User Interface Software Tools

by

James D. Foley

GIT-GVU-91-29
November 1991

Graphics, Visualization & Usability Center

Georgia Institute of Technology
Atlanta GA 30332-0280

USER INTERFACE SOFTWARE TOOLS

James D. Foley
Graphics, Visualization and Usability Center
College of Computing, Georgia Institute of Technology
Atlanta, GA 30332-0280

Developing high-quality user interfaces is becoming the critical step in bringing many different computer applications to end users. Ease of learning and speed of use typically must be combined in an attractively-designed interface which appeals to application-oriented (not computer-oriented) end users. This is a complex undertaking, requiring skills of computer scientists, application specialists, graphic designers, human factors experts, and psychologists.

User interface software is the foundation upon which the interface is built. The quality of the building blocks provided by the software establishes the framework within which an interface designer works. The tools should allow the designer to quickly experiment with different design approaches, and should be accessible to the non-programmer designer.

In this paper we discuss important directions in software tools for building user interfaces:

- Unified representation serving multiple purposes;
- Integration with software engineering tools:
- Interactive programming and by-example creation of interfaces and interface components.

Most of our focus is on the first two areas.

1. Background on User Interface Software Tools

Figure 1 shows the various levels of user-interface software, and suggests the roles for each. The application program has access to all software levels; programmers can exploit the services provided by each level, albeit with care, because calls made to one level may affect the behavior of another level. In this paper we discuss just the interaction technique toolkit and user interface management system layers. See [FOLE90] for discussions of the window manager and graphics layers.

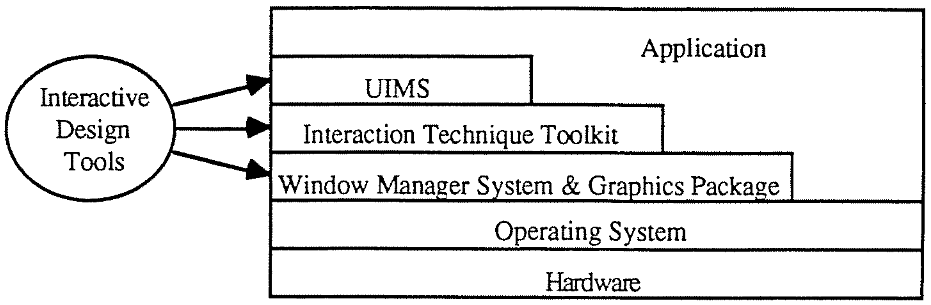

Interactive Design Tools
Application
UIMS
Interaction Technique Toolkit
Window Manager System & Graphics Package
Operating System
Hardware

Fig. 1 User interface software. The application program has access to the operating system, window manager system and graphics package, toolkit, and UIMS. The interactive design tools allow non-programmers to design windows, menus, dialogue boxes, and dialogue sequences.

1.1. INTERACTION–TECHNIQUE TOOLKITS

Interaction techniques are the means by which users interactively input information to a computer system. A typical set of interaction techniques includes a dialogue box, file–selection box, alert box, help box, list box, message box, radio–button bank, radio button, choice–button bank, choice button, toggle–button bank, toggle button, fixed menu, pop-up menu, text input, and scroll bar. Interaction–technique toolkits are subroutine libraries of interaction techniques which are made available for use by application programmers. Widely used toolkits include the Andrew window–management system's toolkit [PALA88], the Macintosh toolkit [APPL85], OSF/Motif [OPEN89] and InterViews [LINT89] for use with X Windows [SCHE86], several toolkits that implement OPEN LOOK [SUN89] on both X Windows and NeWS, and Presentation Manager [MICR89]. In the X Window system, interaction techniques are called *widgets*, and we will often use this term.

The look and feel of a user–computer interface is determined largely by the collection of interaction techniques provided for it. Designing and implementing a good set of interaction techniques is time consuming, requiring experimentation with users to ensure ease of learning and speed of use. This toolkit approach, which helps to ensure a consistent look and feel among application programs, is clearly a sound and well–accepted software engineering practice. Considerable programmer productivity is gained by using an existing toolkit. The interaction technique toolkit is used in the user interface of both applications programs and the window manager itself.

Notice that the previous list of widgets includes both high- and low-level items, some of which are composites of others. For example, a dialogue box might contain several radio–button banks, toggle–button banks, and text input areas. Hence toolkits include a means of composing widgets together, typically via subroutine calls.

Creating composites by programming is tedious. Interactive editors allow composites to be created and modified quickly, facilitating easy changes to user interface details based on user feedback to the design team. Commercially available dialogue box editors include Developer's Guide from Sun Microsystems [SUN90] and UIMX from Visual Edge [VISU90].

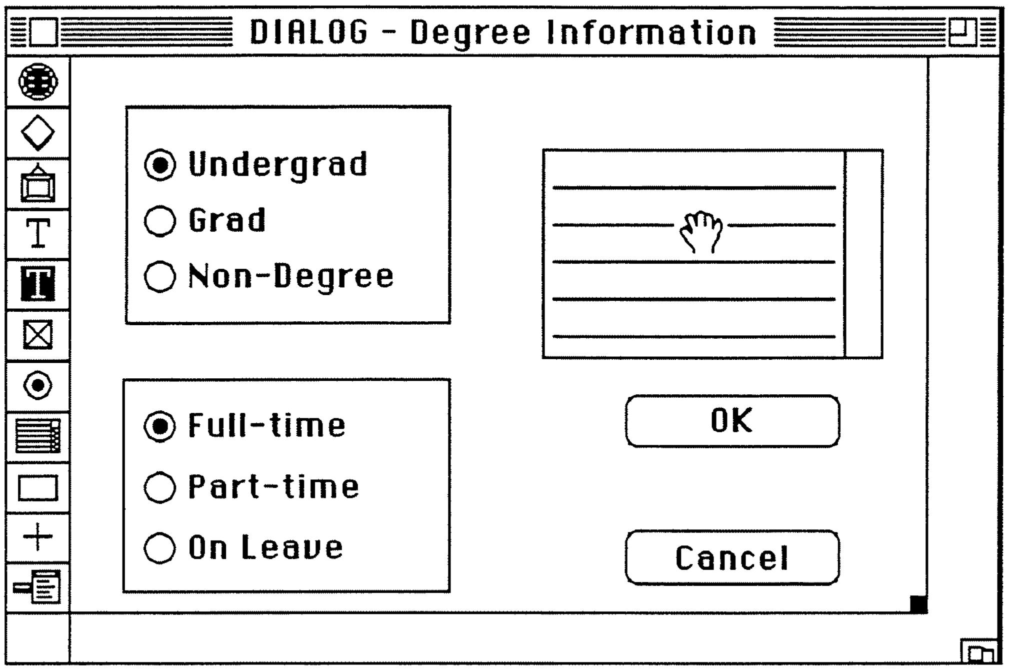
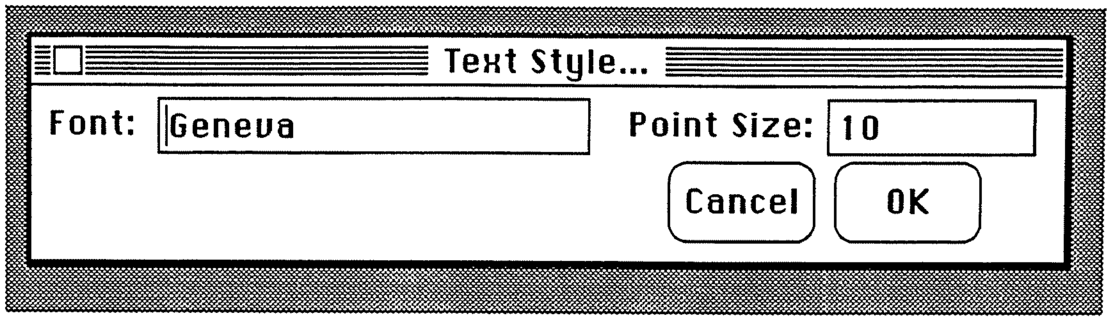


Fig. 2 The Now Software (formerly SmethersBarnes) Prototyper dialogue-box editor for the Macintosh. A scrolling-list box is being dragged into position. The menu to the left shows the widgets that can be created; from top to bottom, they are buttons, icons, pictures, static text, text input, check boxes, radio buttons, scrolling lists, rectangles (for visual grouping, as with the radio-button banks), lines (for visual separation), pop-up menus, and scroll bars. (Courtesy Now Software.).

The output of these editors is a representation of the composite, either as data structures that can be translated into code, as code, or as compiled code. In any case, mechanisms are provided for linking the composite into the application program. Programming skills are not needed to use the editors, so the editors are available to user–interface designers and even to sophisticated end users. These editors are typical of the interactive design tools. Figure 2 shows an example of such an editor.

Another approach to creating menus and dialogue boxes is to use a higher-level programming–language description. In MICKY [OLSE89], an extended Pascal for the Macintosh, a dialogue box is defined by a record declaration. The data type of each record item is used to determine the type of widget used in the dialogue box: enumerated types become radio–button banks, character strings become text inputs, Booleans become check boxes, and so on. Figure 3 shows a dialogue box and the code that creates it. An interactive dialogue–box editor can be used to change the placement of widgets.



```
type
  Str40     = string[40]
  textStyle = record
                font : Str40;
                points (*Name = 'Point Size'): integer
              end
```

Fig. 3 A dialogue box created automatically by MICKY from the extended Pascal record declaration. (Courtesy Dan Olsen, Jr., Brigham Young University.)

Peridot [MYER86; MYER88] takes a radically different approach to toolkits. The interface designer creates widgets and composite widgets interactively, by example. Rather than starting with a base set of widgets, the designer works with an interactive editor to create a certain look and feel. Examples of the desired widgets are drawn, and Peridot infers relationships that allow instances of the widget to adapt to a specific situation. For instance, a menu widget infers that its size is to be proportional to the number of items in the menu choice set. To specify the behavior of a widget, such as the type of feedback to be given in response to a user action on a menu item, the designer selects the type of feedback from a Peridot menu, and Peridot generalizes the example to all menu items.

1.2. User Interface Management Systems

A user–interface management system (UIMS) is built on top of an interaction technique toolkit, and provides additional functionality in implementing a user interface. All UIMSs provide some means of defining admissible sequences of user actions and may in addition support overall screen design, help and error messages, macro definition, undo, and user profiles. Some recent UIMSs also manage the data associated with the application.

UIMSs, like toolkits, can increase programmer productivity, speed up the development process, and facilitate iterative refinement of a user interface as experience is gained in its use. The more powerful the UIMS, the less the need for the application program to interact directly with the operating system, window system, and interaction-technique toolkit.

In some UIMSs, user–interface elements are specified in a programming language that has specialized operators and data types. In others, the specification is done via interactive graphical editors, thus making the UIMS accessible to non-programmer interface designers. The former approach tends to be more powerful; the latter, more accessible.

Applications developed on top of a UIMS are typically written as a set of subroutines or, in contemporary object-oriented environments, as *methods*. The UIMS is responsible for calling appropriate methods in response to user inputs. In turn, the methods influence the dialogue—for instance, by modifying what the user can do next on the basis of the outcome of a computation. Thus, the UIMS and the application share control of the dialogue—this is called the *shared-control* model. The key concept here is that of a user–interface *state* and associated user actions that can be performed from that state. The state can be affected by user actions and by methods.

If a context-sensitive user interface is to be created, the system responses to user actions must depend on the current state of the interface. System responses to user actions can include invocation of one or several methods, state changes, and enabling, disabling, or modifying interaction techniques or menu items in preparation for the next user action. Help can also be made dependent on the current state.

Sequencing and state dependencies can be specified using a variety of linguistic approaches, such as state diagrams, augmented transition networks, and event languages. Green [GREE87] surveys event languages and all the other sequence-specification methods we have mentioned, and shows that general event languages are more powerful than are transition networks, recursive transition networks, and grammars. He also provides algorithms for converting these forms into an event language. ATNs that have general computations associated with their arcs are also equivalent to event languages.

A quite different way to define sequencing is by example. Here, the user interface designer places the UIMS into a "learning" mode, and then steps through all acceptable sequences of actions (a tedious process in complex applications, unless the UIMS can infer general rules from the examples). The designer might start with a main menu, select an item from it, and then go through a directory to locate the submenu, dialogue box, or application-specific object to be presented to the user in response to the main menu selection. The object appears on the screen, and the designer can indicate the position, size, or other attributes that the object should have when the application is actually executed. The designer goes on to perform some operation on the displayed object and again shows what object should appear next, or how the displayed object is to respond to the operation; the designer repeats this process until all actions on all objects have been defined. This technique works for sequencing through items that have already been defined by the interface designer, but is not sufficiently general to handle arbitrary application functionality. User-interface software tools with some degree of by-example sequencing specification include Menulay [BUXT83], TAE Plus [MILL88] and the SmethersBarnes Prototyper [COSS89]. Peridot, mentioned earlier, builds interaction techniques (i.e. hardware bindings) by example. Another way to define user interfaces consisting of interconnected processing modules is with data-flow diagrams. For instance, the NeXT Interface Builder [NEXT90] allows objects to be interconnected so that output messages from one object are input to another object. Type checking is used to ensure that only compatible messages are sent and received.

Further background on UIMSs can be found in [HART89; MYER89; OLSE87].

2. The User Interface Management System – Data Base Management System Analogy

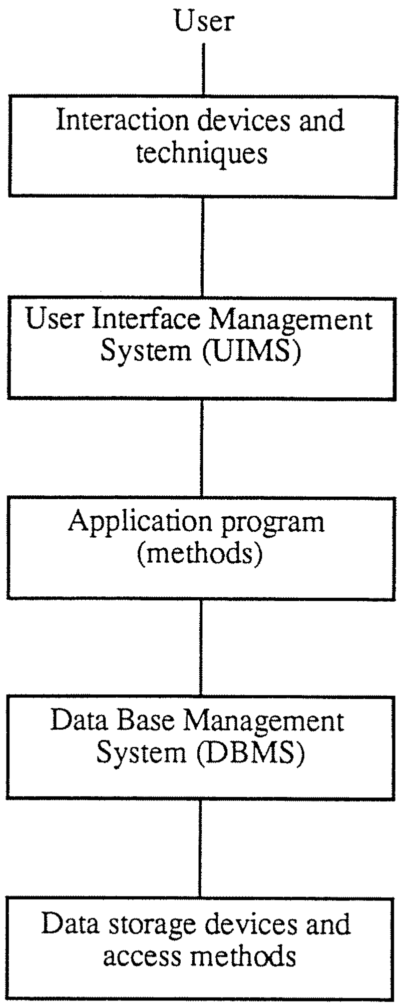

User
Interaction devices and techniques
User Interface Management System (UIMS)
Application program (methods)
Data Base Management System (DBMS)
Data storage devices and access methods

Figure 4. Symmetry between User Interface Management Systems (UIMS) and Data Base Management Systems (DBMS).

There is a useful analogy and symmetry between UIMS concepts and those found in data base management systems (DBMS) as suggested in Figure 4. Just as a DBMS manages data, a UIMS manages the interface. A DBMS hides the user from details of storage organizations, just as a UIMS hides the programmer from details of interaction devices and techniques. A DBMS has one or more specialized languages with which the programmer defines views, report formats, and queries. So too a UIMS may have specialized languages for specifying sequencing, state dependencies and changes, screen organizations, and menu designs.

Driving this analogy further, a key to the success of DBMSs is the use of a unifying representation. In the relational world, the conceptual data base design serves this role. This representation is used at design time and at run time in a variety of important ways, which we list here without elaboration.

Design time uses:

- Form creation for data input using the form-fill-in dialogue style (Form Builder)
- Report template generation for printing/displaying reports (Report Formatter)
- Definition of multiple views of the data
- Generation of diagrams of the conceptual data base organization

Run time uses:

- Helping user by displaying prompts and error messages as the user specifies a query or an update, and in forms-oriented approaches such as Query-by-Example (QBE)
- Integrity checking
- Alerting/triggering
- Processing queries and updates
- Mapping from views into actual relations
- Query optimization -- deciding how most efficiently to access information
- Generating reports

3. Unified Representation for User Interfaces

In UIMS research, the values of a unified representation are only now coming to be recognized. Some of the types of information found in such a representation are typified by UIDE, the User Interface Design Environment [FOLE88, FOLE89, FOLE91A]. The representation provides:

- The class hierarchy of objects which exist in the application
- Properties of the objects
- Actions which can be performed on the objects
- Units of information (parameters) required by the actions
- Pre- and postconditions for the actions.

UIDE is implemented in an expert system shell, using seven types of schemata: object schema, action schema, parameter schema, pre-condition schema, post-condition schema, attribute schema, and attribute type schema. (In the following description, slots, or information contained with a schema, are given in italics).

Instances of the Object Schema represent the objects defined as part of the user interface design. Within this schema, a *Description* slot is a textual description of the object (created by the user interface designer) which can be provided to the user at run time in response to a help request. *Actions on object* is a relation linking an object schema instance to the action schema instances for those actions which can be applied to the object class. The object schemata makes the UIMS aware of the data model assumed by the application.

There is an instance of the Action Schema for each action defined in the user interface. Actions can affect an attribute, an object, or an object class action. *Description* is available for run-time help. The *Action routine name* provides the link to the run-time procedure which actually carries out the action. (In object-oriented programming, this would be the method name.) The *Actions mutually-exclusive with* slot refers to all actions which cannot be available at the same time as this action. This slot can be used to organize menus: mutually exclusive commands, such as "turn x on" and "turn x off", can be assigned the same menu slot. *Inverse action* is the name of the action which is the inverse of this action, if such an action exists. This assists in implementing an undo command. *Parameters*, *Pre-conditions*, and *Post-conditions*, refer to schema instances which further describe the action. For each action there is a Parameter Schema instance for each parameter, or unit of information, required by the action.

Pre-conditions are predicates which must be true in order for an action to be enabled and thus available to the user for selection. Arbitrary predicates can be used, along with pre-defined predicates having to do with the number of instances of different types of objects. Post-conditions, implemented as add-lists and drop-lists, change the values of predicates. Pre- and post-conditions specify just enough of the application's semantics to encode the state of the interface, to allow context-sensitivity in presenting menus, to give context-sensitive help, and for the very limited understanding of the semantics which is necessary for transformations of interface styles.

A Pre-condition Schema is instantiated for each action. A *Description* explains what the pre-condition means. The actual *Expression* of the pre-condition is represented in a form which can be conveniently evaluated at run-time, to determine whether the pre-condition is true or false. A Post-condition Schema contains the *Add-list* and *Drop-list* of predicates.

The types of information encoded by predicates and used in the pre- and post-conditions include:

- The number of objects of a particular object class which are in existence. A creation command increments this number as a post-condition; a deletion command decrements the number.
- The number of objects of different object classes which have been selected as belonging to the Currently Selected Set.
- The existence of a currently selected command.
- The existence of a value for a parameter.
- Any predicates established by the interface designer as being important to defining the context of the interface.

Each attribute of an object is described by an instance of the Attribute Schema, which includes an optional *Default value* for the attribute. The Attribute Type Schema records the attribute *Data type* as being integer, real, enumerated, etc., and gives other information which is specific to the data type.

This unified representation is evolving with time; in its present form, it is not sufficiently robust to serve all of the purposes we have in mind, and it continues to evolve [FOLE91b]. However, it is already quite useful. The following list indicates things that we would like to be able to do with the representation; items indicated with an * have already been developed:

Design time uses:

* Test properties of the interface, such as functional completeness, consistency and reachability [FOLE89, BRAU90]

* Automatically organize menus and dialogue boxes [KIM90, DEBA91]

* Support application of correctness-preserving transformations to the interface specification, allowing the designer to quickly explore a space of design alternatives [FOLE87]
* Automatically create an interface to the application, using menus, dialogue boxes, and direct manipulation [FOLE91a]
* Evaluate the interface design with respect to speed of use, using a key-stroke model type of analysis [CARD80, SENA89]
• Optimize the interface design for speed of use
• Predict learning times and error rates using cognitive models [KIER85] integrated with the representation
* Generate a parser which accepts natural-language commands to the application [KOVA90]

Run time uses:

* Explain why a command is disabled [FOLE89]
* Explain (partially) what a command does [FOLE89]
• Explain what information is needed before a command can be performed
* Provide procedural help, via animation, taking into account the current application context [SUKA88, SUKA90]. Also show, via animation, what sequence of commands must be performed before a currently-disabled command can be performed.
• Generate a guided tour of the application for new users
• Adapt to individual users based on their demonstrated knowledge of and usage of the application
• Provide an undo capability
* Control actual execution of the application, including enabling and disabling of menus items, as well as display of menus, dialogue boxes, and windows [FOLE91a, GIES91]

Because the unified representation includes aspects of data base schemata as well as those of traditional user interface representations, the situation of Figure 5 is more evocative of the state of affairs which becomes possible with this approach.

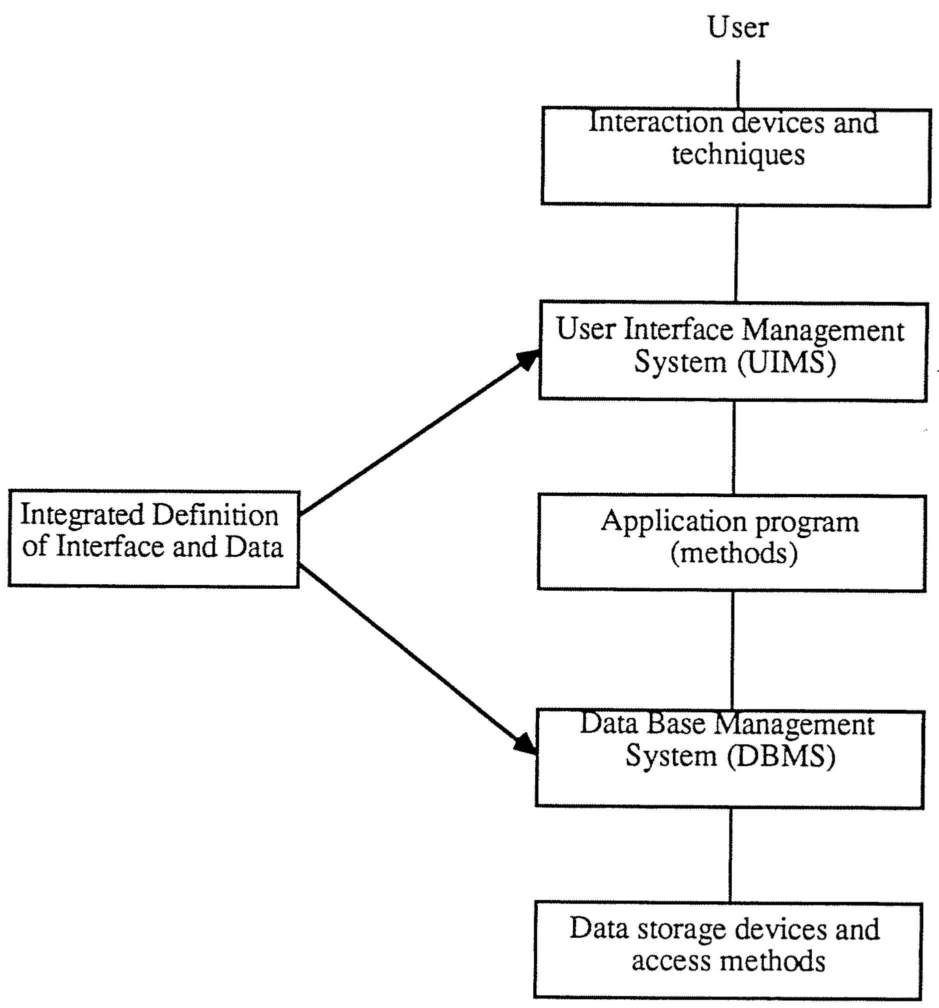

User
Interaction devices and techniques
User Interface Management System (UIMS)
Integrated Definition of Interface and Data
Application program (methods)
Data Base Management System (DBMS)
Data storage devices and access methods

Figure 5. Unified definition driving/controlling UIMS and DBMS.

Because the unified representation includes aspects of data base schemata as well as those of traditional user interface representations, the situation of Figure 5 is more evocative of the state of affairs which becomes possible with this approach.

4. Integration with Software Engineering / Data Engineering Tools

The software and data engineering community has already developed a variety of tools for describing information about applications in general, without any real focus on the user interface. In the meantime, the user interface software community has been developing tools, as discussed above, which include some of the very same information, plus more specialized user interface information. This means that some information has to be specified twice, once by the software engineer and once by the user interface designer. These tools should clearly be merged to speed up the design process, avoid duplication of efforts, and avoid potential inconsistencies between the dual specifications.

As a start in this direction, we have coupled the data modeler component, called D2M2 Edit of the Delft Direct Manipulation Manager (D2M2) [BEEK90] with Sun's DevGuide [SUN90]. Once an object-oriented data model has been created using D2M2's interactive design tool, the user interface designer selects attributes or actions (i.e. methods) of one or more objects, and drags them into DevGuide. We have augmented DevGuide with a rule base developed from the Open Look Style Guide [SUN89] utilizing application semantic information in the specifications, so that a control panel, complete with menus and possibly other widgets, is automatically created [DEBA91]. The designer can then use DevGuide's interactive design capabilities to fine-tune the design, changing either the specific selection of controls or the positioning of controls in the window.

5. Conclusions

We have presented the concept of a unified user interface representation which can be used for a variety of run time and design time purposes, and have drawn analogies to data base management systems. A specific unified model has been overviewed, and current and projected uses have been described. Much work is needed to realize the full potential of this concept; in the meantime, some of the promises of the approach have already been realized.

Acknowledgements: The Graphics and User Interface Research Group at George Washington University and later the Graphics, Visualization, and Usability Center at Georgia Tech provided the intellectual environment supporting this work. Direct contributors to UIDE include D. DeBaar, M. Frank, C. Gibbs, D. Gieskens, M. Gray, W. Kim, S. Kovacevic, L. Moran, K. Murray, H. Senay, J. Sibert and P. Sukaviriya. Financial support has been provided by the National Science Foundation, Sun Microsystems, Inference Corporation, Software Productivity Consortium, Siemens Corporation, the GWU Dept. of EE & CS Industrial Liaison Program, and Georgia Institute of Technology. Thanks to Martin Frank, Mark Gray, Srdjan Kovacevic, Joan Morton, and Piyawadee Sukaviriya for technical assistance in preparing this paper.

REFERENCES

APPL85 Apple Computer, *Inside Macintosh*, Addison-Wesley, Reading, MA, 1985.

BEEK90 Beekman, W., *D2m2edit,* Master's Thesis, Delft University of Technology, July 1990.

BRAU90 Braudes, R., *A Framework for Conceptual Consistency Verification,* DSc dissertation, Dept. of Electrical Engineering and Computer Science, The George Washington University, 1990.

BUXT83 Buxton, W. et al., Towards a Comprehensive User Interface Management System, Proceedings 1982 SIGGRAPH Conference, published as *Computer Graphics*, 17(3), July 1982, pp. 35–42.

CARD80 Card, S., T. Moran, and A. Newell, The Keystroke-Level Model for User Performance Time with Interactive Systems, *Communications of the ACM*, 23(7), July 1980, pp. 398–410.

COSS89 Cossey, G., *Prototyper*, Now Software, Portland, Oregon, 1989.

DEBA91 DeBaar, D. and J. Foley, Coupling Application Design and User Interface Design, *Report GIT-GVU-91-10,* Graphics, Visualization and Usability Center, Georgia Institute of Technology, Atlanta GA 30332-0280, 1991.

FOLE87 Foley, J., W. Kim, and C. Gibbs, Algorithms to Transform the Formal Specification of a User-Computer Interface, *Proceedings INTERACT '87, 2nd IFIP Conference on Human-Computer Interaction*, Elsivier Science Publishers, Amsterdam, pp. 1001–1006.

FOLE88 Foley, J., C. Gibbs, W. Kim, and S. Kovacevic, A Knowledge Base for a User Interface Management System, *Proceedings CHI '88 - 1988 SIGCHI Computer-Human Interaction Conference*, ACM, New York, 1988, pp. 67–72.

FOLE89 Foley, J., W. Kim, S. Kovacevic, and K. Murray, Designing Interfaces at a High Level of Abstraction, *IEEE Software*, 6(1), January 1989, pp. 25–32.

FOLE90 Foley, J., A. van Dam, S. Feiner, and J. Hughes, *Computer Graphics – Principles and Practice*, Addison-Wesley, Reading, MA, 1990.

FOLE91a Foley, J., W. Kim, S. Kovacevic and K. Murray, UIDE - An Intelligent User Interface Design Environment, in J. Sullivan and S. Tyler (eds.) *Intelligent User Interfaces,* Addison-Wesley, Reading MA, 1991.

FOLE91b Foley, J., D. Gieskens, W. Kim, S. Kovacevic, L. Moran and P. Sukaviriya, A Second-Generation Knowledge Base for the User Interface Design Environment, *GWI-IIST-91-13,* Dept. of Electrical Engineering and Computer Science, The George Washington University, 1991.

GIES91 Gieskens, D. and J. Foley, Controlling Interface Objects Through Pre- and Postconditions, *GIT-GVU-91-09,* Graphics, Visualization and Usability Center, Georgia Institute of Technology, Atlanta GA 30332-0280, 1991.

GREE87 Green, M. A Survey of Three Dialog Models, *ACM Transactions on Graphics*, 5(3), July 1987, pp. 244–275.

HART89 Hartson, R. and D. Hix, Human–Computer Interface Development: Concepts and Software, *ACM Computing Surveys*, 21(1), March 1989, pp. 5–92.

KIER85 Kieras, D. and P. Polson, An Approach to the Formal Analysis of User Complexity, *International Journal of Man-Machine Studies*, 22, 1985, pp. 365–394.

KIM90 Kim, W. and J. Foley, DON: User Interface Presentation Design Assistant, *Proceedings of the SIGGRAPH Symposium on User Interface Software and Technology,* Snowbird, Utah, October 1990.

KOVA90 Kovacevic, S., A Compositional Model of Human-Computer Dialogues, *GWU-IIST-90-36,* Department of Electrical Engineering and Computer Science, The George Washington University, Washington DC 20052, 1990.

LINT89 Linton, M., J. Vlissides, and P. Calder, Composing User Interfaces with InterViews, *IEEE Computer*, 22(2), February 1989, pp. 8–22.

MICR89 Microsoft Corporation, *Presentation Manager*, Microsoft Corporation, Bellevue, WA, 1989.

MILL88 Miller, P., and M. Szczur, Transportable Application Environment (TAE) Plus Experiences in 'Object'ively Modernizing a User Interface Environment, *Proceedings OOPSLA'88*, pp. 58–70.

MYER86 Myers, B., Creating Highly-Interactive and Graphical User Interfaces by Demonstration, Proceedings 1986 SIGGRAPH Conference, published as *Computer Graphics*, 20(4), August 1986, pp. 249–257.

MYER88 Myers, B., *Creating User Interfaces by Demonstration*, Academic Press, New York, 1988.

MYER89 Myers, B., User-Interface Tools: Introduction and Survey, *IEEE Software*, 6(1), January 1989, pp. 15–23.

NEXT90 NeXT, *Interface Builder*, NeXT Inc, Sunnyvale, CA 1990.

OLSE87 Olsen, D., ed., ACM SIGGRAPH Workshop on Software Tools for User Interface Management, *Computer Graphics*, 21(2), April 1987, pp. 71–147.

OLSE89 Olsen, D., A Programming Language Basis for User Interface Management, Proceedings of CHI, 1989, ACM, New York, 1989, pp. 171-176.

OPEN89 Open Software Foundation OSF/MOTIF manual, Cambridge, MA, 1989.

PALA88 Palay, A. et al., The Andrew Toolkit: An Overview, *Proceedings 1988 Winter USENIX*, Feb. 1988, pp. 9–21.

SCHE86 Scheifler, B., and J. Gettys, The X Window System, *Transactions on Graphics*, 5(2), April 1986, pp 79–109.

SENA89 Senay, H., P. Sukaviriya, L. Moran, Planning for Automatic Help Generation, *Proceedings of Working Conference on Engineering for Human Computer Interactions,* Napa Valley, California, August 1989.

SUKA88 Sukaviriya, P., Dynamic Construction of Animated Help from Application Context, *Proceedings of ACM SIGGRAPH 1988 Symposium on User Interface Software and Technology (UIST '88)*, 1988, ACM, New York, NY.

SUKA90 Sukaviriya, P and J. Foley, Coupling a UI Framework with Automatic Generation of Context-Sensitive Animated Help, *Proceedings of ACM SIGGRAPH 1990 Symposium on User Interface Software and Technology (UIST '90)*, Snowbird, Utah, October, 1990, in press.

SUN89	Sun Microsystems, Inc., *Open Look Graphical User Interface Application Style Guidelines,* Mountain View Ca, December 1989.
SUN90	Sun Microsystems, Inc., *Open Windows Developer's Guide 1.1, Reference Manual,* Part No l. 800-5380-10, Revision A of June 1990.
VISU90	Visual Edge, *UIMX User's Manual*, Ville St. Laurent, Quebec, Canada, 1990.