MFE: A Syntax Directed Editor for Interaction Specification† Summary Paper

Robert V. Rubin
Joseph N. Pato

Department of Computer Science
Brown University
Providence, RI 02912

ABSTRACT

In this paper we present MFE, an interface interaction generator for event driven, menu based programs. MFE may be thought of as the analogue of a wysiwyg (what you see is what you get) syntax directed editor for MAPLE, a menu application language developed at Brown University. MFE generates all the source code necessary for the specification of the user interface. The prototype system was built using many of the components of the Brown Workstation Environment.

KEYWORDS: user interface management, graphical programming, interaction techniques

1. Introduction

MFE is a prototype interactive system that generates the code necessary for the specification of a program's user interface. In its current implementation, MFE runs under and generates code for the Brown Workstation Environment,[1] a UNIX‡ based bitmap screen oriented software environment. Much recent research has been directed at the definition of user interaction languages,[2, 3] and it has been our experience using one such language, MAPLE,[4] that these languages are as yet unstable and are undergoing many refinements. In MFE we have provided a system that insulates application programmers from learning the syntax and explicit semantics of the underlying interaction specification language.

MFE provides for the editing of menu interactions by providing operations for inserting and deleting menus, and their buttons. It also provides a syntax directed facility for describing the characteristics of the menus, as well as the actions that may occur when a user interaction occurs.

We will briefly describe the capabilities provided by the MAPLE package and its interpreter, and then describe MFE.

2. MAPLE

MAPLE is a subroutine package that may be thought of as a user interface specification language and interpreter for providing menu-oriented finite-state program control. The application program can define a menu to consist of a set of buttons, to be either pop-up or anchored within a part of a window, and to be displayed in a variety of fonts. The MAPLE interpreter automatically handles tracking the locator device, highlights buttons when the cursor is on top of them, and may invoke a programmer specified action routine when the user selects a button. This action routine may use MAPLE specifications to request additional input, i.e., a point on the screen, a string of text, a box, or a choice from a list of alternatives. After a button is depressed, the MAPLE run-time support automatically generates the following state of the interface, which may involve the setting up of a different set of menus, an automatic menu of sub-choices for that button, or leaving the original state intact.

MAPLE uses a modified external control model.[5] After initialization, the locus of control for an application resides within the MAPLE interpreter. Each application module is viewed as a discrete functional unit, available to be called by the MAPLE interpreter when the appropriate input activity has occurred.

† This research was supported in part by National Science Foundation grants MCS-7905992, MCS-8200670, SER-8004974 and MCS-8121806, by the Office of Naval Research under contract nos. N00014-78-C-0396 and N00014-83-K-0146 and by DARPA order n. 4786.

‡ UNIX is a trademark of Bell Laboratories.

3. MFE -- The MAPLE Front End

The user-interface designer, be it the programmer, or the user, uses MFE to specify the menu interface. At the end of a session, MFE will generate a file containinge C code and declarations, as well as the calls to MAPLE and ASH (the Brown University Window Manager).[6] This code need only be compiled to be executed. Executing the code generated by a session with MFE allows the user to walk through the top level interface, all without the programmer having written a line of code. The programmer may then edit the code, using the action routines invoked by the depression of a menu button as the entry points into his modules.

3.1. Interactive Dialogue Specification

MFE is a syntax directed, template editor, which allows a user unfamiliar with MAPLE syntax to specify an interactive user interface. The user designs his menus, and their interactions, in a what you see is what you get environment. Menu layouts, button location, typefaces, and window layout are modifiable via editing commands.

MFE requires a user to specify only the interactions. Because all code is automatically generated, a knowledge of MAPLE, or of C, is not required to generate the source code for the user interface.

MFE is a non-preemptive editor, allowing the user to incompletely specify the characteristics of menus and buttons, and return at a later time to fill them in.

The user interaction paradigm for MFE itself is menu oriented. All modifiable parameters, and commands, are invoked as the result of a button depression.

3.2. Editing the User Interface with MFE

As illustrated by the figures in the appendix, the MFE user is presented with three windows of editing commands. One window is for menu creation, one is for for old menu modification, and one is for button manipulation.

The first set of editing commands are used for setting up a menu in a window. The user selects the layout attributes of the menu, specifies a name for the The menu "Edit-Button" is popped on top of "new button". The user is then free to select from any of the buttons on the menu. In this case, the user has decided to name the button "Box", and assign to its input request type, a box request. Additional selections include the name of the routine to be invoked when the button is depressed, and the prompt to be displayed upon depression of the button.

The Appendix includes the source code generated for this trivial example. Inspection of the code shows that MFE has generated all the declarations and parameters necessary, including those left incomplete, so that after compilation, the user interface may be succesfully executed and experimented with. The action routines include the code necessary to obtain any of the input requested as a result of a button depression

4. Limitations and Futures

MFE has been used for prototyping user interfaces, both at the application program level, and at the operating system command level. Within the limits of a prototype system it has served us well; However as a production tool it has its limitations.

MFE does not allow a user to end a session, and then modify the user interface generated. If modifications are necessary, they must be made either via a conventional editor operating on the generated source code, or by reentering the data into MFE.

MFE provides no encapsulating facilities, that might allow a designer to experiment with menu interactions. All execution is done via the compiled modules.

The grahical layout facilities in MFE are limited. While they allow for window definitions, they do not allow a user to completely layout the screen. This deficiency is caused in part by the MFE editing menus themselves. A window layout facility supporting pan and zoom operations is highly desireable. The layout facility should support menu design within a window, as well as the necessary operation to do full screen layout in a what you see is what you get environment.

menu, and then specifies the location of the menu within a window.

The first set of figures in the Appendix shows the screen after we defining a menu named "SHAPES".

The only button in the menu "SHAPES" is the button "new button". Depression of this button results in a pop-up menu for editing the characteristics of the button.

Their are two sets of editing commands for the buttons in the menu being designed. The "Button Layout" menu in the top part of the screen is used for moving buttons between different user designed menus. The "Edit Button" menu is a pop-up menu which is invoked by the depression of any of the buttons on the menu being developed, including the "new button" button. One depresses a button on the menu being designed to specify the characteristics of that button. The Edit Button menu allows for the nonprocedural specification of the interactions involving a user defined button for a menu. This is essentially a MAPLE specification and may involve five seperate components.

The five components of a button may be specified as follows:

o The button may be assigned a name.

o Buttons are then assigned a type. Button types indicate whether depression of the button will result in a query for text, integers, a box, or any legal type as defined by MAPLE.

o The user may choose to associate a prompt with a button.

o A link to other programs is established by assigning an action routine to be invoked upon depression of the button.

o Transformations to the current menu may be specified. The menu transformations specify whether the current menu should be replaced by another menu, and the manner in which it should be replaced. Their is a stack paradigm in use here, i.e. menus may be reordered via push or pop operations, or any combination of stack operations.

In the example in the Appendix the user has selected the "new button" in the menu "SHAPES".

5. Conclusions

MFE represents a step in the development of a truly interactive graphical programming environment. While limited in the above mentioned respects, it does achieve its primary goals.

MFE demonstrates that the specification of user interfaces may be completely decoupled from the specification of the application program. It does this on two levels:

o MFE demonstrates that the programming of user interfaces may be succesfully accomplished in an environment that is natural to a user-interface team, and not just to one familiar with the syntax of the interaction language.

o MFE frees the programmer from learning the syntax and semantics of yet another language, or subroutine package, for interactive menu specification. The syntax of interaction languages is bound to change as a result of further experimentation and refinement. Interaction editors like MFE provide a vehicle by which programmers may be insulated from these changes.

6. Future Research

MFE was written before many of the subcomponents of the Brown environment were complete. We hope to extend the paradigms presented by MFE with an eye towards providing true graphical programming capabilities within the Brown Program Development System.[7]

References

1. Joseph N. Pato, Steven P. Reiss, and Marc H. Brown, "The Brown Workstation Environment," Brown University TR84-03 (October 1983).

2. Mary Shaw, Ellen Borison, Michael Horowitz, Tom Lane, David Nichols, and Randy Pausch, "Descartes: A Programming-Language Approach to Interactive Display Interfaces," *SIGPLAN Proceedings on Programming Language Issues in Software Systems* **18**(6)(June 1983).

3. Dan R. Olsen Jr. and Elizabeth P. Dempsey, "SYNGRAPH: A Graphical User Interface

Operator," *Computer Graphics* **17**(3)(July 1983).

4. Marc H. Brown and Steven P. Reiss, "MAPLE Reference Manual," Brown University (December 1982).
5. James J. Thomas, chairman, "Graphical Input Interaction Technique (GIIT) Workshop Summary," ACM/SIGGRPAH (June 1982).
6. Steven P. Reiss, "A Screen Handler," Brown University (November 1982).
7. Steven P. Reiss, "PECAN: Program development systems that support multiple views," Brown University (1983).

```
Tue Feb 28 14:21:59 1984        box.d   Page 1


/*****************************************************************************/
/*        include files                                                      */
/*****************************************************************************/


#include        <stdio.h>
#include        "/pro/include/ash.h"
#include        "/pro/include/maple.h"


/*****************************************************************************/
/*        definitions                                                        */
/*****************************************************************************/


typedef char * String;
#define SALLOC(str) (((str) == NULL) ? NULL : \
  ((String) strcpy(malloc(strlen(str)+1),str)))

/*****************************************************************************/
/*        external routines                                                  */
/*****************************************************************************/


int     box_action();


/*****************************************************************************/
/*        MENUS                                                              */
/*****************************************************************************/


menu_init () {
/*****************************************************************************/
/*        SHAPES                                                             */
/*****************************************************************************/
MAPLEmenu_init ("SHAPES");
MAPLEbtn("BOX",box_action,MAPLE_STATE_LOOP,"NULL");
MAPLEbtn_boxnew ("BOX", "Click the two ends of the box", 12);
}
```

Figure 5: MFE generated declarations.
The definitions and menu declarations generated by MFE.

```
Tue Feb 28 14:21:59 1984        box.d   Page 2


/*****************************************************************************/
/*        ACTION ROUTINES                                                    */
/*****************************************************************************/



/*****************************************************************************/
/*        box_action                                                         */
/* This routine invoked by depression of button BOX in menu SHAPES           */
/*****************************************************************************/
box_action ()  {
int lx,by,rx,ty;

MAPLEinq_btn_box (&lx,&by,&rx,&ty);
}

/*****************************************************************************/
menu_setup () {
/*****************************************************************************/
    ASH_WINDOW        top, temp;

    top = ASHinq_top ();
    temp = ASHcreate (0,250,0,0,125,250);
    MAPLEbase_sticky("SHAPES",MAPLE_FMT_DEFAULT ,1,temp);

}


/*****************************************************************************/
 main () {
/*****************************************************************************/
for (;;) {
     menu_init ();
     menu_setup ();
     MAPLEnext_user();
     }
   }
```

Figure 6: MFE generated procedures.
The action routines and mainline generated by MFE.

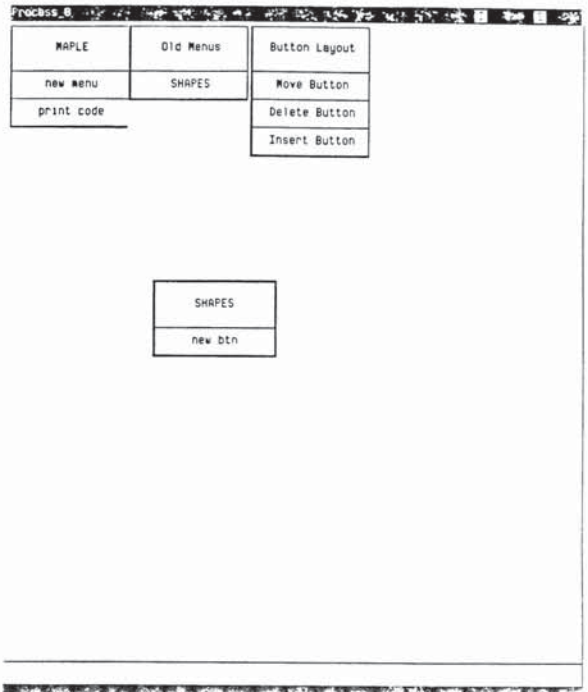

Figure 1:
The menu SHAPES has been defined.

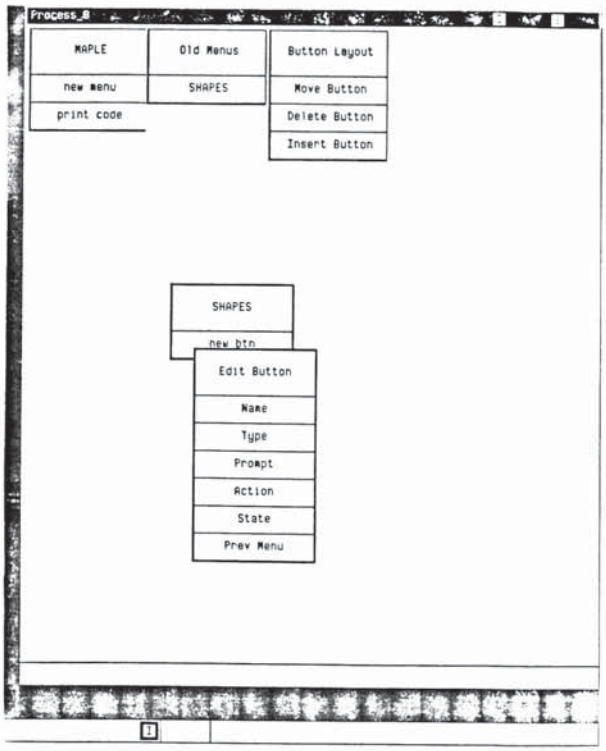

Figure 2:
Depression of the "new btn" button pops up the "Edit Button" menu. Note the button options inside the button.

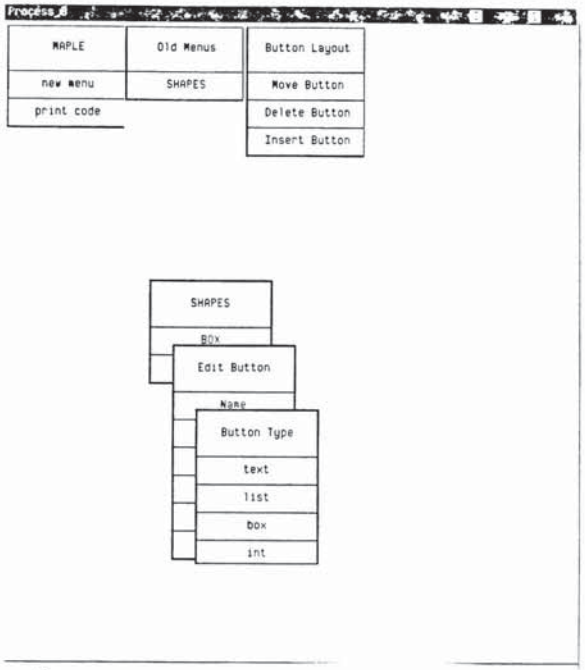

Figure 3:
The user has named the button being edited "BOX". The "Button Type" menu allows use of the MAPLE input facilities for obtaining typed input.

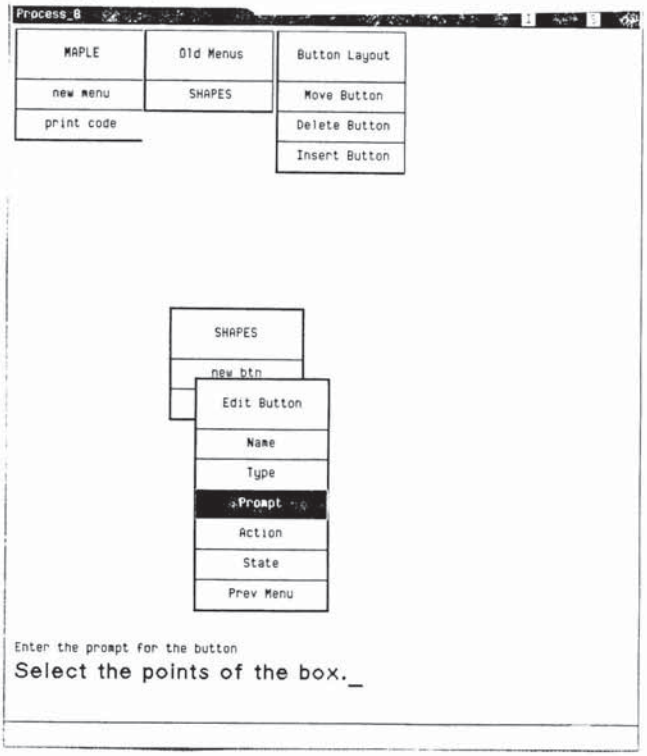

Figure 4:
The interface designer associates a prompt with the depression of the button.