Seeheim architecture calls for three UIMS components. The primary component is the *dialog control* which receives inputs, determines what should be done about them, and requests services from the application. The application code is accessed via some *application interface model*, which is also called the *semantic interface*. The *presentation* consisted of all those issues that control the visual appearance and physical device selection of the actual interface.

This model called for graphical output of application data to be generated by the application under the control of the dialog manager, with the presentation controlling the external appearance. The notion of the dialog control having an influence over the presentation of application data is signified in the diagram by the small circle around the path from the application interface to the presentation. This data display facet of the model has not been fully realized in any UIMS.

In our example drawing application, the images of the icons would be part of the presentation. The drawing of the lines, rectangles, and polygons in the draw area would be done by the application, through the presentation. The presentation, for example, would know where the draw area is actually located relative to the icons. The dialog control might influence this drawing by making sure that the currently selected object is highlighted. In this application it is the dialog control's responsibility to know that after selecting the rectangle icon, any mouse activity in the draw area should be used to draw the desired rectangle. The application interface model might simply be a set of commands such as:

```
DrawLine(X1,Y1,X2,Y2)
DrawRect(X1,Y1,X2,Y2)
StartPoly(X,Y)
AddPolyPoint(X,Y)
EndPoly()
```

The dialog control would then receive the mouse inputs and decide which of the application commands should be called up and what arguments they should receive.

Seattle Revisited

The third workshop was again held in Seattle, in 1986.[8] The primary results of this meeting were:

- The UIMS tools and interface descriptions should support the process of user interface design rather than simply its implementation. This includes all of the issues from software engineering, which have rather unique problems when applied to user interface development.
- Earlier UIMS architectures, including the Seeheim model, suffered serious problems in facing the demands of direct manipulation interfaces. Most of these problems were the result of the separation of input from output, which had been inherited from earlier graphics research and had permeated many UIMS developments.

These workshops served only as focal points for a range of research that began well before 1982 and which is still emerging. It is important to note that as research has progressed, the emphasis has expanded beyond the narrow questions of quick and easy implementation of the interface, to encompass the entire user interface design process.

The Process

In order to design a good UIMS one must first consider the nature of the development process that an interactive application goes through. In many ways this is similar to the standard approaches from software engineering, but there are several unique characteristics of graphical user interfaces that need consideration; particularly in light of the design and implementation tools that should be incorporated into the UIMS.

The 1986 Seattle workshop identified the following relevant software engineering phases:

- requirements
- specification and design
- implementation
- testing
- maintenance

The requirements phase is pretty much standard, although some human factors practitioners have advocated the inclusion of usability criteria in the requirements[9] which specify measurable productivity, error rate, and learnability goals for the end product. The role of the UIMS in meeting usability requirements will be discussed in the chapter on interface evaluation *(Chapter 11)*.

The specification and design phase actually has several parts which can be broken down along the lines of the Seeheim model. The first and most critical part is the design of the application's functionality. Functionality is characterized by specifying the application/UIMS interface. There are a number of forms that the application interface and its specification might take, some of which directly support functional design of the interface and other forms which simply support the implementation. The variety of semantic interface models will be discussed in Chapter 2.

Once the functional design of the interface is complete, proceeding sequentially through presentation design and dialog design and then on to implementation and testing is not appropriate for user interface design. At present the only known method for creating a quality user interface design

is an iterative process of creating a design, implementing it, testing it with users, and modifying it. In addition, the specification of presentation and dialog are heavily intertwined, since the dialog design is controlled by the presentation of the application data and it in turn controls the kinds of icons, dialog boxes, and other presentational items that are required. Compare, for example, the Macintosh Finder with the UNIX command line. This change in presentation of the information radically affects the nature of the dialog. The key issue for the design of user interfaces is flexibility in the tools. The tools must support rapid modification and testing of a design to facilitate convergence to an acceptable user interface.

The iterative nature of user interface design poses several challenges to UIMS developers. The first of these is to create models for dialog and presentation descriptions. Such descriptions should be easily modified and quickly turned into working implementations that can be tested. This includes tools both for editing the dialog specification and for drawing, or otherwise designing, the external presentation of the interface. The challenge of designing elegant dialog and presentation models has been the driving force in most UIMS development.

A second challenge is to be able to evaluate a design. The most attractive approaches are predictive evaluation tools which analyze the dialog specification to identify design weaknesses before implementation. This approach is particularly attractive due to the fact that most UIMSs are characterized by a machine-readable dialog specification which could, potentially, be analyzed.

A third challenge is to provide support tools to evaluate the performance of actual working user interfaces and to indicate where problems exist and where changes should be made. A particularly ambitious goal is not only to identify where problems in the interface design or implementation exist but also to suggest specific remedies or otherwise directly aid in selecting remedies for problems that have been detected.

Few UIMSs have addressed the maintenance and functional testing problem. Many software development environments provide tools for managing test sets and evaluating output. These tools allow regression tests to be created, maintained, and semiautomatically applied to new versions of the software. In an interactive graphics application, test set maintenance is much more difficult. There is no source file that can be applied as the test input, and the output is primarily defined in terms of what appears on the screen. There is a potential for UIMS-based tools to provide assistance here, because of their control of all user inputs. As yet, however, no UIMSs have attacked this problem.