1982

Menu-driven user environment for CAD

Thomas B. Slack
Carnegie Mellon University

Stephen W. Director

A MENU-DRIVEN GRAPHICAL USER ENVIRONMENT FOR CAD

by

Thomas B. Slack & Stephen W. Director

DRC-18-«W-82

April, 1982

A MENU-DRIVEN GRAPHICAL USER ENVIRONMENT FOR CAD[1]

T.B. SLACK AND S. W. DIRECTOR

CARNEGIE-MELLON UNIVERSITY
PITTSBURGH, PENNSYLVANIA

ABSTRACT

For the most part, ptpers which have been published in the CAD area introduce new algorithms and or programs but. in general do not concern themselves with the user-machine environment in which they are to be used. The result is that the circuit designer is often expected to employ a number of different programs, each having a different input language and assuming a different design methodology. This paper describes an approach aimed at creating a more uniform environment for computer-aided design programs.

1. INTRODUCTION

We are interested in developing a uniform means of communication between a circuit designer (called the *user)* and a set of design programs (called *application programs).* In general application programs require some form of input and generate some form of output We define a *user interface* as the program which controls the formal aspects of input and output between an *application program* and the user. Thus, the interface acts as a buffer between the user and the application program. The portion of the interface which decodes the input from the user and passes it to the application program is called the *parser* and the portion of the interface which actually communicates with the user is called the *terminal driver.*

Note that in some instances, there is frequent communication between the application program and the interface (such as during interactive post processing) while in other instances there is infrequent communication between the application program and the interface (such as during a simulation).

After considering some general attributes of a user-machine interface, we describe a general purpose interface called DEULAH which we have developed.

Z USER-MACHINE INTERACTION: GENERAL CONSIDERATIONS

We begin our discussion by considering two different schemes for user-machine interaction: *tutorial questioning,* sometimes called *dialog*,* and *command language.*

In the tutorial questioning method, the interface prompts the user for an instruction, then checks for any errors in the user's response. If an acceptable response has been made, control is passed to the application program for the appropriate action. While this approach allows for a wide degree o^r flexibility for the application programmer, there is an inht-ent lack of standardization amongst different application programs. A major disadvantage of tutorial questioning is that the application program controls the interaction sequence. Furthermore, the parsing operation is distributed between the interface and the application program. Most users, especially expert users would prefer to have more control over the action sequence[1].

In the command language method, information is processed by the interface in whole commands. After reading an acceptable command, the interface relinquishes control to the application program. Most operating systems use a pure command language control structure to interface with the user. For this scheme to be effective, a separate documentation facility must be provided to inform the user of the correct command syntax. An advantage of the command language approach is that the parsing operation can be completely separate from the application program. However, control during parsing is no longer in the hands of the application program.

From the application programmer's point of view, there are advantages to being able to employ both of these methods of interfacing with the user.

Let us now consider the types of responses which might be expected from a user. In general, user responses can be separated into three types: responses which are selected from a small, well-defined set: responses which set values for. or limits on. a variable; and responses which are arbitrary strings.

When the expected response comes from a small well-defined list, the user can be shown the entire set and asked to choose one of them. The majority of user interaction falls into this category. We will refer to this type of response as a *menu response.*

When the expected response is a value, or set of values, the user needs to know the range from which to choose a response. A reasonable default value may also be provided. We will refer to this type of response as. a *range response.*

The third type of response is called a *string response.* As mill be seen, this type of response can usually be handled as either a menu response or range response.

3. THE DELILAH INTERFACE SYSTEM

We now turn to the specifics of the DELILAH user-machine interface. Recall that one of our primary goals is to create an interface which can be employed by a variety

---

'Tkt* worl h»* km [illegible] te part *> tkc Hc*l«fr»ckM« *Oemm*m>* [illegible] IL *Army* ftc*eare»ON [illegible]

~%\•. cftc*ic ttki- aamc beans, ii p.«« aiccl> with i»c acroay* ftAMSON *wku* [illegible] •» • aiixcJ cwcait-tofic level *ummltxm* alw< «»^tap^ M CML'.

of application programs, but which interacts in a consistent manner with the user. In order to nave a uniform means of interaction we have chosen a menu based scheme. More specifically, we have assumed that the principle type response expected from the user is a menu response. (We will discuss range responses in Section 4.) In order to allow the same basic menu scheme to be usabk by a number of different applications programs, we drive the parser by a menu file. This menu file is specific to a given application program and contains a description of the type of user input, and interactive control expected by a given application program. Note that each application program is accompanied by its own menu file.

We hasten to point out here that while we have assumed a basic menu format, the DELILAH system allows the user to enter command lines (a cominand line usually consists of a sequence of items found in one or more menus) via the keyboard. Thus, an expert user can skip menu interaction entirely, if he wishes.

3.1. THE MENU FILE

The menu file contains a description of the type of user input and interactive control expected by the application program. This information is characterized in terms of *menu fists, prompts, help messages,* and *menu nodes* where:

- a *menu fist* is a fast of acceptable user responses and is displayed along the right band side of the screen;
- A *prompt* is a short message, or question, sent to the user when input is desired and displayed at the beginning of the command line:
- a *help messsge n* the information presented to the user when an incoherent response is returned, or upon his request.
- a *menu node* is the association of a menu Hsu a prompt and a help message; a menu node must be defined for every point in the application program where user input is expected.

A *menu definition language* has been developed to formalize this information. We briefly describe this language below. An example of a menu file using this language is shown in Fig, 3-1.

Definition of a menu list consists of:

- The keyword MENU
- A unique identifier which will be used by the menu mode declaration
- A list of menu items
- The keyword END

* Each menu item is an acceptable response for the user to type on the command line or to select from the menu which *h* displayed on the right-hand side of the screen. If the symbol "•" is included at the end of a list of menu hems, then an arbitrary string typed by the user, which is not cootained in the list, will be passed directly to the application program. (Thus we are able to handle a string response). This string is then assumed to be parsed by the application program. Examples of a menu list declaration can be seen in Fig. 3-1. Observe in this example the menu list identified as "linem." has the menu items "Solid." "Dotdash." "Broken." "Dashes." and "Dots."

Sometimes it is necessary to allow the selection of one menu item to cause a new menu list to be displayed. This can be done by associating a menu node identifier with a menu item. We discuss this case after the menu node declaration is defined below.

Definition of a prompt consists of:

- The keyword PROMPT
- Individual prompt declarations, separated by semicolons. Each declaration consists of a unique identifier which will be used by the menu node declaration and the actual string to be presented to the user on the command line, surrounded by single quotes
- The keyword END

Observe the prompt declarations in Fig. 3-1 Actual]} three prompt declarations are made. In the first declaration, the identifier is "changep" and the string sent to the terminal is "Action>".

Definition of a help message consists of:

- The keyword HELP
- An identifier which will be used by the menu node declaration
- The help message, surrounded by single quotes
- The keyword END

Examples of belp messages can abo be found in Fig. 3-1.

The menu node declaration consists of:

- The keyword NODE
- An identifier which can be used in a menu list as discussed below
- A menu list identifier, a prompt identifier and a help message identifier, all separated by semicolons
- The keyword END.

Observe the menu node identified as linen in Fig. 3-1. This decknikm identifies linem. haep. and linen as the menulist, prompt and help message, respectively. Note that menu lists, prompts, and help messages may be used by more than one menu node, if appropriate.

As mentioned above, it is sometimes necessary to have a menu hierarchy. In other words, one or more of the choices in a menu list would need to call forth additional menus. In order to handle such a hierarchy, we allow individual items in a menu list to be associated with a menu node. The appropriate syntax would be:

<menuitem. menunode-ktentifier. n>

where menuitem is a string which is presented as an item in the menu list to the user, menunode identifier indicates

tbc menu node selected if the user makes this choice, and n is used to control the sequence of events after a menu item from the new menu is selected. For purposes of explanation suppose menu node A has an item in its menu hst which calls forth menu node B. If n is omitted or set to 1, after an item from the new list associated with menu node B has been selected, and appropriate action taken by the application program, menu node A regains control, and menulist A is displayed. If n is set to •, once selected, menu node B maintains control until an explicit "reject" string in the menu list associated with menu node B is selected by the user. If n is set to any integer other than 1. menu node B maintains control for the specified number of calls from the application program.

3.X USE OF DELILAH

In order to employ the user interface supported by DELILAH' . the application programmer needs to employ two procedures. *Re»d Menu* and *ReedCommend.* The procedure *ReidMenu* designates the menu file to be used by the particular application program. Normally each application program would need only one menu file. The procedure *ReedCommend* returns the string from the menu list being displayed which is pointed to by the user. If the application programmer requires a range type of responses he would employ a *form node.* This type of node is discussed further in Section 4.

As can be seen from the above discussion, the menu file is closely coupled to the application program. As an example, consider the program segment in Fig. 3-2 and the menu file in Fig. 3-1. In particular, note the relationship betwen the case statements and the menu lists. Furthermore, observe the need for the • in item "modify" for the menu bst identified as topm. Thus, once the modify option is chosen, the modify menu, identified by modifym, is displayed and remains displayed until an explicit reject is chosen.

4. ENHANCMENTS

To this juncture, all that has been discussed above has been implemented. One item which is missing is the form construct If the application program needs a range response, the menu construct as described above is inappropriate. A better construct is that of a *form* which would consist of i bst of prompts, and possibly help messages.

Specifically we define the form by:

- The keyword FORM
- A unique identifier to be used in a menu Bst to invoke the form
- A *form list*
- The keyword END.

' Each Hem 'm the *form list* consists of a prompt mrrounded by quotes, the tetter I F. or S (indicating whether the expected response is an integer constant, floating point number, or a string) an optional default value contained in square brackets, an optional range oif allowable values separated by a colon and contained in angle brackets, an terminated by a semicolon. When a form is invoked, all the prompts and defaults, if any, in the prompt list are displayed on the graphics screen. The cursor is positioned at the end of the first item on the KSL The user can then input the requested item followed by a carriage return. The cursor then will move to the end of the second item in the list This process is repeated until all items are entered. If the displayed default response is desired by the user, he merely has to enter a carriage return.

One case where a form is more suitable than a menu is to enter execution controls for a circuit simulator. For this case, the form might be defined as follows:

```
FORM
•No. of Newton intentions' I[1001<l:1000>:
'Absolute Integration truncation error* Ft.001]:
Relative Integration truncation error' Fl.011:
'Minimum Step Size in nsec* F[.O1)

END
```

At present, the form construct as indicated above is being implemented.

Acknowledgement

The authors would like to thank Michael Bushnell for his review rod constructive criticism of this paper.

REFERENCES

1. Sakailah. Karem A.. *Mixed Simulation of Electronic Integrmed Circuits,* PhD dissertation. Department of Electrical Engineering. Carnegie-Mellon University. November 1981.
2. Mehlmann, Marilyn, *When People Use Computers,* Prentice-Hall New Jersey. 1981.
3. Shneiderman. Ben. -Human Factors Experiments in Designing Interactive Sysems." *Computer,* VoL 12, No. 12PAGES"9-18". December 1979. .

[3] DELILAH is written in PASCAL.

```
PROMPT
        cbangep 'ActionV;
        topp •Command)';
        linetyped line TypeV;
        vtsiblep *visible'>;
END

MENU topm
        Graph;
        <modifyu, modifyn. •>
END;

MENU modif ym
        <line.tinetypen>;
        Visible, Vi«bkn>
END;

MENU Unetypem Solid; dotdash; broken; dasher dots; END

MENU visible Yes; No END


    HELP topfa
•You are at top level:;
    Graph plots graph;
    Modify allows modification of Curves,;
        -Modifications are not immediately apparent but will
          be seen the next time Graph is called..
        -Those curves which are not visible are listed under
          the legend.,
END


    HELP changeh
*I need to know which attribute you wish to change.';
        linetype.;
        Visible-change (on or off).';
END.

    HELP Knetypeh


Type one of the line types or list them with a list
 command.*
END;


NODE topn
    lopm; topp; topfa
END
NODE modif yn
    chaugem; changep; chanfch
END
NODE baetypen
    fanetypem: boetypeh; foetypep
END
NODE vitibkn
    visibkm; visibleh; vUblep
END.
```

FIGURE 3-1

```
——PROGRAM_____

VAR
    ·continue: BOOLEAN;

BEGIN
    ReadMenu (choke);
    CASE choke OF
        graph:  (can routines to make plot);
        modify: (modify existing plot)
                BEGIN
                    ReadMenu (choke);
                    continue * TRUE;
                    WHILE continue DO
                        CASE choke OF
                            line:
                                    BEGIN
                                        ReadMenu (choke);
                                        CASE choke OF
                                         solid:   linetype :« 1;
                                         dotdash: linetype :« 2;
                                         broken: linetype :« 3;
                                         dashes: linetype :* 4;
                                         dots:    linetype :* 5;
                                        END. (CASE)
                                    END. (tine)
                            visible:
                                    BEGIN
                                     ReadMenu (choke);
                                     CASE choice OF
                                      yer   show ^ l;
                                      no:   show :« fr.
                                     END; (CASE)
                                    END; (visible)
                            reject          continue := FALSE;
                        END; (CASE)
                    END. {MODIFY)
    END; (CASE)
END.
```

FIGURE 3-2