RC 14674 (#65767) 6/5/89
Computer Science 18 pages

Research Report

User Interface Management Systems: Survey and Assessment

Peter Y. F. Wu

IBM Research Division
T.J. Watson Research Center
Yorktown Heights, N.Y. 10598

Researh Report

User Interface Management Systems: Survey and Assessment

Peter Y.F. Wu
IBM T.J. Watson Research Center
P.O. Box 704, Yorktown Heights, NY 10598
Vnet: PWU at YKTVMH

May 22, 1989

User Interface Management Systems: Survey and Assessment

Peter Y.F. Wu
IBM T.J. Watson Research Center

ABSTRACT

The term User Interface Management System (UIMS), coined in 1982, referred to a software strategy for dialogue control in computer-user interaction. The strategy attempted to separate specification of dialogue sequences from application development. The concept spawned much further work, intended to generalize the strategy to other applications. Many UIMSs were since reported in the literature, and meaning of the term UIMS was also changing, from a strategy for application development to an integrated system of tools for user interface (UI) design. The paper gives a historical account. The intended separation of dialogue control sequencing from application development is hardly successful in any general sense. The functional distinctions between dialogue control and application program are still very much unclear. Yet UIMS work has made an impact in our views toward application development. We recognized the role of UI design, for which most programmers were not trained. Since the requirements for UI are often inherently fuzzy, the only reliable test for UI quality is in testing by end-users. There is evidently the need for application development to be user oriented: to build UI software around the user, and to test early UI prototypes by end-users to support iterative re-design. Much of UIMS work therefore concentrated in the UI design tools, and the integration of UI design into application development. The paper adopts a classification of UIMSs by the tools for UI design, and presents a survey.

INTRODUCTION

The advent of interactive graphics [Sutherland 63] has brought new possibilities to computer-user interface (or simply user interface, or UI). The potential for higher quality and more usable UI is obvious, but the problem of constructing UI software also becomes greater at the same time. The cost for software production is often unexpectedly high, and statistics reported that UI software takes a substantial share of that heavy cost [Sutton 78]. At least in part due to its graphical and asynchronous real-time nature, software for interactive graphical UI is complicated. On the other hand, application programmers often are not good UI designers. Software engineering techniques such as requirements analysis do not seem to apply very well to UI software design and development. We often need the feedback from end-users testing with prototypes. Quick prototyping and flexibility in iterative design and development are much more effective. The need for a novel approach and new techniques for UI software is evident.

User interface management system, (or UIMS), was a term first used in 1982, as the title of a paper in the SIGGRAPH conference [Kasik 82]. UIMS referred to a software strategy for dialogue control in UI. The strategy separated specification of dialogue sequences from application development. In 1983, Eurographic Seminars held a workshop on UIMS in Seeheim, West Germany [Pfaff 85], and enthusiastic discussions ensued. Envisioned to be the mediating system between application and end-user, UIMS intended to separate UI functions from the application program. Much of the work attempted to sketch out practicable UIMS architectures. In the years following, many UIMSs were reported. But along with the development, the meaning of the term UIMS was also changing. UIMS now commonly refers to a collection of software tools integrated into a system for UI design and development, and the operation of UI software. In recognizing the role of UI design, UIMS work has resulted in changes in the environment to facilitate new approaches to application software development.

This paper presents a historical account of the development of UIMS work. As a commentary, the paper discusses the implications of the UIMS approach to application software development. It uses a software layer model to characterize the run-time application environment, and sorts some terminologies involved. Based on how the UI designer's tools and the environment for UI design, the paper presents a classification of UIMSs and comments on the range of functionalities supported in the end-user's interface. The classification reported here is built upon previous work in [Green 86] and [Myers 89]. In conclusion, the paper also suggests that the future for application development will necessarily be the integration of different tools and environments.

A HISTORICAL ACCOUNT

Although high quality UI is very much conceivable, and in many cases realized, the general question of how to build high quality UI does not have an easy answer. Around the beginning of 1980's, when graphics standards are being developed, the question concerning human-computer interaction and how to handle it was catching attention. The term UIMS, coined in 1982, referred to a software strategy to separate dialogue specification from application development. Later on many attempted to generalize the strategy; UIMS then was conceived to be a system mediating between the user and the application program. Much work tried to draft out an architectural model for the UIMS. Since 1985, hosts of different UIMSs were reported in the literature. We will trace out the brief history of development of UIMSs here: Table 1 presents in chronological order the events and the UIMSs disseminated in the literature, with a brief remark on each one. This however is hardly an exhaustive list; we have selected those which we deem representative. The following will comment on the development and then discuss on the implication of the UIMS approach to application software development.

An Early Bird in 1968

Early in the 60's after the introduction of interactive computer graphics by the Sketchpad project [Sutherland 63], Newman foresaw the need of tools for UI software development. He modelled computer-user interaction by a finite state automaton and implemented a system which he called the Reaction Handler [Newman 68]. The system provides a graphical editor for the UI designer to construct a state transition diagram. The run-time system interprets the information from the state transition diagram to control the interactive operation of the application program. The Reaction Handler was probably the first UIMS, and interestingly, was dated long before the inception of the term.

The Few UIMSs in 1982-83

The term UIMS was coined in 1982. In the paper entitled "A User Interface Management System," Kasik described a framework for developing engineering applications with highly interactive graphics called TIGER [Kasik 82]. The UIMS refers to the software system which handles user interaction. The system comprises two major components. First, a special purpose programming language allows detailed specification of dialogue sequences. A pre-processor compiles the dialogue sequences into a formatted menu file for use at run-time. Second, at run-time system, an interpreter generates systems of menus (according to the formatted menu file) to provide user control to invoke appropriate application procedures. Kasik explained

that the strategy was similar to that of a data base management system (DBMS), in which the functions of a schema as to data would be similar to that of a dialogue sequence specification as to user interaction. The approach separated the specification of dialogue sequences from the application program.

Developed at the University of Toronto, Menulay and MakeMenu were both tools of an integrated system for UI design and development. The strategy is quite different from that of the TIGER UIMS. The system is for UI designers who are not programmers. Menulay provides an interactive graphical environment for the designer to make up networks of menus and keep the information in UI specification files. MakeMenu can then take the UI specification and generate the UI software in C code, to be compiled and linked with the application program. At run-time, a table-driven support system controls the user interaction [Buxton 83].

The Syngraph UIMS was developed as part of the Automated Human Interfaces project at Arizona State University. Syngraph adopted yet another approach. The model for user interaction is decomposed into semantic, syntactic, and lexical levels, much like a formal language. Hence the UI software is a a parser for the language of computer-user interaction. The UI designer prepares the UI description in an extended version of Backus-Naur form (BNF), and Syngraph generates the UI software in Pascal code from the UI description. An interaction techniques library supports the run-time system. At the lexical level of the grammar specification, terminals refer to the primitive interaction techniques in the library [Olsen 83].

1983 Seeheim Workshop on UIMS

November 1983, the Eurographic Seminars held a workshop on UIMS in Seeheim, West Germany [Pfaff 85]. Much enthusiastic effort attempted to sketch out an architectural model for the UIMS, which was conceived to be a system mediating between the user and the application program [Strubbe 85]. Figure 1 depicts the conceived UIMS model [Green 85a]. Such a model is based on a concept of dialogue independence, that the software managing dialogue with the user would be loosely coupled with the application program. However, the functional distinctions between dialogue and application were unclear. Indeed, some examined the premises in the concept of dialogue independence and stated that "the UIMS cannot be as isolated from the application semantics as had originally been presumed [Tanner 85]." Despite the controversy, the following years witnessed the development of many UIMSs.

The Hosts of UIMSs in 1985-87

There was a time of silent working on UIMSs the year after the 1983 Seeheim workshop. But in the following years, many UIMSs were reported. In 1985, most of the

systems reported were language-based, ranging from declarative languages (Cousin [Hayes 85], Domain/Dialog [Schulert 85]) for UI specification to detailed event-based programming (Squeak [Cardelli 85]). Grins [Olsen 85] is grammar based, but comes with a graphical editor for UI specification. Perhaps more remarkable was the work of U of Alberta UIMS [Green 85b]; using three different lanugage models for UI specification, he showed that the three models, state transition diagrams, context free grammars, and event-based languages were progressively more flexible in expressive power but more difficult to use [Green 86].

In 1986, many systems began to break out the modes of the predecessors. Jacob extended the easy of use of state transition diagrams [Jacob 86]. Olsen in the Mike UIMS attempted easier generation of UI from an incomplete declarative language specification using proper defaults [Olsen 86]. With GWUIMS, Sibert began to build an object-oriented framework for UI programming, intended to test different UI models [Sibert 86]. Hill and Myers both at the University of Toronto worked on UIMSs to produce direct manipulation style of UI. Hill built an elaborate event-based programming system Sassafras [Hill 86]. The system was effective for use but hard to learn. Myers took an entirely different approach; the Peridot system attempted to specify direct manipulation UI by direct manipulation, so that it can be a UI design tool for non-programmers. In other word, the designer demonstrates the desired sequence of interaction and Peridot generalizes from the action taken to produce the software to control the dialogue [Myers 86].

Panther [Helfman 87] and Control Panel Interface [Fisher 87] in 1987 both were concerned about generating graphical UI modelled by the control panel. When reduced to the problem of screen space allocation, both were effective UI programming tools.

Some Higher Level UIMSs

Most of these system mentioned above dealt with the UI software problem at a relatively lower level of abstraction. Most provided support for UI software at the binding and sequencing levels. The years following saw more UIMSs intended to provide support at the functional and conceptual levels. Most notable was the Knowledge-Based UIMS [Foley 88b] built on a frame-based expert system which generates UI software from the specification of application functions. Garnet [Myers 89] which is under construction is intended to be a more comprehensive system, in the sense that it provides supporting tools for different roles to contribute to UI design and its incorporation with application functions. Returning to the UIMS problem of separation of dialogue control from the application, Garnet began to treat the UI design problem in the larger context of application development.

The rest of the paper goes on to a commentary, discussing the implications of the UIMS work in application development, and a classification of UIMSs.

IMPLICATIONS OF THE UIMS APPROACH

Since 1982, UIMS work has been going on, and the meaning of UIMS also has been changing. In the 1983 Seeheim workshop on UIMS [Pfaff 85], the UIMS was conceived to be a mediating system between the user and the application program, and much work attempted to draft models of the UIMS architecture. Many UIMSs have been reported since then, and they come in different forms and varieties. The intended separation of dialogue control from the application program did not lead to a general solution. Despite the difficulties, the UIMS work has made an impact on our views toward application development. In this section, we will first sort out the roles and the terminologies to discuss the issues involved with a model for UIMS architecture. Further, we will present a classification of UIMSs, based on the UI design tools provided by the systems.

The Roles and The Terminologies

The UIMS approach to application development recognizes several different roles involved. The UIMS approach recognizes these four basic roles: the application programmer, the UI designer, the UIMS architect, and the end-user. These are considered different roles since they require different skills. Traditionally, the application programmer is responsible for all these roles. By identifying these different roles, the need for different tools for different roles becomes evident. Tools for the UI designer become the major thrust of the UIMS work. The term UIMS now commonly refers to the collection of UI design tools in an integrated system for their use. For this reason, some researchers used the terms UIDS (UI development system) or UIDE (UI development environment) instead [Hill 86, Myers 89]. We suggest clearing up these terms here. UIMS in its restricted meaning should refer to the run-time application environment operating the UI; in the general sense, UIMS may refer loosely to the entire integrated system incorporating all the tools for different roles. On the other hand, the term UIDS (or UIDE) should refer to the sub-system in the general UIMS, specifically the collection of UI design tools and the components supporting their operation. In the following, we will discuss the run-time UIMS environment and the software components. Then we will discuss the UIDS (that is, the UI designer's sub-system) and the tools for UI design.

Models for UIMS Run-Time Architecture

Conceived to be the mediating system between the user and the application program, the UIMSs were portrayed in different models of run-time organization of

software components. The issues involved were communication and control flow among these components. To discuss these issues, we will first introduce a model of the software layers in the application's run-time environment. The model should help us clarify the level of abstraction in user interaction the UIMS is intended to function.

Software Layers

Figure 2 depicts the different software layers in the run-time environment of an interactive application. We may consider everything beneath the application program layer UI software. There is in fact no clear delineation such that beyond which level, it is or is not UI software. When the application program interacts with the user, information travels through all the layers of UI software, each handling the information at its own level of abstraction. Foley [1988a] suggested use of the terms binding, sequencing, functional, and conceptual to describe the different levels of abstraction (instead of the earlier lexical, syntactic, and semantic since they convey unappropriately the idea of formal languages). The I/O device drivers are often parts of the operating system to control privileged access to the devices. The graphics library, or the imaging model of a window system, provide primitive I/O action units, hence at the binding level. The toolkits or widgets provide packaged interaction techniques; they are at the sequencing level. These components support the application environment as collections of procedures called by the software from a high level of abstraction. The difficulty with UIMS is with the dialogue control layer which should operate in the functional level, closely relating to the application semantics. Thus, the software for dialogue control is traditionally embedded in the application program, often referred to as the UI portion of the program. The UIMS approach attempts to separate the dialogue control functionalities from the application program. Although this separation is hardly successful is any general sense, the UIMS approach introduced a new style of control flow in the run-time environment. The following section will discuss further.

Control Flow Styles

Traditionally, the application programmer treats the UI software as packages of interaction techniques, and the application program calls upon these packaged procedures when so desired. The user follows the lead of the application program to interact with it. The style is thus called internal control, or application control; figure 3 depicts the idea. External control refers to a style in which the user takes control in the interaction. The UI software is organized around the user and calls upon application procedures in response to the user's requests, as illustrated in figure 4. To mediate between the user and the application program, most UIMSs adopt the external model in order to control sequencing and scheduling of tasks

initiated by user interaction. Hence the style of external control is also called UIMS control. Thus in the UIMS approach, the roles between the dialogue control layer and the application program can be reversed. It is also possible to have a mixed control style. While neither the UI software nor the application program takes overall control, each may be in control for some of the time. The two may be coroutines, or two communicating processes.

UI DESIGN IN APPLICATION DEVELOPMENT

Besides a new control flow style in user-centered run-time environment, the UIMS work has also focused on the essential role of UI design in application development. While recognizing the different roles involved, the UIMS approach identified the need of tools for the UI designer. Much of the work was in investigating new design tools and systems integrating their use. The next section will discuss the UI tools provided in different UIMSs. In our terminology, this refers to the UIDS or UIDE of the general UIMS.

The Designer's Tools and A Taxonomy

Green [1986] presented a classification of three different models for dialogue specification. Myers [1989] introduced a classification of UIMSs by the different UIDSs they offer. The UIDS refers to the set of tools the UI designer may use to specify the desired UI for an application. Table x shows the classification of the UIMSs alluded to in our historical account. This is largely based on Myers' work of classification by the UI design tools which their repesctive UIDSs offer. There are three broad categories: language-based specification, direct graphical specification, and automatic UI generation. We describe each category below:

A. LANGUAGE-BASED SPECIFICATION

Most of the earlier UIDSs were language-based. The UI designer uses a special purpose language for UI design specification. Such a special purpose language can take many different forms, too:

(1) Forms

Panther [Helfman 87] allocates screen space to different interaction functions according to tabulated parametered specified in a form filled by the UI designer; the screen then models a control panel for user interaction.

(2) Menu networks

TIGER [Kasik 82] supports a hierarchy of menus, which is specified in a structured language with sophisticated features such as aborting or skipping levels.

(3) State transition diagrams

Computer-user interaction often involves changing of one mode to another in response to certain input action. The state transition diagram in then an effective means to model the interactive behavior. Jacob [1985] argued for its effective use in visual programming. The Reaction Handler [Newman 68], probably the first UIMS, also used the state transition diagram to specify interactive behavior of applications. We consider this a language-based specification tool since it is still essentially the state table which characterizes the operation, but systems come with graphical editors for UI designers to construct and edit the state transition diagrams.

(4) Context free grammars

These systems treat interactive dialogues as formal languages. The UI designer specifies the legal interactions in an exteneded Backus Naur form (BNF), and a parser generation system generates the UI software which handles the dialogue. These systems work very well with command dialogue model of user interaction; they do separate dialogue sequencing from the application program, but it cannot produce semantic feedback. Syngraph [Olsen 83] is a grammar-based system.

(5) Event languages

These are essentially programming languages for event handlers. Event languages are explicitly designed for concurrent dialogues. However, these systems require UI designers to be good programmers. These tools work at a low level of abstraction and the larger systems are quite difficult to debug. Squeak [Cardelli 85] and Sassafras [Hill 86] are both event language systems. The Univ. of Alberta UIMS [Green 85] is also event language based, but it comes with tools to translate state transition diagram and BNF grammar specification into event language specification. EDGE [Kleyn 88] formulates event handling using an AND-OR graph, and comes with a graphical tool to construct and edit the graph. It becomes a nice aid to event language programming.

(6) Object-oriented languages

These systems provide an object-oriented framework for UI programming. The framework comes with object having encapsulated interactive behavior which the programmer can use by default, or modify thru inheritance. GWUIMS [Sibert 86] is a system built on Franz Lisp with Flavors. Like event languages, the UI designer has to be a programmer.

(7) Declarative languages

A declarative language is different from a procedural language in that it specifies what should happen instead of how it should happen. Hence, the UI designer does not have to worry about sequences of events in the interaction, but can be

concerned more about the semantic information related to the application. Both Cousin [Hayes 85] and Domain/Dialog [Schulert 85] use some form or slot based communication between the application program and the UIMS.

B. DIRECT GRAPHICAL SPECIFICATION

These systems are intended to achieve a WYSIWYG (What you see is what you get) style of UI design. Menulay [Buxton 83] and Grins [Olsen 85] both allow the UI designer to place display objects including packaged interaction techniques directly on the presentation layout. Grins also allows specification of constraints on these objects to handle semantic feedback. Peridot [Myers 86] furthermore allows the designer to create new interaction techniques by demonstration. These UIDSs are certainly much easier to use, but they are quite difficult to build. With these tools, UI designers do not have to be programmers at all.

C. AUTOMATIC UI GENERATION

To alleuviate the difficulties in the detailed specification of UI design this class of UIDSs generates the UI automatically from the semantic information about the application. Control Panel Interface [Fisher 87] uses the type information of I/O parameters to create graphical interfaces: buttons for Boolean variables, dials for integers, etc. The UI designer can then modify and reorganize these gadgets onto a control panel. Mike [Olsen 86] generates an interface from declarative procedure headings in a profile file. By proper defaults, a complete specification is not necessary. The interface generated is menu-oriented, but the UI designer can modify it on an interactive graphical editor. Mike can even change certain commands to operate on direct manipulation. Mickey UIMS [Olsen 89] generates an interface from declarative type-extensions to Pascal, mapping onto it a predefined UI style. In this aspect, Mickey is similar to Mike, but Mickey has included a wider range of descriptive facilities supported and still maintained the ease of use. Foley [1988b] presented a Knowledge-Based UIMS which generates the UI from a language description call IDL (Interface Definition Language [Gibbs 86]). But since IDL describes the functions supported by the application, we can consider this automatic UI generation from application semantics. The UIDS therefore functions at a fairly high level of abstraction. Furthermore, the UIDS is a knowledge-based system. While maintaining the generated UI consistent with the IDL description, it can also transform a UI design to other alternatives for subsequent selection.

A UIMS SURVEY

Table 2 presents a survey of the UIMSs listed in table 1. The column under UI

Design Tool describes the classification based on the UIMS taxonomy already introduced. The other columns include comments on other aspects related to the UIDSs. We will discuss these below.

Level of abstraction in the UI design tools

Another issue concerning UI design tools is the level of abstraction the tools work with. In other words, this relates to the basic units of meaning the UI designer will have to deal with. Refer to figure 2 for the model of software layers in the run-time application environment. The UIMS approach is intended to provide support at the functional level, to separate dialogue control from the application program. But when we generalize the approach to view the UI design tools, these tools only serve in fairly limited ways. The UIDSs still very often operate at the binding or sequencing levels of abstraction. Of the systems we studied in the survey, one UIDS which provides support at a significantly higher level of abstraction is the Knowledge-Based UIDS [Foley 88b].

Internal UI description and representation

This issue refers to how the UI design information is kept by the UIDS. Quite often this information is kept in a format for run-time access. Pictures and icons as well as other presentation information seem to be best kept for efficient run-time access. Information related to interaction is more often expressed in programming languages, general purpose or specially designed. Hence the UI description then may be used in compile-time to be incorporated with application programs (in the form of header files or packaged procedures), or interpreted at run-time.

Interaction model of the UI functions supported

UI design is quite often biased toward a certain user interaction model which the designer has pre-supposed. Listed in this aspect is our interpretation of the "model of interaction" by examining the UI functions supported by the UIMS. The state machine model seems one easy to prepared for. Although to some extent, it is limited, it can still support a fairly broad range of applications. Another common observed model is the command dialogue model of interaction. More involved interaction models and UI functions tend to be very much application specific, and thus become difficult to be supported by general UIDS tools.

In view of the different roles identified for the process of application development, as well as the differences in the levels of abstraction in the supporting tools, we suggest the future of application development will be a complex environment. The necessity to integrate different tools for different roles is eminent. In the past five to six years, there has been a rich development of various UI design tools. The next step in the coming five or six years shall see the development toward compre-

hensive application development environments, integrating different tools so that programers, designers, as well as end-users can all contribute to the desired operations of application software.

CONCLUSION

We presented a historical account on UIMS work: from the inception of the term to the evolution of its meanings, with remarks on many representative systems reported in the literature. As a commentary, we discussed the issues involved in the UIMS architectural model, and the implications of UIMS approach in application development. The UIMS approach brought a new style of flow control to the run-time application environment, namely, the external control model, so that software is built around the user instead of forcing the user to following prescribed interaction patterns designed by the application programmer. In recognizing the role of UI design in application development, UIMS work has resulted in the burgeoning of new tools for UI design. To clear up some terminologies, we use the term UIMS to refer either strictly to the external control run-time environment, or loosely to the integrated system for the design, development, and operation of UI software. The UIDS (or UIDE) refers to the collection of UI design tools in the UIMS. Based on a taxonomy of UIDSs, we classified a number of UIMSs surveyed in the report, and discussed other issues characterizing the UI design tools of the UIMSs.

REFERENCES

Buxton, W., M.R. Lamb, D. Sherman, K.C. Smith. 1983. "Towards A Comprehensive User Interface Management System," *ACM Computer Graphics*, Vol.17, No.3, (SIGGRAPH'83), pp.35-42.

Cardelli, Luca and Rob Pike. 1985. "Squeak: a Language for Communicating with Mice," *ACM Computer Graphics*, Vol.19, No.3, (SIGGRAPH'85), pp.199-204.

Fisher, G.I. and K.I. Joy. 1987. "A Control Panel Interface for Graphics and Image Processing Applications," Proceedings SIGCHI+GI'87: *Human Factors in Computing Systems*, Toronto, Ontario, Canada, April 1987, pp.285-290.

Foley, James D. 1988. "Models and Tools for the Designers of User-Computer Interfaces," *Report* GWU-IIST-87-03, Dept of EE and CS, George Washington University, March 1987. (*Theoretical Foundations of Computer Graphics and CAD*, R.A. Earnshaw, ed., published 1988, Springer-Verlay, pp.1121-1151.)

Foley, J., C. Gibbs, W.C. Kim and S. Kovacevic. 1988. "A Knowledge-Based User Interface Management System," Proceedings, CHI'88: *Human Factors In Computer Systems*, Washington, DC, May 1988, pp.67-72.

Gibbs, C., W.C. Kim and J. Foley. 1986. "Case Studies in the Use of IDL: Interface Definition Language," *Report* GWU-IIST-86-30, Dept of EE and CS, George Washington University, Washington, DC 20052.

Green, Mark. 1985. "Report on Dialogue Specification Tools," Proceedings of Eurographic Seminars workshop on *User Interface Management Systems*, Seeheim, West Germany, November 1983, p.9-20, published 1985 by Springer-Verlay, Pfaff, ed., 224 pages.

Green, Mark. 1985. "The University of Alberta User Interface Management System," *ACM Computer Graphics*, Vol.19, No.3, (SIGGRAPH'85), pp.205-213.

Mark Green. 1986. "A Survey of Three Dialogue Models," *ACM Transactions on Graphics*, Vol.5, No.3, July 1986, pp.244-275.

Hayes, Philip J., Pedro A. Szekely and Richard A. Lerner. 1985. "Design Alternatives for User Interface Management Systems Based on Experience with COUSIN," Proceedings CHI'85: *Human Factors in Computing Systems*, San Francisco, Calif., April 1985, pp.169-175.

Helfman, J.I. 1987. "Panther: A Specification System for Graphical Controls," Proceedings SIGCHI+GI'87: *Human Factors in Computing Systems*, Toronto, Ontario, Canada, April 1987, pp.279-284.

Hill, Ralph D. 1986. "Supporting Concurrency, Communication, and Synchronization in Human-Computer Interaction - The Sassafras UIMS," *ACM Transactions on Graphics*, Vol.5, No.3, July 1986, pp.179-210.

Jacob, Robert J.K. 1985. "A State Transition Diagram Language for Visual Programming," *IEEE Computer* (Aug'85) Vol.18, No.8, pp.51-59.

Kasik, David J. 1982. "A User Interface Management System," *ACM Computer Graphics*, Vol.16, No.3, July 1982, pp.99-106.

Kleyn, M.F. and I. Chakravarty. 1988. "EDGE - A Graph Based Tool for Specifying Interaction," Proceedings of ACM *Symposium on User Interface Software*, Banff, Alberta, Canada, October 1988, pp.1-14.

Myers, Brad A., and William Buxton. 1986. "Creating Highly-Interactive and Graphical User Interfaces," *ACM Computer Graphics*, Vol.20, No.4, (SIGGRAPH'86), pp.249-258.

Myers, Brad A. 1989. "Encapsulating Interactive Behaviors," Proceedings of CHI'89 Conference, Austin, Texas, May 1989, pp.319-324.

Newman, William M. 1968. "A System for Interactive Graphical Programming," Proceedings of the AFIPS Spring Joint Computer Conference, pp.47-54.

Olsen, D.R. Jr. and E.P. Dempsey. 1983. "SYNGRAPH: A Graphical User Interface Generator," *ACM Computer Graphics*, Vol.17, No.3, (SIGGRAPH'83), pp.43-50.

Olsen, D.R. Jr. and E.P. Dempsey. 1985. "Input/Output Linkage in a User Interface Management System," *ACM Computer Graphics*, Vol.19, No.3, (SIG-

GRAPH'85), pp.191-197.

Olsen, D.R. Jr. 1986. "MIKE: The Menu Interaction Kontrol Environment," *ACM Transactins on Graphics*, Vol.5, No.4, Oct'86, pp.318-344.

Olsen, D.R. Jr. 1989. "A Programming Language Basis for User Interface Management," CHI'89 Conference Proceedings, Austin, Texas, May 1989, pp.171-176.

Pfaff, Gunther E., ed. 1985. *User Interface Management Systems*, Proceedings of Eurographic Seminars workshop, Seeheim, West Germany, November 1983, Springer-Verlag, 224 pages.

Rhyne, Jim, et al. 1987. "Tools and Methodology for User Interface Development," Report from 1986 ACM SIGGRAPH Workshop on Software Tools for User Interface Management, *ACM Computer Graphics*, Vol.21, No.2, pp.78-87, (April 87).

Schulert, Andrew J., George T. Rogers and James A. Hamilton. 1985. "ADM - A Dialog Manager," Proceedings CHI'85: *Human Factors in Computing Systems*, San Francisco, CA, April 1985, pp.177-183.

Sibert, J.L., W.D. Hurley and T.W. Bleser. 1986. "An Object-Oriented User Interface Management System," *ACM Computer Graphics*, Vol.20, No.4, (SIGGRAPH'86), pp.259-268

Strubbe, H.J. 1985. "Report on Role, Model, Structure and Construction of a UIMS," Proceedings of workshop on *User Interface Management Systems*, Seeheim, Germany, Nov'83, pp.3-8, Springer-Verlag, 1985.

Sutherland, Ivan E. 1963. "SketchPad: A Man-Machine Graphical Communication System," Proceeding, 1963 Spring Joint Computer Conference, Baltimore, Maryland, published by Spartan Books.

Sutton, J.A. and R.H. Sprague Jr. 1978. "A Study of Display Generation and Management in Interactive Business Applications," IBM Research Report RJ2392, Nov'78, 20 pages.

Tanner, P.P. and W.A.S. Buxton. 1985. "Some Issues in Future User Interface Management System Development," Proceedings of workshop on *User Interface Management Systems*, Nov'83, Seeheim, Germany, pp.67-79, Springer-Verlag, 1985.

Table 1. The UIMS development chronology	
1968	Newman. **The Reaction Handler** *first* UIMS, based on state transition diagrams.
1982	Kasik. **TIGER** language-based for menu networks.
	Jun'82: Seattle, Washington. ACM workshop on Graphical Input Interactin Techniques - concepts of UIMS articulated.
1983	Buxton. **Menulay** and **MakeMenu** direct graphical specification for menu networks.
	Olsen. **Syngraph** grammar based UIMS: dialogue specification in BNF.
	Nov'83: Seeheim, West Germany. Eurographic Seminar workshop on User Interface Management Systems - pursuing UIMS architectural model.
1985	Hayes. **Cousin** declarative lang for UI spec, UIMS/App communicate via *slots*.
	Schulert. **ADM, Domain/Dialog** declarative lang for UI spec, programming framework.
	Cardelli. **Squeak** language for mice programming, event-based.
	Olsen. **Grins** constraint-based I/O linkage, with graphics editor for direct spec.
	Green. **U of Alberta UIMS** event-based: incorporated with STD, CF Grammar, and EB lang.
1986	Jacob. **STD for visual programming** graphical/textual, augmented State Transition Diagrams.
	Hill. **Sassafras** event-based with debug/interpret aids for direct manipulation UI.
	Myers. **Peridot** using direct manipulation to create direct manipulation UI.
	Olsen. **Mike** use defaults for automatic generation and tools for modification.
	Sibert. **GWUIMS** lang based: OO programming for application development.
	Nov'86: Seattle, Washington. ACM Workshop on Software Tools for User Interface Management - confusing terminologies.
1987	Helfman. **Panther** tabular setup for screen space function allocation.
	Fisher. **Control Panel Interface** automatic generation from application semantics.
1988	Foley. **Knowledge Based UIMS** textual IDL, automatic generation and selection.
	Kleyn. **EDGE** event-based, with graphical editor using AND-OR graphs.
	Oct'88: Banff, Alberta. ACM Symposium on User Interface Software. - many UI software tools.
1989	Olsen. **Mickey UIMS** declarative type-extensions to Pascal language.
	Myers. **Garnet** comprehensive UIMS, with "interactors."

FIGURE 1. MODEL OF UIMS ARCHITECTURE

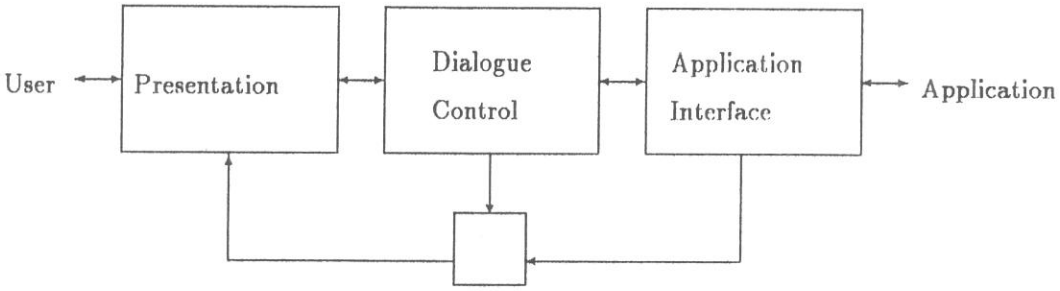

FIGURE 2. SOFTWARE LAYERS IN THE APPLICATION RUN-TIME ENVIRONMENT

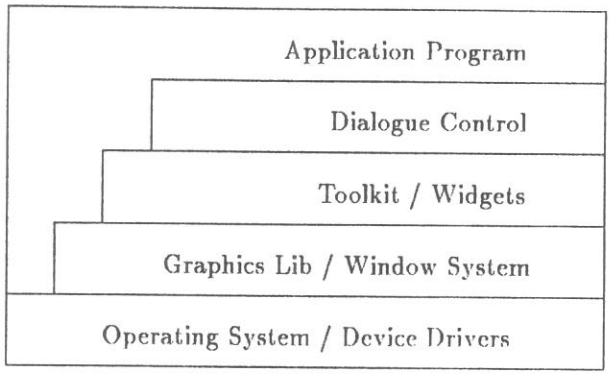

FIGURE 3. INTERNAL (APPLICATION) CONTROL

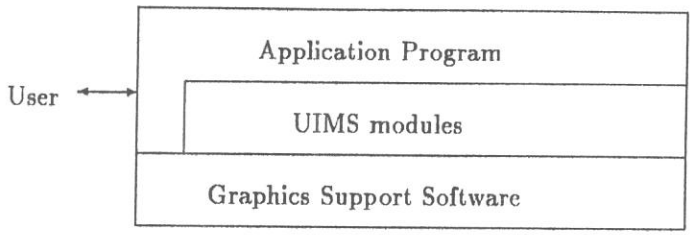

FIGURE 4. EXTERNAL (UIMS) CONTROL

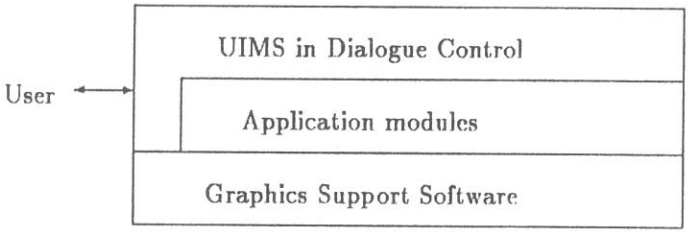

TABLE 2. THE UIMS SURVEY

UIMS	UI Design Tool	Internal Description	Level of Abstraction	Model of Interaction
Reaction Handler	STDs	state tables	func: interaction states	state machine model
TIGER	lang for menu networks	formatted menu files	seq: menu networks	menu networks
Menulay MakeMenu	direct graphical spec	graphical layouts and control tables	seq: menu networks	menu networks, and graphical layout
Syngraph	BNF: grammar spec	dialogue parser	seq: dialogue parsing	state machine model
Cousin	decl lang: slot-based values	decl lang: slot-based values	func: slot-based values	state machine model
Domain Dialogue	decl lang: states/tasks grouping	decl lang: insert files, and dialogue descript'n	seq/func: grouping techniques into tasks	state machine model
Squeak	event lang: mouse programming	proc lang: C	bind: event handling	event-based, async interaction
GRINS	direct graphical spec	dialogue parser	seq: dialogue parsing	state machine model, and graphical layout
U Alberta UIMS	event lang, with BNF and STDs	control tables, and C procedures	bind: event handling, with BNF and STD	event-based, async interaction
Visual lang spec	Augmented STDs	state tables	func: interaction states	state machine model
Sassafras	event language	proc lang: InterLisp-D	bind: event handling	concurrent dialogues, direct manipulation
Peridot	direct graphical spec	proc lang: InterLisp-D	seq: demonstrating interaction techniques	direct manipulation
MIKE	decl lang: profile files	UI profiles, Pascal procedures	func: default interface from cmd semantics	cmd dialogue model
GWUIMS	object-oriented lang	object-oriented formulation	bind/seq: object-oriented programming	direct manipulation, and others
Panther	table entries	C include files for table values	func: screen space allocation	control panel model
Control Panel Int	auto: type info in application semantics	parm types, and control mechanisms	func: screen space allocation	control panel model
K-based UIMS	auto: from IDL (Interface Def Lang)	schemata in ART, knowledge frames	func: frame-based spec of UI functions	cmd dialogues, direct manipulation
EDGE	event-based: AND-OR graph editor	proc lang: InterLisp-D	bind: event handling	event-based concurrent dialogues
Mickey UIMS	declarative type-ext'ns to Pascal	proc lang: Pascal	func/seq: type ext'ns in Pascal	cmd dialogue model
Garnet	comprehensive: many tools integrated	layouts, objects, and constraints	seq: objects, interactors, and contraints	direct mainpulation