INTEGRATION OF WINDOWS BINARIES IN THE UNIX-BASED RHIC CONTROL SYSTEM ENVIRONMENT*

P. Kankiya, J. Jamilkowski, L. T. Hoff, BNL, Upton

Abstract

Since its inception, the RHIC control system has been built on top of UNIX or LINUX and implemented primarily in C++. Sometimes equipment vendors include software packages developed in the Microsoft Windows operating system. This leads to a need to integrate these packaged executables into existing data logging, display, and alarms systems. This paper describes an approach to incorporate such non-UNIX binaries seamlessly into the RHIC control system with minimal changes to the existing code base, allowing for compilation on standard LINUX workstations through the use of a virtual machine. The implementation resulted in the successful use of a Windows dynamic linked library (DLL) to control equipment remotely while running a synoptic display interface on a LINUX machine.

INTRODUCTION

Large enterprises that involve the integration of multiple engineering groups such as the Collider accelerator Department at BNL, rely on a consolidated control system framework. It is the primary responsibility of the control system to provide an interface for device management and data acquisitions.

With the rapid growth in software development techniques, it is not uncommon for merchants supplying control equipment to also provide software support in form of a packages like portable executable(PE) library files - the Windows equivalent of ELF files. It is time consuming to reproduce these PE files with in house code t on Unix platforms. I some cases this is not even an option. There needs to be a technique to take advantage of such pre-built programs without rewriting the complete interface for each target platform.

In this paper we propose a technique that can be used to incorporate Windows executables into a Unix-based control infrastructure, while maintaining compatibility with our legacy suite of applications used for archiving, logging, storage and alarming.

EXISTING COMPONENTS OF SOFTWARE DEVELOPMENT

The RHIC control system consists of two physical levels: console level computers and front-end computers (FECs). FECs provide access to accelerator equipments. A code generation mechanism called “adogen” facilitates the development of ADO code and associated database.

For the scope of this paper, the flow of data at RHIC from a device to the user consists of four stages. The control equipment tier represents the hardware devices in use at accelerator complex. The rudimentary device access to such equipment is provided by the ADO classes. The Accelerator Device Object (ADO) is implemented as a C++ class [1] that contains a data structure and full set of actions or "methods" that are needed to control and monitor the accelerator device(s). To publish this data across controls applications, an interface layer known as adoIfServer has been implemented (see Figure 1). The ADO protocol implemented by the adoIfServer serves as the RHIC Controls System API for application programs. The communication between ADO class and the client applications is achieved through client-server model utilizing TCP/IP communications[2].

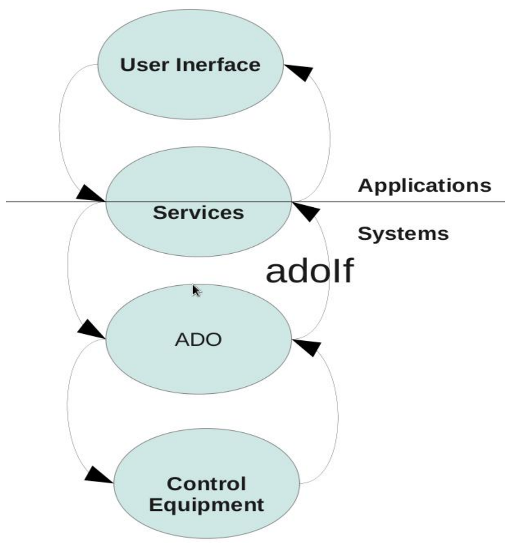

Figure 1: SW architecture of RHIC control system.

The latter two stages refer to multiple applications which make use of information made addressable via ADOs. These applications are responsible for fundamental data interaction. Example of these tools include parameter Editing Tool (PET), live data graphing through the General Purpose Monitor (GPM) application,and logged data can be plotted through the LogView application. Console software has been primarily developed in the C++ language.

* Work supported by Brookhaven Science Associates, LLC under Contract No. DE-AC02-98CH10886 with the U.S. Department of Energy.

ISBN 978-3-95450-139-7

PROCESS OF INTEGRATION OF WINDOWS BINARIES

For the purpose of combining of foreign binaries into our control systems, we needed to replicate the software components in our existing development process to the foreign OS host , windows XP in our case.

The first step was to install a cross-platform compiler utility. Cross-compilation is the technique of running programs on a "host" system to generate object code for a different "target" system. We have used here an efficient and highly reliable industry solution called Cygwin[3].

Cygwin is a Unix-like environment and command-line interface for Microsoft Windows. Cygwin provides native integration of Windows-based applications, and other system resources with software tools available in a UNIX environment.

Cygwin permits installing inetd, syslogd, sshd, Apache, and other daemons as standard Windows services, allowing Microsoft Windows systems to emulate Unix and Linux servers. It consists of a library that implements the POSIXx system call APIs utilizing Win32 system calls, and a GNU development tool chain (including GCC and GDB) to allow software development.

We also needed to install a remote version control system, which was required to the suite of libraries and applications that RHIC control system is founded upon. To provide access to this vast code base, we install the ClearCase Remote Client (CCRC). This product provides access to the IBM Rational® ClearCase® version control system from various network connections over multiple operating systems. With the help of CCRC it is possible to access resources in remote repositories and load them into local Rational ClearCase "web views" as ordinary files and directories under Rational ClearCase source control.

Synchronizing the existing Linux code repository with the remote client repository is easily performed with the CCRC[4] user interface. Once synchronized, it is important to link Cygwin to the versioned objects present in Clearcase environment. This is easily accomplished using Unix links.

Before starting software development the standard library code such as adogen and Cdev[5] must be built.

To build an application that includes a Windows shared object library (eg a .DLL or .lib file) provided by the vendor, makefiles do not need to be modified as long as the DLL is in the search path. Note that the ADO specific makefile should make sure to include the pertinent header file.

Henceforth, an ADO server program can be coded on Windows workstation in the exact same fashion as a control system developer uses to code in his or her endemic Linux development environment. Once operational, this ADO server should be registered with the portmap. This will attach the running program to the list of aodIfServer clients[6].

Now, since the interface between the vendor's software library and a main control room user is an ADO, we can successfully compound this application with every utility that employs ADOs as their elementary inputs. In other words a synoptic display can be used to control parameters, the data received can be logged and alarms could be submitted to the alarm system without any tweaks in the already implemented code for these applications.

PRACTICAL IMPLEMENTAION

To demonstrate the integration of a Windows executable, a test bench was set up to simulate Windows Xp OS, using a virtual machine[7]. To validate suffested scheme, a program called SimpleMan was compiled under Cygwin. This is standard test program which is used at RHIC control systems as a demo tool for testing ADO features in conjunction with user interface tools. This successfully demonstrated the execution of a Windows software object from a Unix file system.

Other Test Cases with Limitations

To demonstrate the use of above methodology, we designed a control interface for a Laser assembly. This laser, as supplied by the manufacturer, is attached to a PC hosting the Windows XP operating system. Following the industry trend this set-up was also provided with a software package constituting a custom application designed to perform Laser related operations and a shared object library in the form of a Windows DLL.

The issue with this configuration was that the custom application could not have been used to incorporate laser operations into our Controls System. With help of function prototypes provided with the DLL, it was relatively easy to write an ADO class

In process of developing an interface to this device, it was discovered that the DLL has a dependency on the Microsoft Visual Studio suite. For instance, the event-loop or the message dispatcher, is inherently implemented by the MFC library of visual studio framework[8]. A method to avoid these dependencies need to be investigated.

A second situation that could benefit from suggested technique development of control software for a laser wavelength meter which was packaged with a set of APIs written in Windows and packaged as a DLL.

This test case highlighted another concern in reference to the proposed method. It is possible to run into compiling issues due to unresolved name mangling when trying to export the DLL symbols. As a result, the compiler will generate the "unresolved external symbol" error message. A proven solution to surpass this error needs to be devised[9].

LIMITATIONS

Windows systems will typically not have access to Linux file systems outside version controlled environment. The RHIC controls system uses the relational database SYBASE to store, among other things, configuration data. Application software of all kinds routinely access the SYBASE database, to retrieve such data. Access is provided via a combination of vendor-supplied software libraries, and in-house developed libraries. The latter has been optimized and specialized for the LINUX environment, and provides added value in functionality such as failing over to a redundant backup SYBASE server. However, these libraries are not readily ported to the Windows environment.

CONCLUSION

A well defined method to incorporate a Windows portable executable file into existing Unix/Linux braced control infrastructure has been laid out. The approach highlights the use of a cross compiler and a supplementary version control system on a remote platform. This will aide in successfully producing an executable program on an external target platform which can be helpful in integrating equipment control with native data acquisition, logging, archiving and alarming tools. The outstanding issues of the solution have been listed out. The outstanding issues described here must be resolved before further attempts to integrate Windows binaries in RHIC Controls System are made.

REFERENCES

[1] D.S. Barton, et al., "The RHIC Control System", Nuclear Instruments and Methods in Physics
[2] L.T. Hoff, J.F. Skelly, "Accelerator devices at persistent software objects"
[3] "http://www.cs.rpi.edu/~magdon/courses/cs1/labs/lab1/cygwin.pdf"
[4] "https://pic.dhe.ibm.com/infocenter/cchelp/"
[5] J. Chen et al., "CDEV: An Object-Oriented Class Library for Developing Device Control Applications", Proc. ICALEPCS'95
[6] S. Sathe, L.T. Hoff, T.Clifford, "Client Server design and implementation issues in accelerator Control system environment", Brookhaven National Lab.
[7] Tal Garfinkel, Mendel Rosenblum, "A Virtual Machine Introspection Based Architecture", Computer Science Department, Stanford University.
[8] "Visual C++ - Exploring New C++ and MFC Features in Visual Studio 2010", Msdn.microsoft.com. Retrieved 2012-11-19.
[9] http://cygwin.com/cygwin-ug-net/dll.html

ISBN 978-3-95450-139-7