P. 11

A MEMORY EFFICIENT USER INTERFACE FOR CLIPS MICRO-COMPUTER APPLICATIONS

aurthors

Mark E. Sterle
Richard J. Mayer
Janice A. Jordan
Howard N. Brodale
Min-Jin Lin

Knowledge Based Systems Laboratory
Department of Industrial Engineering
Texas A&M University
College Station, TX 77843
(4009) 845-8500

April 24, 1990

A MEMORY EFFICIENT USER INTERFACE FOR CLIPS MICRO-COMPUTER APPLICATIONS

ABSTRACT

The goal of the Integrated Southern Pine Beetle Expert System (ISPBEX) is to provide expert level knowledge concerning treatment advice that is convenient and easy to use for Forest Service personnel. ISPBEX was developed in CLIPS and delivered on an IBM PC AT class micro-computer, operating with an MS/DOS operating system. This restricted the size of the run time system to 640K. In order to provide a robust expert system, with on-line explanation, help, and alternative actions menus, as well as features that allow the user to back up or execute "what if" scenarios, a memory efficient menuing system was developed to interface with the CLIPS programs. By robust, we mean an expert system that (1) is user friendly, (2) provides reasonable solutions for a wide variety of domain specific problems, (3) explains why some solutions were suggested but others were not, and (4) provides technical information relating to the problem solution. Several advantages were gained by using this type of user interface (UI). First, by storing the menus on the hard disk (instead of main memory) during program execution, a more robust system could be implemented. Second, since the menus were built rapidly, development time was reduced. Third, the user may try a new scenario by backing up to any of the input screens and revising segments of the original input without having to retype all the information. And fourth, asserting facts from the menus provided for a dynamic and flexible factbase. This UI technology has been applied successfully in expert systems applications in forest management, agriculture, and manufacturing. This paper discusses the architecture of the UI system, human factors considerations, and the menu syntax design.

USER INTERFACE ARCHITECTURE

The UI architecture was developed as a result of the requirements of the ISPBEX system for memory efficiency and fast execution speeds. By designing menus that could be stored on the hard disk during program execution, main memory could be reserved for execution of the CLIPS program. Thus, more rules could be incorporated into the system, more facts could be maintained, and the program could run faster. This architecture, illustrated in figure 1, consists of two components, the FIFTH programming environment and the menu interpreter.

FIFTH Programming Environment

The FIFTH programming environment, was developed by Cliff Click and Paul Snow and is maintained by the Software Construction Company. FIFTH facilitates the compilation and debugging of text description menus by providing high level utilities to access the low level commands of Forth. The programmer uses a simple text editor to create the text descriptions which specify the size, content, and actions of each menu. This eliminates the need for complicated key sequences and instructions and thus allows rapid menu creation. The menus are compiled into microprocessor-like binary format instructions (e.g. push, pop) and stored on the hard disk in a file.

Menu Interpreter

The second component of the system, the menu interpreter, is written in C. The menu interpreter loads and interprets the binary instructions for a menu only when a call is received from the expert system. The menu interpreter is extendible because new commands can be easily defined and compiled into the C code providing a system that can meet the needs of the particular application. A menu selection history utility was also developed so the user could review or modify data that was previously entered into the system for a problem scenario.

Advantages of the UI Architecture

The design of the UI architecture allows the programmer to design a robust and user friendly system by providing a memory management method for developing the menus. Since a text editor can be used to create the menus, quick prototyping is simplified. The extendible menu interpreter allows the programmer to implement a system that can be tailored to the user's needs. All of this went into the development of a UI to meet the requirements for memory efficiency and fast execution speeds.

HUMAN FACTORS CONSIDERATIONS

Human factors design considerations were included in the development of the UI. Since this system would be utilized by Forest Service personnel with various levels of computer expertise (managers as well as technicians), it had to be easy to learn and convenient to use. One design consideration which simplified use of the system involved requiring the user to remember as few keys as possible. Another consideration included the use of help menus and explanation files. Also, backing up to allow the user to execute similar scenarios made using the system easier. Ten keys were selected for this implementation and were mapped to the functions described in table 1, titled Mapping of Keyboard Keys to Functions.

Cursor Movement, Option Selection, and Data Entry

The first set of keys shown in the table allow the user to move the cursor or leave a menu. The arrow keys are used to move the cursor to the appropriate selection so the user may select an item by pressing the enter key. Or the user may type the first letter of the selection to move the cursor to the item. If a selection is made by mistake, the user can de-select the item by moving the cursor to the selection and pressing the enter key. If the selection requires information to be typed in the space provided, the data can be entered after moving the cursor to the selection without hitting the enter key first. The data is checked for validity, type, and length which is defined in the menu description and if a mistake is made (syntax or out of bounds) an error message is displayed and the user is allowed to re-enter the information. When the user presses enter again, the cursor moves automatically to the next selection. The del key will cause the cursor to backspace and delete one character at a time. The menu can be designed so that the user goes directly to the next menu after making a selection (that is, after pressing the enter key) or the user can be permitted to review selections and leave the menu by pressing the end key.

Function Keys

The second set of keys are the function keys. Help and explanation menus can be accessed where provided by moving the cursor to an item and pressing the F1 key. The user may then press any key to return to the previous menu where the cursor will be on the same item. General information on use and movement of the cursor and function keys is provided from any position on any menu by pressing the F2 key. Pressing the F10 key will cause a question to appear asking if the user wants to end the ISPBEX session and exit to MS/DOS. Typing an *n* will simply return the user to the previous menu.

Viewing Results

The third set of keys shown in the table allow the user to view and leave a result file. To view a result file in which information determined by the application program during a session has been stored, the user can use the page up and page down keys to move through the text. The up and down arrow keys can be used to move up or down one line at a time through the file and the right and left arrow keys can be used to view files that are wider than 80 columns. The esc key will return the user to the previous menu from which the result file was accessed. No editing can be performed on these files as the information they contain is determined by the program.

Human Factors Design Benefits

Several factors were deemed necessary in order for the WCA to be successfully implemented. First, by keeping the number of keys required to a minimum and providing the user with individual selection help utilities, a system can be designed that is easy to learn as well as convenient to use. With this system, the user does not have to remember complicated key sequences or details about system implementation, and thus, first time or infrequent use becomes less trying. Also, the user can learn from the expert system because files are created by the system which give explanations and details for why certain results were suggested and the logic that went into making the decisions. Second, by backing up to previous menus, the user can execute similar scenarios and soon begins to understand the subtleties involved in the complicated reasoning processes that the experts used to make those decisions. Additionally, the user can save time with the history facility when executing problem scenarios that have similar data input to ones already evaluated because no time is lost due to re-entering all the data. These factors were considered necessary for the successful implementation of this system.

TEXT MENU SYNTAX

The text menu syntax was designed to help the programmer develop menus rapidly and easily. Figure 2, Text Menu Syntax, is an example of a text description for a menu showing the identification and location of an infestation of southern pine beetle, *Dentroctonus frontalis* Zimn. (Coleoptera: Scolytidae). It also includes the help menus associated with the input data. First, the main menu name is declared, followed by the help menu declarations. Next, the help menus are defined. Following the help menu definitions are the commands that are executed from the main menu upon making a selection. And finally, the main menu is defined. The following paragraphs will explain some of the syntax shown in figure 2 in more detail, how the menu is called from a CLIPS program, and the additional commands that are available.

UI Function Call

The main menu is identified by the name SPBDATA and is called by a CLIPS rule during program execution. The call looks like this:

```
(ui "spbis.mnu" spbdata history ?id-code)
```

The user defined function call to the menu interpreter is *ui*. The name of the file containing the compiled menu definitions is *spbis.mnu* in this example. The main menu name, *spbdata*, is next. Any number of menus (usually related functionally) can be stored in one file. The items entered by the user in the menu are stored in another file called *history* and will appear the next time this menu is accessed. A unique history file name must be used for each menu. A *NULL* can replace a file name if saving the information is not desired. The parameter, *?id-code*, is passed to this menu for display as the infestation number at the equals sign on the main menu shown in figure 2 . The number *0*, shown in the figure, represents the first parameter passed to the menu. Additional parameters would be numbered in increasing order (*1*, *2*, etc.).

Help Menu Definition

The help menus are defined within a set of brackets with the menu name following the closing bracket. The *0 0*, located after the opening bracket, refers to the minimum and maximum number of selections, respectively, that must be made before leaving this menu. This indicates no selections are to be made and the user can leave the menu and return to the main menu by pressing any key. *Menu-begin* and *menu-end* indicate the start and finish of the menu's display area. The ^ symbol specifies the border limits. Finally, *3 8*, followed by the command *display*, indicates the position on the CRT screen, row and column, to display the help menu.

Menu Selection Commands

Each item that can be selected from the main menu has three sets of brackets associated with it. The brackets contain menu commands, summarized in table 2, that can be executed from the menu. The following is a brief description of each of these commands.

The first set of brackets contain commands which are executed when the cursor is moved to that item and the enter key is pressed. The *prtmsg* command causes the message in the quotes to be displayed at the bottom of the main menu as shown in figure 3. The *1 20 readi* command specifies that an integer between 1 and 20 is to be entered for this item. Similarly, the *"0123456789WLD" reads* command restricts the user to entering an integer or the special characters WLD. Other possible commands available for defining input are: *read*, which reads any printable keyboard input; *reada*, which reads an alphabetic character; *readr*, which reads a real number; *readan*, which reads alpha-numeric input; *readdate*, which reads a date. All the numeric read commands have range checking. If an out of bounds number is entered an error message will tell the user to enter a value between the specified bounds. These commands help the user by checking the input to avoid errors in data entry and thus, data integrity can be maintained.

The second set of brackets contain commands that are executed when the the F1 key is pressed. The *exec* command causes a help menu containing detailed information about the item pointed to by the cursor to be displayed. The user returns to the main menu after leaving the help menu. A sub-menu can also be called with the *exec* command that allows information related to the items in the main menu to be entered and then returns the user to the menu it was called from.

The third set of brackets contain commands that will be executed upon leaving the main menu. Two commands, *readwrd* and *readstr*, will read and store the word or string entered by the user for a selected item. The *assert* command causes the string within the square brackets to be asserted to the fact base of the CLIPS program.

Other Menu Modifiers

Placing the *exit* command in the first set of brackets will cause the user to leave the menu if the cursor is next to that item when the enter key is pressed. Otherwise, the user is required to enter the minimum to maximum number of items specified by the numbers that precede the *menu-begin*. If these numbers are equal, but not zero, the user will leave the menu as soon as the minimum/maximum number of entries have been performed. If they are different, the user must press the end key to exit the menu.

The asterisk is used to specify cursor placement on the screen for an item. When enter is pressed on the selected item the line following the asterisk will be highlighted. The programmer can also use the ampersand/tilde combination to control cursor direction. This is especially useful when the user must fill in several items because it will allow the use of arrow keys to wrap around the menu.

Advantages of the UI Menu Syntax

Changes to menus during the prototyping phase of knowledge acquisition can be made quickly by the programmer or an expert who has minimal knowledge of programming and computers by using a simple text editor. Development time for the system is thus reduced. Utilities for error checking of user data entry are extendible and can be specifically tailored for the application and user's needs. This saves the user time because errors are caught immediately and there is no reason to rerun the entire program. Methods for cursor movement, data entry, and leaving a menu can be specified which make the system less cumbersome to use.

FUTURE ENHANCEMENTS

Three enhancements to the UI would improve the performance, maintainability, and versatility of the system. First, the FIFTH programming environment, currently written in Forth, and menu interpreter, which is written in C, should be rewritten in one language. This would provide a single environment for creating the menus and allow easier modification and enhancement of the UI system. Second, the current UI operates only on the IBM PC AT class machines running with MS/DOS and should be rewritten to port to other operating systems, such as UNIX. And third, the UI was designed specifically to run with CLIPS and should be rewritten as a stand alone package that can be used with other software systems.

SUMMARY

From a programmer's view point, there are four advantages gained by using this UI. First, because the menus are stored on the hard disk (instead of in main memory) during program execution, a more robust system can be implemented. Second, the compact size of the binary files leads to efficient memory usage. Third, since the menus can be built rapidly using a text editor, fast prototyping speeds up the knowledge acquisition phase and development time is reduced. And fourth, because new commands can be added to the text menu syntax, the system is extendible and can be tailored to the user's specific needs and requirements.

A user benefits from the use of this system in four ways. First, detailed help menus can easily be associated with any item and displayed using a common function key. Second, backing up and saving menu choices allows the user to repeat similar scenarios without having to re-enter all the information. Third, asserting facts from the menus provides for a dynamic and flexible factbase. And fourth, requiring the user to remember as few keys as possible makes learning and remembering how to use the system easier. This UI technology has been applied successfully in expert systems applications in forest management, agriculture, and manufacturing.

Mapping of Keyboard Keys to Functions

• *Arrow keys*	Moves the cursor up, down, right, and left.
• *Enter*	Causes the item next to the blinking cursor to be selected or de-selected from a menu.
• *Del*	Allows you to backspace and deletes values already typed in.
• *End*	Allows you to leave a menu after the appropriate information is entered.
• *F1*	Causes help information to be displayed if available for the item that the cursor is on.
• *F2*	Causes explanations for the user keys to appear.
• *F10*	Allows you to end the expert system session and return to DOS.
• *Page up*	Causes the previous page of a result file to be displayed.
• *Page down*	Causes the next page of a result file to be displayed.
• *Esc*	Allows you to leave a result file.

table 1

Menu Commands

• *prtmsg*	Causes message to be displayed at bottom of main menu.
• *readi*	Specifies that an integer is to be entered for this item.
• *reads*	Restricts user to entering the special characters designated.
• *read*	Reads any printable keyboard input.
• *reada*	Reads an alphabetic character.
• *readr*	Reads a real number.
• *readan*	Reads alpha-numeric input.
• *readdate*	Reads a date.
• *exec*	Causes a help menu containing detailed information about the item pointed to by the cursor to be displayed.
• *readwrd*	Read and store word entered by the user for a selected item.
• *readstr*	Read and store string entered by the user for a selected item.
• *assert*	Causes the string within the square brackets to be asserted to the fact base of the CLIPS program.
• *exit*	In the first set of brackets causes the user to leave the menu when the enter key is pressed.

table 2

User Interface Architecture

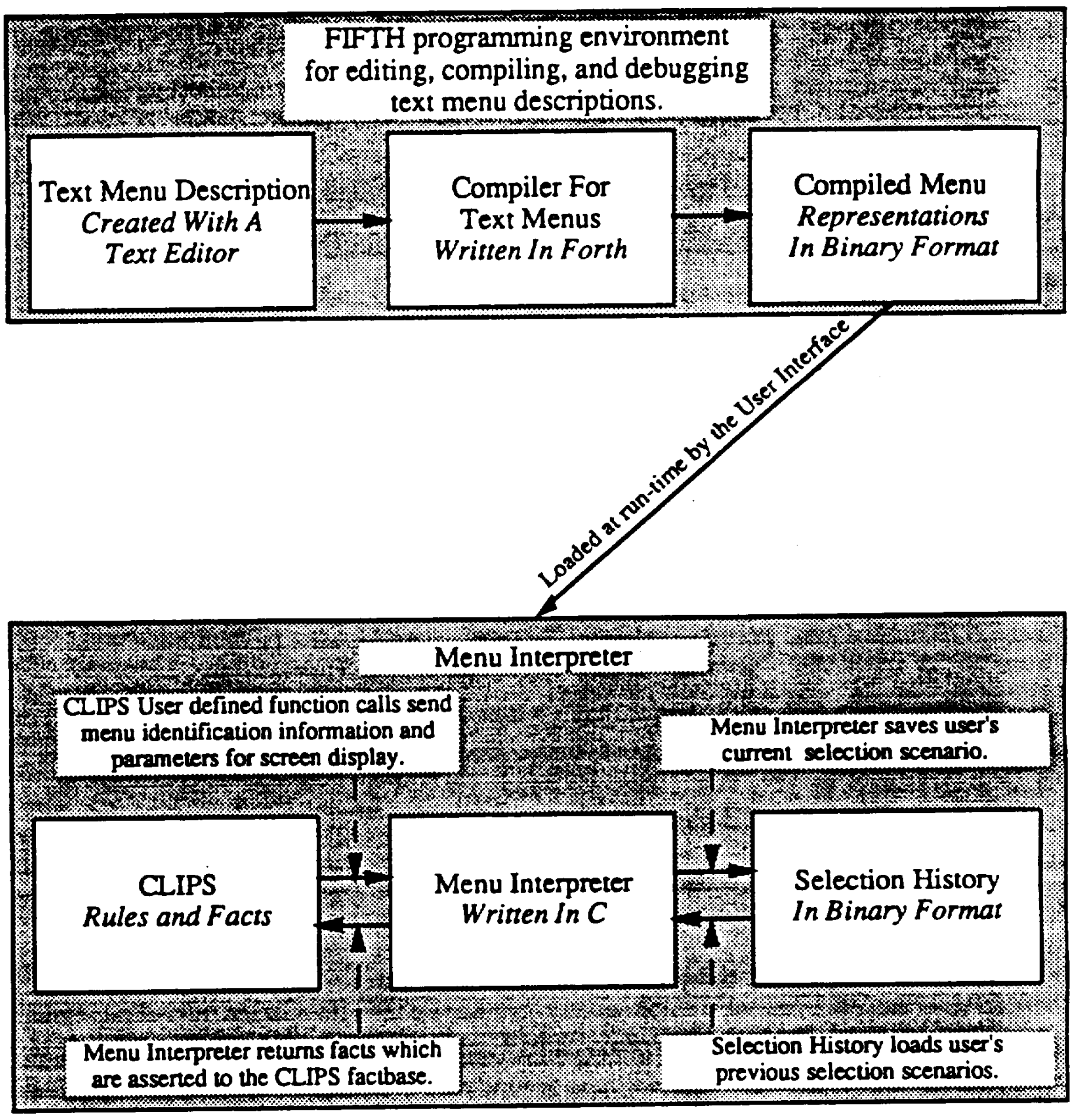

FIFTH programming environment for editing, compiling, and debugging text menu descriptions.
Text Menu Description *Created With A Text Editor*
Compiler For Text Menus *Written In Forth*
Compiled Menu *Representations In Binary Format*
Loaded at run-time by the User Interface
Menu Interpreter
CLIPS User defined function calls send menu identification information and parameters for screen display.
Menu Interpreter saves user's current selection scenario.
CLIPS *Rules and Facts*
Menu Interpreter *Written In C*
Selection History *In Binary Format*
Menu Interpreter returns facts which are asserted to the CLIPS factbase.
Selection History loads user's previous selection scenarios.

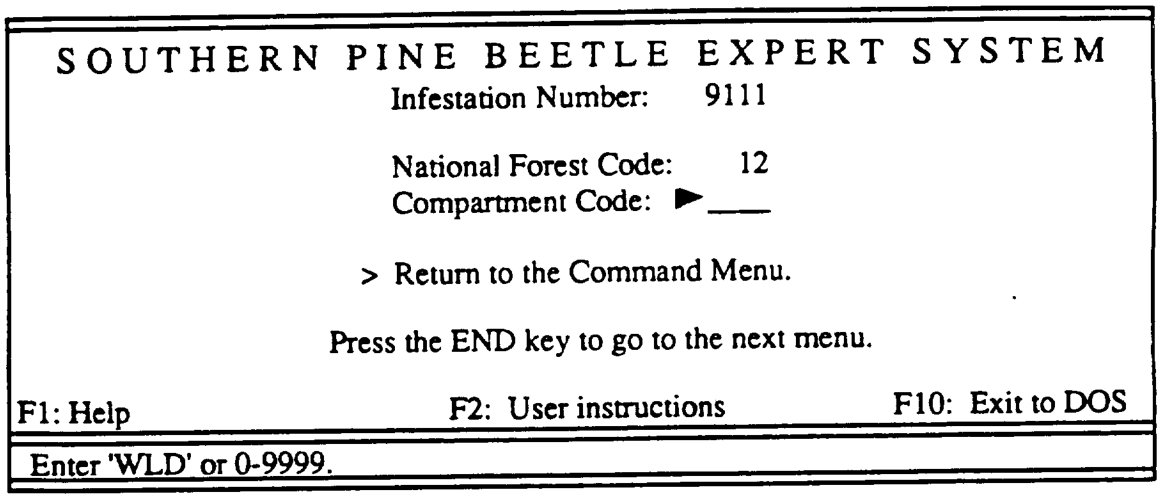
figure 1

Text Menu Syntax

```
SPBDATA
!00000898
define spbdata
var help1
var help2

{ 0 0
menu-begin
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
          Enter number from 1 to 20 for the National Forest Code.
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
menu-end
3 8 display
} define help1
{ 0 0
menu-begin
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
Enter number from 0 to 9999 for general forest or the letters WLD for wilderness.
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
menu-end
3 8 display
} define help2

{ " Enter 1-20 for National Forest. " prtmsg 1 20 readi }{ help1 exec }{ [national-forest readwrd ] assert }
{ " Enter 'WLD' or 0-9999. " prtmsg "0123456789WLD" reads }{ help2 exec }{ [comp readwrd ] assert }
{ [ backup command ] assert exit } { } { } 3 5
menu-begin
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
   S O U T H E R N   P I N E   B E E T L E   E X P E R T   S Y S T E M
                         Infestation Number:    = 0

                         National Forest Code:   &__~
                         Compartment Code:   &____~

                       *> Return to the Command Menu.

                   Press the END key to go to the next menu.

F1: Help                        F2: User instructions              F10: Exit to DOS
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
menu-end
1 1 display
end
```

figure 2

Example Menu Output



```
SOUTHERN PINE BEETLE EXPERT SYSTEM
                    Infestation Number:    9111

                    National Forest Code:     12
                    Compartment Code: ►____

                  > Return to the Command Menu.

             Press the END key to go to the next menu.

F1: Help              F2: User instructions          F10: Exit to DOS
Enter 'WLD' or 0-9999.
```

figure 3