A distributed object-oriented control system for the CODA Data Acquisitio n Toolkit

W.G. Heyes, J. Chen, D.J. Abbott, P. Banta[a], E. Jastrzembski, C. Timmer

Thomas Jefferson National Accelerator Facility (TJNAF)

12000 Jefferson Ave. Newport News, VA 23606 USA

[a]ADC Telecommunications Inc.

2240 Cambell Creek Blvd. Richardson, TX 75082 USA

The CODA data acquisition system used in all experiments at Jefferson lab is a software system designed to manage a distributed network of data acquisition components. At Jefferson lab data transport is over commercial networks such as ATM, FDDI and Ethernet. However, CODA is designed to perform the control and configuration of any distributed data acquisition system and the data transport method used can, and do, differ at other sites. This paper covers the control and configuration aspects of CODA.

Key words: Data Acquisition Systems; Object-Oriented Methods; Software Tools;

1 Introduction

Thomas Jefferson National Accelerator Facility (Jefferson Lab), is an electron accelerator with an energy of 4GeV. There are three halls containing particle detectors. The Jefferson Lab data acquisition system CODA[1] has been developed for use in all three experimental halls and detector test areas to implement data acquisition systems with a large range of event sizes and data rates. CODA is designed to configure, control and monitor data flow through distributed processes running on a heterogeneous network of processors, embedded processors running VxWorks[2] and Unix systems.

The original control system in CODA was based on Sun RPC. The system suffered from several problems. Firstly the functions of control, monitoring, state tracking and GUI were all combined in a single process. This lead to situations where the GUI was unresponsive when the control program was busy with RPC transactions. Configuration information was maintained in ASCII files, as the number of processes being controlled grew management became increasingly difficult. Final ly, since the system contained no *name server*, it was necessary to *hard wire* some configuration information into the distributed processes. This lead to restricti ons to the operation of the system. For example, two users could not run CODA at the same time from the same UNIX machine. A new control, configuration and monitoring system was devised to address these issues and is the subject of this paper.

2 Configuration and name server.

Two of the problems of the earl y CODA system, configuration and *name server* have the same solution. The configuration information for the new system is contained in a relational database. The msql[3] databa se was chosen as a simple, fast, and robust database with the additional advantage of being freely distributable within the academic research community. The msql database is served by a daemon on a well known host and TCP/IP port. Any process can query the database daemon using a TCP/IP connection. Each CODA process is described in the *process* database table. This table contains a translation between process name and network parameters such as the host on whic h the process is running and the TPC/IP port on which it is listening. Therefore the database also acts as a name server since any process can query th e database and retrieve information about any other process using only the process es name.

The use of a relational database for configuration is a powerful feature of CODA . Each msql database represents an *experiment*. Within an *experiment* da tabase there is a *sessions* table which has one row per instance of the run control prog ram. Multiple copies of run control can run using different sessions and different subsets of hardware. The particular subset of processes in use by a session are listed in *run configuration* tables, the names of these tables are arbitrary and listed in the *RunTypes* table. The indiv idual run configuration tables define the subset of components used for that particular run type, the connections between them, and any dynamically loaded libraries required at run time.

To ease the use of the configuration database two tools have been written. The *dbedit* tool allows the database tables to be listed and modified in a GUI. A second tool *codaedit* allows run configurations to be generated in a grap hical manner. Components can be placed in a drawing area and wired up using data links in an i ntuitive way. The configuration generated is saved directly to the database.

There are many other tables and data fields in the database which cannot be desc ribed in detail here.

3 Scriptable components and control

Each process in CODA co ntains a Tcl[4] interpreter. The Tcl language is enhanced by the Itcl extension which allows an object oriented approach to Tcl s cript writing. For each class of CODA component there is an Itcl class which implement s the functionality of that particular class of component. The base component classes all inherit from the c lass CODA which provides the core functionality shared by all classes of component. Optionally each CODA process c an be run in *interactive mode* whereby a Tcl prompt appears and commands can be given directly to the process from the command line. The power of this approach for diagnostics is readily apparent.

A simple RPC system has been implemented which is loosely based on the Tcl-DP sy stem[5] for inter process communication using TCP/IP. Any CODA process can issue the Tcl command *NS_ServerInit [name]* where *[name]* is the replaced by name of the It cl object which constitutes the CODA component. This command sets up a TCP/IP server socke t which listens for connections and enters the port and host in the *process* table in the database. Any other process can issue the Tcl command *DP_ask [name] [Tcl c ommand]*. DP_ask looks up the host and port corresponding to [name] in the database then creates a TCP/IP socket connection with the server process. The Tcl command is then sent over the socket connection and executed on the server end. The Tcl result is returned ove r the connection. In this way we can ask ROC2 to download using *DP_ask ROC2 down load TEST-CONFIG*.

The DP_ask mechanism is used extensively in CODA for control of remote processe s. It has the advantage that the set of commands which can be issued using DP_ask is not pre- defined in a stub file as in traditional RPC but can be chosen at run time.

4 Object-orient ed run control

A typical data acquisition system can be best described in the following way, there are readout controllers (ROC), sending data to a event builders (EB) that build incoming data into predefined format which eventually are written to a tape or disk by event recorders (ER). This clearly forms hierarchy of systems, subsystems and components within these subsystems. This simple hierarchy can be easily mapped to C++ class hierarchy which can describe the data acquisition system adequately. In CODA the central control of the system is performed by the process rcServer in which *real* data acquisition objects are mapped directly onto instances of C++ classes. A state transition is initiated by sending a control message to the *daqSystem* object in rcServer which in turn send out messages to all its *daqSubSystem* objects these in turn send the messages to individual components (using DP_ask) . State transitions are actually carried out by a state transitioner which initiat es sending messages and monitors the status of the components. There is a *daCompMonitor* objec t that monitors all components' status and ends a run if any of the components behave abnormally. There are also abstract C++ classes which do not correspond to any real physical system/components, *daqRun* class is an example of this. This class encapsulates all the attributes of a data acquisition run; the event number, data rate, event rate and status etc.

The rcServer has two clients, a Motif GUI and client API library for the common device interface CDEV[6]. There are about 300 C++ classes involved among which 20 classes are dedicated for network communication, and 150 classes are for Motif interfaces. This version of Run Control not only works with CODA version described in this paper (tested on Solaris 2.5) but also works with the earlier CODA version 1.4 (tested on HP-UX 9.05). The client GUI is very light in CPU and network usage. In addition, there is an access control mechanism to allow exclusive control of a server by a client.

5 Conclusions

It is difficult to describe such a complex system in such a short paper. Certain ly much is lost in the way of clarity and completeness. The CODA system as described here h as been used in the commissioning data taking runs of the CLAS detector in hall B, December 1 996 and February 1997. The control features of the system performed very well in *re al* on-line conditions. The main features of the system, embedded Tcl for control, c onfiguration and diagnostics, a relational database as a *name server* and central system configuration, and the C++ object oriented run control are all novel features. Tcl has been ver y useful for on-line diagnostics. During the coming years we intend to increase our use of the common CDEV interfa ce to the run control server, and to the DP_ask inter-process communication system, to al low custom control GUIs to be constructed which monitor and control both CODA and EPICS at the same time. At that time CODA and EPICS will be truly integrated and become a very pow erful data acquisition environment. CODA is availible via anonymous ftp from ftp.jlab.org i nterested parties should mail heyes@jlab.org.

References

[1] G. Heyes et.al. The CEBAF On-line Data Acquisition System *Proceedings of the CHEP Conference* (April 1994) 122-126.

[2] VxWorks and Tornado are registered trademarks of Wind River Systems, USA.

[3] mSQL was developed as part of the Minerva Network Management Environment. Copyright (c) 1993-1995 David J. Hughes.

[4] John K. Ousterhout *Tcl and the Tk Toolkit* (Addison-Wesley, Massachusetts, 1994)

[5] The tcl-DP distributed programming package for Tcl. Brian Smith et al. Department of computer science, Cornell University.

[6] http://www.jlab.org/cdev - The CDEV home page.