ON THE GENERATION OF GRAPHICAL OBJECTS AND IMAGES FROM WITHIN CLIPS USING XVIEW

Terry Feagin

University of Houston - Clear Lake

Abstract. A variety of features that support the generation and manipulation of graphical objects and images in CLIPS are described. These features provide the CLIPS programmer with the ability to develop an enhanced user interface. Windows and objects within the windows (such as buttons, sliders, text, etc.) can be created, hidden, redisplayed, given new values, properties, or positions, and manipulated in various other ways. Menus can be generated for any window or object and displayed when desired. Interaction with the user is primarily via mouse movements and mouse button selections made over windows, items, or menus. User input is recorded in the form of fact assertions. Limited animation of objects is also supported.

INTRODUCTION

In the development of heavily interactive expert systems, it is important to provide a user interface that will enhance and accelerate the the overall process. Meaningful dialog should be accomplished with minimal effort. Clear and unambiguous communication that expedites the handling of sensitive or emergency situations and that provides intuitive mechanisms for giving commands and for receiving responses should be supported.

Computer graphics has always been recognized as a valuable aid for facilitating the flow of information between the user and the application. Modern computer graphics allows users to interact with their applications via two-dimensional color images that can move or be influenced by the user with a mouse, light pen, keyboard, joystick, or other graphic input device.

XView was developed by Sun Microsystems (both XView and Sun are trademarks) as a high level toolkit for developing graphic applications for the X Window System. It also facilitates the conversion of SunView applications to X Window applications because many of the features are similar. However, the display of graphic images in SunView is usually accomplished using "pixwins" and the conversion of these features to XView requires the direct use of the lower level Xlib routines. Also, the use of implicit dispatching (which allows for the concurrent input of text and mouse selections) in XView is more complicated than in SunView because of the subtle timing and synchronization issues of the X Window System. Nevertheless, the use of XView simplified considerably the development of the graphical extensions to CLIPS described below. In the current version, digitized color images or simple colored shapes or text may be displayed in the background of a window and the CLIPS programmer may elect to display buttons, sliders, text, or other items (with which the user may interact) on top of the background image. In addition, the user may interact with menus for an item or for the window itself if the CLIPS programmer has made provision for it.

The use of graphical input and output for an expert system conveys much more information to the user. The more traditional command-line interface is restricted to character input and output and is more susceptible to errors and misunderstandings. In a graphics-oriented interface, one can use a more natural vehicle for conveying ideas such as providing an image or icon that is easily recognized and indicating state changes via changes in the image. In serious or emergency situations, one could sound alarms and flash the images to gain the user's attention. Also, with the extensive use of menus and help windows, one can provide special explanations, diagrams, and assistance for novice users.

GRAPHICS PRIMITIVES

A number of primitive graphics commands are provided for the CLIPS programmer that facilitate the creation of windows, objects within windows, images within windows, etc. Most of these primitives are summarized below:

create-window	-	causes a new window to be created and its attributes to be specified
remove-window	-	causes an existing window to be destroyed
remove-all-windows	-	causes all existing windows to be destroyed
hide-window	-	causes an existing window to be hidden from view
show-window	-	causes an existing window to be displayed
open-window	-	causes an existing window to be opened or de-iconified
open-all-windows	-	causes all existing windows to be opened
close-window	-	causes an existing window to be closed to an icon
close-all-windows	-	causes all existing windows to be closed
set-window	-	allows various attributes of an existing window to be altered
get-window	-	retrieves the value of a given attribute of a given window
load-window-image	-	places a specified raster image into the window's background
draw-rectangle	-	causes a rectangle with prescribed attributes to be drawn in window
draw-circle	-	causes a circle with prescribed attributes to be drawn in window
draw-line	-	causes a line with prescribed attributes to be drawn in window
draw-polygon	-	causes a polygon with prescribed attributes to be drawn in window
create-item	-	causes a new item of specified type and attributes to be created
remove-item	-	causes an existing item to be destroyed
remove-all-items	-	causes all existing items within a window to be destroyed
remove-all-items-in-all-windows	-	destroys all items in all windows
hide-item	-	causes an existing item to be hidden from view
show-item	-	causes an existing item to be displayed (if the window is displayed)
set-item	-	allows various attributes of an existing item to be altered
get-item	-	retrieves the value of a given attribute of a given item
animate-item	-	allows an existing item to be moved within a window at a given rate
create-item-menu	-	causes a menu and selections to be created for a given item
remove-item-menu	-	destroys the menu for a given item
get-alert-window	-	displays an alert window and blocks user until he/she responds

In addition to these primitives, there are a number of functions and commands that permit one to reset the default values for various attributes (such as the size and position of a window). The use of defaults makes the system much easier to use. Also, there are commands that permit the user to list the items and the windows currently defined. This feature is especially useful for debugging purposes.

GRAPHICS INPUT

In addition to the graphics output primitives described above, there are a number of ways to provide graphics input to the system. The simplest mechanism is to select a button item within a window by depressing the mouse select button (usually the leftmost button on the mouse) while the mouse cursor is directly over the item. The result of this action by the user is the automatic assertion of a fact into the CLIPS database of facts. In this case the fact asserted would be:

(selected-item "item-name")

If the item were a text item with space where the user could enter text (e.g. NAME: _____), then when the user selected this item and typed in the name "username", the fact asserted would be:

(selected-item-text "item-name" "username")

The adjustment or selection of other items results in similar facts being asserted. The net effect is that, if the CLIPS programmer has provided rules that activate and fire when a particular item is selected or adjusted, appropriate actions can be taken that will allow the user to interact with the executing CLIPS program.

Another mechanism for interacting with the user involves the use of menus. The CLIPS programmer creates the menu using the create-window-menu or the create-item-menu commands described above. If the user depresses the menu-select button (usually the rightmost button on the mouse), while the mouse cursor is over an item (for which a menu has been created earlier) then the menu is displayed. If the user releases the menu-select-button while the cursor is over a particular menu selection, then a fact is asserted according to the selection made. For example, if at some point in time a menu were created via the command:

(create-item-menu "item-name" "load" "save" "quit")

then the item "item-name" would have a menu associated with it. Later, if the user depressed the menu-select button while the cursor was over the item, the menu:

load
save
quit

would be displayed and if the user selected the first selection, the following fact would be asserted:

(selected-item-menu "item-name" "load")

which could then be used to cause further actions to take place.

It is also possible to create menus that are not associated with any particular item, but that are associated with a window (or its background). If the user depresses the menu-select button while the cursor is over a window, but not over an item, then the menu associated with the window (if such a menu has been created earlier) would be displayed.

THE ISSUE OF CONTROL

In developing a graphics extension to CLIPS, one invariably faces the question of control. A graphics interface requires that events be accepted, queued up somehow, and handled appropriately in a timely fashion. Systems like SunView and XView generally require that the programmer define all the graphics entities (windows, items, menus, etc.), their interrelationships, any events that may interest the programmer, and callback routines which should be invoked whenever particular events occur, and *then* turn over the control of the program to a main loop. The reason for this is the desire to be able to deal effectively with multiple streams of input (mouse, keyboard, etc.) The execution of the program then proceeds according to the events that occur and the appropriate, related callback routines. In such a case, the programmer has given control of the program to the central main loop.

For the present extension of CLIPS, it was decided to avoid (to the greatest extent possible) changing CLIPS in any major fashion and to maintain the central control loop of CLIPS. Therefore, the dispatching mechanism of XView is called explicitly after each rule firing (to dispatch any events that might occur) 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 directly to the CLIPS> prompt.

CONCLUSIONS

A graphics extension to CLIPS has been developed using the XView toolkit. The project is now complete and over fifty new, user-defined functions have been added to CLIPS. Most of the new functions and commands are graphics-oriented. Work is now underway to enhance these capabilities and examine their usefulness within various applications.