TOWARDS A COMPREHENSIVE USER INTERFACE MANAGEMENT SYSTEM

W. Buxton, M. R. Lamb,
D. Sherman & K. C. Smith

Computer Systems Research Group
University of Toronto
Toronto, Ontario
Canada M5S 1A4

ABSTRACT

A UIMS developed at the University of Toronto is presented. The system has two main components. The first is a set of tools to support the design and implementation of interactive graphics programs. The second is a run-time support package which handles interactions between the system and the user (things such as hit detection, event detection, screen updates, and procedure invocation), and provides facilities for logging user interactions for later protocol analysis.

The design/implementation tool is a preprocessor, called MENULAY, which permits the applications programmer to use interactive graphics techniques to design graphics menus and their functionality. The output of this preprocessor is high-level code which can be compiled with application-specific routines. User interactions with the resulting executable module are then handled by the run-time support package. The presentation works through an example from design to execution in a step-by-step manner.

CR Categories and Subject Descriptors: D.2.2 [Software] Tools and Techniques - User Interfaces; H.1.2 [Information Systems] User/Machine Systems - Human Information Processing; I.3.4 [Computer Graphics] Graphics Utilities - Software Support; I.3.6 [Computer Graphics] Methodology and Techniques - Interactive Techniques.

General Terms: Human-Computer Dialogue, Interaction Management.

Additional Keywords and Phrases: Dialogue Design and Specification, and Dialogue Run-Time Support.

1. INTRODUCTION

Traditionally, interactive graphical programs have been written using conventional programming languages, low-level tools, and often *ad hoc* techniques. The cost of doing so has been time, frustration, and quality of end product.

A *user interface management system* (UIMS) is intended to reduce this cost. The objective is to free the applications programmer from low-level details so as to be able to concentrate on higher applications-specific aspects of the user interface. A UIMS typically consists of a package of tools which support the implementation, debugging and evaluation of interactive human-computer dialogues.

An analogy can be drawn between a UIMS and a data base management system (DBMS). The DBMS mediates between programmer and data, enforcing a consistency of technique among all programmers in accessing that data. It provides portability because only the lowest-level DBMS routines are hardware-dependent. Similarly, the UIMS mediates between the applications programmer and the input events, encouraging a consistency both of graphical layout representation and of input processing mechanisms.[1]

This paper presents a UIMS developed at the Computer Systems Research Group, University of Toronto. The system comprises a set of tools for designing and implementing menu-based dialogues and for providing run-time support for more general event-driven interactive programs. The module for designing/implementing menu-based dialogues has the property of permitting hand-written programs to be integrated with code automatically synthesized from the programmer's specifications.

1.1 The Need for a UIMS

Our current understanding of human-computer interaction is extremely limited. We cannot sit down and design an interface for an application and know, *a priori*, how well it will perform. We must accept at the outset, therefore, the inevitable intertwining of specification, design, and implementation (Swartout & Balzer, 1982). Design then becomes an iterative process, each iteration consisting of three phases: *design*, *implementation* and *evaluation* (Buxton & Sniderman, 1980). The motivation to develop improved tools can therefore be seen as a desire to increase the number of iterations that we can afford to pass through in this loop. The hoped-for consequence will be an improvement in the quality of the user interfaces which we produce.

The UIMS presented herein aids in the *design* and *implementation* phases by providing the following set of tools: a graphics package; sketch editors; a standardized graphics communications protocol; table-driven run-time support of the user dialogue; and, most recently, a graphical layout and interaction specification preprocessor which generates C language programs. Underway are the development of interaction analysis tools to aid in the *evaluation* phase of the cycle.

1. This is not to imply that data-base management is independent of the user interface. Rather, the tools applied to each of these free the applications programmer so that their mutual influence can be better taken into consideration.

Each of the modules of the existing system has limitations. However, when viewed together, the various pieces begin to shed some light on what would comprise a comprehensive set of tools for the user interface designer. In presenting our experience in a longitudinal way, it is hoped that we will make some contribution towards the realization of such a comprehensive set of tools.

1.2 U of T UIMS Structure

In its current state, our UIMS consists of two main modules. The first is a preprocessor which enables the program designer, using interactive graphics techniques, to design and specify the graphical layout and functionality of menu-based user interfaces. It is with this module that the applications programmer establishes the relationships between what the user sees, does, and hears, and the application-specific semantics underlying the program being implemented.

The second main module is the run-time support package. It handles things such as event and hit detection, procedure invocation, and updating the display - all according to the schema specified using the preprocessor. An important aspect of this implementation is that the code executed by the run-time module can be generated automatically (complete with comments) from the data specified to the preprocessor.

The applicability of the current implementation of the preprocessor is restricted to menu-based interaction. Within this domain, however, it is quite flexible, and supports various types of displays (colour or monochrome) and sound output. The run-time support package is more general and does not depend upon the preprocessor. It is designed for supporting event-driven software and a variety of input devices. These two packages do not stand alone. They are just two components in a more general "kit" of prototyping tools to support iterative and experimental programming (Deutsh & Taft, 1980). In discussing these tools, we find it useful to distinguish between *general design tools* and *programmers' packaged units*.

The general design tools facilitate the specification of graphical information and functional relationships, the writing of application-specific code and the debugging of successive versions of applications programs. Many of these tools are standard components in modern operating systems. Besides the two modules mentioned above, our inventory of such tools includes: (1) the UNIX[2] operating system, with its highly sophisticated command interpreter, C language compiler, hierarchical file system and many utilities; (2) the GPAC device-independent graphics package (Reeves, 1975); (3) routines to save and retrieve pictures stored as standard format metafiles; (4) text editors; (5) graphical sketch editors; and (6) interactive symbolic debuggers.

Programmers' packaged units are ready-made modules which can be integrated into applications software. They are documented and tested *building-blocks*. They are generally packaged versions of blocks of code which are observed to be reccuring and of general utility. Our library of such modules, therefore, is continually expanding. Representative of such packaged units are: (1) specialized iconic cursors which can be used as meaningful tracking symbols; (2) directory windows which permit graphical user reference to any file of a given type; (3) a graphical potentiometer with a "knob" which the user can "slide" up and down to change the value of a program parameter; (4) a graphical piano keyboard which audibly plays the notes pointed to; (5) routines to manipulate lightbuttons within the above menu-driven system (e.g, to flash, deactivate and highlight individual menu items); and (6) an audio support package, which drives a digital sound synthesizer (Buxton, Fogels, Fedorkow, Sasaki & Smith, 1978). Many of these modules will be illustrated in examples which follow.

2. UNIX is a trademark of Bell Laboratories.

2. EVOLUTION OF OUR UIMS

Our UIMS has evolved over several years, in parallel with the development of innovative software for graphics applications. It is implemented in the C programming language on a PDP-11/45, which was acquired in 1975 and runs UNIX version 6.

First came GPAC, a device-independent graphics package (Reeves, 1975), onto which successively higher levels were built. Interaction techniques, specialized cursors and directory windows followed, in 1976-78. As part of the Structure Sound Synthesis Project, the table-driven run-time support package of the UIMS was put together in 1979. Finally, in 1982, MENULAY, the graphical menu layout and function specification preprocessor was built.

2.1 Table-driven run-time support system

With the development of a number of menu-based interactive programs in the late 1970's, it became apparent that the user interfaces all had a large number of common features, but that each programmer implemented them with a "personal" touch. So personal, in fact, that the code was difficult for others to understand or support. Accordingly, we began to develop a more uniform methodology which not only captured the best ideas of these various approaches, but also made them available in a fully-documented and well-supported environment. The basis of our new methodology was an externally-controlled, table-driven system which supported event-driven interactions. The programmer supplied the system with the applications-specific routines and information about what types of events were active when, and what and how such events were to affect the program's behaviour. At run-time, then, responses to user interactions (such as event and hit detection, display updates, and procedure invocation) were handled by this package.

The present package, an extension of that developed in 1979, is primarily oriented towards screen and menu-based systems. It is capable of supporting event-driven interactions that go well beyond simple light-button selection (examples are: Buxton, Sniderman, Reeves, Patel & Baecker, 1978; and Buxton, Patel, Reeves & Baecker, 1982). The tables used by the system have nine user-specified fields per entry. These fields include:

the name of a user-specified procedure which is to be invoked upon the detection of a specified input event;

the input event which is to trigger that procedure;

fields to contain (variable) parameters of that procedure;

the x and y hit area for procedures triggered by pointing events;

any text (or the name of a graphics file containing a picture) to be displayed;

the x and y co-ordinates of the item to be displayed;

the item's size and colour.

The system also supports dummy table entries for setting various parameters, such as whether subsequent hit-areas are to have boxes drawn around them or not. As well, it provides for functions to be invoked in special cases arising in menu-based systems: (1) if the user misses all of the light-buttons, (2) when entering or (3) when exiting from the menu.

Finally,the package provides easy-to-use routines for setting-up and switching among the menus, or "states" of the application-defined system.

While this package was of value, it was still a tedious task to specify the data which made up the tables. This was especially true during the early stages of development, when the graphic design, functionality, and syntax of the

user interface were continually being revised due to experimentation. Consequently, we were motivated to develop another tool which would facilitate the specification and modification of these data. The result was the preprocessor, MENULAY, described in the next section.

3. MENULAY AND MAKEMENU

3.1 Concept

The preprocessor which serves as the front end of our UIMS is known as MENULAY. The package is designed to enable the user interface designer (who is not necessarily the applications programmer) to specify rapidly and naturally the graphical and functional relationships within and among the displays making up a menu-based system. Specifications made using the package are converted into the C programming language and compiled through the use of a companion program MAKEMENU. The resulting code can be linked with application-specific routines.

MENULAY enables the designer to define user interfaces which are made up of networks of menus. These may be structured in a hierarchic manner, or in an arbitrary fashion. Furthermore, the method of interacting with these menus is open, and up to the designer. A prime objective of the tool is to minimize the bias imposed by the path of least resistance, which may favour one interaction technique over another. MENULAY is a product of itself. It therefore gives the user interface designer a feel for the nature of the interaction sequences being specified, while at the same time indicates the range of tools available.

3.2 Functional Flow

The entire sequence as set out in Figure 1 can be performed by the

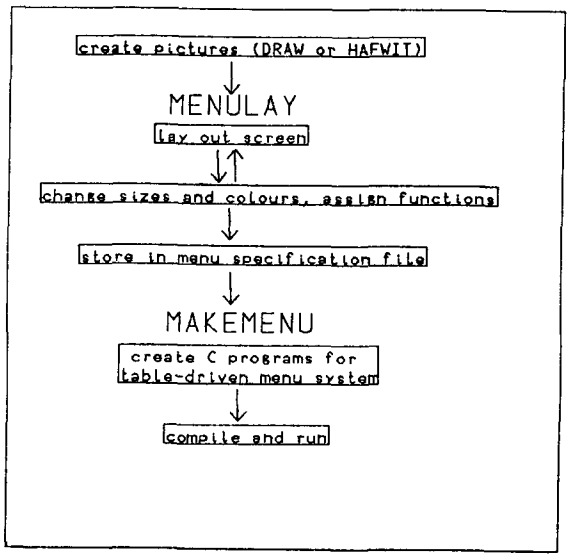

Figure 1. A typical Sequence for Constructing an Interactive Dialogue

applications designer in an interactive graphics environment. A graphics tablet and four-button puck are used for all input to MENULAY (except for such character-oriented input as the typing of the names of application-specific functions to be called upon hit detection). The MAKEMENU program creation and compilation options are also specified graphically.

3.3 Automatic Code Generation

MENULAY's elegance lies in the automatic conversion into C language programs of the graphical interface specified by the applications designer. This conversion is performed by MENULAY's companion program, MAKEMENU. The programs generated from MENULAY's compact specification files are syntactically correct, complete and internally documented with liberal comments. They are designed for compilation with the same table-driven support system on which MENULAY itself is built. Because these programs are in C, however, they can be adapted when required by using a text editor. Where necessary, these changes can be recorded automatically, and repeated whenever the menu specification files are changed[3].

Where the applications designer specifies names of application-specific functions, the programs generated by MAKEMENU contain unresolved external references ("hooks"). By writing functions with the required names and referencing the appropriate file names when MAKEMENU is called, the applications programmer can add any amount of application-specific programming to the layout and sequence information specified by the designer. The two sets of code (computer-generated and programmer-authored) are completely compatible, and can share global variable names, external function references, and so on.

Work is presently underway on a decompiler which will be able to reverse the MAKEMENU process. This decompiler, called UNEMEKAM, will convert a C program which makes use of the table-driven menu software into a menu specification file which can be edited graphically using MENULAY.

3.4 Command Hierarchy

MENULAY has one main command level, containing seven basic commands. These are *layout*, *size*, *colour*, *function*, *get save*, *tryout* and *exit* These are illustrated in Figure 2.

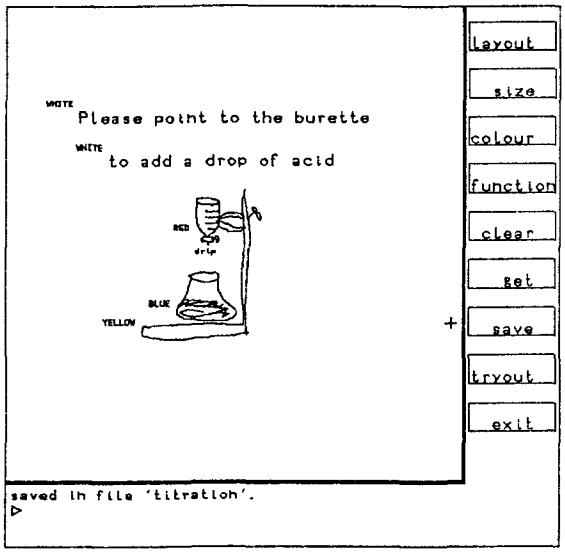

Figure 2. MENULAY Commands

Layout allows the designer to create textual items (e.g., light buttons) on the screen; add pictures either taken from the system library of graphical icons or created by the user with a sketch editor; change the position of any item; and delete items from the screen (see Figure 3).

Size enables the designer to change the scale of any item on the screen, either by "sliding the knob" on a graphical potentiometer or by typing in a scale factor, as shown in

3. Hand-coding changes to MAKEMENU output obviously causes problems in "unmaking" menus. Using the facility is a concession to reality: the state-of-the-art does not yet permit us to make a totally comprehensive UIMS.

Figure 4.

Colour allows one to specify (or change) the colour of any item on the end-user's screen (see Figure 5). This is independent of the device on which the designer is using MENULAY; the high-resolution display we most commonly use, for example, does not support colour graphics.

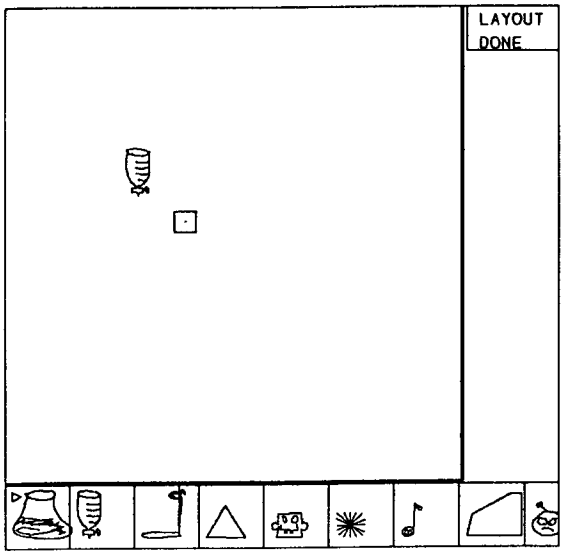

Figure 3. MENULAY - Laying Out the Display Graphics

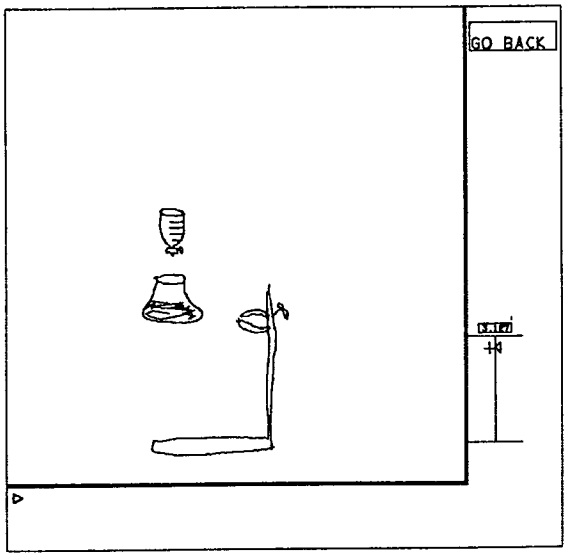

Figure 4. MENULAY - Changing the Size of an Item (the stand) with a Graphical Potentiometer

Function enables the designer to specify what will happen when the end-end-user points to light buttons. The light buttons may be text or pictures. The designer may also specify a function to be invoked if the user activates an input event without pointing to any one light button. In addition, it is possible to specify functions to be called when the menu being designed is entered and exited. This is shown in Figure 6.

The function names may be taken from MENULAY's library of utility functions or be written by the applications

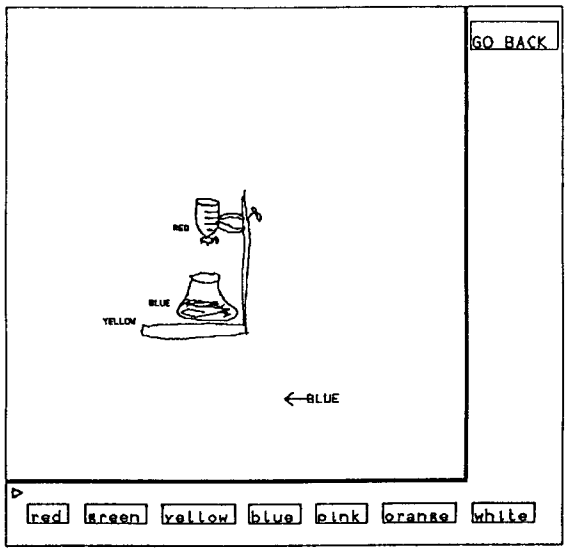

Figure 5. MENULAY - Assigning Colours to the Items on the Screen

programmer before compilation of the program. The writing of these functions is done entirely independently of (and either before or after) the creation of the menu specification file with MENULAY.

Function also allows programs of up to 50 characters - such as short *print* statements - to be typed directly into MENULAY. MAKEMENU takes care of creating a new function name for the function table to enable the code typed in by the user to be loaded.

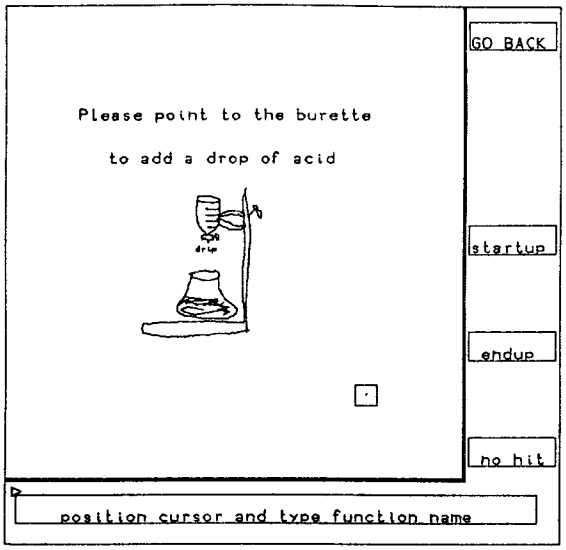

Figure 6. MENULAY - Assigning Function Names to the Active Items (Light-Buttons)

Get (and *save*) allow the user to retrieve from (and store into) a menu specification file the details of the layout and functional relationships which the designer has specified. The menu specification file is extremely compact and is thus a very efficient storage format. Each item of the menu screen is referenced by x and y co-

ordinates, hit area, size, colour, function, text (if a textual item) or standard graphics format file name (if a graphical icon).

Tryout gives the user the opportunity to invoke *makemenu*, the code generation program, by specifying graphically the options and files he wishes to access (see Figure 7).

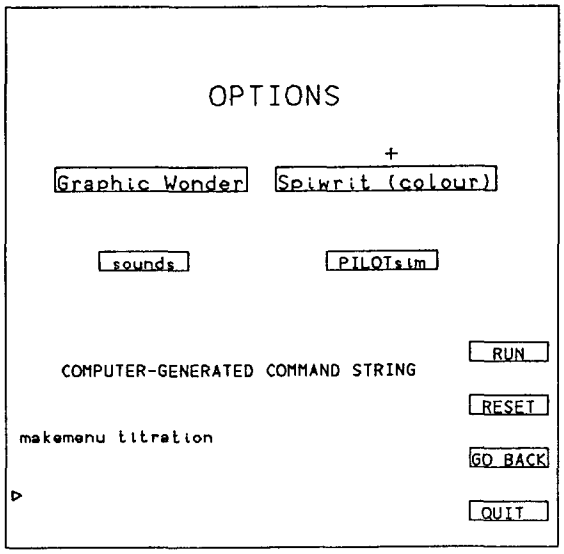

Figure 7. MENULAY - Generating 'C' Code to Implement Specified Interface (by Invoking MAKEMENU)

Once the program has been compiled, *tryout* then runs it for the user.

Exit is used to exit the program. The user is given a new menu with "YES" and "NO" options to confirm that he does indeed wish to exit.

MENULAY has levels of use: "novice" and "expert". In *novice* mode, only the basic commands (as set out above) are accessible. All input is done through one button on the cursor puck, and instructions are given at various points in the program. In *expert* mode, the other three buttons on the puck can be used to perform special functions (e.g., displaying a grid while positioning items; flipping from one menus of pictures or colours to another; assigning a function name which is identical to the text in the menu item), and fewer instructional diagnostics are displayed. The system maintains on disk a profile for each user (a file called "userpro"), initially tagging each person as "novice" and upgrading them based on experience with the system and on specific request.[4]

From any point within MENULAY (except during *layout*, where typing text causes the creation of light buttons containing the text typed), the user can access the UNIX shell (command interpreter) by typing at the terminal. (When this happens, the MENULAY screen fades and the user's scroller is reset to the full screen until the user returns to MENULAY.) This means that the full range of terminal-based commands can be accessed instantly without leaving MENULAY. For example, any calculations which the designer wishes to make in the course of the graphical layout specification can be made by invoking the on-line desk calculator. Similarly, the applications designer who is also a programmer may compose a function by invoking the text editor. Or the designer may send the programmer comments about the implementation by electronic mail before there's a chance to forget them.

3.5 Applications

MENULAY has already been used to construct the user interface for each of the following programs: (1) MENULAY itself; (2) the DRAW sketch editor, which is used to generate light-button graphics; (3) a computer-assisted instruction (CAI) program which teaches children about birds and their nests[5] (see Figure 8);

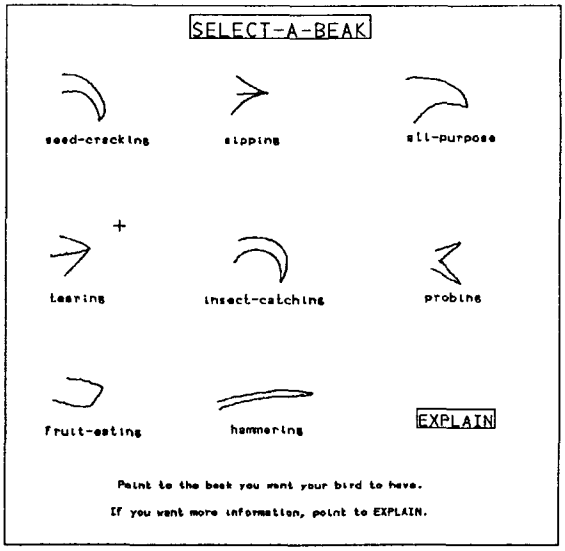

Figure 8. Beak Selection Sequence from 'Land Birds and Their Nests'

and (4) a graphical piano keyboard and musical notation editor (Kuzmich, in preparation: see Figure 9).

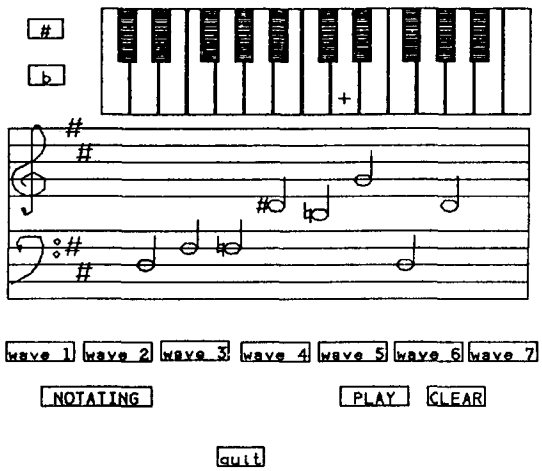

Figure 9. 'Melody Manipulations' music notation editor and teaching tool

It can be used to assemble, in minutes, sequences of frames for CAI in virtually any area of instruction where graphics is helpful. It has also proven useful for laying out

5. "Land Birds and Their Nests", designed by Young Naturalist Foundation, Toronto, Canada, and programmed by CSRG (in press).

figures such as Figure 1 of this paper. Appendix I presents a walk-through of a CAI example.

It is notable that since graphical icons for the user's screen are referenced by file name, these pictures can be changed without even having to recompile the applications program. Thus, a program could be fully constructed by an applications designer who is not a graphic artist, and an artist brought in later to revise the pictures.

4. UNIFORMITY AND PORTABILITY

It is important that a UIMS be portable in a number of senses. The system described satisfies these criteria theoretically and we are in the process of proving its portability in practice.

4.1 Output Device Independence

At the most primitive level, a user interface management system must support a number of different output devices with different characteristics. In the case of MENULAY and MAKEMENU, for example, the device independence is achieved by using the GPAC graphics package with such varied devices as a high-resolution vector display with 16 intensity levels and a low-resolution 16-colour raster display. The run-time support package (without MENULAY as of yet) has also been made to run on various alphanumeric terminals.

4.2 Input Device Independence

The UIMS must also be able to support alternative input techniques. The primary input device used in MENULAY at present is a graphics tablet with a four-button puck and the typewriter keyboard. A set of Allison sliders[6] are also used, but are only available through pre-programmed packages. Other pointing devices (such as mice, light-pens, or touch-screens) could be used in place of the tablet with the provision of the appropriate GPAC device driver. The run-time support package, however, can be driven by virtually any event-generating device that has a GPAC driver.

4.3 Language Independence

At a further level of portability, the UIMS should be structured to facilitate the ability to generate code in different programming languages. The output of MENULAY is a metacode which is translated into high-level language by MAKEMENU. To output code in a different language would involve rewriting this program and providing run-time support in a compatible format.

4.4 Machine Independence

At a higher degree of portability is the capacity to transfer a system such as MENULAY either to a more powerful machine than the PDP-11/45 (e.g., a VAX-11/780) or a less powerful one (e.g. an APPLE microcomputer).

As has been noted, MENULAY and MAKEMENU are written in C (a standardized language which is relatively portable) and use GPAC, a device-independent graphics package (which is itself written in C). Provided the necessary hardware drivers are available, GPAC and thus MENULAY/MAKEMENU could be transferred at reasonable cost to any system which will support UNIX and C.

4.5 Applications Program Portability

In contrast with MENULAY, the applications programs generated by MAKEMENU can be ported even to systems which do not support UNIX. With a cross-compiler and a basic graphics package, for example, applications programs such as computer-assisted instruction frames could be compiled to run on many microcomputers.

6. This device is a continuous belt slider. It is a treadmill with a 9 by 1.5 cm surface exposed which is used as a motion-sensitive input device. The mechanical section was developed by Allison Research Inc., 2817 Erica Place, Nashville, Tenn. 37204. The electronics used here were developed in house.

5. COMPARISON WITH OTHER SYSTEMS

Other UIMSs do exist, and have had an influence on the evolution of our system. The most distinguishing feature of MENULAY is its natural way of integrating graphical design specification with human-written applications programming. By way of comparison, we review briefly three systems: TRW's FLAIR; Olsen's automatic code generation design; and Kasik's TIGER.

5.1 FLAIR

FLAIR (Functional Language Articulated Interactive Resource) (Wong & Reid, 1982), is a user interface dialogue design tool which enables a system designer to construct graphically a user dialogue for an applications program. It is largely driven by voice input and incorporates text picture construction and editing (at the graphical primitive level) as well as dynamic frame layout. Its high-level features include the ability to define and control a menu hierarchy, graph and map generation, an on-line calculator and relational data base access for graphical entity storage and retrieval.

FLAIR is more advanced than MENULAY in its use of multiple input techniques and in its ability to permit the applications designer to specify a wide range of end user interactions. However, we are in the process of extending MENULAY's capabilities to permit the specification of a much wider range of user interactions.

FLAIR contains a powerful set of internal utilities, but appears to be rather limiting in its integration with application-specific code. FLAIR is a language and package unto itself, with no apparent "hooks" into other programming languages. This suggests that if the FLAIR "language" does not permit the applications programmer to program a certain algorithm conveniently, then that algorithm will be inaccessible. MENULAY, on the other hand, creates menu specification files that are converted into fully-documented C programs which, as noted earlier, are automatically integrated with any code the programmer may have written for the specific application.

5.2 Olsen's Model for Automatic Code Generation

Olsen (1982) describes research into the automatic generation of interactive graphical systems to facilitate faster and cheaper generation of interactive user interfaces. This work has not yet progressed beyond the design stage.

Olsen points out the useful distinction between these *design* of the application program interface and thes *writing* of the program itself. He observes that it is the design aspect of the program creation which is suited to automatic program generation. This is because of the high cost in time and effort of hand-coding and the increased reliability of automatically generated software.

Olsen envisages the use of Pascal procedure definitions for the characterization of interactive commands in the applications program. We feel that MENULAY is a significant improvement over this idea in that the command menus and interaction relationships are specified in the very way in which the end user will interact with the applications program, i.e. by pointing. Olsen does not address the possibility of having the specification technique use the same devices and interfaces as those the end-user will ultimately face.

5.3 TIGER

Kasik (1982), describes a UIMS which, like our UIMS, takes care of the bookkeeping associated with screen layout, interrupt handling and the definition of interactive dialogue sequences. This UIMS, called TIGER, has as its core the language TICCL, which permits the applications programmer to concentrate on the logical functions which he wishes to perform rather than the physical, low-level steps which must be taken to accomplish the task.

TICCL can be used to describe algorithms which combine graphical primitives in response to user interactions as well as to define user interaction sequences. TICCL code operates at a higher level than the Pascal code which is used for the non-graphical portion of the applications programming.

TICCL is useful as a mechanism for specifying user interaction at a higher level than is otherwise available to its designers. Such a language combined with a higher level package such as MENULAY would permit even more flexibility in user interface prototyping. While TIGER does not currently incorporate a module comparable to MENULAY, TICCL is a powerful language, and could support such a tool.

To the extent TICCL is used for constructing graphical primitives from user interactions, it is more advanced than our table-driven menu system for which MENULAY acts as preprocesor. On the other hand, we feel that our programmers' packaged units, together with the flexibility of GPAC and its integration into programmers' C code, provide a useful alternative set of interaction response tools.

6. FUTURE DIRECTIONS

6.1 Protocol Analysis

The recording of sequential data about end-user interactions is essential to the evaluation of the interaction techniques used in an applications program. With a menu-driven system based on cursor and tablet, this data consists of a time-stamped record of each user input, recording the x and y tablet co-ordinates of the cursor and the input event which was activated.

We are in the process of developing tools for the analysis of this data, stored in a so-called "dribble file". As part of the process of developing the interaction sequences for an applications program, the designer will ask an end-user to spend a session with the program. Afterwards the new tools will facilitate the analysis of the "dribble file" for that session in a number of ways. First, they will allow the designer to "play back" the user interactions in real time, so as to get a feel for the flow of the user-computer dialogue. Second, they will draw for the designer a "spiderweb" which superimposes graphically all of the hand motions of the user in his interaction with the program (see Figure 10). The spiderweb makes it easy to spot the

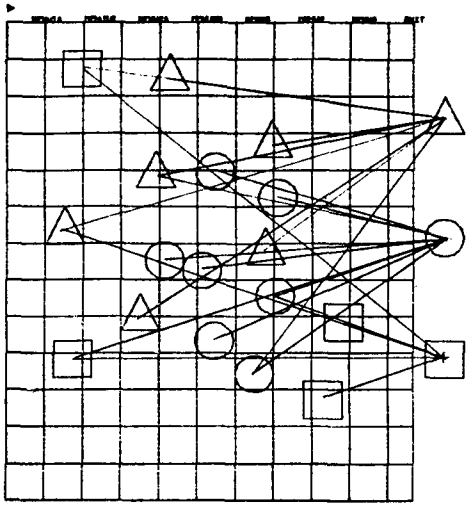

Figure 10. Tracing of Coordinates of Events Recording a User's Session

points at which the user is being forced to repeatedly make hand motions that are uncomfortable or excessively long.

To function properly, the dribble file data must be recorded at every user input. This implies that the recording be done at the level of event recognition, and that the information always be available unless the programmer has specified otherwise. To do so, however, requires special support in the event-detection mechanism of the graphics package used. Furthermore, the event recognition routines themselves should be able to function in a mode which permits input from sources other than the physical input devices. The stored data files should be able to be used as the source of input events.

6.2 Window Management

MENULAY presently generates code which operates within a single window on the screen. While multiple *frames* or *levels* within a program can be created very easily, multiple *windows* can not.

It is intended to expand the capabilities of the table-driven menu system (and therefore of MENULAY) to permit the designation of multiple windows by the applications designer. Windows would have attributed to them specific cursor tracking symbols, background colours, and input events. As at present, specific light buttons (text or graphical) will be locatable at any place within a particular window.

7. ACKNOWLEDGEMENTS

MENULAY and MAKEMENU are the highest level of a user interface management system which is built on years of work at the Computer Systems Research Group. We acknowledge with thanks the contributions of Ron Baecker and Leslie Mezel, former directors of the Dynamic Graphics Project; Bill Reeves, author of the GPAC graphics package; and other major contributors to our inventory of graphical tools and techniques: Tom Duff, Greg Hill, Tom Horsley, Sanand Patel, Rob Pike, David Tilbrook, Mike Tilson and Martin Tuori.

We also gratefully acknowledge the helpful comments made by the referees and by Dave Kasik.

Interactive graphics research at the Computer Systems Research Group has been funded for many years by the National Sciences and Engineering Research Council, and more recently by the Social Sciences and Humanities Research Council.

8. REFERENCES

Buxton, W., Fogels, A., Fedorkow, G., Sasaki, L. & Smith, K. C. (1978). An Introduction to the SSSP Digital Synthesizer. *Computer Music Journal* **2(4)**, 28 - 38.

Buxton, W., Patel, S., Reeves, W. & Baecker, R. (1982). Objed and the Design of Timbral Resources. *Computer Music Journal* **6(2)**, 32 - 44.

Buxton, W., Sniderman, R., Reeves, W., Patel, S. & Baecker, R. (1978). The Evolution of the SSSP Score Editing Tools. *Computer Music Journal* **3(4)**, 14 - 25.

Buxton, W. & Sniderman, R. (1980). Iteration and the Design of the Human-Computer Interface. *Proceedings of the 13th Annual Meeting of the Human Factors Association of Canada*, pp 72 - 81.

Deutsh, L & Taft, E. A. (1980). Requirements for an Experimental Programming Environment. *Technical Report CSL-80-10*, XEROX PARC.

Kasik, D. (1982). A User Interface Management System. *Computer Graphics*, **16(3)**, 99 - 106.

Kuzmich, N. (in preparation). Melody Manipulations. Music

Dept., Faculty of Education, University of Toronto.

Olson, D. (1983). Automatic Generation of Interactive Systems, *Computer Graphics* **17(1)**, 53 - 57.

Reeves, W. (1975). *A Device-Independent Interactive Graphics Package* M.Sc. Thesis, Dept. of Computer Science, University of Toronto.

Swartout, W & Balzer, R. (1982). An Inevitable Intertwining of Specification and Implementation. *Communications of the ACM* **25(7)**, 438 - 440.

Wong, Peter C.S., and Eric R. Reid (1982). FLAIR - User Interface Dialog Design Tool, *Computer Graphics*, **16(3)**, 87 - 98.

APPENDIX 1: A Walkthrough of a CAI application

The following is a brief account of an applications designer's use of MENULAY to create a lesson to help teach chemistry titration. The time taken in this instance was less than ten minutes.

The designer begins by typing "draw" to invoke the DRAW program and then uses the graphics tablet to input freehand pictures of a burette, a beaker and a stand. Each picture is scaled down in size by pointing to the command "SIZE" and then sliding the "knob" on the displayed potentiometer (like that in Figure 4). The "knob" is slid by positioning the cursor over it and holding down the main button on the cursor puck while sliding it up or down. Each picture is stored in a disk file (by pointing to the command "SAVE", and typing in or pointing to the file name).

Next, MENULAY is invoked (by selecting "EXIT" and then "MENULAY" from a new menu), whereupon an explanation is displayed together with MENULAY's command menu. The user selects "LAYOUT" and sees the newly created pictures in a menu at the bottom of the screen. Selection of the burette causes a copy of it to be tracked as the cursor. It is anchored in the work area by pressing or releasing the main button on the cursor puck (see Figure 3). The same is done to the beaker and the stand. Typing at the keyboard causes that text to be displayed at the current cursor position. Any item in the work area (whether text or graphics) can be repositioned by pointing to it and dragging it to a new position, again anchoring it either by releasing the button or, if it was released immediately upon pointing, by pressing the button again.

To change the scale of any item, the user selects "SIZE" in the main menu (displayed in Figure 2), selects the item, and then changes its size, again with a graphical potentiometer (see Figure 4). To set or change the colour of any item, the user selects "COLOUR" in the main menu and chooses a tint from the menu at the bottom of the screen. This tint is tracked (as shown in Figure 5) until another tint is selected. Any items pointed to are assigned the currently tracked colour. The colour of each item is displayed next to the item if the hardware device does not support colour graphics.

The designer specifies that the function named *drip* is to be invoked when the end-user points to the burette (by selecting "FUNCTION", selecting the burette, and typing "drip": see Figure 6). The entire set of interface specifications is now stored in a file (by pointing to "SAVE" and typing a file name.

Finally, the user selects "TRYOUT" and then chooses Spiwrit (a colour raster screen) as the display device (see Figure 7) and references "drip.c", an application-specific C source file. This causes the interaction specifications to be automatically converted into C language programs which are compiled and linked with the application-specific code. The resulting binary file is then executed (see Figure 11).

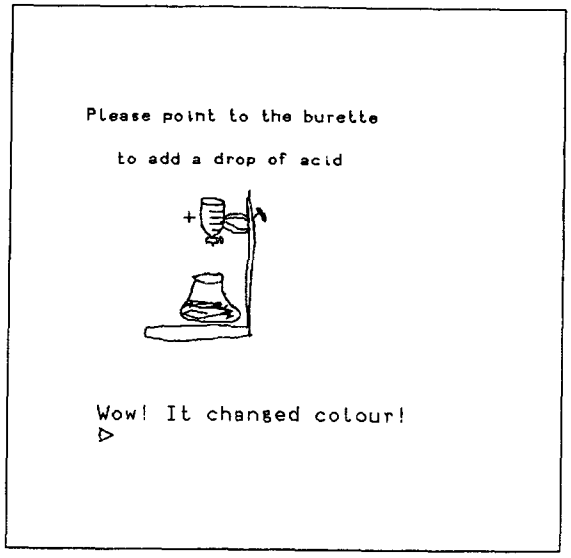

Figure 11. Running the Titration Simulation

Here is the source code for the *drip* routine - the only code which the programmer had to write:

```
#include "/u0/dave/master/menuglobal.h"
#define beaker picture("beaker");
int drops 0;

drip()
{
        type("Added a drop of liquid...");
        sound(DRIPPING);
        drops = drops + 1;

        if(drops == 3)
        {
                type("Wow! It changed colour!");
                resetcolour(beaker, PINK);
                sound(BUZZING);
        }
}
```