Conf-920966--12

Fermi National Accelerator Laboratory

FNAL/C--92/259

DE93 004409

Received by OSTI

Using Workstation GUIs in HEP, X-Windows, Motif and the Nirvana Project

M. Edel, J. Kryiakopoulos, P. Lebrun, B. Ren and J. Kallenbach

Fermi National Accelerator Laboratory
P.O. Box 500, Batavia, Illinois 60510

K. Iourcha

Petersburg Nuclear Physics Institute
188350 Gatchina, Leningrad district, Russia

October 1992

Presented at the *Computing in High Energy Physics Conference*, Annecy, France, September 21-25, 1992

MASTER

Operated by Universities Research Association Inc. under Contract No. DE-AC02-76CHO3000 with the United States Department of Energy

DISTRIBUTION OF THIS DOCUMENT IS UNLIMITED

Disclaimer

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

Using Workstation GUIs in HEP, X-Windows, Motif, and the Nirvana Project

Mark Edel, Joy Kryiakopoulos, Paul Lebrun, Baolin Ren, Jeff Kallenbach
Fermi National Accelerator Laboratory, P.O. Box 500 Batavia, IL 60510, USA
Konstantine Iourcha
Petersburg Nuclear Physics Institute, 188350, Gatchina, Leningrad district, Russia

We present four small, high quality, Motif based tools for high energy physicists and discuss some of the less obvious work that is necessary to fully take advantage of graphical user interfaces (GUIs). Histo-Scope and NPlot are interactive data display programs. Histo-Scope is for viewing data as it is collected in running analysis or data acquisition programs, NPlot, for plotting data from text files. Visajet is an interface for the ISAJET event simulation program. It gives users a fast way to configure ISAJET and displays ISAJET events in a three dimensional phase space display that users can rotate and manipulate with the mouse. NEdit is a GUI style plain text editor.

Introduction

For the vast majority of users, the menu/window/dialog style of interface is a very important achievement. Software that once took weeks to learn can often be used immediately without training. Graphical user interface (GUI) environments seamlessly integrate graphics and mouse interaction, giving users the ability to directly manipulate graphics on the screen, and optimally allocate screen space with overlapping windows. Unfortunately for HEP, these benefits come primarily by trading programming time for user time. Providing software that actually takes advantage of them is harder than most people think.

This paper presents a project dubbed "Nirvana" which provides a library of graphical user interface tools for the high energy physicists' workstation that employ GUIs to their fullest advantage. It reviews our experience with this new technology and its applicability to HEP.

The Real Advantages of GUIs

Replacing a command line or character based interface with panels of buttons and text fields may make a program look more attractive, but it is just as likely to make the program harder to use if done improperly. The real power of graphical interfaces comes from very careful design, consistent application of design rules, and knowledge of the task. Applying these rules can require huge amounts of extra work that is almost invisible in the end product. Unfortunately, the same is true for most of the benefits that people associate with GUIs. Taking advantage of direct manipulation, for example, usually means hand coding animated graphics with little or no support from the GUI libraries.

Below are some examples of the kinds of design guidelines that contribute to an effective interface. A well written GUI application should:

- Be consistent in its reactions to user actions regardless of context, and consistent with other applications on the same system.
- Make optimal use of limited screen space.
- Be efficient for expert users and have a natural transition between novice and expert.
- Present clear choices. All of the capabilities of a program should be made immediately apparent and available, in any order, from the top level of the program.
- Make every operation undoable, or warn the user that the operation is irreversible. Every dialog should have a cancel button.
- Provide positive visual (or audio) feedback for every user action. Give users a sense of completion, so that they don't worry that some further action is required of them.

- Be continuously responsive to user input. When the system is busy, the program needs to indicate it clearly. Long operations should have progress indicators.
- Make it difficult for the user to enter bad data, and when an error does occur, explain what is wrong as specifically as possible.

The end result of this kind of careful design is an interface that seems almost invisible. For physicists, this means they can concentrate on their analysis rather than on the software.

The Nirvana Tools

The computer of choice for HEP analysis and data acquisition problems is usually a high performance CPU running Unix or VMS. The GUI environment on most of these computers is X Windows with the Open Software Foundation's Motif window manager and widget set. Despite bugs, poor design, and poor performance of X/Motif, it is possible to realize nearly all the advantages experienced by users of better designed GUI environments.

The appropriate software tools in a workstation GUI environment are small inter-operable applications. Smaller applications with simpler interfaces can be used together effectively on a single screen because of the workstation's support for windows, cut and paste, and inter-process communication. Giving the small applications interfaces that are consistent with one another ensures that users' knowledge of the operation of one will easily transfer to the operation of the others. Consistency is fragile and can be easily undermined by one bad interface.

Histo-Scope & NPlot

Histo-Scope is a tool to select and display histograms, n-tuples, and scalar values from a program as data is being created or analyzed. Using Histo-Scope, physicists can interactively "browse" through the large quantities of statistical data that their analysis and data acquisition programs gather as they run. It is intended to complement existing physics applications, providing immediate access to data while a program is running, as well as new interactive methods for viewing data.

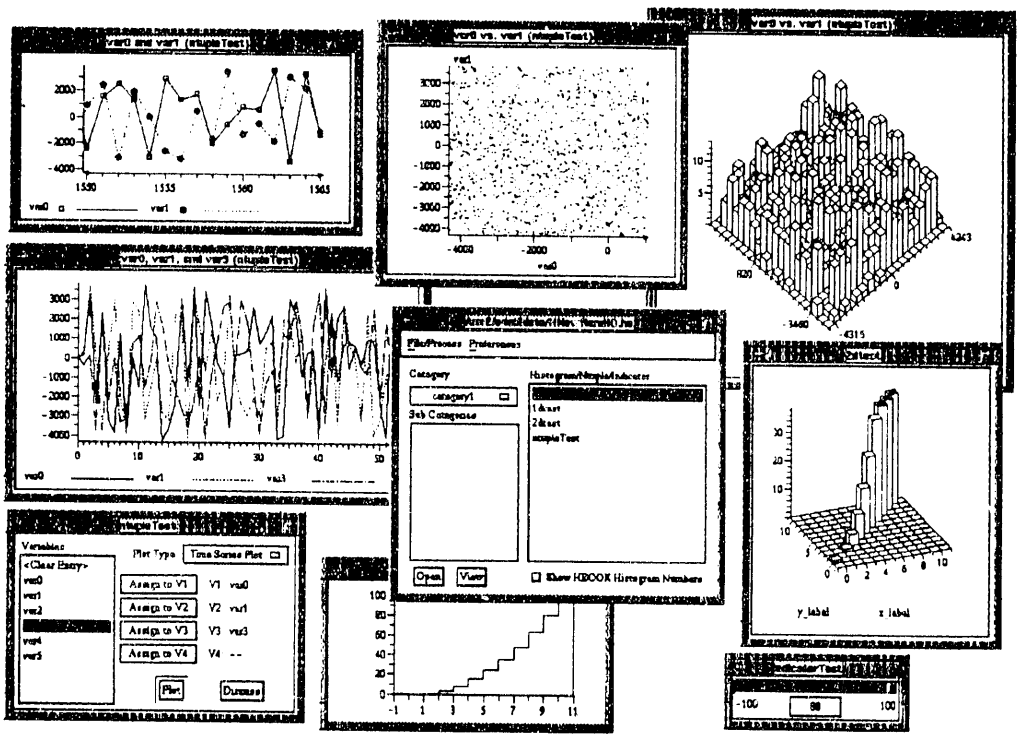

Figure 1. Viewing Data with Histo-Scope

Histo-Scope has two parts. The first is a small library of routines which can be inserted in physics analysis or data acquisition code without significantly changing its behavior. No restrictions are

placed on the user analysis process except that it must periodically call an update routine. It does not need to run on a workstation, and is not linked with any graphical user interface code.

The other part is the "scope" process. Invoked upon user demand, the scope requests and displays data continuously from the analysis process. The scope concentrates on interacting effectively with users. It responds to mouse and keyboard input and provides the interactive graphing and plotting that enable users to view their data quickly and effectively. The scope program can also read HBOOK and Histo-Scope format files.

NPlot is a tool for quickly plotting columnar data from text files. It is a simple re-packaging of the Histo-Scope n-tuple interface with a file reader.

Both Histo-Scope and NPlot produce highly interactive graphs and plots. These include: multi-variable graphs, two and three dimensional scatter plots, and one and two dimensional histograms. Users can rescale, zoom, and pan these plots by dragging on axis scales and other sensitive areas. The three dimensional widgets, 2D histogram and 3D scatter plot, can be rotated accurately using the mouse as a hand on a "virtual trackball" sphere surrounding the plot. The plots can also be combined with animation sliders to reflect additional variables or to rebin histograms.

NEdit

NEdit[1] is a GUI style plain text editor. It was originally intended as a project benchmark, to establish standards for program structure, file handling, accelerator keys, and general GUI operation, but not to be released for widespread use. However, it has since proved itself to be a superior replacement for the conventional Unix text editors. NEdit serves as both a good introduction to Motif based tools, and an illustration of the advantages of GUIs. It supports programmers with features such as: auto-indent, block indentation adjustment, and parenthesis matching. People who are used to character based editors like edt, vi, and emacs are usually hesitant to try a new editor because of the time investment in learning the commands. NEdit, on the other hand, requires no such investment. Though it appears much simpler, the functionality that it provides is as complete. A naive NEdit user can usually work faster than an expert edt user!

Visajet

Visajet[2] is a GUI front end to the ISAJET[3] Monte Carlo event generation program. Physicists can use Visajet instead of composing an ASCII file of "input cards" ISAJET uses for its run parameters. Running Visajet allows the physicist to see all the options that are available, explore them, glance through default values, and change them in preparation for a run. Visajet then allows the physicist to start ISAJET under control of a run panel and provides graphical routines to visualize the event data as ISAJET is running. Like Histo-Scope, the Visajet process is separate from the ISAJET process, allowing the event generating process to reside on another machine.

Programming Utilities

To develop the above applications, we created a library of common software components. These include: PostScript drawing routines which parallel X calls for hardcopy output; dialogs for printing, opening files, saving files and help; modal dialogs for errors warnings and simple prompts; interactive plotting widgets; support for Greek, superscript, and subscript characters; and convenience routines to simplify Motif programming.

The most sophisticated components that we have produced are, of course, the interactive plotting widgets used in Histo-Scope and NPlot. These can be used like any other Motif widget, and supply the complete direct-manipulation interface that they do in the programs.

Our Experience

Although the products that we have created work very reliably and are easy to use, their development was painful because the design and documentation for both Motif and X-Windows are extremely poor. When we started the project, there were serious bugs in both X and Motif that actually precluded their use in a number of areas. Though the quality of X and Motif has been improving continuously, many bugs still exist and we have had to put serious effort into working around them.

Even though we were working on machines capable of incredible graphics performance, we had to invest considerable development time to achieve adequately fast, smooth animation because of inefficiencies in X-Windows. For simple menu and dialog interaction, the combination of X and Motif require a 10 MIP machine with 16 megabytes of RAM to achieve the graphics performance of a Mac Plus (a 0.5 MIP machine with 256K of RAM).

Creating consistent interfaces in the Motif environment was challenging. We began with the Motif Style Guide. Where there were holes in the Motif standards, we filled in from the Macintosh and Microsoft Windows. Unfortunately, the Motif Style Guide is mostly holes, and interface styles already vary widely among the commercially available Motif software packages. The Open Software Foundation does not seem to consider consistent interfaces to be an important goal. Properly designed programs must support two different pointer focus modes, different menu selection styles, different font sizes, and endless user tailoring of appearance. There are no standard types for exchanging data other than text so, for example, programs are not guaranteed to handle cut and paste of graphics from other programs.

As of yet, there is no really quick way to generate good graphical interfaces. There are aides that can make programming GUIs easier, and some environments are much easier to program for than others. For example, we could develop interfaces much faster if physicists were willing to move to the NeXT computer. From the point of view of GUIs, NeXT Step is superior in all respects, and tailor made for users like us who would like to develop GUIs less expensively for limited numbers of users. Of course, NeXT Step is proprietary, does not run on the fastest CPUs, and except for developing GUIs, its programming environment is no better . For Motif, Interface building tools are available and we employed them in the initial stages of all of the products. Motif interface builders are good for initially laying out panels of buttons and controls, but become progressively more burdensome as you complete an interface and try to make it efficient for expert users.

Despite the problems, as our experience with the technology and our library of GUI components grows, it becomes easier and faster for us to create similar software.

Conclusion

Commercial software developers usually estimate the percentage of development time spent on the user interface to be about 75%. In HEP, this additional time is hard to justify where a software product may have less than 50 users.

Because GUIs can quadruple development time for a project, they are not necessarily desirable on all HEP applications. To realize the benefits of GUIs in HEP, we need to identify programs that are heavily used, or that depend on the unique properties of GUIs. This paper is too short to adequately cover the topic of creating complete, efficient, and consistent interfaces, except to say that it is well understood, and the benefits can be considerable. There are a number of good books on the subject, such as Designing the Interface[4], by Ben Schneiderman. Also see [5].

[1] NEdit Users' Guide, Fermilab Computing Division Library document #PUO135

[2] Visajet Users' Guide, Fermilab Computing Division Library document #PUO136

[3] ISAJET 6.34, A Monte Carlo Event Generator for P-P and Pbar-P Reactions, F. Paige and S. Protopopescu, Brookhaven National Laboratory, Upton, NY 11973.

[4] Designing the Interface, Ben Schneiderman, Addison Wesley, 1987.

[5] Programming Graphical User Interfaces (Notes from 1/22/91 Talk). Fermilab Computing Division Library document #EN0089.

DATE
FILMED
2/8/93