ANL/CP--74827

DE92 004716

Man-Machine Interface Builders at the Advanced Photon Source[1]

Mark D. Anderson
Advanced Photon Source, Argonne National Laboratory,
Argonne, Illinois 60439

Abstract

Argonne National Laboratory is constructing a 7-GeV Advanced Photon Source for use as a synchrotron radiation source in basic and applied research. The controls and computing environment for this accelerator complex includes graphical operator interfaces to the machine based on Motif, X11, and PHIGS/PEX. Construction and operation of the control system for this accelerator relies upon interactive interface builder and diagram/editor type tools, as well as a run-time environment for the constructed displays which communicate with the physical machine via network connections. This paper discusses our experience with several commercial GUI builders, the inadequacies found in these, motivation for the development of an application-specific builder, and design and implementation strategies employed in the development of our own Man-Machine Interface builder.

1 Introduction

Experimental physics facilities have traditionally challenged state-of-the-art computing techniques and hardware, yet often are bound to less than current interface and interaction models. The success of interface technologies descendent from Xerox PARC research (and the Apple Macintosh), coupled with the extremely rapid advances being made in these and related technologies (e.g., inexpensive, powerful graphics workstations, network oriented graphics systems, GUI toolkits, etc.) has raised the expectations of scientists and engineers involved with these facilities. Command line interfaces are no longer adequate for the dialogue between scientific staff and the complex instruments they operate.

[1] Work supported by U.S. Department of Energy, Office of Basic Energy Sciences under Contract No. W-31-109-ENG-38.

2 Advanced Photon Source (APS)

Particle accelerators are physically large, complex experimental physics machines of linear or circular geometry with lengths of several meters to many miles. These machines accelerate particles (e.g., electrons, positrons, heavy ions) to high energy thus forming a beam. Accelerators can be composed of several particle energizing, extraction, accumulation, injection, acceleration, and storage ring stages. The control of these systems is accomplished through sophisticated hardware and software.

The Advanced Photon Source (APS) is a 7-GeV[2] circular positron accelerator and storage ring with circumference of about two-thirds of a mile. The facility will be used for research in a variety of areas, including chemistry, physics, medicine, materials science, and biotechnology. Commissioning of the facility is scheduled for 1996.

As in other large software projects, tens or hundreds of man-years of effort are frequently put into accelerator control systems, and, as in other software projects, an attempt is being made to maximize the return on this huge investment. Software for this facility is being written in a portable, vendor-neutral, standards-based manner. The APS has adopted ANSI-C, POSIX[3], X11, Motif, and PHIGS/PEX[3] as the vehicles for software development, capitalizing on the functionality and investment protection offered by these standards.

The APS control system consists of high-performance Unix-based workstations at the operator inteface level connected to VME-based processors at the field level using TCP/IP protocols over high performance networks. The VME processors control front-end electronics and provide equipment interfaces. This paper discusses the operator interface level of this system. For a complete discussion of the APS control system, see [McDowell et al. 1991] or [Dalesio et al. 1991].

2.1 User/Operator Environment

Given the complexity of a system such as APS and the information density necessary to monitor and control it, a direct manipulation, graphical interface mechanism is required for efficient, comprehensive control. Primary components of this graphical interface mechanism are X11 and Motif. X11 has been chosen for its power, portability and momentum (i.e., there is no

[2] 1 GeV $= 10^3$ electron-volts

[3] proposed use

obvious contender). Motif has been chosen for its functionality as well as its portability and vendor neutrality.

2.1.1 X11 and Motif

X provides the basic windowing and bitmapped graphics functionality required in the APS environment. Augmenting the capabilities of Xlib and Xt, Motif provides a rich set of widgets supporting a large set of interaction techniques and physical metaphors. However, Motif does not offer a complete or sufficient set for experimental physics environments. Various instruments familiar to scientists and engineers which can extend the metaphoric quality of an interface include, for example, meters, indicators, bars, and strip charts. Hence, domain-specific interface objects are a necessary part of the environment and any tools used will support these.

2.1.2 Another Dimension

A dimension of machine or system interfaces not present in most other applications which are wholly software-based and conceptually modelled is *external dynamism*. *With external dynamism the interface both represents the current state of the physical machine or external system and provides the facilities to effect changes in that system.* Implicit in this definition is the ability of the external system to change state independently of the interface.

This dimension adds an additional level of complexity to traditional graphical user interfaces.

2.1.3 Objectives in Accelerator Systems Control

Some key objectives for accelerator systems control toolkits of interest here are identified in [LANL 1988]. These include:

1. reduction of requirements for conventional computer programming in applying control systems to accelerators, and

2. provision of a friendly interface to the user for implementation of an application.

It is these objectives which can be exploited with graphics tools and interface builders.

3 GUI Builders

In the past year, several Motif GUI builders have appeared and are commercially available. This initial generation of builder tools offers the following features and functionality, with the promise of reduced programmer effort, shorter development cycles, greater flexibility in interface design (through interface prototyping), and increased project productivity:

- interactively draw or paint Motif interfaces, set widget and gadget resources
- tie callback routines to the interface at interface development time (with at least one product offering a built-in C interpreter)
- create UIL or C source code (K & R or ANSI-C)
- promote standards (X11R4 and Motif 1.X)
- cut development time, build and modify an interface without time-consuming compile-link cycles
- lower learning curve of X Toolkit and Motif
- import UIL code from existing interfaces
- prototype instantly, eliminate "bad" interfaces early in development
- WYSIWYG operation

3.1 Generation 1

Evaluation of most of these products yielded the conclusion that this initial generation of products was largely a step above manual coding for simple Motif interfaces, but inadequate for larger systems. Generally, using these interface builders was *cumbersome*. Some characteristics of or comments on this first generation of GUI builders are:

- no obvious support of multiple application contexts or for modularizing of a generated interface
- lack of a grouping function, to select and operate on a set of widgets
- interface developer (despite claims to the contrary) required to have a very significant command of Motif widgets and resources

- builders tended to coerce developers into setting resource values in the application rather than in resource files (although there were switches to say 'put this in an app-defaults file' you generally had to do that extra operation for all resources you set)
- navigation through the constructed widget hierarchy was very awkward
- very large interfaces could not be broken down into conceptual sub-interfaces or 'scoped'
- properties not able to be expressed in the builder could be hand-edited into the output of the builder, but this almost invariably meant that the constructed interface could not be re-input and operated on by the builder
- interface editing was very much on a per-widget basis (e.g., to set the major interface trees to a specified background color that resource had to be edited for all affected widgets).
- could not incorporate custom widgets into the builder (as a feature this was usually advertised, but 'Not Yet Implemented')
- could not customize the builder to make it accessible for non-Motif and Xt-literate staff
- absence of any integral draw or paint facility for static rendering

3.2 Generation 2

Just currently being released is the second generation of most of the GUI builders. Many of the issues raised above have been expressly addressed and many of these inadequacies no longer exist. These new products offer more features, including:

- expanded customization features (e.g., editing of widget class resources can be tailored to reflect project or organization style guides)
- widget grouping functions (e.g., move sets of widgets as a whole)
- multiple-widget selection - allow resource specifications to apply to sets of widgets

- drag-and-drop operations (more direct manipulation interaction styles)
- widget hierarchy browsers, with selection and editing functions operating on this model of the interface, as well as the WYSIWYG manifestation
- greater support for large software projects, including internal utilities and interfaces to external software tools (e.g., editors, debuggers), including data and message-driven tool sets
- extensibility - full integration and support of custom widgets
- customizability - expand or simplify the builder according to user community abilities
- better integration of interface and application code (application code can be stored as part of the project, thereby eliminating the difficult and tedious reintegration of code and GUI after each modification)

This second generation of GUI builders is clearly a significant step forward, addressing nearly all of the inadequacies of the earlier versions. For a majority of software projects requiring a Motif interface, these tools offer a significant productivity enhancement. Notice, however, the continued lack of an integral draw or paint tool as part of the builder, for example.

4 MMI Builder

Despite all the obvious enhancements in these second generation builders, GUI development for particle accelerator control systems and other complex, dynamic, physically-interfaced systems requires another dimension of functionality. It is for the accommodation of this other dimension as well as a result of the inadequacies found in the first generation of Motif GUI builders that APS has chosen to develop its own interface development tools based on Motif.[4]

The required functionality of a builder tool in the accelerator control domain exceeds that of a "standard" GUI builder, due largely to the dynamic

[4]Had the current generation of GUI builder products been available at the time of APS MMI project initiation, a significantly different development path may have been taken.

nature of what is ultimately being interfaced. Hence, let this "larger" builder be referred to as a *Man-Machine Interface*[5] *(MMI) Builder.*

$$\text{GUI Builder} \subset \text{MMI Builder}$$

This tool creates not only the interface with which a human interacts (human — computer), but also provides a dialogue mechanism by which a human operator or experimenter communicates with, controls, and monitors the physical accelerator complex (human — machine).

4.1 MMI Builder Requirements

The requirements for an MMI builder are a superset of those for a GUI builder. As alluded to earlier and explored elsewhere, these include for GUI's [Rubin, T. 1988] and GUI builders [Hartson, R. and Hix, D. 1989] such high-level notions as *functionality*, *usability*, *completeness*, *extensibility*, *escapability*, *direct manipulation*, *integration*, *locality of definition*, and *structure guidance*. The MMI Builder must incorporate or provide the above, as well as additional functionality for control and monitor functions.

4.2 Definitions and Semantics

Motif widgets and gadgets implement interaction techniques, frequently using physical metaphors for their expressive power (e.g. push buttons). The classification schemes employed in describing Motif widgets are typically class hierarchy based, such as Manager-type objects, or Primitive-type objects, depending upon widget ancestry. Alternatively, the data types best or most easily operated on by a widget class can be used to classify their functions, such as text-oriented widgets (XmText or XmTextField), numerically oriented widgets (XmScale), or state oriented widgets (XmToggleButton).

For control functions, an additional classification scheme can be employed. In the APS MMI, control system objects (whether based on physical entities or not) can be *Controller*, *Monitor*, *Static*,[6] or *Composite* objects. Static objects have representational data associated with them. Controller and monitor objects have representational details, as well as external system

[5]The term 'Man-Machine Interface' has been used somewhat more extensively in Europe than in the U.S.; its usage here can be ascribed to conversations with staff at the ELETTRA Synchrotron Radiation Facility in Trieste, Italy.

[6]This nomenclature is adopted from work done at Los Alamos National Laboratory's Ground Test Accelerator (GTA) Facility and other sources.

(or data source) information. It is through this external system data that the actual dialogue between screen objects and the physical objects they are "attached to" is specified and managed. Composite objects have representational data and other objects (controller, monitor, static, or composite) associated with them.

4.2.1 Static Objects

Static objects are simple statically rendered entities, used frequently for showing non-dynamic properties and relationships. For example, diagrams or schematics of hardware subsystems can be drawn and used as the backdrop for a system of monitors and controllers, mimicking the physical object they are modelling in software. Note that this object type does not implement an interaction technique, and hence is not a widget per se.

4.2.2 Monitor Objects

Monitor objects reflect the current state or value of their associated process variables. As such, they must reflect the latest value of their data. Additionally, they may convey other information, such as alarm state of hardware.

4.2.3 Controller Objects

Controller objects have both monitor and control functions, providing for the change of state or value of external hardware and software. Since they are tied semantically to an external data source, for consistency they too provide a monitor function. For example, a scale (valuator) in Motif both sends or sets a value, but also maintains and displays that value.

4.2.4 Composite Objects

Composite objects are container objects which provide a scope for their constituent objects and allow for application of attributes or resources across a set of objects. This object type allows for hierarchical relationships of various types.

4.3 Architecture

The overall software architecture of the APS Control System is based upon a layered toolkit structure. The APS MMI is one of the topmost layers in this toolkit. Objects in the APS MMI are meta-objects of static, monitor, controller, or composite type, which are implemented via Motif widgets and gadgets, custom widgets, Xlib functions, and access/communication layers found in the control system.

The APS MMI is logically a small, simple pipeline similar to those found in traditional computer graphics:

$$\boxed{\text{mmi builder}} \Longleftrightarrow \text{mmi display list} \Longrightarrow \boxed{\text{mmi manager}}$$

A display list generator (*MMI Builder* or editor tool) produces a graphical display list which contains both representational and functional data describing a screen object and its connection information (channel or process variable identifier) for run-time resolution of dynamic data. This data, the *MMI Display List* is typically stored in a display list file. This display list file is then parsed and executed by a display list consumer, the run-time system or *MMI Manager*, which renders static objects, creates network connections to remote systems for subject channels or process variables, and maintains the correct current visual state of dynamic objects on the screen or display.

4.4 Implementation

The MMI Builder is an interactive display editor currently being developed which implements direct manipulation and drag-and-drop interaction mechanisms using the Motif widget set. It is this component of the MMI system that most closely mimics traditional GUI builders and graphical drawing tools.

The MMI Display List is a syntactic (and semantic) definition of objects and operations in the MMI. The display list contains display data, colormap definitions, attribute and object specifications, and external system data.

The MMI Manager is a run-time display list parser and interpreter which renders static objects, instantiates Motif objects, initiates network connections for monitored and controlled external variables, and manages the dialogue between human, computer, and external accelerator systems and sub-systems. This component lexically analyzes, parses, and interprets the display list and utilizes lower software layers in the APS control system for network connection and monitor function handling.

5 Status and Future Work

The computing environment for the APS control system is Unix-based, relying heavily upon the network transparency offered by X. Code is being written to ANSI-C specifications. Current hardware platforms include SUN4, HP 9000/700, and DECstation 5000.

The Display List and Manager components of the APS MMI are stable and implemented. Most current work is being directed toward the Builder, with the result probably being an integrated Builder/Manager tool.

The syntax of the Display List and structure of the Manager are such that the entire MMI pipeline can be extended as additional requirements are identified. Hence an evolutionary software design is supported, as APS approaches its commissioning date in 1996.

Future work in the APS MMI system will be in the realm of higher-dimensional graphics. Interactive 3-D graphical models of the accelerator complex and sub-systems, including the linac, synchrotron, storage ring, and beamlines will prove very helpful for diagnostics and physical trouble-shooting, as well as for beam-tuning and related operational activities. Scientific visualization techniques can be expected to be employed on the system as a whole for beam-tuning, data analysis, and experimental operations. It is in this realm that PHIGS+/PEX (or possibly GL) will be incorporated into the MMI scheme, consistent with the standards-based work currently being done with POSIX, X11, and Motif.

References

[LANL 1988] *Los Alamos Accelerator Automation Toolkit Workshop*. 1988. Los Alamos, New Mexico.

[Dalesio et al. 1991] "EPICS Architecture", *ICALEPCS '91*. November 1991. Tsukuba-shi, Ibaraki, Japan.

[Hartson, R. and Hix, D. 1989] Human-Computer Interface Development: Concepts and Systems for Its Management. *ACM Computing Surveys*. (March 1989), pp. 5-92.

[McDowell et al. 1991] "Standards and the Design of the Advanced Photon Source Control System", *ICALEPCS '91*. November 1991. Tsukuba-shi, Ibaraki, Japan.

[Rubin, T. 1988] *User Interface Design for Computer Systems.* Ellis Horwood Ltd., England.