Supporting Adaptive Interfaces in a Knowledge-based User Interface Environment

by

Piyawadee "Noi" Sukaviriya
and James D. Foley

GIT-GVU-92-19
July 1992

Graphics, Visualization & Usability Center

Georgia Institute of Technology
Atlanta GA 30332-0280

Supporting Adaptive Interfaces in a Knowledge-based User Interface Environment

Piyawadee "Noi" Sukaviriya
James D. Foley

Graphics, Visualization, and Usability Center
College of Computing
Georgia Institute of Technology
E-mail: noi@cc.gatech.edu, foley@cc.gatech.edu

ABSTRACT

Developing an adaptive interface requires a user interface that can be adapted, a user model, and an adaptation strategy. Research on adaptive interfaces in the past suffered from a lack of supporting tools which allow an interface to be easily created and modified. Also, adding adaptivity to a user interface so far has not been supported by any user interface systems or environments.

In this paper, we present an overview of a knowledge base model of the User Interface Design Environment (UIDE). UIDE uses the knowledge of an application to support the run-time execution of the application's interface and provides various kinds of automatic help. We present how the knowledge model can be used as a basic construct of a user model. Finally, we present adaptive interface and adaptive help behaviors that can be extended to the current UIDE architecture utilizing the user model. These behaviors are options from which an application designer can choose for an application interface.

KEYWORDS: User Interface Design Environment, User Model, Adaptive Interface, Adaptive Help

INTRODUCTION

Developing an adaptive interface requires a user interface which can be adapted, a user model, and an adaptation strategy. In simple cases, the user explicitly defines his or her own model and adaptation strategy. For example, an experienced user may be aware of her expertise level and request the system for full menus and brief prompts. A novice on the contrary may request short menus and lengthy prompts. In more interesting cases, the adaptive interface infers information about the user based on some aspects of how the user works with the application. A novice user who manages to use more advanced features quite frequently should be asked whether she wants full menus and less elaborated prompts.

A user *model* is a collection of information about a user of an application. The model might be as simplistic as the novice/expert distinction [1], device and interaction technique preferences [2], and screen layout preferences. In more interesting cases the model can be cognitively-oriented, representing for example how much the user knows about each application command, how the user currently conceptualizes the application, and how the user maps tasks onto command sequences. In general, the model can change over time as the user works with the application and learns (or forgets) various aspects of the application. Our interest is in modeling the user's procedural knowledge of how to use the application to carry out tasks.

Research on adaptive interfaces in the past suffered from a lack of supporting tools which allow an interface to be easily created and modified [3]. Also, adding adaptivity to a user interface so far has not been supported by any user interface systems or environments. We have been working for some time with UIDE, the User Interface Design Environment. UIDE uses knowledge of an application at design time to help automate the user interface design process, and at run time to control execution of the application and to provide various forms of run-time help. The UIDE knowledge model is declarative and object-oriented, dealing with objects, attributes of objects, actions on objects, pre and post conditions of those actions, and the interaction objects and interaction techniques through which the user interacts with the application. The model represents some of the application's semantics and the interaction mechanisms used to access those semantics.

The UIDE application model and user model will allow us to provide several forms of adaptive behavior. With respect to on-line help, we will be able to provide explanations of how to carry out various tasks based on the user's current knowledge: given several ways to carry out a task, the one which best maps onto the user's current knowledge of the application will be used. Similarly, the amount of detail and extent to which detail is repeated in an explanation can be adapted. Also, by combining the application model with a command trace, we will be able to reorganize menus and dialogue boxes and suggest new macro commands.

The UIDE adaptive project is in its first of the 3-year span. Our intention is to provide adaptive behaviors both on interfaces and help as interface design options. Upon an application designer selecting an adaptive behavior for an application, a user model will be automatically created. Data will be collected at runtime for evaluation of when the adaptive behavior should be triggered. Performance data are recorded to confirm or deny whether the adaptation improves the usage of the interface. Our long term goal is to make UIDE self-adapted; that is continually being able to evaluate adaptations automatically using collected user performance data and pre-defined analysis techniques.

RELATED WORK

User interface systems that utilize explicit knowledge representations about application actions and data models are HUMANOID [4], User Interface Design Environment (UIDE) [5,6], and MASTERMIND [7]. While HUMANOID's emphasis is on runtime support of control sequencing and automatic presentation of objects, UIDE is focused on supporting the interface design process throughout its life cycle–from its inception to the actual execution stage. In fact, MASTERMIND is a proposed design to merge HUMANOID and UIDE to combine the strength of their representations. All these systems have or will have explicit knowledge entities about application actions, objects, and interaction techniques, which can be used as a basis for constructing user models. UIDE has a more refined knowledge model separating interface actions/objects from those of an application, allowing user performance information captured for similar interface styles to be transferrable from one application to another.

With in the past few years, a number of research efforts have expanded UIDE's runtime support. These efforts include automatic generation of animated help [8,9], automatic layout of menus and dialogue boxes [10], a compositional model of interface component [11], and automatic generation of coordinated text and animated help [12]. These research efforts refine the UIDE's knowledge representation model and its architecture such that automatic support for user models, adaptive interfaces, and adaptive help have become less than far fetch extensions to the original UIDE. In addition, adaptive behaviors that are plausible in the UIDE framework are extensions of previous research such as generation of animated and textual help, and organization of menus and dialogue box layouts.

Unlike some other user models in the HCI field which revolved around user's mental models [13] or stereotypes of users [1], the user model under UIDE research is somewhat similar to those used in AI research–the user model of the person the system is interacting with [14]. One purpose of our user model is similar to that of UC [15]; it collected information while interacting with a user and used the information to assume how much help information should be presented to the user. In addition, the history of interactions in our user model is also used as a basis for suggesting changes in an interface layout.

The user model representations in UIDE are based on representations that are used primarily to construct interfaces, be they adaptive or not. Our work is quite different from other research where user models are usually built from scratch. According to Sanderson's compiled list of user model features [2], the UIDE user model falls in the same category as that of Sanderson's N-CHIME. N-CHIME's user model is explicitly represented for each individual user, dynamic over an interactive session, persistent throughout non-consecutive sessions, and its information is inferred from interacting with users. Interestingly, a number of elements in N-CHIME's user model such as number of times an action has been selected, user experience with various interaction techniques, etc., are similar to UIDE's user model. However, N-CHIME research centered around building a domain-independent adaptive interface, while our research has always emphasized on a comprehensive interface design environment where adaptive behaviors of an interface are some of the options a designer can choose.

The next section discusses the knowledge representations of UIDE that are used to generate application interfaces. This section should give readers a background on the kind of information captured for interface purposes. The following section will then show how a subset of this information can be used in constructing a user model.

KNOWLEDGE BASE MODEL

The UIDE knowledge model defines generic classes of various representation entities which are used to capture information about an application and its interface. A knowledge base of an application consists of instances of these generic classes; each instance pertains information specific to an application aspect it represents or the particular interface features chosen for the application.

An application is described in the knowledge base as a set of actions that the user can perform within the application, and objects on which these actions operate. Knowledge classes which are used to capture this information include: *action*, *parameter*, *parameter constraint*, *pre-condition*, *post-condition*, and *application object*. The knowledge base of an application consists of instances of actions referred to as *application actions*. Each action is linked to instances of parameters representing all parameters required by the action. Each parameter has a set of parameter constraint instances, which altogether state the boundary of acceptable values for the parameter. Pre-conditions and post-conditions of the action are described in the predicate language and are also linked to the action. A class hierarchy of application objects must be specified and each object has a list of actions that can apply to it.

Currently, the knowledge base of an application is considered a high-level abstraction of an application. It is the application-specific part of the UIDE knowledge base. Specifying it is viewed as enlisting the functionality–the semantics–of the application and does not commit the application to any particular interface styles. Readers who wish to find more detailed class descriptions should refer to

[9, 16, 17]. Research on higher-level goal-oriented task representations is under way to capture bigger units of application-specific semantic knowledge.

A set of user interface services that can be chosen for different applications are described as interface actions and interaction techniques. These representations reside in a domain-independent part of the knowledge base. An *interface action* is an instance of the *action* class. As the matter of fact, its structure and its associated entities such as parameters, pre- and post-conditions, etc., are similar to those of an application action. Interface actions represent mid-level intermediary means to interface with application functionality. Interface actions such as "selecting an action" or "selecting an object" in essence represent more concrete steps in an interface. Pre-conditions of interface actions include those which state the sequencing control, i.e., an object can be selected only after the action "delete" has been selected. Interface actions neither capture nor imply the input devices used for the actions.

The same interface action which is chosen for different application actions are represented as 2 instances in the knowledge model. For example, select-object can be used for selecting an object for a rotate action and a delete action. Two instances of select-object are created; one is linked to the rotate action, the other to the delete action. This is mainly because the designer may want "selecting an object" for the two actions to be done by different sets of interaction techniques. Though such a design may not be consistent, the representations accommodate such a flexibility.

Device-level knowledge is captured in *interaction techniques*. An interaction technique describes a meaningful sequence of lexical input and output, sometimes in relation to certain types of interface objects. An example of interaction techniques are pulling down a menu to select an item using a mouse, typing a string using a keyboard, and touching an object through a touch panel. An interface object can be either an OLIT or Motif widget or a view of an object defined in an application. Each interface object has associated with it interaction techniques which the user can use to interact with the object. An interaction technique is uniquely specified concerning input device or devices employed by the technique. In addition to being an active agent dealing with user inputs at runtime, an interaction technique also holds declarative information about its characteristics. This information is used by animated help to demonstrate interaction details at runtime.

Once an application is described in the knowledge base as actions, parameters, objects, and so forth, interface actions can be linked to actions and parameters to specify the interactions which will interface with each application function. This linking process is as the matter of fact automated in [16] using semantic information of actions and parameter types. In the same fashion, interaction techniques can be linked to interface actions to specify the actual interactions which will carry out these interface actions. More than one interaction technique can be linked to an interface action for alternative interactions. Interface objects such as dialogue boxes, menus, buttons, etc., will have to be specified and named in the process so they can be referenced in the specifications.

The UIDE runtime architecture is designed such that the UI representations are used to guide the dialogue sequencing. This means that UIDE does not hard-wire the dialogue sequence and bindings to interaction techniques in the runtime code. At any point in an interface session, UIDE keeps track of which application actions have all their pre-conditions satisfied thus are available to be used. The interface actions and interaction techniques associated with these enabled actions are also enabled. This allows UIDE to control which interaction techniques can be used by the use on which objects at any point in time, therefore control dialogue sequencing through these representations.

As you can see, the knowledge base is formed by not only components which are instances of knowledge classes, but also by hierarchical connections among components. Our knowledge model can be viewed as consisting of layers of representations which describe interface functionality at different levels of abstraction. At the highest level, representations capture more abstract components which essentially are application semantics. Representations at the interface action level capture interface activities independent of input devices used. The lowest level representations describe mouse and keyboard interactions and are designed to capture lexical groupings which define related lexical input and output. Through these layers of representations, a designer has different granularity of control over interface features which she can choose to interact with different parts of application functionality. Also, should the design decisions change, the designer only has to change the linkage among these layers and the resulting interface will be changed automatically at runtime.

USER MODEL AS A SUBSET OF THE KNOWLEDGE BASE MODEL

By walking through the representations of an application, the UIDE runtime component keeps track of which action is being interacted with, which parameter of which action is being entered, and which interaction techniques can be used and which is being used by the user. As the consequence of this architecture, the runtime component constantly visits different parts of the representations for various types of information. This makes it plausible to select the already existing components of the UIDE knowledge model as a start for constructing user models.

UIDE knowledge components revolve around concepts such as actions, parameters of actions, interface actions, and interaction techniques. Certain assumptions can be made upon a user's successful usage of each unit. Here are some examples. A successful invocation of an action can be assumed that the user knows about the action, knows about the parameters of this action, knows at least one set of interaction techniques for interacting with the action, etc. Upon using one interaction technique successfully, the system can assume that, to provide help on an unfamiliar action which employs the same technique, help will not have to elaborate on the technique. The user should have no difficulties executing the technique, but only needs to know that the technique could be used. Currently, we are not at a stage where we know how many successful uses of different components actually mean the user has understood about these components. This is part of our adaptive interface research and can only be confirmed by usability testing. Our current research emphasis is on constructing a model where related statistics can be collected.

Our approach is to use an overlay of the UIDE knowledge model as a user model to record information pertaining user history of interactions. However, much of UIDE representations deals with links among semantically related components. These links are not necessary in the user model since they can be inferred from the UIDE knowledge base. Those links hence are droped out in the user model. Each user has a copy of her own user model stored.

In most cases, we can assume there is an entity in the user model corresponding to a component in the knowledge base. For example, each application action would have a corresponding "shadow" object in the user model. Each of the action's parameters also has a "shadow" parameter object in the user model. However, not every component of the UIDE knowledge base needs to be duplicated in the user model structure. For example, information about the same interaction technique employed by two different actions should be the same "shadow" object for this class of technique.

Following are the categories of information which UIDE can collect for the user model using the current knowledge base structure.

Statistical History of Interactions

Upon a successful completion of each application action, its shadow in the user model has one count added to a slot marked "completed by the user." The shadows of the action's parameters also get one count for completion. After each action is completed, the date and time are stored. All interface actions and interaction techniques which are linked to the action and its parameters get one count as being completed by the user.

A more interesting recording is: when the user cancels an action and requests help on the action right after the cancellation, one count is added to "user having some difficulties" with the action. If the action is cancelled and no help is being requested, the action is marked "attempted by the user." Timestamps are stored with all of these recordings. A number of successful completion following usage with difficulties should override the fact that the user had difficulties in the past. Again, these numbers are still undetermined. It may prove useful to consider the state of user interaction when an action is cancelled, for example, action A is cancelled after the action is selected and its first parameter is entered. Such a recording can be easily done in the UIDE framework, even if the parameter values must be recorded. Uses of interface actions and their parameters are also recorded in the same fashion.

Two kinds of statistics related to interaction techniques are recorded. One is the success/failure usage for each interaction technique, which is stored in one place though the technique may be used in various application actions. This is based on the assumption that mastering an interaction technique is a transferrable skill. The other statistics is about which technique is chosen to perform an action or to enter a parameter. Since interface actions for each application action have separate shadows in the user model, their shadows can record the frequency of use for all interaction techniques associated with them. That is the slot for the technique in each interface action's shadow will get one count when the user chooses to use the technique of the action.

Chronological History of Interactions

Chronological history can be recorded rather straightforwardly in UIDE. Taking advantage of UIDE's layers of representations, history of user actions can be recorded at various levels. Recording events is possible, but normally takes much more space. Also, their related semantic activities are not represented in the data. By recording application actions and specific parameter values the user chooses, the semantic information is captured. As part of recording each application action, interface actions and chosen interaction techniques are also be recorded. Recording information at high-level of abstractions and storing specific parameter values make low-level event recording unnecessary. UIDE uses chronological recording to detect repeated patterns which can be suggested to users as macros.

History of Help Requests

It is often useful to remember the kind of help questions the user had in the past. History of help requests is stored in our user model to predict how much help to present to the same user the next time the same help is requested. History of help requests is recorded both statistically and chronologically with timestamps.

Since UIDE generate help on actions automatically [9], we will focus on help on actions in this research. The shadow for each action will have slots to record help-related statistics. For example, an action gets one count on help (and perhaps a recording of how long the user spends in each help session) when help on the action is requested. The action is marked visited every time help on the action is requested. Recording help is not necessary for action parameters since UIDE mentions or demonstrates parameters every time help on an action is requested.

Chronological recording of help requests is recorded separately. It is merely a list of actions for which help is requested and timestamps when the requests occurr.

USER MODEL SUPPORT FOR ADAPTIVE INTERFACES

The user model can be used to evaluate when an interface should adapt, and provide help which is adapted to the user. In the past, some adaptive systems have adapted the user interface without any user control, leading to user confusion and lack of acceptance [3]. Our approach for adaptive interfaces is to have the system suggest adaptations to the user, who will always maintain control over whether the adaptation is accepted. We discuss in this section adapting the interface, and in the next section adaptive help.

We are developing two types of interface adaptation: redesign of menus and dialogue boxes, and addition of new commands (macros). Because a reorganization requires relearning, and at least a temporary loss of productivity, reorganizations should not be made frequently, and should be based on a model which considers both the temporary loss of productivity due to relearning and the longer-term productivity gain due to the reorganization.

A simple power-law learning model [18] is used to predict how rapidly the user will return to the pre-reorganization level of performance, and a keystroke model [18] is used to predict performance using the reorganized interface.

The system will periodically examine a trace of user actions and consider alternative menu structures and dialogue box designs, evaluating each of them against the keystroke performance model applied to the trace and redesign. If the best reorganization found is predicted to improve user performance within a reasonable number of uses, then the reorganization will be suggested to the user, who will be free to accept or reject the suggestion.

To suggest command macros, the chronological history of interactions will be periodically examined to find repeated sequences of commands. The records of the sequence and timing of actions, parameters values specified by the user, and the interaction technique used for each action or parameter are considered. It will be necessary to identify which command parameter values are literals of the macro, and which are to be variables specified by the user when the macro is invoked. Also, variable-length repeating sequences of commands might take as a variable the number of iterations. In general, identifying macros faces the same challenges as does programming tasks by example, such as are discussed in [19].

USER MODEL SUPPORT FOR ADAPTIVE HELP

UIDE currently supports context-sensitive animated help [8,9] which is automatically generated using the knowledge of an application. Animated help demonstrates steps to perform actions by displaying mouse moving on the screen to represent cursor movement, with the appropriate button clicked when it is called for, and characters flowing from a keyboard to a dialogue box to represent character inputs. Animated help shows how to perform an action within the current context using the current context as part of the demonstration. UIDE also supports textual help explanations WHY an interface object is disabled [20], and how to enable it [21]. The textual and animated help are being integrated [12] to provide a more complete yet automatically generated help for users by showing and explaining procedural information.

Both animated help and help on how to enable a widget depends on the layer of UIDE representations to extract interface actions used for invoking an action and entering parameters. In particular, animated help uses links to interaction techniques to animate device-level activities in detail.

We are developing 2 types of help adaptation: selection of preferred procedural steps and variation of help contents. Unlike adaptive interfaces, adaptive help which varies help contents does not require relearning and can be done without users being aware of it. This is similar to human-to-human communication where a speaker often adjusts the conversation to fit the context and listeners.

The statistical history of interactions allows UIDE to assume that a technique used frequently by a user is a preferred technique. When help on an action is requested, of which multiple interaction techniques can be used, help can demonstrate how to perform this action based on the technique which is used quite often. However, if the purpose of help is to introduce alternatives to the user, other interaction techniques should be demonstrated.

Planning techniques are applied to tailor help content specific to the user context [9]. Occasionally, animated help cannot demonstrate within the current context because pre-conditions of the action for which help is requested are not all true. A planner is invoked to search for short paths which will satisfy the pre-conditions. The planning algorithm may find that a condition can be satisfied by one of several actions. The statistical history of interactions can be used as planning heuristics to choose a more appropriate action to pursue further planning.

Help can also adapt in between sessions in UIDE using the statistical recording of help requests. The first time help on an action is requested, UIDE would elaborate how to perform the action in detail using both textual and animated help. After a couple requests on the same action, animation can be faster and briefer leaving out continuity of animation. A repeated request on the action over a period of time may result in animation being left out altogether. On the contrary, if the user fails to select an appropriate interaction technique or to accurately locate related interface objects on the screen, help should start using animated demonstrations again.

USER MODEL REVISITED

The information tokens captured in the UIDE user model is all based on history of interactions. User choices of interactions are used to assume what she knows, what she prefers, what she did wrong, and what she has trouble with. This may not cover all information UIDE needs in the long run to make accurate assumptions. However, gathering these numbers allows UIDE to make basic assumptions about users, which typical interfaces cannot make. This information seems to map nicely with what human experts partially use when trying to evaluate the knowledge of a user. For procedural help purposes, history of interactions provide an excellent basis for determining how much details of help should be presented within a narrow time window.

Performance data can be extracted from timestamp recordings. These data are used primarily to support power-law analysis to confirm or reject an adaptation. Though performance time data are used, UIDE adaptations are not aimed at making users type faster or move the mouse quicker. We are more interested in interfaces which are dynamic and keep up with the users. And this is reflected in an interface which provides more or less help information as appropriate, and suggests better menu and dialogue box layouts to ease user tasks.

Notice that, until now, the kind of data UIDE accumulates does not need stereotypical models to set up the initial set of data. The user model starts out having no information about the users. More information is accumulated as the users interact with UIDE interfaces longer.

In this paper, we only discuss using components from the UIDE knowledge base as a basic construct for a user model. Other information may be needed and may be computable or derivable from this basic set of data. Some information may need to be obtained by inquiring users directly. The need for additional information will become clear when we have adaptive behaviors implemented and when we try them out with real users.

CONCLUSIONS

In this paper, we presented the knowledge model of UIDE which has been refined substantially since reported in [5,6]. We emphasized that this same knowledge model could be used as a basic construct of a user model. Details of this user model were elaborated. Adaptive interface and adaptive help support which could be extended as part of UIDE runtime support and which utilized the user model were explained in the previous 2 sections.

UIDE has been developed in Automatic Reasoning Tool (ART) and Smalltalk-80. It is currently transferred to a C++ platform. This paper reports the implementation which is in progress, and the design of the user model and adaptive behaviors to be implemented in this platform.

ACKNOWLEDGEMENTS

This work has been supported by the Siemens Corporate R&D System Ergonomics and Interaction group of Siemens Central Research Laboratory, Munich, Germany, and the Human Interface Technology Group of Sun Microsystems through their Collaborative Research Program. The work builds on earlier UIDE research was supported by the National Science Foundation grants IRI-88-13179 and DMC-84-205-29, and by the Software Productivity Consortium. We thank the members of the Graphics, Visualization, and Usability Center for their contributions to various aspects of the UIDE project: J.J. "Hans" de Graaff, Martin Frank, Mark Gray, Todd Griffith, and Srdjan Kovacevic. We also thank colleagues and former students at the George Washington University who contributed to UIDE: Hikmet Senay, Christina Gibbs, Won Chul Kim, Lucy Moran, and Kevin Murray.

REFERENCES

1. Rich, E. Stereotypes and User Modeling. In *User Models in Dialog Systems*. Eds. A. Kobsa and W. Wahlster. Berlin: Springer-Verlag, 1989.

2. Sanderson, P. *Structure Design if an Adaptive Human-Computer Interface*. A Ph.D. Dissertation, University of Pittsburg, 1991.

3. Browne, D.; P. Totterdell; and M. Norman. *Adaptive User Interfaces*. London: Academic Press, 1990.

4. Szekeley, P.; P. Luo; and R. Neches. Facilitating the Exploration of Interface Design Alternatives: the HUMANOID Model of Interface Design. In *Proceedings of Human Factors in Computing Systems, CHI'92*. May 1992, 507-515.

5. Foley, J.D.; C. Gibbs; W.C. Kim; and S. Kovacevic. A Knowledge-based User Interface Management System. In *Proceedings of Human Factors in Computing Systems, CHI'88*. May 1988, 67-72.

6. Foley, J.D.; W.C. Kim; S. Kovacevic; and K. Murray. UIDE–An Intelligent User Interface Design Environment. In *Architectures for Intelligent Interfaces: Elements and Prototypes*. Eds. J. Sullivan and S. Tyler, Reading, MA: Addison-Wesley, 1991.

7. Neches, R.; J.D. Foley; P. Szekeley; P. Sukaviriya; P. Luo; S. Kovacevic; and S. Hudson. Knowledgeable Development Environments Using Shared Design Models.

Paper submitted to 1993 International Workshop on Intelligent Interfaces.

8. Sukaviriya, P., and J.D. Foley. Coupling a UI Framework with Automatic Generation of Context-Sensitive Animated Help. In *Proceedings of the ACM SIGGRAPH Symposium on User Interface Software and Technology*. October 1990,. 152-166.

9. Sukaviriya, P. *Automatic Generation of Context-sensitive Animated Help*. A Ds.C. Dissertation, George Washington University, 1991.

10. Kim, W.C., and J.D. Foley. DON: User Interface Presentation Design Assistant. In *Proceedings of the ACM SIGGRAPH Symposium on User Interface Software and Technology*. October 1990,. 10-20.

11. Kovacevic, S. A User Interface Representation Model Based on a Compositional View. Paper submitted to 1993 International Workshop on Intelligent Interfaces.

12. Sukaviriya, P. and de Graaff, J. Automatic Generation of Context-sensitive "Show & Tell" Help. Paper submitted to 1993 International Workshop on Intelligent Interfaces.

13. Gentner, D. and A.L. Stevens *Mental Models*. Hillsdale, NJ: Lawrence Erlbaum, 1983.

14. Wahlster, W. and A. Kobsa. User Models in Dialog Systems. In *User Models in Dialog Systems*. Eds. A. Kobsa and W. Wahlster. Berlin: Springer-Verlag, 1989.

15. Chin, D. KNOME: Modeling What the User Knows in UC. In *User Models in Dialog Systems*. Eds. A. Kobsa and W. Wahlster. Berlin: Springer-Verlag, 1989.

16. Kovacevic, S. *A Compositional Model of Human-Computer Interaction*. A Ds.C. Dissertation, George Washington University, In progress.

17. Foley, J.D.; D. Gieskens; W.C. Kim; S. Kovacevic; L. Moran; and P. Sukaviriya. A Second Generation Knowledge Base for the User Interface Design Environment. *Report GWU-IIST-91-13*. Washington, DC.: Dept. of EE & CS. The George Washington University, 1991.

18. Card, S.K.; T.P.Moran; and A. Newell. The Psychology of Human-Computer Interaction. NJ: Lawrence Erlbaum Associates Publishers. 1983.

19. Cypher, A. Eager: Programming Repetitive Tasks by Example. In *Proceedings of Human Factors in Computing Systems, CHI'91*. 1991, 33-39.

20. Gieskens, D. and J.D. Foley. Controlling User Interface Objects Through Pre- and Post-conditions. In *Proceedings of Human Factors in Computing Systems, CHI'92*. May 1992, 189-194.

21. de Graaff, J. H. Automatic Generation of Context-sensitive Textual Help. A Master Thesis. Delft University, the Netherlands. In progress.

The Graphics, Visualization, and Usability Center is an interdisciplinary center which serves as a focal point for these specialties on the Georgia Tech campus. Intellectually, our interests span relevant aspects of Computer Science, Psychology, Industrial and Systems Engineering, Design, Mathematics, and Computer Engineering.

The GVU Center has three missions: education, service, and research. In our educational role, we teach the principles and methods of computer graphics, visualization, and usability to members of the academic community ranging from undergraduate students to graduate students and faculty. A series of short courses are offered through the Division of Continuing Education. Our service mission is carried out through the Scientific Visualization Lab, a joint undertaking with Information Technology (the campus-wide computing service), which provides state of the art computer graphics hardware and software capabilities to the entire Georgia Tech community. Our research spans the areas of Monte Carlo and radiosity methods for global illumination, virtual reality applications in medical procedures, medical imaging, model-based image understanding, animation of physical objects, application of psychophysics to enhanced image generation, algorithm animation, stereo graphics, image quality, expert systems in graphics and user interfaces, knowledge-based processing of medical images to assist in diagnosis, visualization of multiparametric data, user interface software, usability, adaptive user interfaces, and computer-supported collaborative work.

By integrating education, service, and research missions together in a single unit, the Center is developing a collaborative environment where researchers unfamiliar with computer graphics can come for help in integrating scientific visualization into their research work, graphics experts and graduate students can share their knowledge with one another and find difficult new research problems, and students can learn in an intellectually stimulating melting pot of closely-related ideas and collaborations between researchers from multiple disciplines. We believe this environment prepares our students for effective professional careers and encourages research which is both relevant and challenging.

Our lab facilities will be housed (as of January) in 2800 square feet of newly-renovated space. Equipment includes seven Silicon Graphics systems, a Pixar Image Computer, twelve Sun SPARC systems, color recorder and digitizer, Sony video disk, complete suite of video and audio editing equipment, several stereo shutters, a NeXT, and several Macintoshes.

Current members of the Center are:

Dr. James Foley	Center Director and Professor, College of Computing,
Dr. Al Badre	Associate Professor, College of Computing and Psychology
Dr. Marc Berger	Professor, Department of Mathematics
Dr. Greg Corso	Associate Professor, Psychology
Dr. Elizabeth Davis	Associate Professor, Psychology
Dr. Norberto Ezquerra	Associate Professor, College of Computing
Dr. Brian Guenter	Assistant Professor, College of Computing
Dr. Larry Hodges	Assistant Professor, College of Computing
Dr. Laurie Hodges	Research Scientist, Georgia Tech Research Institute
Dr. Scott Hudson	Assistant Professor, College of Computing (effective Jan. 1992)
Dr. Daryl Lawton	Associate Professor, College of Computing
Ms. Beth Mynatt	Research Scientist, Multimedia
Mr. William Putnam	Research Scientist, Multimedia
Dr. Bill Ribarsky	Sr. Research Scientist, Physics, and Client Services, Inf. Tech.
Dr. Holly Rushmeier	Assistant Professor, School of Mechanical Engineering
Dr. John Stasko	Assistant Professor, College of Computing
Dr. Noi Sukaviriya	Post Doctoral Fellow, GVU Center
Ms. Mary Trauner	Manager, Associate Director for Client Services, Inf. Tech.
Dr. Neff Walker	Assistant Professor, Psychology
Ms. Joan Morton	Administrative Assistant