S33-61

A Graphical Interface to CLIPS Using SunView

Terry Feagin
University of Houston - Clear Lake

Abstract

The importance of the incorporation of various graphics-oriented features into CLIPS is discussed. These features, which have been implemented in a version of CLIPS developed for a popular workstation, are described and their usefulness in the development of expert systems is examined.

Introduction

When developing expert systems that are intended to interact heavily with the user (as opposed to those systems that operate in a primarily independent manner), it is essential to provide an interface that enhances and accelerates the process, that allows meaningful dialog with the least effort, that provides clear and unambiguous two-way communication, that expedites the handling of sensitive or emergency situations, and that provide intuitive mechanisms for giving commands to and for receiving responses from the expert system. Computer graphics has long been recognized as a valuable aid in facilitating the flow of information between users and their computer-based applications. Instead of allowing only a few one-dimensional streams of characters (i.e., input and output files and a command line interface), modern computer graphics admits the possibility of interacting with the user via a number of two-dimensional color images that can move or be influenced directly by the user using a mouse, light pen, keyboard, joystick, or other graphic input device. The images are often organized into windows and user-interaction is often provided via menus that are mouse-selectable.

There are, of course, several enhanced versions of CLIPS that provide support for graphic-based interactions. However, these are primarily provided to enhance the giving of commands or setting conditions at the top-level (which level of support, it might be added, is also provided by the system described below).

For example, the Macintosh interface allows users to clear, load, reset, run, etc. by making conventional menu selections. However, there is no direct support for opening windows or generating menus from within an executing expert system.

Because of the extensible nature of CLIPS, it is not difficult to develop such support by adding user-defined functions. This paper, as well as several other papers presented at this conference, offers the expert system developer the ability to support directly such graphic-oriented interfaces to the user.

In an expert system, it is often desirable to convey to the user a set of conditions that may be true or false or indeterminate (inactive, disabled, etc.). Additionally, it may be important to show a precise measurement or reading. In the traditional command-line versions of CLIPS, these conditions would normally be exhibited via printed messages. In a graphic-oriented interface, the natural vehicle for conveying such information would be to provide an image or icon that might be immediately recognized by the user and to alter the image in a way that might graphically depict the level of the condition. For example, in an expert system developed to assist the operators in a nuclear power plant, excessive temperature conditions might be indicated by flashing red in an image depicting a thermometer. In an expert system developed to control a chemical reaction, the pH of a solution might be indicated by showing a dial. If the pH exceeds certain limits (either high or low), then the dial could be repainted, for example, in red for low pH and in blue for high pH values. If a serious or emergency condition holds, a graphic-oriented interface might be set up to flash the whole screen or window in color, to set off audible alarms (such as a beep or buzzer sound), to present the operator with an alert window with various possible actions or options designated, and to allow the operator to view and evaluate the consequences of his/her actions on the system via explanatory text revealed in windows and additional diagrams of equipment, meters, or fault nets.

Specific Advantages of a Graphical Interface

Most of the advantages of providing a graphical user interface to an expert system are obvious. Human operators are usually more receptive to a new environment if it is intuitive, pleasing to the eye, and easy-to-learn. A well-designed graphical interface assists the operator in *visualizing* the problem at hand, the relationships between entities in the system or variables in the problem, the ways in which she/he can or cannot affect the behavior of the system or the solution of the problem, the current state of the system or the solution process, the distinction between essential and non-essential characteristics or conditions, and any hierarchical organizational relationships.

Any change in the system can be identified almost immediately and the more significant changes can be allowed to trigger the more visually stimulating graphical effects (such as flashing lights or images and alert windows). This kind of separation of more significant from less significant events is difficult to accomplish as effectively with a simple command line interface.

Animation of windows or objects within windows can be used to represent higher level concepts such as a sequence of actions or events that are particularly difficult to represent in a simple command line interface. This is especially helpful if the speed of the animation can be varied.

In a graphical user interface, it is easier to control and restrict the user's input when it is important. Typographical errors can be eliminated. Other types of user input errors such as clicking on the wrong object are possible, but can be readily monitored and the user can be requested to confirm any unusual input.

Even a simple presentation of images within windows can be effective. The images can be used to present aspects of the problem or system that are otherwise difficult or impossible to present. In many of the science and engineering disciplines, there are times when some kind of two-dimensional image is the best way to represent a problem or method of solution. For example, in an expert system that might be used to help solve heat transfer problems, it may be desired to show the temperature distribution over the surface of some physical object like a flat plate of copper. The actual temperature distribution could be displayed as a color-coded image within one of the windows and used to show progress toward a solution.

In an object-oriented approach within CLIPS, one may wish to identify specific graphic objects or items that represent the objects about which the system is reasoning. As the attributes of the objects change, the graphic representations (position, size, shape, color, motion, etc.) could be made to change as well, thereby giving the user a view of the reasoning process that might be difficult to provide with the more usual command line interface. As the new object-oriented version of CLIPS emerges, this advantage may become even more significant.

Another advantage of a graphical approach concerns explanation facilities. If one creates an expert system in which the user can simply depress a mouse button over an object to signify that the user requires an explanation of the reasoning process or simply requires help regarding the meaning of the object, then the expert system can be more readily understood and may even be used for training new users. Also, if a user enters an unusual, expensive, hazardous, or dangerous request of the system, the system can ask for confirmation with an alert window complete with a cautionary warning. Security can be enforced by requiring passwords at critical points before actions are taken.

A graphics environment is less tiring to the user. Graphical output is generally able to convey more information with less eyestrain than simple text. In a command line interface, a user may miss an important detail that can become lost in line after line of alphanumeric characters. For input, many users often find that using a mouse is easier than typing.

Some Graphics Primitives Useful In Creating Expert Systems

The kind of graphics primitives that one might select for creating an expert system will undoubtably vary from one application to another. A general expert system shell that proposes to support the user in all of the ways mentioned above must therefore be able to support a wide variety of graphic objects and functions. In this section, several kinds of graphic objects and functions are described and some examples of how they might be used in an expert system are given.

WINDOWS - A CLIPS programmer should be able to call functions that cause windows to be created, opened, closed, hidden, exposed, and destroyed. The size and position of any window should be adjustable from within CLIPS or directly by the user. Other useful attributes might include scrollbars, labels, and colors. The windows should have the same appearance as non-CLIPS windows. It should be possible to retrieve most window attributes directly.

ITEMS - There should be the ability to support a number of graphic objects or items within each window. It should be possible to create, hide, show, and destroy items. It should be possible to label the items, and move them about under user control or program control. Several especially useful types of items might be identified. For example, a button item would be useful for selecting conditions or indicating operator actions. Most graphical interfaces provide for this type of object. Such items should be displayable as general graphical images in color or as simple labeled buttons. Another useful type of item would be text items. These items could be used to prompt the user for input with text strings and enable the user to enter filenames, passwords, or other text input. Such objects could also be used for short messages to the user. Other types of items might also be defined. Animation of items would be also useful, particularly in simulations (another area where the use of CLIPS is growing rapidly).

Items should be selectable and the result of a user selecting such an object should be the assertion of a fact describing the event. It would also be useful for items to be highlighted when selected, thereby providing positive feedback to the user. It would also be useful to be able to get most item attributes directly.

MENUS - Menus should be supported for both windows and items within windows. Menus should be displayed according to the conventions supported by the windowing system in general. Whenever a menu selection is made, a fact should be asserted describing the selection. It should be possible to remove a menu and create a new menu for an item or a window.

DRAWING PRIMITIVES - Certain simple drawing primitives should be provided as a minimum, including the ability to draw lines, draw polygons, fill regions , load images from raster files into windows, and save window images (all in color, of course). It would also be important to be able to get the pixel value at a particular location within a window.

OTHER FEATURES - Other helpful features of a graphical interface to CLIPS would include the ability to change the color definitions in the colormap segment for a window, to cycle a window's colors, to repaint a window, to cause an item to be highlighted, to change the menu for an item, to remove all items in a window, to remove all windows, and to remove all the items in all the windows.

There are also a number of functions that might be provided for debugging purposes such as the ability to print a list of all the windows or a list of all the items in a window for examination.

Implementation : Some Questions

Most of the functionality for graphical objects described above is supported for C programs executing on Sun™ workstations under the windowing system SunView™. Making this functionality available to CLIPS programs is somewhat complicated by a number of issues:

How many of the hundreds of options available under SunView are really important for supporting the development of expert systems ? What features not currently supported by SunView should be added ?

Should the central control loop remain within CLIPS or should control be given to the main loop in SunView and returned to CLIPS only for the handling of predesignated events ? The latter callback mechanism is the one normally used when developing SunView applications. Also, how should multiple simultaneous input streams be treated ?

Should the SunView distinction between a window for images or drawings (known as a "canvas" in SunView) and a window for button and text items (known as a "panel") be maintained or should a "new" type of window be adopted that would incorporate the essential features of both panels and canvasses ? The second approach would be less confusing and present additional power to the CLIPS programmer.

Implementation : Some Answers

Out of the hundreds of options available to general SunView applications, it was decided that only those most useful to the expert system developer would be supported. In the current version of the system, the most significant functions supported are as follows:

create_window - causes a window to be created with attributes as specified,
remove_window - causes a window to be destroyed, releasing resources used,
remove_all_windows - causes all windows to be removed,
hide_window - causes a specified window to be hidden from view,
show_window - causes a hidden window to be exposed,
open_window - causes a closed, iconified window to be opened,
close_window - causes an open window to be iconified or closed,
set_window - allows resetting of a window's attributes,
get_window - allows retrieval of a window's attributes,
set_window_color - allows redefinition of the particular colors used in a window,
set_window_fg - sets the window's foreground color,
set_window_bg - sets the window's background color,
draw_window - allows drawings to be created within a window,
load_window_image - allows a user-specified image to be loaded in a window,
save_window_image - causes the window's image to be saved in a file,

create_item - causes an item of specified type to be created in a window,
remove_item - causes an item to be removed permanently from a window,
remove_all_items - causes all the items in a window to be removed,
remove_all_items_in_all_windows - causes all items to be removed,
hide_item - causes an item to be hidden from view,
show_item - causes an item to be exposed,
set_item - allows for resetting of attributes of an item, including its image,
get_item - allows for retrieving attributes of an item,
highlight_item - allows the item to be highlighted for a flashing effect,
animate_item - permits the item (i.e., its image) to appear to move within a window at a specified rate of speed,

create_item_menu - causes a user-specified, item-dependent menu to be created for the item,
remove_item_menu - causes menu to be removed from the item,
set_item_menu - allows menu attributes to be established,

get_alert_window - causes an alert window to be displayed and blocks user from entering input (except to indicate a response to the alert),
cycle_window_colors - allows the colors in a window to be cycled,
repaint_window - allows the window to be repainted,
snooze - causes CLIPS to sleep for a user-specified number of milliseconds,
list_windows - causes a list of the presently defined windows to be produced,
list_items - causes a list of presently defined items to be produced,

In almost all of the above functions, the number of arguments has been limited in order to make the syntax of each function easier to remember. The arguments are generally ordered in such a way that the most significant arguments appear first, thus allowing the CLIPS programmer to omit some of the less significant arguments (thereby implicitly specifying default values for such arguments).

In addition to the explicit function calls listed above, a user can interact with the system in a number of ways, primarily by making mouse movements and mouse button selection over windows, items, and menus. Text entry is also supported.

Regarding the issue of control, it was determined that the central control loop of CLIPS would be maintained and that SunView's Notifier (the dispatcher which allows client programs to register event handlers and receive notifications later when the respective events occur) would be called explicitly after each rule firing and implicitly during any blocking or non-blocking read. This allows the user to obtain good response to graphics input events while CLIPS is firing rules and also when the user is entering commands or function calls directly to the CLIPS prompt.

It was also determined that the distinction between a canvas and a panel in SunView would be superfluous. The features of both have therefore been combined by adding the essential features of a canvas (namely, most two-dimensional graphics primitives such as drawing lines, constructing images, setting colors, and getting pixel values) to the panel type of window. Therefore, to the CLIPS programmer, there appears to be a core type of window that allows for buttons, simple text interactions, and color graphics output.

Another feature that was not directly supported under SunView is the dynamic movement of items under user control. By dragging the item with the middle mouse button depressed, the user can reposition the item at will. Afterwards, the new position of the item is reported to CLIPS as a new fact assertion. These last two features make the package much simpler and more useful for developing expert systems.

The animation of an item is also not directly supported under SunView. However, given the growing interest in using CLIPS for the development of simulations (whether or not such simulations are a part of an expert system), the animation feature as given above has also been provided.

Conclusions

The project has now been successfully completed and over thirty-five new functions (mostly graphics-oriented) have been added to CLIPS. Work is now underway to enhance these capabilities even further and study their usefulness within several existing expert systems.