Edgar H. Sibley
Panel Editor

A single adaptable user interface (AUI) which allows the user to switch between any number of different dialogue modes at any time—even in the middle of a command—can be useful to a variety of users who are neither beginners nor experts. It can also be used in applications where different dialogue modes are appropriate for the various parameters of a single command. An implemented user interface management system (UIMS) suggests the practicality of AUIs and their automatic generation.

The Adaptable User Interface

Eliezer Kantorowitz and Oded Sudarsky

THE AUI CONCEPT

Software systems interact with users in a large variety of ways (*dialogue modes*). These methods may be classified as the menu-type and the command-language-type. In a *menu-type-dialogue mode* (MM), the user controls the system solely through the selection of options from a number of choices presented. It is assumed that only choices that make sense are presented to the user, and that it is, therefore, not possible to select options that are not permitted. In a *command-language-type dialogue mode* (CLM), the user controls the system by instructions given in a certain command language. The user must know this language in order to use the system. In contrast with an MM, where the user is guided by menus and can only enter legal choices, the user of a CLM may enter erroneous instructions. A CLM should, therefore, be designed to detect improperly formulated instructions and to correctly recover from their effects.

The advantages of an MM over a CLM are well known. Since the users do not have to learn any command language, they may become productive with a new system after a very short time. A new user may explore the operations provided by a system simply by browsing through the menus. If such a system also provides adequate help for every menu option, the user may operate the system without ever needing a manual.

Menu systems may, however, be less satisfactory for frequent users who have to work through a large number of menus to get their work done. Waiting for a menu to appear on the screen, finding the right menu entry, and making the selection can take time. Proficient users who do not need guidance tend to prefer a concise CLM where a few keystrokes, entered at full typing speed, achieve the same effect as a number of relatively slow menu selections. Isaki and Schneidermann observed in [7] that "Knowledgeable users often remark that they would prefer to type commands and believe that they can work more rapidly by just typing commands."

Realizing the different needs of beginners and experts, many systems provide two distinct user interfaces: a menu interface and a command-language interface. In these systems, the user has to select in advance (in the beginning of the session or before each command) one of these two interfaces that is best suited to the task. If a user wants to change the dialogue mode, the current command must be completed first, and then the user must request that the system switch to the desired user interface. There are cases, however, when it is useful to change the dialogue mode in the middle of a command. To demonstrate this, we will employ an example command for drawing a black box with the lower left corner at (0,0) and the upper right corner at (1,1):

```
BOX BLACK (0,0) (1,1)
```

The first case to be discussed is that of a user who is neither a beginner nor an expert but is somewhere between these two extremes. Such a user may realize that he or she has forgotten some command language element while in the middle of composing a command and may need the assistance of menus [3, 5]. Assume that the user has typed the word BOX of the example command and is uncertain about what colors are available. Shifting to an MM will facilitate color selection from a menu. A similar situation occurs when the user has made a mistake, e.g., entered a color that does not exist. After reading the error message, the user can switch to a menu of colors in order to select an available color.

The need to assist users who cannot complete a command is realized in current systems, which enable commands to be programmed such that users are prompted for missing parameters. These systems, however, do not

This research was supported by the Gutwirth Fellowship Fund and by the Nicholas M. and Gisela Deutch Fellowship in memory of their martyred parents and families.

give the users the freedom to employ at any time the dialogue mode that is most productive for their level of experience. The users' freedom in changing dialogue modes solves another class of problems which we will discuss next.

One of the criteria for deciding whether to use a CLM or an MM is the nature of the input data and the physical properties of the I/O devices. In our example, the coordinates of the corners are known by their exact numerical values ((0,0) and (1,1)), and it is, therefore, appropriate to enter them by typing them at the keyboard (a CLM). If a corner is only known by its position on the screen, however, it must be entered by moving the cursor with a mouse or another locating device. By our definition, this method for entering coordinates is an MM, where the points of the screen are the choices and the selection is made with the mouse. We observe that depending on the nature of the actual input data, the same command is sometimes entered in a CLM and, in other cases, in an MM. Sometimes it is required to employ two different dialogue modes within the same command. For instance, one of the corners of the box is known by its position on the screen (and must, therefore, be entered by pointing at it) while the other corner is known by the numerical values of its coordinates (and should be entered through the keyboard).

It is sometimes useful to have more than one MM or more than one CLM. As an example, let us consider a system with two different CLMs for the same command language: a voice-input mode and a keyboard-input mode. Voice input is preferred when the user has to operate away from the terminal or is otherwise occupied. On the other hand, keyboard input may be quicker or less error prone in noisy environments.

We propose an adaptable user interface (AUI), which will allow the user to switch dialogue modes in the middle of a command. An *adaptable user interface* is defined as an interface that:

- supports a number of different dialogue modes. More than two modes may be provided;
- allows the user to switch between dialogue modes *at any time*, i.e., even in the middle of a command;
- makes the switch between dialogue modes smoothly and naturally;
- makes it easy for the user to learn how to use the different dialogue modes, especially the CLMs, which usually require a longer training period.

In order to enable simple and natural switching between dialogue modes, a number of assumptions and requirements are proposed. The central assumption is that all the dialogue modes of an AUI are different representations of a single underlying *dialogue language*. This common language is assumed to be constructed of a number of elementary syntactic components, to be called *tokens*. Every token is required to have a distinct representation in each of the dialogue modes. In an MM, a token is represented by a single menu selection, while the corresponding representation in a CLM is an atom of the command language. For example, in the GUIDE system, to be described later, CLM tokens are represented by character strings. Each token may be entered in any one of the available dialogue modes, independent of the modes employed for the other tokens. Two subsequent tokens may thus be entered in two different modes.

Beginners and casual users will employ the AUI in an MM. As they become more familiar with the system, they will gradually learn the CLM instructions that they need. A user can exploit the CLM commands already learned and employ an MM for all the other commands. Users will not have to learn CLM commands that are rarely used since they may be entered in an MM.

Additionally, each token can be entered in the most suitable way. For example, in the BOX command, each of the two corners is given by a single token. One corner of the box may be entered using a mouse while the opposite corner can be entered by typing its coordinates. The implementation of a user interface is usually a major effort. This is especially true for an AUI, in which several input devices must be monitored simultaneously. It is, therefore, desirable to have a user interface management system (UIMS) [10, 11, 15, 16] that automatically generates AUIs. Nonetheless, none of the existing UIMSs seem to allow the easy production of AUIs. Most systems can only generate single-dialogue-mode user interfaces. In systems that offer several modes, the end user is usually required to select a single dialogue mode at the beginning of the session. The Workspaces system [1] allows only partial adaptability (keyboard parameters must be entered first, followed by the parameters given by other input devices). The IOT [16], Switchboard [14], and Sassafras [6] systems may possibly be extended by the user interface designer with code that supports several dialogue modes; however, writing such code is a difficult task that requires insight in parallel programming of I/O devices.

THE GUIDE SYSTEM

In order to test the practicality of AUIs and of their automatic generation by a UIMS, the GUIDE (Graphic User Interface Design Environment) system was implemented [13]. Further design goals of GUIDE were:

- specification and modification of a user interface should be simple and require no programming skills. This will enable the system to be used by human factor experts who are not necessarily programmers. The ability to easily modify the user interface is important since human behavior may not be precisely predicted, and some debugging may be required;
- extending the user interface with new I/O devices and associated dialogue modes should be easy and require only minimal modification of the system.

An application program developed with GUIDE has three main modules called the *lexical*, *syntactic*, and *semantic* components. The lexical component identifies the tokens in the stream of input events. The syntactic component analyzes the stream of tokens it receives from the lexical component and invokes the semantic component when required. The *semantic component* is the collection of application routines written by the application programmers in some ordinary programming languages.

The syntactic and lexical components constitute the user interface of the application program. GUIDE generates this user interface from specifications given through interactive graphic design tools. The user interface specifications do not cause the generation of any code; rather, they are stored in a relational database and are later interpreted by a *run-time environment.* The code of this run-time environment is identical for all GUIDE-developed applications; only the database and the semantic component are different. A change in the user-interface specifications only requires a modification of the database. The effect of such a change can be seen immediately, since no compilation and linkage are required. This facilitates rapid prototyping of user interfaces since the designer can try several alternative solutions within a short time.

The Syntactic Component

The syntactic component of the user interface employs a *recursive transition network* (RTN) as the definition of the dialogue language. An RTN interpreter executes this definition when it analyzes the stream of input tokens. RTNs were chosen for the syntax representation because they are as powerful as deterministic, context-free grammars yet easier to use than BNF representation, especially for nonprogrammers [2, 4, 8]. An overview and comparison of current methods for specification of the syntax of dialogue languages is found in [16].

An RTN is constructed from a number of *subnets.* Each subnet is represented by a directed graph. The following kinds of *states* (nodes) and *transitions* (edges) may appear in a subnet (see Figure 1):

- *initial state*—the state in which the execution of subnet begins;
- *return state*—causes control to return to the calling subnet;
- *subnet call state*—causes control to pass to another subnet (or, recursively, to the same subnet);
- *application call state*—causes an application routine to be executed;
- *input state*—causes control to wait for the reception of a token from the lexical component. A menu may be associated with this state; if required, this menu will be displayed automatically when this state is reached;
- *output state*—causes the display of a message to the end user;
- *plain transition*
- *return transition*—appears after an application call state and is traversed if the routine has returned the return code associated with this transition;
- *option transition*—appears after an input state and is traversed if the user has selected the menu option associated with this transition;
- *parameter transition*—appears after an input state and is traversed if the user has picked an object of the type associated with this transition.

It is noted that a subnet may recursively call another subnet or itself. This enables the grammar of the dialogue language to be defined in a modular way in much the same way as a program is constructed from subroutines.

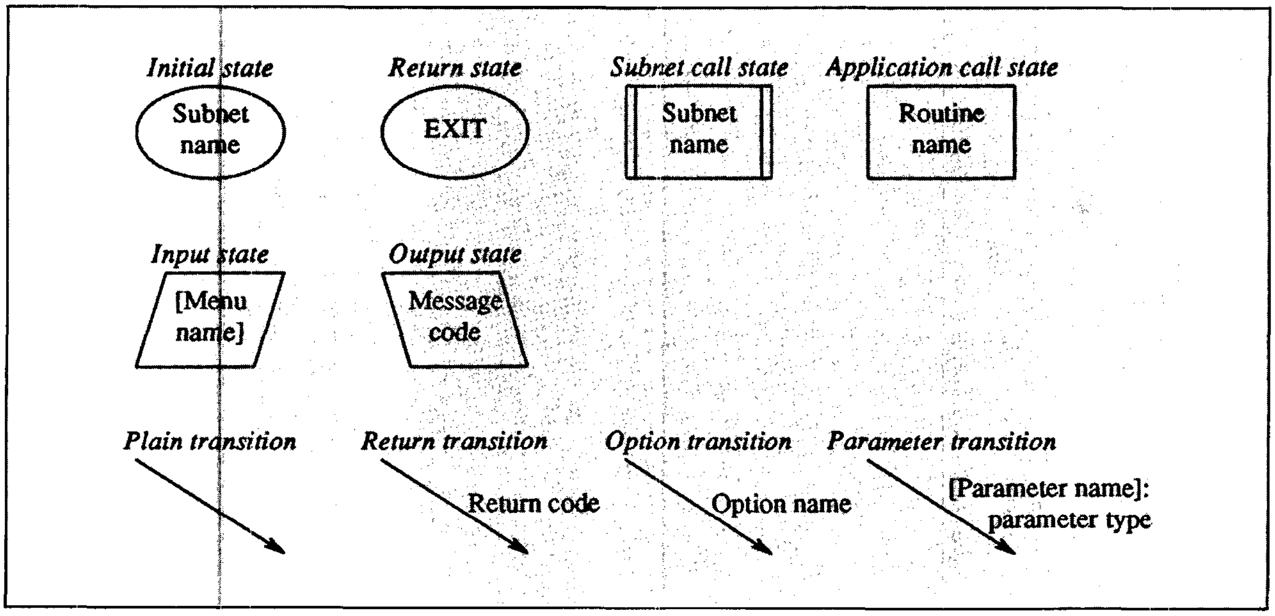


FIGURE 1. The Graphic Representation of RTN States and Transmissions

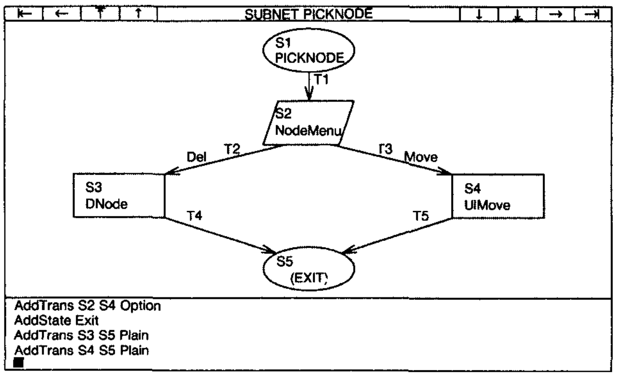
Figure 2 shows an example of an RTN subnet called PickNode. The execution of this subnet by the RTN interpreter starts at the initial state S1 and then immediately passes to the input state S2. A menu called NodeMenu is associated with this state. The user has now to select one of the two options in this menu. If, for instance, the option named Del is selected, the option transition T2 will be traversed, and the application call state S3 will be reached. The semantic procedure DNode associated with this state will be called by the RTN interpreter. Finally, the return state S5 will be encountered, and the execution of the subnet will terminate.

GUIDE provides an interactive graphic editor called SYNEDIT. This editor allows the user interface designer to construct the RTN which defines the syntax of the dialogue language. A SYNEDIT screen is shown in Figure 2. SYNEDIT checks the consistency and completeness of the RTN to make sure that it can be executed. The user interface for SYNEDIT itself was generated by GUIDE.

The Lexical Component

The lexical component of a GUIDE-generated user interface manages all the input and output of the program. The output consists of text messages, menus, icons, and *links*. A link is a line that connects two icons. Icons and links may be used to represent the different objects on which the application operates. They can, for instance, be employed to show the nodes and edges of the graphs that appear in some applications.

GUIDE's lexical component is called by the RTN interpreter (the syntactic component) when it encounters an input or output state. The lexical component currently supports two dialogue modes. Every menu, icon, or link can, therefore, be selected in one of two ways: by pointing with a mouse (an MM) or by typing the option's or object's name (a CLM). Text typed at the keyboard appears in a special CLM text area at the bottom of the screen (see Figure 3 and the examples in Figures 2 and 7). If the user employs the mouse, the name of the selected menu option or object will be copied by GUIDE into the CLM area as if the user has typed them in. The CLM area will, therefore, in all



```
AddTrans S2 S4 Option
AddState Exit
AddTrans S3 S5 Plain
AddTrans S4 S5 Plain
■
```

FIGURE 2. An RTN Subnet being Edited by SNYEDIT

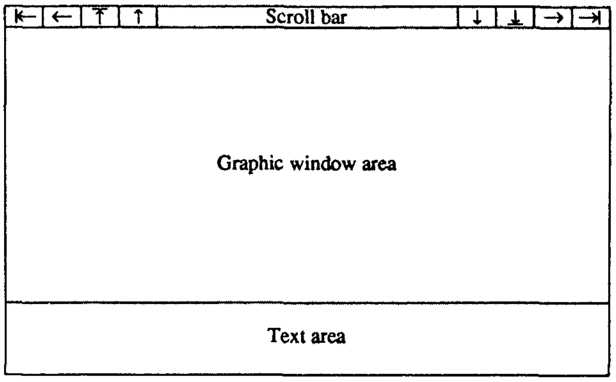


FIGURE 3. Screen Layout for a Guide-Developed Application

cases show the command language representation of the command being entered. This helps the user to learn the command language. Note again that each of the tokens in the command may be entered in a different dialogue mode. Furthermore, note that the user does not have to tell the system to switch between dialogue modes—but simply uses whichever device (mouse or keyboard) wanted.

The different dialogue modes are managed solely by the lexical component. When the RTN interpreter receives a token from the lexical component, it has no knowledge of the mode in which this token was entered. This makes the system relatively easy to adapt to future dialogue modes and input devices (e.g., a speech recognizer) since only the lexical component will have to be changed, while the syntactic and semantic components will remain unchanged.

GUIDE includes a program called LEXEDIT that allows the user interface designer to define icons, links, menus, and messages. In the icon editor screen (see Figure 4) the icon is drawn using graphic primitives and text fields. The icon is shown twice: life-sized in the corner of the screen and enlarged in the main window. In the link editor screen (see Figure 5), the designer can specify the attributes of the link: line style, arrowheads, and link shape. Text fields can be placed along the link. In the menu editor screen (see Figure 6), the menu's graphic appearance is drawn. The name and the rectangular region occupied by each option in the menu can be defined. The designer has a choice of three menu styles: static, pop-up, and pull-down. This choice of facilities allows the implementation of many of the currently popular user interface styles.

EXAMPLES OF GUIDE-DEVELOPED APPLICATIONS

In order to test the applicability of GUIDE in various areas, user interfaces for three different applications were constructed. The applications are:

- a directed graph editor,
- a specification program for management information systems (MISs), and
- the RTN editor of the GUIDE system.

The directed graph editor is a small program. It was

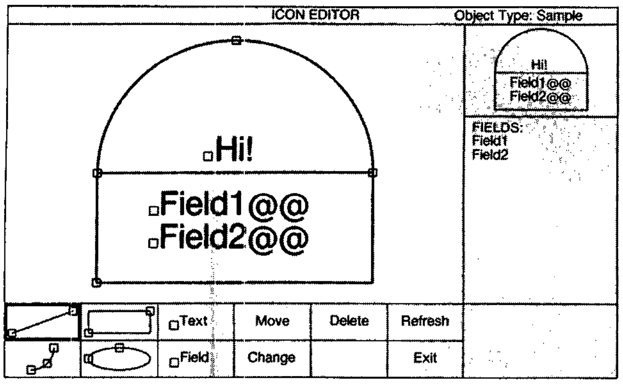


FIGURE 4. LEXEDIT's Icon Editor Screen

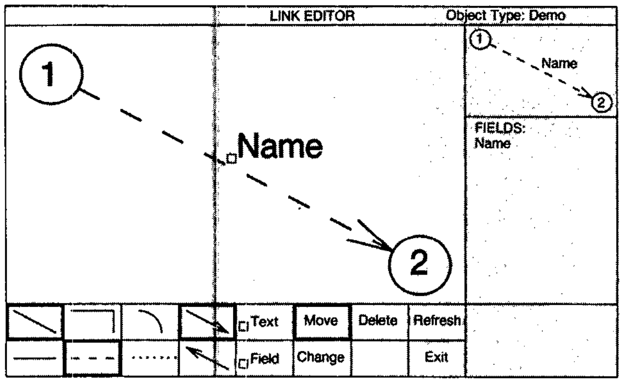


FIGURE 5. LEXEDIT's Link Editor Screen

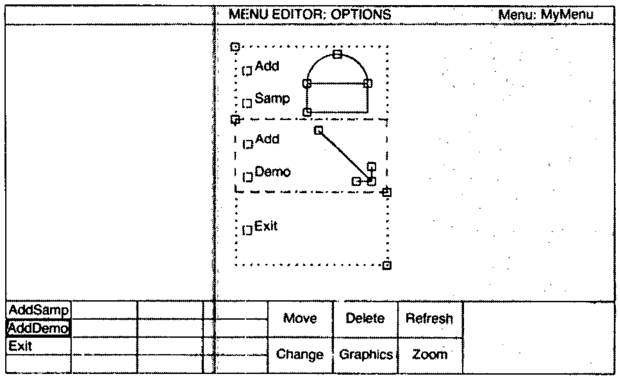


FIGURE 6. LEXEDIT's Menu Editor Screen (in Option Specification Mode)

implemented as a simple example for the GUIDE user's manual. The program lets the end user edit a directed graph interactively.

The entire user interface for this application was developed in less than two hours. The user interface designer used LEXEDIT to define the shape of the icons which represents the nodes of the graph, the link which represents the edges, and the menus. The designer used SYNEDIT to construct an RTN with commands to add, move, and delete nodes and edges. Figure 2 shows one of the subnets of this RTN being edited. The resulting user interface, as it appears to the end user, is shown in Figure 7.

The use of AUIs can be illustrated with this directed graph editor. Suppose the end user wants to add an edge from node v to node w. One way to do it is to type at the keyboard the command:

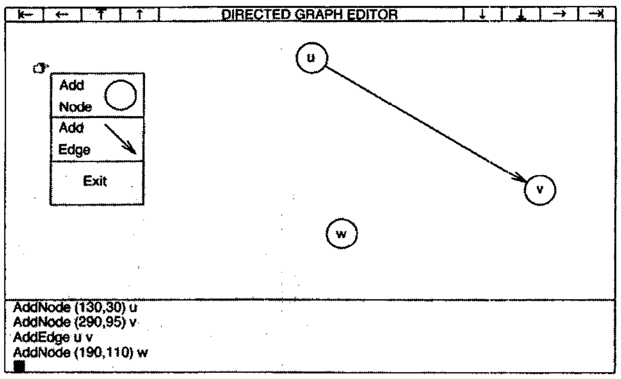
```
AddEdge v w
```

If the user has forgotten the command name `AddEdge`, however, selection can be made from the pop-up menu as shown near the upper left-hand corner of Figure 7 and then point at the nodes v and w with the mouse. Entering a node name at the keyboard may, however, be preferred when the node is not currently visible because scrolling has moved it off the screen. In fact, there are $2^3 = 8$ possible ways to enter the three tokens of this command.

A second application developed with GUIDE is a program to assist system analysts in the development of design specifications for MISs. It was developed by D. Reider at the computer science department at Technion [12]. The program supports a top-down design methodology combining data modeling and structured analysis. The system analyst employs three kinds of tools: an organizational structure diagram, an entity-relationship diagram (ERD) (see Figure 8), and data-flow diagrams (DFDs). Each of these diagrams is composed of different kinds of icons and links. The design methodology requires that certain complex relationships be maintained between the different diagrams.

The application lets the user construct these diagrams with an interactive graphics editor. The consistency and completeness of each diagram and the relationships between the diagrams are checked automatically. The diagrams are stepwise-refined and are, therefore, changed frequently. The resulting diagrams can be quite complex and would be difficult to manage manually. The user interface for this application was developed in about forty hours.

A third application is the RTN editor SYNEDIT, which is employed in GUIDE to specify the syntax of dialogue languages (see the section "The Syntactic Component" and Figure 2). The RTN states (shown in Figure 1) are represented as icons, while the transitions are displayed as links. The editor is employed to con-



```
AddNode (130,30) u
AddNode (290,95) v
AddEdge u v
AddNode (190,110) w
```

FIGURE 7. The Directed Graph Editor Screen (Shown with a Pop-Up Menu)

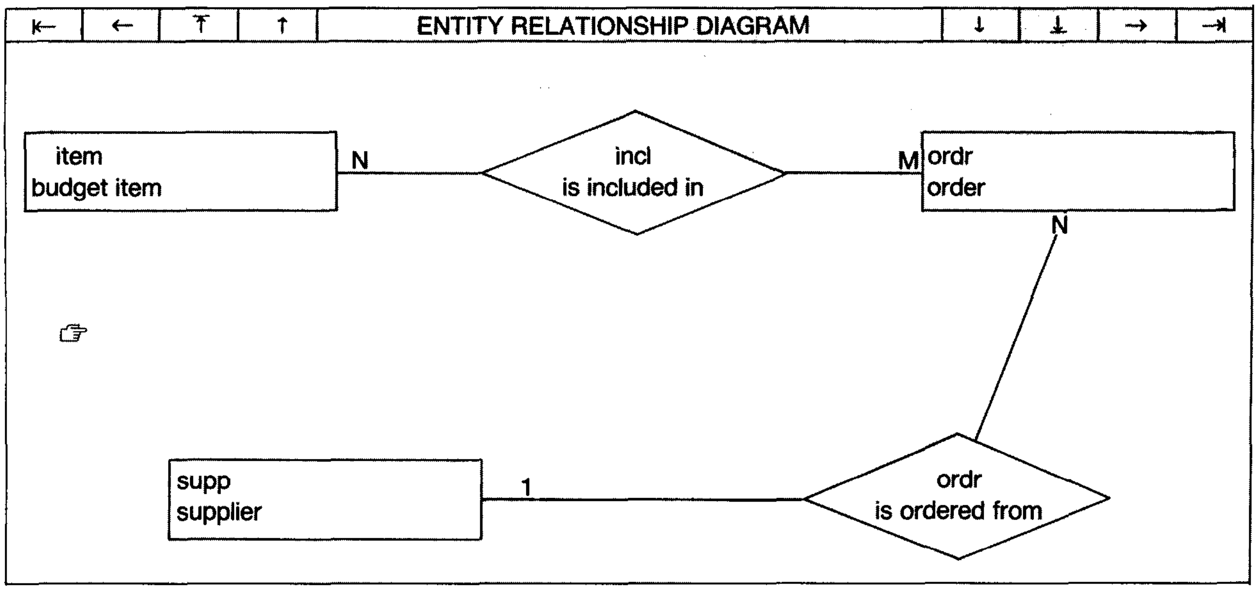


FIGURE 8. An ERD being Edited in the MIS Specification Program

struct the subnets which constitute the RTN and to check the correctness of the RTN. Since SYNEDIT could not be used to enter the specifications of its own RTN, these data had to be entered manually into the specification database. The rest of the specifications for SYNEDIT were input with LEXEDIT.

IMPLEMENTATION

GUIDE was implemented on an IBM PC AT-compatible computer. Most of the software was written in the dBASE III PLUS database programming language and compiled with the Clipper compiler. The use of a relational database to store user interface specifications allowed rapid and simple development of GUIDE. Since the dBASE language cannot be fully compiled, however, the system is somewhat slow. A reimplementation of GUIDE in a compilable programming language is expected to result in satisfactory performance.

CONCLUSIONS

It has been argued that it is sometimes useful to employ a number of different dialogue modes (such as typing text at the keyboard and pointing with a mouse) within the same command. This cannot be achieved in the current systems that provide different dialogue modes by applying an essentially separate user interface for each dialogue mode. In order to meet these needs, this article introduces the *adaptable user interface* (AUI) concept which integrates a number of dialogue modes into a single user interface.

The freedom of the AUI user to adapt dialogue modes to actual needs is useful at all levels of experience. A novice user starts by using menu modes (MMs) where guided by choices presented by the system and gradually learns how to utilize the command language. An experienced user employs the faster command language modes (CLMs) but can switch to menus in the middle of a command if uncertain about how to finish it. An AUI also enables the user to choose the dialogue mode that is most suitable for the nature of input data and the working environment.

The implementation of the GUIDE system suggests that both AUIs and a UIMS which generates AUIs are practical on personal computers. It also shows that all dependencies on I/O devices and dialogue modes can be isolated in the lexical component of the user interface. This facilitates future additions of any number of new dialogue modes to the user interface of an already existing application.

User interfaces for three different applications were implemented relatively quickly and conveniently using GUIDE. In these applications, switching between different dialogue modes appears natural and gives the user a new degree of freedom in exploiting the system.

Acknowledgments. We thank David Reider for his kind help in system analysis and in the development of one of the examples.

REFERENCES

1. Enderle, G. The flexible configuration of interaction environments using GKS and Workspaces. In *Proceedings of the Seeheim Workshop on UIMS.* Springer-Verlag, New York, 1985.
2. Green, M. Report on dialogue specification tools. In *Proceedings of the Seeheim Workshop on UIMS.* Springer-Verlag, New York. 1985.
3. Grimes, J. D. A knowledge oriented view of user interfaces. In *Proceedings of the 12th Hawaii International Conference on System Sciences, Vol. 1* (Honolulu, Hawaii, Jan. 4–5.) ACM, New York, 1979, pp. 158–163.
4. Guest, S. P. The use of software tools for dialogue design. *Int. J. Man-Machine Studies 16,* (Apr. 1982), 263–285.
5. Heffler, M. J. A human-computer interface that provides access to a diverse user community. In *Proceedings of the 14th Hawaii International Conference on System Sciences* (Honolulu, Hawaii, Jan. 8–9). ACM/IEEE, New York, 1981, pp. 601–610.
6. Hill, R. D. Supporting concurrency, communication, and synchronization in human-computer interaction—The Sassafras UIMS. *ACM Trans. Graph. 5,* 3 (July 1986), 179-210.

7. Iseki, O., and Shneiderman, B. Applying direct manipulation concepts: Direct Manipulation Disk Operating System (DMDOS). *Softw. Eng. Notes 11*, 2 (Apr. 1986), 2–26.
8. Jacob, R. J. K. Using formal specifications in the design of a human-computer interface. *Commun. ACM 26*, 4 (Apr. 1983), 259–264.
9. Myers, B. A. User-interface tools: Introduction and survey. *IEEE Softw.* (Jan. 1989), 15–23.
10. Olsen, D. R., Jr., et al. ACM SIGGRAPH workshop on software tools for user interface management. *Comput. Graphics 21*, 2 (Apr. 1987), 71–174.
11. Pfaff, G. E., Ed. User interface management. In *Proceedings of the Seeheim Workshop on UIMS*. Springer-Verlag, New York, 1985.
12. Reider, D., Kantorowitz, E., and Raz, Y. Specification of management information systems combining data modeling and structured analysis. In *Proceedings of the 4th Israel Conference on Computer Systems and Software Engineering* (Tel Aviv, Israel, June 5–6.) IEEE-CS, New York, 1989, pp. 34–44.
13. Sudarsky, O. A user interface management system adaptable to various user experience levels. Thesis, Dept. of Computer Science, Technion-Israel Institute of Technology, Haifa, Israel, 1988.
14. Tanner, P. P., et al. A multitasking switchboard approach to user interface management. In *Proceedings of SIGGRAPH '86: 13th Annual Conference on Computer Graphics and Interactive Techniques* (Dallas, Tex., Aug. 18–22.) ACM/SIGGRAPH, New York, 1986, pp. 241–248.
15. Thomas, J. J., and Hamlin, G. Graphical Input Interaction Technique (GIIT) workshop summary. *Comput. Graphics 17*, 1 (Jan. 1983), 5–30.
16. van den Bos, J., Plasmeijer, M. J., and Hartel, P. H. Input-Output tools: A language facility for interactive and real-time systems. *IEEE Trans. Softw. Eng. SE-9*, 3 (May 1983), 247–259.

CR Categories and Subject Descriptors: D.2.2 [**Software Engineering**]: Tools and Techniques—*user interfaces*; D.2.6 [**Software Engineering**]: Programming Environments—*interactive*; D.2.m [**Software Engineering**]: Miscellaneous—*rapid prototyping*; D.3.4 [**Programming Languages**]: Processors—*interpreters, run-time environments*; H.1.2 [**Information Systems**]: User/Machine Systems—*human factors*; I.3.6 [**Computer Graphics**]: Methodology and Techniques—*interaction techniques, languages*

General Terms: Human Factors, Languages

Additional Key Words and Phrases: Dialogue languages, recursive transition networks, user interface management systems, user interfaces

ABOUT THE AUTHORS:

E. KANTOROWITZ is currently a professor at the Computer Science Department at Technion–Israel Institute of Technology and an ACM member. His research interests are in adaptable user interfaces, fault tolerant distributed database systems, and he is involved in the design of an industrial database-oriented ship design system. Author's Present Address: Computer Science Department, Technion–Israel Institute of Technology, Haifa 32000, Israel. KANTOR@TECHSEL.BITNET

O. SUDARSKY recently completed his Master of Science studies and is an ACM member. His research interests are adaptable user interface management systems. Author's Present Address: 11 Mishmar HaYarden St., 18412 Afula, Israel.