GUIDE: Graphical User Interface Development Environment

Tamar E. Granor
Norman I. Badler
MS-CIS-85-19

Department of Computer and Information Science
Moore School/D2
University of Pennsylvania
Philadelphia, PA 19104

May 1985

* Acknowledgement: This research was supported in part by NSF grants MCS8219196-CER, MCS-82-07294, 1 RO1-HL-29985-01, U.S. Army grants DAA6-29-84-K-0061, U.S. Air Force grant 82-NM-299, AI Center grants NSF-MCS-83-05221, US Army Research office grant ARO-DAA29-84-9-0027, Lord Corporation, RCA, and Digital Equipment Corporation.

GUIDE: GRAPHICAL USER INTERFACE DEVELOPMENT ENVIRONMENT

Tamar E. Granor
Norman I. Badler

Department of Computer and Information Science
Moore School of Electrical Engineering
University of Pennsylvania
Philadelphia, PA 19104

Appeared in Trends and Applications 1985, IEEE Computer Sociey, Washington Chapter and National Bureau of Standards, Silver Spring, Md., May, 1985.

Abstract

GUIDE is an interactive graphical system for designing and generating graphical user interfaces. It provides flexibility to the system designer while minimizing the amount of code which the designer must write. The GUIDE methodology includes the notions of "tool," "task," and "context." GUIDE encourages designers to tailor their systems to individual users by inclusion of "user profiles," allowing different control paths based on the user's characteristics. GUIDE also provides a method for invoking application routines with parameters. Parameters may be based on user inputs and are computed at invocation time. Help messages are created along with the objects to which they refer. GUIDE handles the overhead required to display help messages.

1. Introduction

For many system designers, the most difficult part of building a system is the user interface. While the designer is familiar with the application and its operations, the tools of interface design may be unfamiliar, especially when the system uses graphics. GUIDE, the Graphical User Interface Development Environment, facilitates the generation of interactive systems by allowing a designer only to know how the system should look and behave externally.

GUIDE is an interactive system which can be used by an application designer to generate the user interface for a wide class of program. A number of goals have been set for GUIDE.

1. The designer need not write any interface code.
2. The designer provides "action routines" which implement the actions of the application system. Action routines may have parameters.
3. The designer is able to specify multiple control paths based on the state of the system and a profile of the user.
4. Inclusion of help messages is as easy as possible.
5. GUIDE's own interface may be generated with GUIDE.

To achieve the first goal, the designer uses GUIDE to describe the physical appearance and control path for the system. GUIDE generates Pascal code which, together with the designer-supplied action routines, makes up the application system. The generated interface can be edited directly or the designer may modify the design using GUIDE and re-generate the interface.

In the second case, the designer provides "action routines" which are invoked by the GUIDE-generated interface. Actual parameters for these routines are computed in a manner specified by the designer. The parameters may be based on user inputs.

The third goal is that designers be able to provide multiple control paths. The choice of path can be based on a profile of the user, the contents of which is specified by the designer. The profile may contain such items as the user's "skill level." Conditions involving fields of the user profile, as well as other variables, may be included in the control path.

To achieve the goal that inclusion of helps be as simple as possible, help messages for any object are specified at the same time as the object is created and may be edited any time the object is edited. GUIDE handles the overhead required to display helps.

Insuring that GUIDE's interface may be generated using GUIDE has affected the design of the system in several areas, most notably in the structure of menus.

Throughout this paper, examples will be drawn from the furniture layout system described by Foley and van Dam [Foley 82], which consists of manipulating a pre-defined set of symbols representing furniture (desk, chair, *etc.*) until a satisfactory layout is achieved. The user may add symbols to or delete symbols from the layout, may title or re-draw the current layout and may change the window into the layout world.

2. Background

In recent years, much attention has been given to user interfaces. Researchers have studied what kinds of commands are most easily learned and remembered [Barnard 82, Black 82], what kinds of interfaces are easiest to use [Card 82, Savage 82] and how to make it easier to provide a good interface [Buxton 83, Kasik 82].

One approach for simplifying interface implementation that has been studied recently is the use of a User Interface Management System (UIMS). A UIMS mediates between the application and the user in much the same way that a data base management system mediates between the application and the data [Kasik 82, Thomas 83]. UIMS implementations have been reported by several groups [Buxton 83, Kasik 82, Bloom 83, Kamran 83, Olsen 83a, Olsen 83b, Roach 82, Rubel 82, Wong 82].

GUIDE is a UIMS generator, meaning that it allows the system designer to specify the relationship between the inputs, outputs and control paths of his application. The designer does not write the control code; GUIDE generates it.

GUIDE is also similar to a compiler-compiler (*e.g.*, YACC [Johnson 75]). The user of a compiler-compiler provides a grammar which is processed. The compiler-compiler then produces as output the tables needed to compile programs. The output is used by an end-user (to compile programs). With GUIDE, the designer provides a description of the desired interface which is processed to produce the interface itself. The description of the interface is developed interactively, however, rather than specified as a grammar. This interface, together with other code provided by the designer, is presented to the end-user.

The Graphical Input Interaction Techniques Workshop [Thomas 83] characterized UIMS's along three scales: internal vs. external control, single- vs. multi-thread control and simple vs. hierarchical dialogues.

GUIDE generates an external control UIMS, meaning that the UIMS is in control and invokes the

application as a slave. It permits multiple threads of control, allowing the user to have several commands under construction at the same time. GUIDE provides hierarchical tools, thereby allowing hierarchical dialogues.

The control hierarchy for a system with a GUIDE-generated interface is shown in Figure 2-1. The user is in control of the system. When he does something (*i.e.*, causes an event), the GUIDE-generated interface is activated. It invokes the toolkit to determine what the user did. The toolkit, in turn, invokes the window manager and so forth until the appropriate hardware driver is queried. The interface can also invoke the application package, which has access to the window manager.

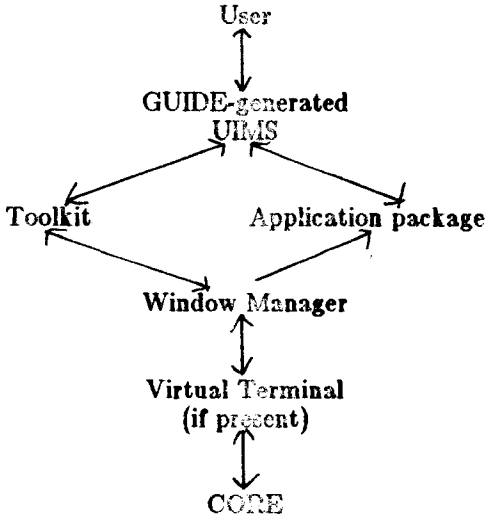

Figure 2-1: Control Flow using GUIDE

The flow of data in a system using GUIDE is shown in Figure 2-2. Data flows from the user to an input device, triggering a CORE event. CORE, in turn, informs the virtual terminal, which passes the information to the window manager and so forth until the GUIDE-generated interface is reached. For output, the interface invokes either the toolkit or the application package, which pass the data down the hierarchy until it is displayed.

3. Building Blocks

The basic building blocks of GUIDE are "tools," "tasks" and "contexts." A "tool" is a technique for graphical input and/or output. A "task" is a transformation of data which can be achieved by use of one of a set of tools. A "context" describes the appearance of the screen at some time, along with the possible actions. A context consists of tasks, user-defined pictures and decisions. The relationship between contexts, tasks, tools, user-defined pictures and classes, and decisions is shown in figure 3-1.

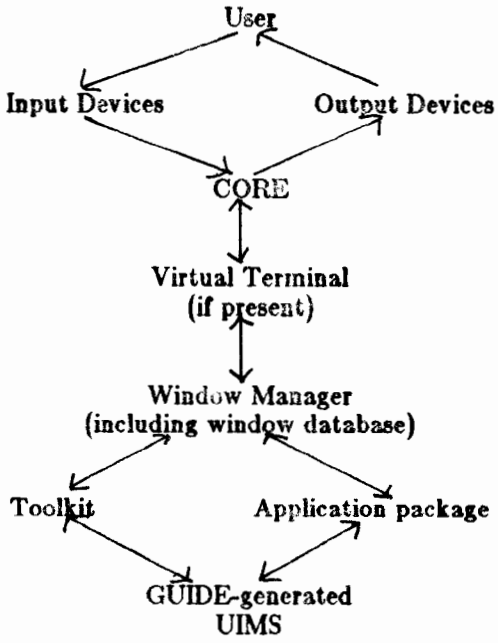

Figure 2-2: Data Flow in a GUIDE-generated Interface

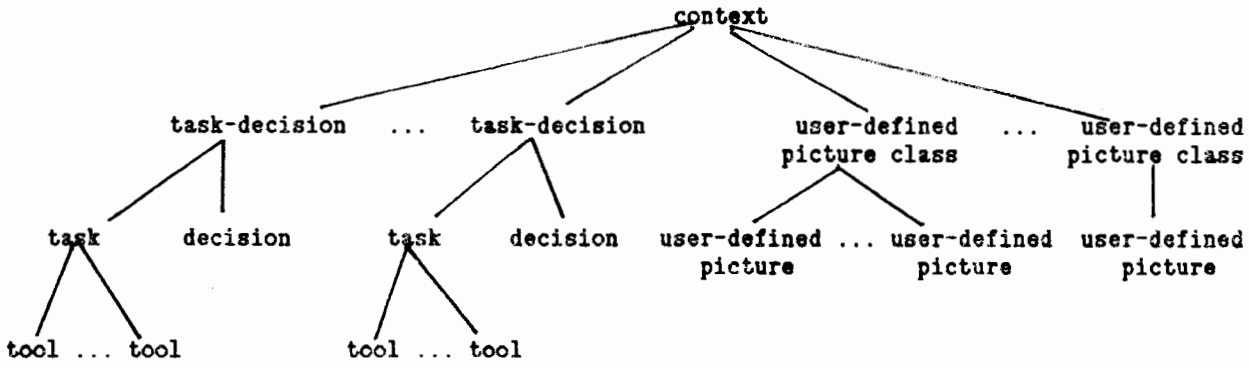

Figure 3-1: Structure of Entities in GUIDE

3.1. Tools

The set of tools available in GUIDE is predefined. Some of the tools available in GUIDE are menus, lists, forms and potentiometers. Each tool has zero or more methods (routines) for drawing itself, echoing itself and processing input. Some tools may have no drawing routines because they have no physical appearance. For example, a button corresponding to a physical button device would not need to be drawn on the screen. A tool may also have a number of options controlling its appearance and performance. For example, a menu may be vertical or horizontal or it may be laid out by the designer. Such options mean that there is no one physical appearance for a given tool type.

The designer may instantiate as many tools as desired. The designer chooses the characteristics for each instance which, in turn, determine which routines are used for drawing and echoing the tool and processing input to the tool. The designer may, if desired, provide alternative routines for the processes.

In the layout system, a menu tool might be provided for choosing commands, while a list tool could be used for choosing an item of furniture upon which to operate.

3.2. Tasks

Tasks allow a designer to provide the user with options while reducing the difficulty of doing so. A task contains several tools for achieving the same goal plus optional actions to be executed when the task is used. For example, the "choose command" task may include both a menu tool and a keyboard tool. The user may use either tool to achieve the same end. However, the designer need specify the associated control flow only once.

A small set of tasks is predefined, corresponding to frequently used real world tasks (*e.g.*, "choose command"). In addition, a designer may define as many tasks as desired. Each task contains one or more tools, and for each tool, routines which convert the output of the tool to the type desired by the task and convert from the task type to the tool type.

Any instance of a predefined task may be edited to change the set of tools available either by adding or removing tools.

The layout system would include a "choose command" task and a "choose object" task. The "choose command" task would contain the menu tool mentioned above plus any other command selection tools desired (*e.g.*, a keyboard tool). The "choose object" task would contain, at least, the above-mentioned list tool. Other tasks would be provided for picking points within the room, changing the window into the room and getting help.

3.3. User-Defined Pictures

User-defined pictures are items which the designer wants to display. Generally, they are graphical representations of the application's data structures. The designer specifies a name, a drawing routine and a (starting) viewport for each picture, and the GUIDE-generated interface displays the picture in the specified location.

User-defined pictures are organized into classes. A class is a set of different graphical instantiations of the same data type. Classes may be modified dynamically, when the application system is running. The designer may provide routines which operate on members of a class; the particular picture upon which to operate is passed as a parameter.

In the layout system, the display of the current layout would be a user-defined picture. If multiple views of the layout were to be provided, all of the views would be contained in one class.

3.4. Contexts

A context describes the state of the system at some time. It includes all of the objects (tasks and user-defined pictures) which may be displayed or used. When changing contexts, a context may be pushed onto or popped from a stack of contexts.

The presence of a window manager is assumed; it is expected to handle the details of placing the viewports ("windows") on the screen and identifying viewports when a "pick" is made.

Each context also contains "decisions" which determine the new state of the system after an input from the end-user. A decision contains conditions plus instructions on the change in control if the conditions are met. Decisions are organized in a priority order (like a LISP *cond*) so that the first decision to have its conditions fulfilled will be chosen. A null condition may be expressed, forcing the change in control to occur. The default, if all conditions fail, is to remain in the same context.

Conditions are boolean expressions involving task values, system variables and user profile fields. For simplicity in generating code, the syntax used is that of expressions in Pascal with modifications to handle task values and user profile fields.

Expressions are also used for specifying parameters, defaults and initial values and to indicate which values to restore when popping a context off the stack. Expressions may refer to task values, application variables and user profile fields. They may contain arithmetic, boolean and logical operators as well as function references. GUIDE generates code to evaluate expressions when they are used.

The layout system would contain one main context in which most interaction would occur. The context would contain the "choose command," "change window" and "help" tasks, along with the current display of the room.

4. Actions

The actual work of the application system is done in "action routines" supplied by the system designer. GUIDE provides a method for specifying what routine is to be executed when, and for specifying the parameters for an action routine. Parameters may be based on end-user inputs. The designer may provide several ways of entering the necessary inputs, differentiated by conditions based, among other things, on the end-user's profile.

Each task may have one or more action routines associated with it. The action to be executed when a task is used depends on conditions specified by the designer. For example, a "choose command" task would have a separate action for each possible command.

Many action routines require parameters. While it is possible to require that all parameters be specified (by means of "current" values) before an action is chosen, this enforces a particular style of programming and interaction. GUIDE, therefore, allows the designer to provide an expression for each parameter. The expression may include system variables and task values. The designer may also specify

any number of contexts to be visited prior to executing the action routine. These "parameter contexts" contain special tasks for the purpose of providing any inputs necessary to compute the parameters of the action routine.

More than one sequence of parameter contexts may be provided along with several ways of computing the parameters. In both cases, conditions are used to determine which option to select.

In the layout system, a parameter context for the "add_symbol" routine (associated with "add symbol" command) would include the "choose object" task for selecting the symbol and the "choose location" task for positioning it. This context might also contain all of the tasks in the main context, if commands may be interrupted and stacked.

5. User Profiles

At many points in GUIDE, the designer may provide several alternatives differentiated by conditions. One of the major motivations for this capability is to allow each system to be tailored for each user. The essential feature in doing so is the "user profile."

The designer constructs the user profile to contain any desired information about the user. Some items which might be included are the user's "skill level" (skill with the system) and "access level" (right to access information in the system). Other contents depend on the nature of the system being implemented, but will generally include the user's preferences in dealing with the application system, for example, level of error messages to be displayed.

User profile fields may be used in expressions. It is expected that the primary use will be in conditions affecting the control flow of the application.

The designer specifies the method whereby user profile fields receive initial values and are updated. A number of simple methods will be provided for the designer to choose from. The designer may provide and specify more complex methods, if desired. Any user can access, at most, only his own profile. Access to each field in the profile may be controlled by conditions based on the profile. In systems where security is an issue, access to some fields may be prohibited. For such systems, a separate system or sub-system must be provided to maintain the user profile data base.

6. Helps and Prompts

The "help" and "prompt" messages associated with an object are created when the object itself is created. As with all other items, they may be edited freely. For each object, the designer may specify both a brief help message and the name of a file containing a longer help message. In addition, each object may have several pairs of helps and several prompts distinguished by conditions, allowing messages to be geared to individual users.

A "help task" is provided containing several possible ways of triggering the help system. The designer may edit this task to eliminate any methods that are felt to be inappropriate for the system.

Prompts are displayed at a location specified by the designer when the corresponding object is displayed.

7. Output

The output from GUIDE consists of several files. The designer may request that the prototype be stored in a file for later examination or modification. The contents of this file can be read into GUIDE and edited during another session. A journal file containing a complete record of each session using GUIDE is also created. Various errors may occur in the interface design; error messages will be stored in a third file. Lastly, the major output of GUIDE consists of Pascal code which can be linked with the application code and the toolkit to form the complete interactive application system.

8. Interface

Since one of the stated goals of GUIDE is that its interface may be generated using GUIDE, there is no one pre-determined interface to GUIDE. This section describes an initial version of an interface, built by calling GUIDE's application routines from a non-graphical bootstrap program. It is expected that later versions will be generated using this initial version both as an interface and as a base upon which to build. Later versions may be adjusted to user preferences and experiences.

The initial interface is primarily menu-driven with extensive use of forms, especially for instantiation of tools and tasks. Whenever possible, information may be entered by picking items from the display.

The methods for creating GUIDE entities and editing them are virtually identical. In each case, the appropriate command is selected. Then, the entity to be modified is specified. Last, the values of individual fields are entered using a form. In general, only those fields which are being changed need to be entered.

In this initial version the display is fairly static. In later versions the user will be able to look at any part of the interface being designed. Both the graphical representation and the underlying data structures of the objects being created may be viewed. The user of GUIDE will have control over most of the contents of the screen.

The initial version also assumes that routines already exist for drawing the user-defined pictures. However, there is no reason why a later version cannot provide links to pre-existing systems such as a graphic editor and a palette program for color selection.

9. Current Status

GUIDE is currently being implemented on a VAX/11-785 running VMS. Graphics are handled by a local implementation of CORE [Stluka 82].

The initial implementation is expected to be completed in mid-1985. It will contain a small toolkit of those tools most commonly used. Several features will be omitted from the implementation, including

journal files and an "undo" task. It is expected that these features could be added with little difficulty.

Early testing has shown that the action routines for GUIDE can be used as a subroutine package. This capability is being used to generate a first interactive interface for GUIDE. One of the early tests of this version will be to generate a new version of GUIDE.

Acknowledgements

This research has been partially funded by the Department of Computer Science, University of Pennsylvania and by Army Research Office contract #DAAG-29-84-K-0061 and NSF CER Grant #MCS-82-19196.

References

[Barnard 82] Barnard, P., Hammond, N., MacLean, A. and J. Morton.
Learning and Remembering Interactive Commands.
In *Human Factors in Computer Systems*, pages 2-7. Institute for Computer Sciences and Technology - National Bureau of Standards, U.S. Department of Commerce and Washington, D.C. ACM Chapter, Gaithersburg, MD, March, 1982.

[Black 82] Black, J. and T. Moran.
Learning and Remembering Command Names.
In *Human Factors in Computer Systems*, pages 8-11. Institute for Computer Sciences and Technology - National Bureau of Standards, U.S. Department of Commerce and Washington, D.C. ACM Chapter, Gaithersburg, MD, March, 1982.

[Bloom 83] Bloom, Douglas A.
A User-Oriented Interface Control (of an Interactive Computer Graphics System).
Master's thesis, University of Pennsylvania, May, 1983.

[Buxton 83] Buxton, W., Lamb, M. R., Sherman, D. and K. C. Smith.
Towards a Comprehensive User Interface Management System.
Computer Graphics 17(3):35-42, July, 1983.

[Card 82] Card, Stuart K.
User Perceptual Mechanisms in the Search of Computer Command Menus.
In *Human Factors in Computer Systems*, pages 190-196. Institute for Computer Sciences and Technology - National Bureau of Standards, U.S. Department of Commerce and Washington, D.C. ACM Chapter, Gaithersburg, MD, March, 1982.

[Foley 82] Foley, J.D. and van Dam, A.
Fundamentals of Interactive Computer Graphics.
Addison-Wesley, 1982.

[Johnson 75] Johnson, S. C.
YACC: Yet another compiler-compiler.
Computer Science Technical Report 32, Bell Labs, 1975.

[Kamran 83] Kamran, Abid and Feldman, Michael B.
Graphics Programming Independent of Interaction Techniques and Styles.
Computer Graphics 17(1):58-66, January, 1983.

[Kasik 82] Kasik, David J.
A User Interface Management System.
Computer Graphics 16(3):99-106, July, 1982.

[Olsen 83a] Olsen, Dan R. Jr.
Automatic Generation of Interactive Systems.
Computer Graphics 17(1):53-57, January, 1983.

[Olsen 83b] Olsen, Dan R. Jr. and Dempsey, Elizabeth P.
SYNGRAPH: A Graphical User Interface Generator.
Computer Graphics 17(3):43-50, July, 1983.

[Roach 82] Roach, J., Hartson, H. R., Ehrich, R., Yunten, T. and D. Johnson.
DMS: A Comprehensive System for Managing Human-Computer Dialogue.
In *Human Factors in Computer Systems*, pages 102-105. Institute for Computer Sciences and Technology - National Bureau of Standards, U.S. Department of Commerce and Washington, D.C. ACM Chapter, Gaithersburg, MD, March, 1982.

[Rubel 82] Rubel, Andrew.
Graphic Based Applications - Tools to Fill the Software Gap.
Digital Design , July, 1982.

[Savage 82] Savage, Ricky E., Habinek, James K., and Thomas W. Barhart.
The Design, Simulation and Evaluation of a Menu-Driven User Interface.
In *Human Factors in Computer Systems*, pages 36-40. Institute for Computer Sciences and Technology - National Bureau of Standards, U.S. Department of Commerce and Washington, D.C. ACM Chapter, Gaithersburg, MD, March, 1982.

[Stluka 82] Stluka, F. P., Saunders, B. F., Slayton, P. M. and Badler, N. I.
Overview of the University of Pennsylvania CORE System.
Computer Graphics 16(2):177-186, June, 1982.

[Thomas 83] Thomas, James J.
Graphical Input Interaction Techniques Workshop Summary.
Computer Graphics 17(1):5-30, January, 1983.

[Wong 82] Wong, Peter C. S. and Reid, Eric R.
Flair - User Interface Dialog Design Tool.
Computer Graphics 16(3):87-98, July, 1982.