Prototyping User Displays Using CLIPS

Charles P. Kosta
Ross Miller

Center for Productivity Enhancement
University of Lowell
Lowell, MA

Dr. Patrick Krolak
Matt Vesty

Transportation Systems Center
Cambridge, MA

Abstract

CLIPS is being used as an integral module of a Rapid Prototyping System. The Prototyping System consists of a display manager for object browsing, a graph program for displaying line and bar charts, and a communications server for routing messages between modules. A CLIPS simulation of physical model provides dynamic control of the user's display. Currently, a project is well underway to prototype the Advanced Automation System (ASS) for the Federal aviation administration.

A prototype, as defined by *The American Heritage Dictionary*, is an original type, form, or instance that serves as a model on which later stages are based or judged.

LEVELS OF FUNCTIONALITY

The prototyping of user interfaces has evolved into four distinguishable levels. The first level is the "straw man" stage, when a basic screen design is developed that approximates how the interface should look. The purpose of this phase is to work out aesthetics issues only; it does not give any indication of the usability of the display. Using C or another script -like language, the second level prototypes static responses using limited scenarios. At this phase the objects can react to user input, but the responses do not deviate from an internal script. The third level incorporates a dynamic response from the system. During this phase the dynamic system attempts to mimic the real system as closely as possible in such areas as responding to user events and simulating (or generating) user scenarios. While using this level prototyping users should not be able to tell that they are using a prototype and not the real system. The highest level of prototyping contains everything in the previous three levels plus the ability to capture and report on usage metrics.

The function of prototyping is to demonstrate whether or not a model serves a useful purpose. At the first level, we are trying to find out if the screens are discernible; do they portray right meaning. The second level asks whether or not the prototype can respond in an intuitive manner. The third level utilizes scenarios that in turn simulate events to which the user must react. The highest level uses metrics to modify the behavior of the running system. It is important to note that the first three levels also have metrics, but they are not integrated into the prototype; they are external: surveys, video taped sessions, subjective comments of the user community.

USER DISPLAYS

Typically, static mock-up displays are the first prototypes created for most applications. They help determine spatial and size constraints for various data models. Dynamic displays are later generated to allow users to interact with the prototype.

Today's prototypes not only deal with data models, but with user models as well. For example, icons must somehow depict a similar meaning for all users. Supporting this trend is the rapidly increasing role that windowing systems are playing in today's computing environments. Specifically, the method in which information is distributed into windows and icons is important for users who are trying to understand the state of an active system.

New techniques are being developed daily that strive to go beyond the borders of windows of information into what have been termed widgets. Widgets are typically some graphical representation, in the form of an icon or window, that provide movements and actuators upon some object. An example of this type would be a sliding bar widget. In a similar manner to the sliding bars used on stereo equipment, the user can select the slide bar with the mouse and move it along the axis to set or adjust some scalar value. Widget complexity is limited only by the creator's imagination, and they can be as simple as a small radio knob dial or as complicated as the entire front panel of a virtual computer. In general, prototyping systems are becoming increasingly object oriented with data items taking on object properties. These

properties can be linked to widget functionality on the display and when an object value changes the corresponding widget can be updated.

This paper will attempt to explain one particular system that was designed to elicit user requirements through the use of prototyping user interactions. The project is called User Requirements Prototyping System (URPS). URPS is positioned at the prototyping interactions (third) level on functionality. This does not mean that the two lower levels (static and responsive) are excluded — they are also available. What we have not included as yet is a method to obtain metrics from the running prototype.

OBJECT RENDERING

Information can be represented (rendered) in different manners. A temperature can be rendered as a number, a picture of a mercury thermometer that has more pixels filled as the temperature increases, or as a square block that changes from blue to red. Any one of these methods may be appropriate in a given situation. Any object can be rendered in some manner, although the method is usually based on object functionality as far as the user interface is concerned.

WINDOWING

It is important to consider the user model as a guide to object rendering. Current windowing systems allow the designer to choose different techniques for window (or object) management. The three main types are tiled, overlapping, and pop-up windows. Tiled windows are those that split up the screen into smaller tiles — no window ever covering up another — and is based upon the user's ability to deal strictly with base spatial concepts. Overlapping windows allow for the possibility of data being covered up and are usually equipped with the ability to resize, move, and place one window over another. In user models terms, overlapping windows represent the "desktop" paradigm. Pop-up windows are interesting in that they can represent a user model that goes beyond the "desktop" into models that are based on a virtual technical assistant working with the user's "desktop." In particular, current pop-up windows are used for displaying a message about the system that you must deal with immediately (like a high priority memo on your desktop); displaying a menu that represents either local or global choices about the window below it; and displaying pop-up windows that act like post-up notes from the system.

DYNAMICS

Allowing dynamic changes to happen on the display is useful. Most user design prototypes find it necessary to know if the user can use and interact with the data that is presented. Current techniques make use of C language (object-code linkability), specially designed scripting languages, or message passing constructs to facilitate dynamics. URPS takes a combined approach in the form of an expert system shell call CLIPS (C Language Integrated Production System). Event messages travel between objects via a FACT construct. Programmability is available at both runtime via CLIPS rules and link time via C code though CLIPS.

CURRENT SYSTEMS

There are many systems currently available for prototyping user displays. Two will be discussed briefly.

The first is a low cost solution available through COSMIC called TAE+ (Transportable Applications Environment Plus). TAE was developed by NASA Goddard as a tool for building consistent, portable user interfaces in an interactive alphanumeric terminal environment. TAE also supports rapid prototyping of user interface screens and interactions, and allows the direct reuse of those screens in the final applications. TAE+ now supports X Window and MOTIF widgets.

VAPS (Virtual Application Prototyping System) is a much more elaborate, commercially available package that runs on silicon graphic workstations. The user can build prototypes by interactively laying out the display and then attaching scripts to each object. The scripts are C functions that are modifiable by the user. VAPS supports a wide range of input devices, and a designer can first prototype a control panel using just graphics and a mouse. Later, a touch sensitive screen can be added. VAPS, a sophisticated product that can prototype very realistic screens, is a product of Virtual Prototypes.

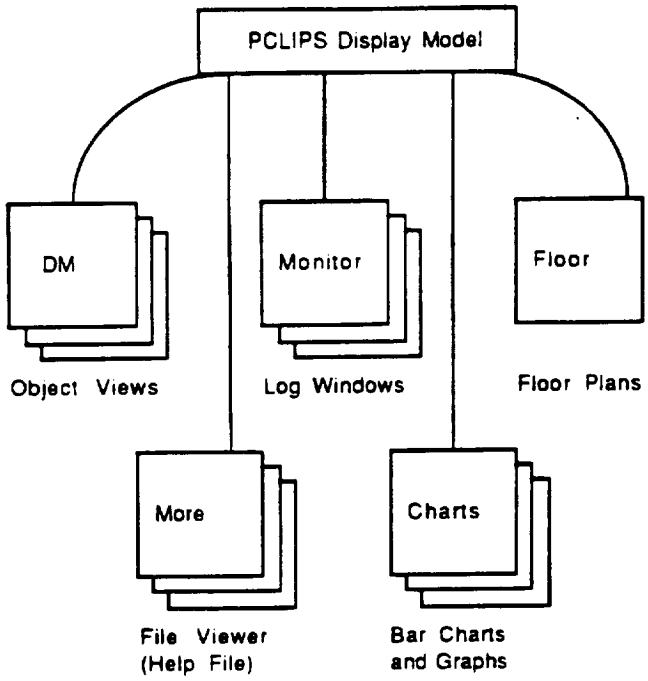

Figure 1. PCLIPS Display Model

DISPLAY MODELS

Rendering Models are based on the display devices available. These devices range from very low capability displays and very high level displays. To examine a few of these differences, three examples will be discussed here: the ANSI terminal, the IBM PC and the X Windowing System.

Using inverted text and special symbols whenever possible, the standard ANSI terminal can provide many rendering possibilities, although tiled windows seem to be the favorite on these systems. It is, however, possible to write, or use, a package can provide both overlapping and pop-up styles. Pictorially, widgets tend to be square and numbers are usually depicted with numerals. Artistically speaking it is possible to have icons that are intuitive.

The next step up from the ANSI terminal the IBM PC. The extended ANSI capabilities of the PC, along with the speed of the system, can support interesting pictorial effects. But one can always choose to tackle the graphic modes (using or buying a package). The biggest problem here is in choosing what level of graphics to support. Bit image graphics on the PC can provide a good medium for widgets; however, screen management is usually still up the programmer.

Lastly, the X Windowing System (and other windowing systems) provide window management features and widget management as well. A detailed explanation of the X Windowing System can be found in other places - it is referenced here to show that display models can vary greatly with device availability.

PROTOTYPING THE ISSS

The original work in this area was done to support the rapid prototyping of the maintenance and control consoles for the Federal Aviation Administration's (FAA) new air traffic control system, the Advanced Automation System (AAS). The purpose of the project is to develop a rapid prototyping system for a man-machine subteam to use in identifying user requirements in terms of the graphical interface. This information could then become the basis for a requirements document for the user interface.

The user displays were separated into functional groups where corresponding object structures and icons were created to represent the various objects. Functionally, the objects represented hardware and software objects that were in some state of usability. Widgets were built using the "traffic light" concept. Green means the object is functioning fine;

yellow means there is a degradation of the object; and red means that the object is dead. Blue is used to represent available but nonallocated resources.

CLIPS is being used as an event-based system. CLIPS is well qualified for this role due in part to the features of the production system model. It addition to events, CLIPS facts are being used to recreate the display model in the form of a fact base (knowledge base). These facts hold the object oriented system data about the actual objects and all the corresponding widget functionality. CLIPS rules function as receptacles for events that occur both by the simulation system and user's (display-based) events. See Figure 1.

PCLIPS is a parallel version of CLIPS that allows multiple CLIPS experts to communicate via a broadcasting function called *remote assert* (*rassert*). By using this method any number of CLIPS experts can be initiated. URPS presently has two: one that serves as a simulation of the prototyped system and another that maps simulation events to the user's screen. A display manager controls usage of the user's screen. Widgets communicate with the display manager in order to gain access to the display space and to update the data.

EVENT-BASED FUNCTIONALITY

There are two major types of widgets: an icon class made up of bit-image graphics and the other, an icon-which is surrounded by a colored box; both represent the state of the object. The box type is our GENERIC class. For this demonstration we have only one icon class; it is called TERMINAL.

```
(deffacts DisplayManager "Base Object Classes for Display Manger"
;
; template: (map-dm-icon <widget-class> <widget-state> <icon-filename>)
; template: (map-dm-state <widget-class> <widget-state> <box-color.)
;
  (map_dm_icon termina up "ik:i_terminal_ok")
  (map_dm_icon terminal down "ik:i_terminal_err")
  (map_dm_icon terminal degraded "ik:i_terminal_warn")
  (map_dm_icon terminal standby "ik:i_terminal_standby")
  (map_dm_icon terminal spare "ik:i_terminal_spare")
  (map_dm_state generic up GREEN)
  (map_dm_state generic down RED)
  (map_dm_state generic spare WHITE)
  (map_dm_state generic standby BLUE)
  (map_dm_state generic degraded YELLOW)
)
```

NOTE: The *generic_display_update* and *icon_display_update* use facts sent from CLIPS to the Display Manager to control widgets. *ask_for_something* receives events from the Display Manager.

```
(defrule generic_display_update "Catch all Generic Status Changes"
  (status ?type ?object ?state)
  (dm_object ?object ?)
  (map_dm_state generic ?state ?signal)
=>
  (rasser dm turncolor ?object ?signal)
)
;
(defrule icon_display_update "Catch only TERMINAL Status Changes"
  (status CC ?object ?state)
  (dm_object ?object icon)
  (map_dm_icon terminal ?state ?fname)
=>
  (assert dm chg-icon ?object ?fname)
)
;
; (Select . . .) facts are remotely asserted by the
; Display Manager when the user does something These
; are much like user events.

; Currently, the default action is to  open up a
; subview.  If the object SELECTed does not have a
; subview, then it does not have a "map_dm_windows"
; fact either.  Another rule with a lower salience
; catches lost User Events in case there is no sub
; view.
;
(defrule ask_for_something "Catch User Events"
 ?r1<-(select ?obj)
 (dm_window ?obj ?w ?h $?Window_Stuff)
 (map-dm-window ?obj ?x ?y)
=>
 (rassert dm open-window ?obj ?x ?y ?w ?h $?Window Stuff
```

```
(retract ?r1)
)
```

DISPLAY MODEL FUNCTIONALITY

Functionally, the display is separated into views. These views consist of collections of such widgets as object views, monitor logs, bar charts, and pop-up menus. Object views are windows controlled via remote asserts (*rasserts*) to the Display Manager Screen control, and pop-up windows are also controlled by Display Manager requests. Log windows, bar charts, and the floor plan are separately running programs that join the PCLIPS session upon start-up.

IMPLEMENTATION ISSUES

The Commodore Amiga was chosen as a platform for the following reasons: Low cost, useful resolution (640 X 400), choice of bitplanes, dynamically loadable icons, commercially available image-based tools, and multiprocessing capabilities. The first challenge was porting Clips 4.3 over to the Amiga -- no problem -- just a 5 week delay! The next challenge was in designing the actual display functions. Following this came the PCLIPS functionality; being able to allow multiple CLIPS experts to join together to form a PCLIPS Environment. This was accomplished via the recoding of a PCLIPS server which runs in the background. The server manages incoming requests to join a PCLIPS session and distributes remote *asserts* to all currently listed CLIPS processes. Once we had tools working we were then able to attack the problem of rapid prototyping the ISSS.

CONCLUSIONS

After weeks of designs and redesigns, we have found widgets, object oriented programming and image based icons to be important concepts in the development of new user interfaces. Widget technology is important for encapsulation of data and needs further study. Object Oriented approaches were definitely the way to go in our prototyping system. These approaches were used to determine the level of granularity for the prototype and also to specify functionality of object classes -- no one object was coded better or worse than another in the same class. Image based view facilitated the involvement of art-types who felt they had much more feedom with paint programs than when they were asked to layout displays based on geometrical (graphical) shapes.

Additionally, an interactive configuration tool was created to help in the layout of widgets within views, allowing objects to be positioned over bit-images (pictures). This is part of a far more interesting problem: whether to deal with image based objects or grahpical based (lines, cubes, geometry . . .) objects. One interesting group discussion led to the idea of rendering graphical objects on top of bit image backdrops.