A Scientific Workstation Operator-Interface for Accelerator Control

V. Paxson, V. Jacobson, E. Theil
Lawrence Berkeley Laboratory, Berkeley, California 94720

M. Lee
Stanford Linear Accelerator Center, Stanford, California 94305

S. Clearwater
Los Alamos National Laboratory, Los Alamos, New Mexico 87545

Abstract

Research in human factors has demonstrated that people use computers more efficiently and effectively if they have a highly visual interface to the machine. Today's scientific workstations provide sufficient power to implement such interfaces. By using these workstations, an operator interface for an accelerator control system can be built which is powerful, flexible, and easy to learn.

We discuss such a system currently being developed on a Sun-3 workstation. The system is designed as a set of building blocks which can be run independently or linked together. This *toolbox* approach gives the operator the ability to execute precisely those programs needed for the task at hand. Each program runs in a separate window and communicates with other running programs via a common data base. When the operator makes a change in one window, the effects are then shown in the other windows.

Introduction

Modern scientific workstations have enough power and capabilities to implement an operator-interface for an accelerator control system that greatly increases the effectiveness with which the accelerator can be controlled. To explore this possibility, we have developed a prototype system on a Sun-3 workstation. We begin by discussing the factors which make workstations ideally suited for this task, the "toolbox" design philosophy we have followed to assure that the operator-interface system has a high degree of generality and portability, and the value of using a *modeling* program to predict the accelerator's behavior. We then discuss the three main components of the system: a set of prototypes for control "tools", a language for describing pictures in a high-level, portable fashion, and an accelerator simulator which can both show the effects of lattice element errors on the beam trajectory and, given a trajectory, find the errors which caused it.

The Opportunities Presented by Workstations

One of the main factors determining how productively people use computers is the richness of the user-computer interface [1]. From the days of keypunches and stacks of printer output up until today, getting information into and out of the machine has proven to be a major bottleneck. The principal goal in entering information into the machine is that it be easy to specify and error-free; in extracting information from the machine, that it be presented in a way that highlights its key features. As the user-computer interface has grown richer, the user's productivity has increased manifold.

Recent years have seen hardware and software advances [2] which have made possible a user-computer interface of power and flexibility much beyond what has been possible in the past. There are four key advances, all of which are integrated into scientific workstations:

- **High-resolution graphics** — A modern high-resolution monitor can display over one million points, in hundreds of different colors if need be, providing for an extremely high information content. Such pictures truly can be worth a thousand words, though care must be taken that the visual information present is not overwhelming.

- **Windows** — Given high-resolution graphics, it becomes possible to display a number of different images at the same time. When working on several related tasks, the user can see all of the tasks *together*, and thus quickly grasp the interconnections between them. Tasks not immediately of interest can be made *iconic* — that is, they shrink to a small size, to avoid cluttering the screen, and display a picture suggestive of their function. They continue to run, and when the user wishes to return to the task, she or he simply "opens" it and it assumes its full size again.

- **Mice** — With the user-computer interface more visually oriented, a new form of input is possible: the pointing device, typically implemented as a mouse. For some types of input, pointing is superior to using a keyboard: in particular, with a mouse the user can designate objects without needing to know their names or how to describe their positions. If the objects are clearly-identifiable pictures, the user will make fewer mistaken designations than if using a keyboard.

- **Pop-up menus** — In general, menus aid the user in effectively using the machine by listing the operations possible at any given point. With pop-up menus, the user can summon the list on demand, without having it always present and taking up display space. Thus the visual information in front of the user is kept limited until such time as the *user* decides to add to it by calling for the menu.

Because of scientific workstations have these features and are low-cost, they are the natural choice for developing the next generation of operator-interfaces for accelerator control.

Toolboxes

When dealing with complicated systems such as accelerators, the tasks that an operator might wish to do are so diverse that it is virtually impossible to anticipate all of them and write a specific control program for each. The key to controlling such a system is to determine the fundamental components of the possible tasks and implement each *component*, rather than each possible combination of components. This approach yields a set of *tools*, and with them *generality:* given these tools and a way to link them together, *any* control task can be constructed from its basic pieces. We shall present examples of such tools shortly.

Modeling Programs

A key component of our operator-interface system is the use of a modeling program to predict the behavior of the accelerator. Not only does the modeling program enable us to develop and test the system off-line, but even more importantly, by using the model the system can compute the effects changes would have on the accelerator without the changes actually being made. Thus the operator can propose a change, see how it would affect the machine, and *then* decide whether to go ahead and make the change. With this approach the operator is encouraged to explore different ways of controlling the machine without the risk of damaging it or the cost of spending beam-time.

We use COMFORT [3] for our modeling program, though we have taken care that little knowledge of COMFORT is incorporated in the system. Our system stores models in an internal format, and uses conversion programs to translate this format into the format COMFORT expects and back. Because knowledge of COMFORT is isolated, it can easily be replaced with a comparable modeling program.

Prototypes of Control Tools

To explore the possibilities of operator-interfaces using a workstation we built several control tool prototypes. Each one takes as input a file describing

Work supported in part by the U.S. Army Strategic Defense Command and the Department of Energy, contracts DE-AC-03-76F00515 and DE-AC-03-76F00098.

the accelerator lattice: a *model* of the accelerator. Since these programs are prototypes, they lack the uniformity that the completed interface system must have, and they are discussed here not as examples of a final product but for what they reveal about what *can* be done.

We initially implemented four tools:

- **Beam Position** shows the beam envelope in the X and Y planes, X in red and Y in blue. As with the other displays, this one can be stretched or shrunk to whatever size the operator wants, and the display provides for *zooming:* the operator can blow up an area of interest or back out and see the position of the envelope relative to the beampipe.

- **Top View** draws a top view of the accelerator showing the various magnets to scale. The beam envelope, magnified by a factor of 1000, is superimposed on top of the image, making **Top View** particularly useful for understanding what is physically happening to the beam. For example, the focussing of the beam by quadrupoles is readily apparent.

- **Twissplot** plots any one of five Twiss parameters (beta, tune, dispersion, derivative of dispersion, and alpha) in the X and Y planes. Below the plot is a pictorial representation of the lattice. By clicking the mouse over a lattice element the operator can identify the element and then enter a new value for it. When the operator clicks on the "COMPUTE" button, **Twissplot** runs the modeling program and displays the new Twiss parameters. The previous parameter values are plotted in contrasting colors.

- **Working Diagram** shows the current tune point on a picture of the working diagram. Using a pop-up menu the operator can select the order of resonances to plot. The operator designates a new tune value by pointing to its location on the diagram (after possibly zooming in) with the mouse. The operator can then instruct **Working Diagram** to run the modeling program in an attempt to effect the tune change. If the modeling program is able to find a satisfactory group of magnet settings, **Working Diagram** deletes the old tune marker and draws the new one.

These tools are linked together using the **Control** program (See Fig. 1). **Control's** display shows the icon image of each of the tools. The operator activates an instance of the tool by clicking the icon. When a tool is activated it is given a copy of the current model being worked on. Whenever the operator makes a change with *any* tool (for example, selecting a new tune using **Working Diagram**), the updated model is sent to *all* of the other active tools. In this way, each tool always reflects the current state of the model. Furthermore, when a new tool is activated it is given a copy of the previous model as well. This approach gives the operator a great deal of flexibility. For example, if the operator makes a tune change and *then* decides to see how the change affected the beam envelope, she or he can activate **Beam Position** and see not only the current beam envelope but also the previous envelope.

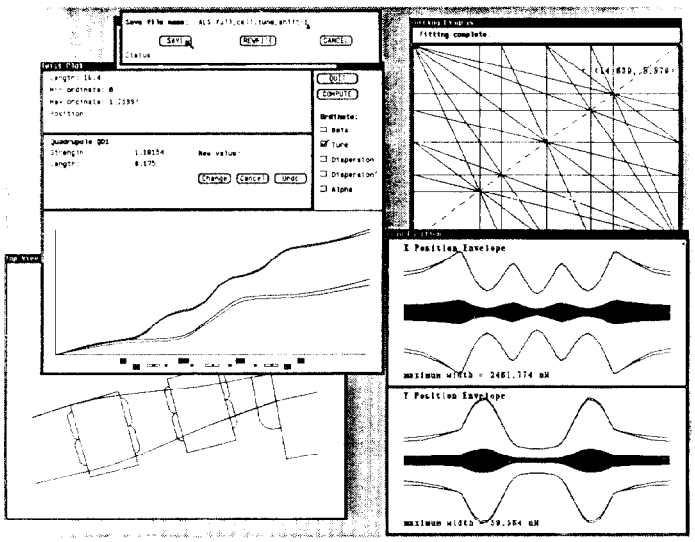

Fig. 1: The four control displays, after a tune change has been made.

Because **Control** maintains both the current and the previous versions of the model, the operator can *undo* a change by summoning the pop-up menu and selecting the "Previous Model" option. **Control** also supports saving models to disk and loading them back. These features are designed to encourage experimentation and provide for quick recall of successful configurations.

We developed a sixth tool — **Orbit Correction** — which has not yet been integrated into **Control** (See Fig. 2). **Orbit Correction** explores an idea that we later developed more fully in **PLUS** (discussed below): providing a tool which the operator can use to experiment with both *manual* and *automatic* control of the machine. **Orbit Correction** has two displays. The top one shows the current readings at the BPMs, the bottom one the current corrector strengths. The operator can change a corrector's strength by clicking the mouse above or below the image of the corrector. The change in strength is proportional to the distance from the corrector to the mouse. Because the tool can compute the new orbit resulting from the change very quickly, the operator instantly sees the effects the change has on the orbit. Thus, the operator can *manually* correct the orbit by simply adjusting correctors and watching the effects, continuing until the beam has been flattened.

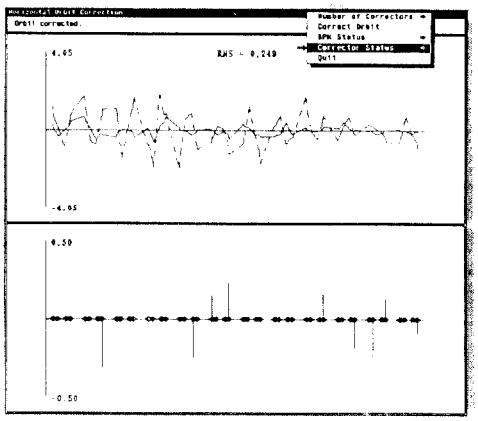

Fig 2: The Orbit Correction display, showing the orbit before (dashed) and after (solid) the tool has flattened the orbit.

Orbit Correction can also search for the proper corrector settings *automatically*. The operator selects the number of correctors to use, turns off any faulty monitors or correctors, and then picks the "Correct Orbit" pop-up menu option. The subroutines used to correct the orbit are fast enough to provide close to real-time feedback (under 10 seconds to smooth an orbit scanned at 72 BPMs, using 48 correctors), but do not take into consideration limits on corrector strengths (again, see **PLUS** below).

Picture Description Language

To fully exploit the opportunities presented by workstations' high-resolution graphics, one needs a way to compose pictures easily and quickly. We addressed this need by designing and implementing PDL (Picture Description Language), and a tool, **Picture,** for displaying pictures and interacting with them.

The main goal in designing PDL was that it be possible to describe pictures with a minimum amount of effort. A few basic shapes are provided — squares, polygons, lines — along with ways to stretch, rotate, and piece together the shapes to make aggregate shapes. These aggregate shapes can then be transformed and moved around just like the basic shapes. For example, to represent quadrupole magnets by using a box 2.5 units wide by four-thirds of a unit high, with the diagonals drawn in, we would write:

```
a quad is:
        box scaled 2.5, 4/3
        line from last box . top left to last box . bottom right
        line from last box . top right to last box . bottom left
```

Now we can refer to "quad" to mean that shape. To get the quad shape twice its normal size, rotated 30 degrees counter-clockwise, and named "QF-1":

quad "QF-1" scaled 2 rotated 30

Numerous synonyms are available for the language's keywords. These cut down on the need for memorization.

Pictures described using PDL are compiled into an intermediate form which is not specific to any particular graphics package. To display pictures one writes an interpreter which will read the intermediate form and perform whatever operations are necessary to draw it using the local graphics package. Because pictures are not compiled directly into machine-specific graphics, the PDL compiler is easily portable to other machines. Therefore a library of pictures can be composed without worrying about them becoming obsolete due to a change in hardware or graphics software. Also, the people who create the pictures need not have any special knowledge about graphics or programming.

In our system, PDL pictures are interpreted and displayed by the **Picture** tool. Applications send messages to **Picture** specifying objects to highlight, display text, zoom areas, and pop-up menu choices, and receive messages telling which object the user has clicked on or which menu choice has been selected. With this structure, applications do not need to know *anything* about graphics, windows, or menus.

Our prototype picture application is **Beamline** (shown in Fig. 3). It draws a beamline switchyard and responds to commands from the user to identify, activate, and deactivate components in the beamlines, zoom in on a beamline or back out, and extend the current object of interest from an individual magnet to the beamline that the magnet belongs to. The program is under 300 lines long.

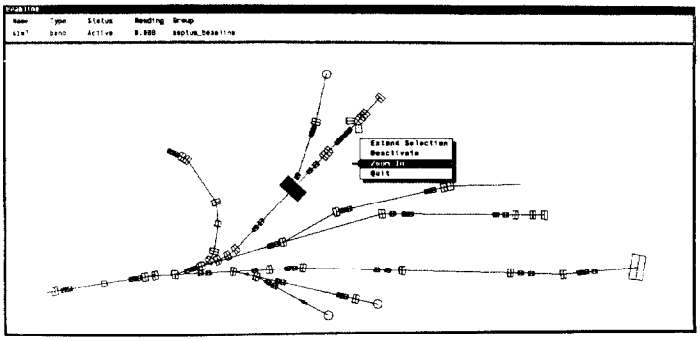

Fig 3: Beamline display showing the Bevatron Switchyard

PLUS

The last tool we developed, and the most sophisticated in terms of its user-interface, is **PLUS** [4], a program which simulates the effects of lattice errors and correctors on a beamline and, given trajectory data, finds the sources of errors or the corrector settings necessary to compensate them (see Fig. 4). Since **PLUS** can generate simulated trajectories, it can be used off-line for training and study of the accelerator. When using measured beam data, its error-finding capabilities make it well suited for commissioning, and its error-compensation for day-to-day tasks such as beam steering.

When using **PLUS**, the operator chooses the type of error or control to simulate: focus, kick or entry error, or beam steering. The **PLUS** display can show either two sets of beam readings (typically a measured set and the set predicted by the model) or the differences between the two. The lattice elements or the BPMs are shown below the readings. To control an element the operator clicks on it (or selects it from a pop-up menu, if the name is known and the location is not). The element and any others it is ganged with are highlighted, and its name and current value displayed below it. Using the menu, the operator attaches a knob to the element. A user-adjustable scale is shown to the left of the readings. As the operator holds down a mouse button and moves the mouse up or down, the knob value tracks the mouse motion. When the mouse button is released the element is changed to the new value, and **PLUS** plots the resulting trajectory.[1]

[1]PLUS also has an option to continually plot the trajectory corresponding to the knob value as the knob moves, making it easy to tweak an element until a desired trajectory is achieved.

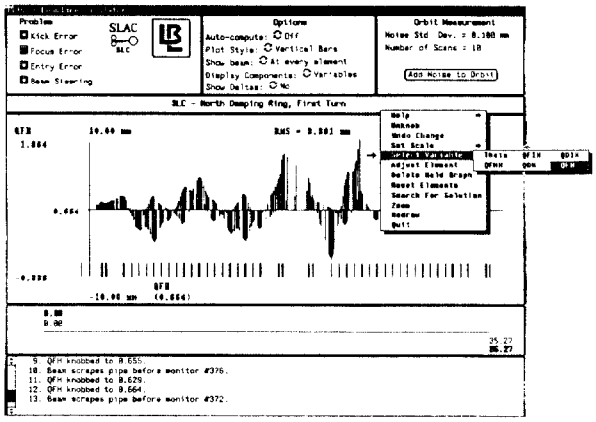

Fig 4: PLUS being used to search for focussing errors

To use **PLUS** to find errors, the operator selects a group of elements which are possible error candidates and instructs **PLUS** to search for a set of errors among them which will match the current trajectory. **PLUS** uses a powerful optimization package, together with its ability to generate simulated trajectories, to find the errors. When found, **PLUS** writes the values into a log window below the main display and plots the resultant trajectory. For beam-steering, the operator selects a monitor or set of monitors and moves the mouse through the desired monitor range (maximum and minimum acceptable reading), which are then marked on the display. Because the optimization package can deal with constraints, the operator can assure that the solution **PLUS** finds will be physical — none of the correctors will be set beyond their capabilities.

Summary

Our operator-interface prototypes demonstrate some of the richness which a scientific workstation provides for large, complex tasks such as accelerator control. By emphasizing control *tools*, and making their use highly visually-oriented, one can design an operator-interface system which is tailored to the operator's needs, rather than the limitations of the hardware and software.

When using graphics and window packages to write a complicated software there are two dangers. The difficulty of mastering the packages can overwhelm their usefulness, and the software can be so enmeshed in the particulars of the packages that it becomes impossible to transport it to another environment. We have attempted to address one instance of these difficulties by creating the PDL picture description language. When using PDL for creating pictures, programmers need not master the intricacies of specific graphics, windows, and menu implementations, and since the PDL compiler generates an intermediate form, the use of the language is not limited to any particular environment.

Much work remains for our system. Our toolbox needs many more tools before it is complete, and the operator-interface — the use of the mouse, the way of accessing pop-up help messages, the visual layouts — needs to be fully uniform across all of our tools. The prototypes have shown that these goals are attainable, and that with them we will gain an operator-interface which will be significantly easier, more effective, and more natural to use and learn with than previously possible.

References

[1] B. Shneiderman, **Designing the User Interface,** Addison-Wesley, 1987

[2] E. Theil, V. Jacobson, V. Paxson, "The Impact of New Computer Technology on Accelerator Control", presented at this IEEE Conference, 1987

[3] M.D. Woodley, et. al., "COMFORT, Control Of Machine Functions OR Transport Systems", 1983 IEEE PAC, Sante Fe, New Mexico

[4] M. Lee, et. al., "Modern Approaches to Accelerator Simulation and Online Control," presented at this IEEE Conference, 1987