TOWARD THE ELETTRA NEW INJECTOR CONTROL SYSTEM

D. Bulfone, M. Chiandone, M. Lonza, L. Pivetta, C. Scafuri
Sincrotrone Trieste, Trieste, Italy

Abstract

The control system for the new ELETTRA booster injector is currently in the design phase. Some of its main components and technologies, both hardware and software, have already been selected or are in the final testing phase. Our choices regarding Linux, C++ toolkits for GUI development and CORBA are discussed. In order to acquire some operating experience and to perform the final tests under realistic working conditions, the candidate tools and technologies are being used to develop systems and programs that are integrated in the existing ELETTRA control system. The experience gained with some application examples is also presented.

SELECTING A GRAPHICAL USER INTERFACE (GUI) TOOLKIT

Since the beginning of operations 10 years ago, the ELETTRA control system has used graphical control panels. All the currently used control panels are based on the Motif toolkit. The new control system will, of course, also use a large number of graphical control panels. Although we will continue to support Motif for the legacy applications, we selected a more modern graphical toolkit for the new applications.

Constraints and Selection Criteria

The new toolkit must fulfill several constraints:

- It must be based on a modern, fully object-oriented design; moreover it should be possibly based on an object-oriented programming language.
- It must have a rich set of native widgets.
- Designing and adding a new widget should be an easy and well documented process.
- It must be independent of any single hardware platform and operating system
- It must be portable at least on Linux and HP-UX, possibly also on Windows.
- It must run with acceptable performances on our current control room consoles (HP9000 C200).
- Integration into the existing ELETTRA control system must be possible and straightforward.

Another fundamental criterion for the selection is the expected lifetime and evolution of the toolkit. It should be available, and possibly improve, for many years. In such a span of time the control system consoles hardware will be upgraded several times, while the various software elements of the control system will follow a different evolution. In other words: code is the asset to be preserved. We must be able to deploy the control system software (recompile and eventually adapt) on the evolving hardware platform as it becomes convenient. We think that the availability of the full source code is a necessary and discriminatory factor for fulfilling this criterion. This belief derives from our past experiences.

An important aspect that received due consideration is whether the toolkit is used by a large and active community of developers, which usually insures code quality and the thrust for improvements and innovation.

Candidates and Final Choice

Keeping in mind the above constraints, we evaluated a number of modern GUI toolkits:

- java/Swing
- gtk [1]
- wxWindows [2]
- Qt [3]

Our choice is Qt. It fulfills all our requirements. It is available in source form, it is written in C++ and has a sound, well designed object-oriented architecture. Qt is also the basis of the Linux KDE desktop [4]. Other important features of the Qt toolkit are its extensive and clear documentation, its support for graphical database access and the availability of a panel-designer/code-generator directly in the toolkit distributions. Other toolkits have also panels-designers/code-generators, but they are separate products; therefore their integration with the toolkit is not always complete and often out of date.

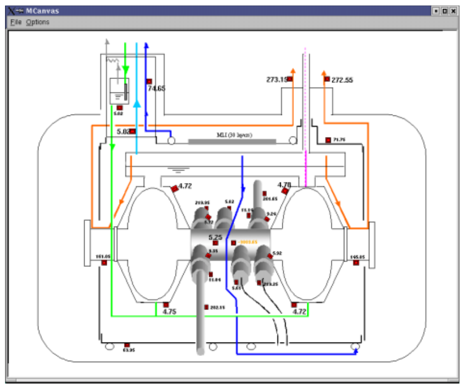

Figure 1: 3HC cavity panel

The portability of Qt based designs have been successfully tested on Linux, HP-UX 11.0 and Windows. Some Qt based control panels are already in use at ELETTRA (Fig. 1) for the monitoring of the 3rd Harmonic Cavity and cryogenic plant. Developing such panels demonstrated also the perfect compatibility of the new C++ based graphics library with the old C based RPC library.

Although java/Swing is largely used in the accelerator community, we have ruled it out as our main GUI for a couple of practical reasons. Its performance is rather poor on the existing operator consoles. It requires *"native interfaces"* in order to call functions of our legacy C RPC libraries; these *"native interfaces"* must be compiled for each platform we intend to use so that the advantage of having a single binary usable on every platform is lost.

MIDDLE-WARE FOR DISTRIBUTED COMPUTING

The control system of the new ELETTRA injector is a two-layer distributed architecture. Communications between the layers are based on TCP/IP protocol over Ethernet. The TCP/IP protocol will generally not be exposed directly to the programmers who will make use of suitable middle-ware tools. The chosen middle-ware will be a determining factor both for the software architecture and for the performance of the control system.

Why CORBA

The first thing that we looked for was a middle-ware package that could support natively an object-oriented programming model. This has led us to the choice of a suitable CORBA [5] implementation. CORBA is a mature and well defined standard. Although other middle-ware packages, like SOAP or XML-RPC, are gaining momentum, we think that on one hand the process of their standardization is not yet settled enough and , on the other hand, that the programming model offered is rather a procedural one (a modern RPC) than an object-oriented one. Interoperability between different implementations and performance are still, at least for us, two open questions.

Interoperability between different implementations of the same middle-ware is very important. This capability will prevent us from becoming locked to a single vendor or open source project. It will also give us the possibility to choose the implementation which fits better the needs of a given server or client. For example we could choose an implementation with a small memory footprint and a limited set of features at the expenses of speed for a system with limited resources. In this respect, an important characteristic of CORBA is that it mandates the use of well defined language-mappings and standard interfaces for interacting with the middle-ware. In this way the structure and code of CORBA clients and servers can be used with different CORBA implementations with very small or no changes at all.

Which CORBA

There are many CORBA implementations available. We decided to select a leading implementation for our control system. This implementation must comply with some requirements. As for the GUI, we want to have the full source code available, licensing policy must be clear and not too restrictive. The middle-ware must be available with bindings for C++. As a minimum, it must have a robust implementation of the Naming Service [6].

Our first selection restricted the candidates to a small number of implementations: omniORB 3.0.5 [7], TAO 1.3.1 [8], mico 2.3.7 [9] and ORBacus 4.0.1 [10]. All of them satisfied our basic requirements and showed a reliable behaviour without any evident major bug. The discriminating factor has thus been the speed performance. A series of measurements has been carried out in a realistic scenario.

Tests and Results

The testing setup consisted of two computers. The client was a desktop PC with a Pentium III CPU at 455 MHz running Linux with 2.4 kernel. The server machine was a Motorola MVME 5100 board with PPC 7400 CPU at 400 MHz running Linux with a 2.4 kernel. This VME board will be the standard field-level board [11]. The machines were connected by means of 100 Mbit/s switched Ethernet. The benchmark consisted of a collection of methods reflecting typical control system actions, e.g. setting and reading of basic types like integer and double precision numbers, setting and readings of arrays of various sizes, etc. Clients and servers were compiled using the different CORBA implementations under test. It should be emphasized that the client and server code needed very little adaptations for switching from one implementation to the another. Execution times were then collected by taking the mean of 1000 successive calls for each method. All the implementations demonstrated complete interoperability. For our selection we considered only the results between client and servers using the same CORBA. Some preliminary tests showed that mixing two implementations did not improve the performance.

The results of the benchmark (a summary in Tab. 1) show that the fastest CORBA of the group is omniORB. The results show also that all the tested CORBA implementation have reasonable performance and a correct behavior versus large data sets, scaling linearly with size (Fig. 2). OmniORB has thus been chosen as the preferred CORBA implementation for the new ELETTRA injector control system. However, we do not rule out the possibility of using also other implementations for special tasks.

TOWARD A CORBA BASED APPLICATION PROGRAMMING INTERFACE

The CORBA middle-ware by itself is not enough to build a practical control system. The number of CORBA features effectively used is just a fraction of those available. Client and server programs will mostly follow some standard patterns. So it is natural to capture these patterns and features

Table 1: Times in microseconds

Call	omni	mico	tao	orbacus
Void	403	652	704	1055
GetInteger	372	686	703	965
GetDouble	374	684	701	963
GetInteger[512]	746	1086	1143	1395
GetInteger[1024]	909	1260	1314	1613
GetDouble[512]	907	1484	1304	1538
GetDouble[1024]	1371	2147	1711	1966

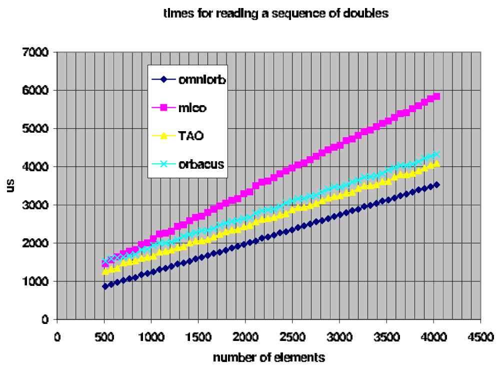

Figure 2: times for reading an array of doubles of increasing length

into a library of objects and functions, offering the control system programmers a lean and effective Application Programming Interface (API).

The other fundamental issue in the design of the control system, and as a consequence in the design of the API, is the device object model. We have chosen to base our design on the so called generic interface model. In a very rough sketch: the control system is composed of a collection of devices,accessed by means of unique names. Each actual device belongs to a device class. The class of the device defines the list of attributes which can be read or written to get or set the status of the device. Each attribute is distinguished inside a device by its unique name. Other issues, like asynchronous calls, publish/subscribe on changing attribute values are beyond the scope of this paper. This object model is adopted by many accelerator control systems, for example see [12] [13].

A Pilot Project for Testing the API concepts - First Results

We are now in the phase of consolidating the specifications and starting some pilot projects to test different designs and implementations of the API and the generic device model. On the server side some efforts are being spent to design a good interaction model with the underlying real-time extensions to the Linux kernel. On the client side, we are testing the integration of the CORBA API with the Qt GUI toolkit and ways to optimize the performance. The monitor application (Fig. 3) for the new digital Beam Position Monitor [11], one of our pilot projects, has an effective graphical interface capable of displaying X and Y beam position diagrams, Fourier transforms and beam position density maps reliably with a rate of 25 frames per second. These good results make us confident that the selected components, both hardware and software, and the overall architecture of the new control system, although not yet complete and fully optimized, will allow us to build effective, reliable and performing application software.

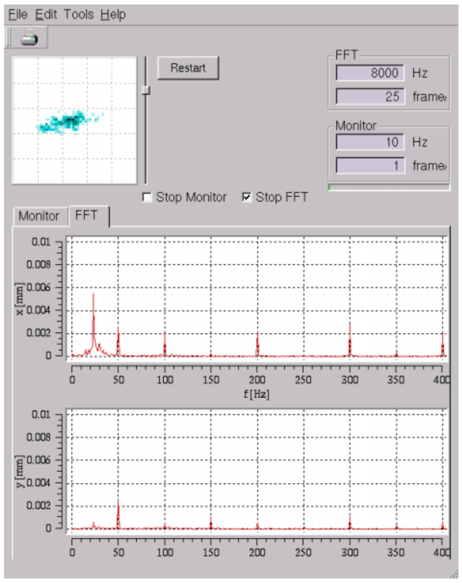

Figure 3: digital BPM monitor panel

REFERENCES

[1] http://www.gtk.org

[2] http://www.wxwindos.org

[3] http://www.trolltech.com

[4] http://www.kde.org

[5] Object Management Group, “The Common Object Request Broker Architecture: Architecture and Specifications”,1998. ftp://ftp.omg.org/pub/docs/formal/98-12-01.pdf

[6] M. Henning, S. Vinoski, “Advanced CORBA Programming in C++”, Chapter 18, Addison-Wesley, 1999.

[7] http://omniorb.sourceforge.net

[8] http://www.cs.wustl.edu/ schmidt/TAO.html

[9] http://www.mico.org

[10] http://www.orbacus.com

[11] D. Bulfone et al., “New Front-End Computers Based on Linux-RTAI and PowerPC”, These proceedings.

[12] K. Kostro et al. , “Controls Middleware - The New Generation”, EPAC 2002, Paris, 2002.

[13] J-M. Chaize et al. , “Tango - An Object Oriented Control System”, ICALEPCS 1999, Trieste, 1999.