ISBN: 978-3-95450-254-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS1991-S13MMI06

A Virtual Control Panel Configuration Tool for the X-Window System

Jeffrey O. Hill, Leo R. Dalesio and Debona M. Kerstiens
Los Lamos National laboratory, Los Alamos, NM 87545

Abstract

Computer Graphics Workstations are becoming increasingly popular for use as virtual process control and read back panels. The workstation's CRT, keyboard, and pointing device are used in concert to produce a display that is in essence a control panel, even if actual switches and gauges are not present. The code behind these displays is most often specific to one display and not reusable for any other display. Recently, programs have been written allowing many of these virtual control panel displays to be configured without writing additional code. This approach allows the initial programming effort to be reapplied to many different display instances with minimal effort. These programs often incorporate many of the features of a graphics editor, allowing a pictorial model of the process under control to be incorporated into the control panel. We have just finished writing a second generation software system of this type for use with the X-window system and the Experimental Physics and Industrial Control System (EPICS). This paper describes the primary features of our software, the framework of our design, and our observations after initial installation.

BACKGROUND

The EPICS[1] control system consists of an input-output controller subsystem that communicates via a software bus with many general purpose or application specific control system components. The input-output system provides the time critical and hardware specific portions of the control system. The software bus or "channel access" provides standardized communication between control system components over a local area network. EPICS has the following general purpose components: an alarm manager, an archive subsystem, a timing subsystem, and the operator interface which this paper describes.

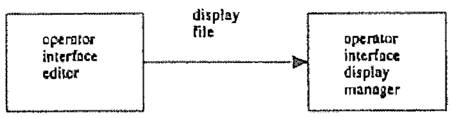
A first generation operator interface, developed for the telescope control system, has been deployed at the Argonne National Laboratory and elsewhere by the Los Alamos National Laboratory[2]. This operator interface was next ported for use on the EPICS control system and used on the Ground Test Accelerator and related test stands at LANL. This first generation operator interface proved the effectiveness of the virtual control panel technique for reducing the applications programming effort[2].

This paper describes the design and implementation of a second generation operator interface. We wanted to replace the obsolete graphics platform on which the first generation was built with the open X Window System standard. Other goals were faster display startup and update rates, and more effective configuration tools[3]. We also wanted to build a proper foundation on which we could build future extensions and enhancements, while spending less of our time on software maintenance.

*Work supported and funded under the Department of Defense, US Army Strategic Defense Command, under the auspices of the Department of Energy.

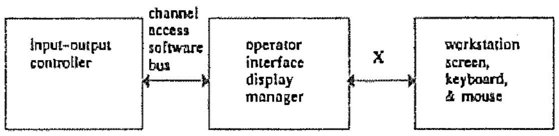
FEATURES

The second generation operator interface or OPI consists of an editor which is used to create and configure virtual control panels and a display manager which activates them. A display



file containing the virtual control panel description is the only form of communication between the two programs.

Once activated, displays can be used to monitor and control


input-output controller
channel access software bus
operator interface display manager
X
workstation screen, keyboard, & mouse

process variables. The display manager accomplishes this by responding to external events and updating the screen. External events include keyboard input, mouse input, or process variable state changes.

Operator interface displays are configured with a graphics editor capable of manipulating the normal complement of graphics object primitives such as rectangles, lines, ovals, arcs, and text. In addition to these primitives, the editor can also configure a full complement of process variable control and read back components such as indicators, meters, buttons, and menus. Once created, one or more graphics objects can be selected for cutting, copying, pasting, moving, or scaling. These techniques can be used to move groups of objects between several operator interface displays under edit on the same workstation. The editor also supports features for productive alignment and even distribution of object groups. The editor saves a finite list of all previous operations so that they may be individually undone at the operator's discretion.

A graphics form for modifying attributes is provided for each type of object which can be created by the editor. The forms have entries for every operator modifiable attribute even if some attribute modifications, such coordinate translations, can be carried out more efficiently with the mouse. These graphics forms or property sheets provide a simple and consistent method for entering the more complex configuration required by process control and read back components such as plots or indicators. For example, a bar indicator might require entry of a process variable name, labeling option, direction of increase, and a color modifier (the color could be static or based on an alarm condition).

The operator may choose to create a private color pallette for his display or share one created for another display. Both the number of colors and the hue of each color are configurable from the editor.

Text is displayed within a bounding box. This allows text object coordinates to be scaled the same as any other object on the screen. If a group of objects containing text is scaled then the text font size is appropriately scaled also. If the exact font specified by the operator during display configuration is unavailable in the future, then a reasonable alternative font is used.

Graphics primitives may be modified for dynamic operation. For instance, the visibility or color of a rectangle may be based on the state of a process variable.

Operating displays detect loss of network connection to process variables. Any graphics objects which are attached to a lost connection are displayed in a special color with the process variable name and an explanation of the problem. Graphic objects are automatically drawn again in their correct colors once normal network communication resumes.

Macro subsitution of process variable names allows a single display description to be used for a series of replicated devices at run time. For example, when the display is configured the operator could type in 'POWER_SUPPLY_$(DEVICE_NUMBER)' for graphic object's process variable name. When the display is activated the operator could specify 'DEVICE_NUMBER = 4' and the resulting process variable name used would be 'POWER_SUPPLY_4'.

DESIGN

We chose to use an internal display list definition to describe the contents of the display files configured by the editor and executed by the display manager. This approach gave us the most flexibility to add information into the display list definition that was specific to process control applications.

Our display list can take either a binary or ASCII form. We use the binary form when the display is executed for improved performance and compact storage. ASCII display files can be safely converted to any future binary display file format. This allows us to rearrange the binary display file format in future releases of the operator interface. This feature has already proved itself valuable when we realigned the binary display file so that it would execute properly on both the SPARC and 68k processor architectures. The ASCII display list format also allows an easy way to convert between the binary display list formats of incompatible architectures. For example, one might first convert a big endian processor architecture display file to ASCII followed by conversion to a little endian processor architecture binary display file.

Additions to the display list description often require no code changes to the display configuration editor. We have a source file which describes the binary and ASCII format of the display list. This file is run through a macro processor which generates two C language include files. One of these is a binary description of the display list format in the form of C structures. The other is a description of the display list format in the form of arrays of a structure describing structure fields. It is this additional information that allows the editor to configure a new addition to the display list description without, in many cases, writing additional editor code. This design has significantly reduced the number of lines of code in the editor, thus making it easier to write and maintain.

Both the display manager and editor were carefully written to use only the parts of the host operating system that are used by the X window system libraries. This should make it an easy task to port the operator interface to any of the numerous environments that currently run the X window system. In addition, care was taken to place all external variables in allocated memory so that our code would easily port to shared memory, multitasking environments such as VxWorks. This gives us the option of creating a stand-alone system that hosts both the input-output controller and operator interface software without the need for a communications network.

Under UNIX, the display manager can run each display with a separate process or it can run many displays from one process. The latter provides for improved display startup time, because we don't have to wait for process creation to activate a display. A file descriptor manager library based on the system call select ( ) provides the display manager with the ability to respond asynchronously to events from both the X window system and channel access simultaneously.

The display manager does not queue up each update from channel access prior to making a graphics update on the screen. If so the display manager would grow further and further behind if updates were generated more rapidly that they could be consumed in the form of graphics updates. Instead, the display manager writes each update into memory and sets a modified data tag. When all updates have been dispatched as above then a list is scanned for modified data and the screen is updated. This method allows display manager to react asynchronously when the updates are infrequent and gracefully degrade to synchronous operation when the updates are continuous.

The display manager and editor both share a common custom graphical user interface tool kit written specifically for process control applications. When the project began the XView and Motif tool kits were unavailable. The project athena and HP widget sets were available at the time but did not provide the features we needed. Attempts to use the Xt subclassing techniques to extend the functionality of these tool kits still did not provide the features we needed.

Another factor contributing to our choice of tool kits was our decision not to create virtual windows with the X libraries for each of our output-only graphics devices such as meters, bars, and indicators. This choice improves start up time and decreases overhead since some of our displays have a large number of these devices. Keyboard and mouse input events generated under the X window system are associated with the virtual window that generated them. The X window system also provides a shifting type coordinate translation for all drawing operations performed within a virtual window. Both of the above are the normal benefits of using virtual windows. However, our output-only graphics devices do not receive mouse or keyboard input. Similarly, we must already perform a scaling type coordinate translation so that graphics devices will appear on the display with the proper height, and width as specified by the display list. All X tool kits investigated create a virtual window for each object.

Each of our user input devices such as menus, keyboard input, valuators, and bottons does have an X library generated virtual window associated with it. We use the hash table routines built into the X libraries to translate window identifications provided by X into a pointer to our structures containing the graphics input device configuration and state information.

CONCLUSION

We have been using our new operator interface in some installations for over a year now. The ASCII version of the display list has allowed us to continually add new features without sacrificing upward compatibility.

In both the first and second generation each display could start up several others. We prefer to set up a hierarchy where each time a display initiates another we see an increasingly detailed view of the process. The second generation operator interface's order of magnitude improvement in display start up time has made this organization practical. Likewise, faster start up times allow us to draw many reusable small displays instead of one display with a large portion of the process on it.

Our experience with the X window system has been positive. Finally, we can write graphics code that does not become obsolete with the graphics hardware. The client private color table and scaleable fonts that appear to be in release five of X11 would have saved us effort if they had been available at the start of this project.

Graphical user interfaces are arguably a revolution in computer accessibility. However, the time spent writing graphical user interfaces is a large and growing percentage of the total time devoted to a project. We have tried to reduce this percentage by providing a software tool that allows process control based on graphical user interfaces to be configured without writing any new source code.

References and Comment

1. Our control system has been renamed from LAACS to The Experimental Physics and Industrial Control System, or EPICS, since the last ICALPECS to better reflect the collaborative effort now in place.
2. Richard Rothrock, "A Workstation Operator Interface for Accelerator Control Systems: A Method for Separating Graphics Code from Control Functionality", Europhysics Conference on Control Systems for Experimental Physics, Sept. 28 – Oct. 2, 1987, Villars sur Ollon, Switzerland.
3. L. R. Dalesio, "The Ground Test Accelerator Operator Interface: The Second Generation", International Conference on Accelerator and Large Experimental Physics Control Systems, October 30 – November 3, 1989, Vancouver, Canada.