AN OBJECT ORIENTED DISTRIBUTED FRAMEWORK TO ADD CONTROLS PARAMETERS TO THE NSCL EVENT STREAM

R. Fox, C. Molhoek

Introduction

Increasingly, experiments at the NSCL and other laboratories require knowledge of control-system parameters if they are to be effectively analyzed. At the NSCL, this is evident where:

- Experiments with the S800 require magnet settings so that software aberration correction can be performed.
- Radioactive beam experiments that are sensitive to the settings in the A1200 separator/analyzer system can make use of knowledge of the A1200 settings to correct for run to run differences in the system.

As the NSCL coupled cyclotron upgrade comes on-line, we anticipate that experimenters will increasingly need to periodically record settings from several control systems along with physics event data. This contribution describes the design and implementation plan of an object oriented framework, along with modifications to the NSCL data acquisition system to support this requirement.

The paper is divided into three sections:

1. "Distributing Control Information" describes a distributed object oriented encapsulation of the S800 control system which supports generalization to other control system types.
2. "Integration with the DAQ System" describes modifications to the NSCL data flow architecture that support the injection of data buffers from a variety of sources outside the primary event stream.
3. "Conclusions" describes the current state of the project as well as what we have learned to date.

Distributing Control Information

A heterogeneous set of control systems controls devices at the NSCL:

- Commercial Off The Shelf (COTS) systems such as Vista[1] and Labview[2]
- Locally written systems based on the FERMI Linac Control System [3]
- Other ad-hoc systems.

Providing a uniform access mechanism to these systems is challenging. This section describes a set of patterns in object oriented design that supports such uniform access. We then describe the implementation of this system for the S800 control system which is based on the Vista COTS product. Finally we describe how Common Object Request Broker Architecture (CORBA) products and parallel object hierarchies allow us to transparently distribute this framework to systems that do not participate directly in the control system.

Object Patterns For Control System Access

Our design requirements are:

- Uniform mechanisms for accessing (read-only) control system data regardless of the underlying control system product.
- Transparent, location independent distribution of control system data outside the boundaries of the control system computers.

Control systems consist of many types of I/O channels. The access mechanisms and representations of each type of channel vary both with the type of channel and the underlying control system. We want to insulate the user from both types of variability. To start with, we represent the value of each control item as a textual string. This is suitable for a wide variety of display and storage applications. One can then model the resulting control system object hierarchy as shown in 1 below.

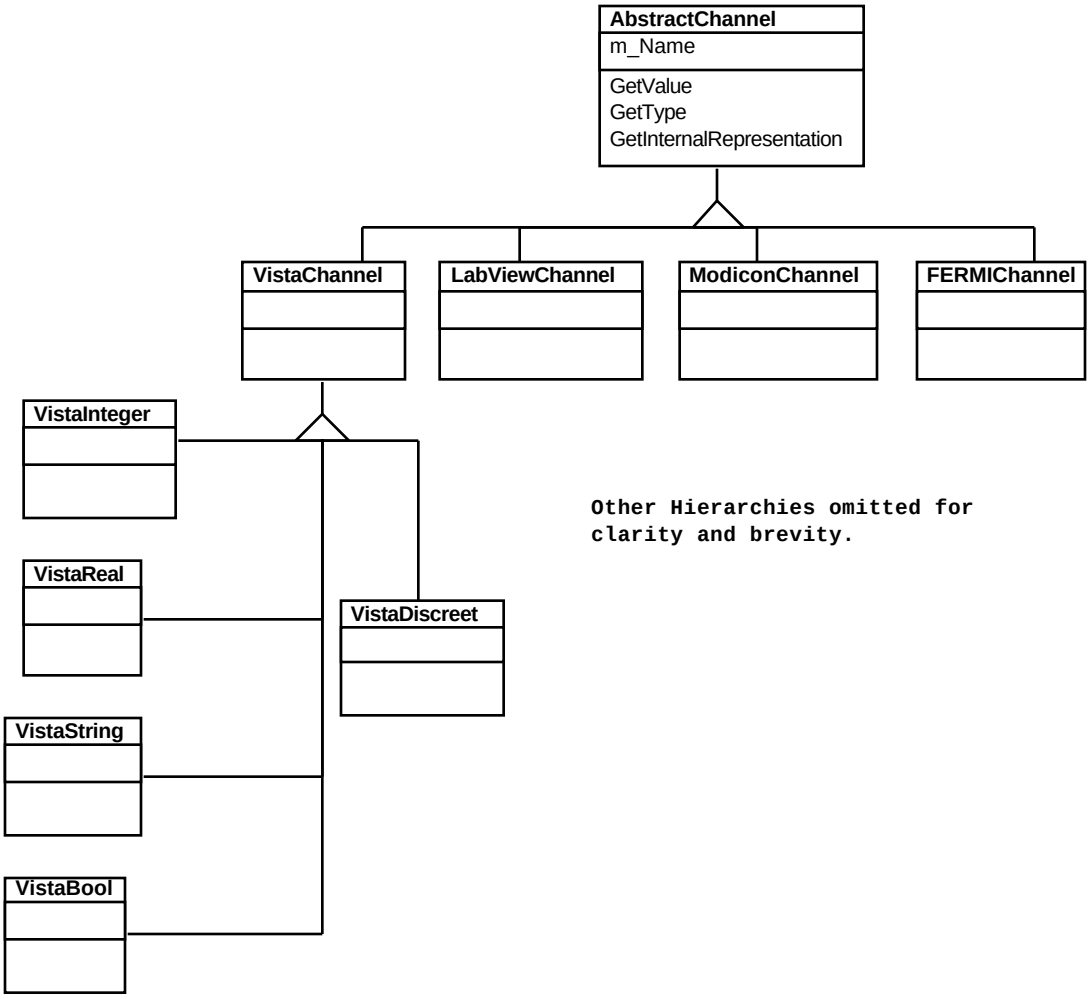

Figure 1 Control Point Hierarchy

AbstractChannel provides a set of interfaces that allow the client to fetch the channel, Get the type of channel represented, and get the internal (non-textual) representation of the channel where appropriate. Each control system in turn has a sub-hierarchy of classes that represent the set of channel types it supports. The sub-hierarchy for the Vista control system is shown in full.

Given the name of a channel, we are still faced with the problem of creating the appropriate type of channel within the appropriate control system hierarchy. An abstract factory[4] is an object which can create objects with identical interfaces from one of a variety of orthogonal class hierarchies. Normally, a programmer selects a concrete subclass of the abstract factory and uses it throughout the program to create instances of objects from only its hierarchy.

We modify the behavior of the abstract factory pattern. The specialization of the pattern to fit our needs are shown in Figure 2. Our modification, resulting in a new pattern we can call the Factory

Selector Pattern, hides behavior in the creational method of the top level of the factory hierarchy (a pure virtual method in the Abstract Factory Pattern).

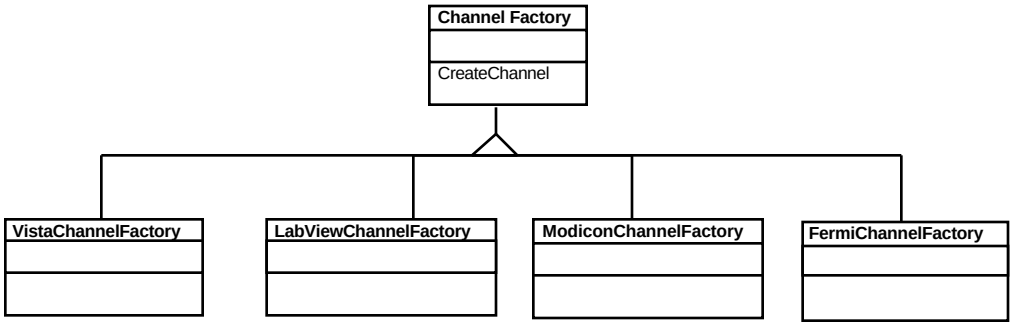

Figure 2 Specialized realization of Abstract Factory Pattern

The CreateChannel member function for ChannelFactory determines from the channel name to which control system the channel belongs. It then creates an object of the appropriate ChannelFactory Subclass. The last step in creating a channel is to delegate the channel instantiation to the control system specific factory object. The control system specific factory then uses its knowledge of the mapping of channel names to underlying channel types to create and return, to the client, an appropriate object of a type derived from AbstractChannel. While our specialization of this pattern uses a fixed mapping between channel names and underlying control system, it is possible to conceive of a channel and factory registration system which allows entire control systems to be dynamically registered with the factory selector.

The S800 Control System

The target system for this deployment was the S800 control system. We would like to integrate parameters from this control system into the streams of physics data which are acquired by the data acquisition system. The S800 control system is a Vista control system which is built on top of the NSCL accelerator control system. For the purposes of this implementation, we will only be interacting with the Vista control system, however we have shown how our hierarchy of classes allows our software to easily extend to other control systems as well.

Parallel hierarchies and Transparency

The control system computers at the NSCL do not directly provide access from arbitrary computer systems. To support this on the S800 Vsystem based control system would require additional client licenses for software on all potential data acquisition systems. We require read access to be distributed to the data acquisition system from the control computers. A natural match for our object oriented implementation of access classes to the control system would be to use a distributed object system such as a CORBA compliant client server system to distribute access to control objects across the hierarchy.

Unfortunately, CORBA bindings to C++ are not entirely transparent. One must bind local objects to remote objects through object identifiers. We would like to hide the existence of CORBA and even the distributed nature of the system from the user, however. This is done in our case through

parallel object hierarchies, on the server and client nodes. Client nodes invoke methods in a channel factory object which in turn bind to a corresponding server channel factory object, and interact with that object to obtain an object identifier for a server channel object. The server factory object also maintains a database of instantiated channel objects so that multiple clients can share instances of the same server channel.

The client factory object takes the CORBA object identifier and wraps a client channel object around it producing a Channel object which operates in a manner completely indistinguishable from a local channel object. The client program can then make calls against the client channel which get transparently mapped to server object calls.

Integration With the DAQ System

This section discusses the integration of control parameters with the data acquisition system. We break this discussion up into:

- A description of the relevant parts of the NSCL data acquisition system data flow architecture.
- Generalizations made to that flow
- The control system buffer injector process.

NSCL DAQ System Data Flow overview

The data acquisition system at the NSCL is based on an active buffer manager[5] shown in Figure **3**.

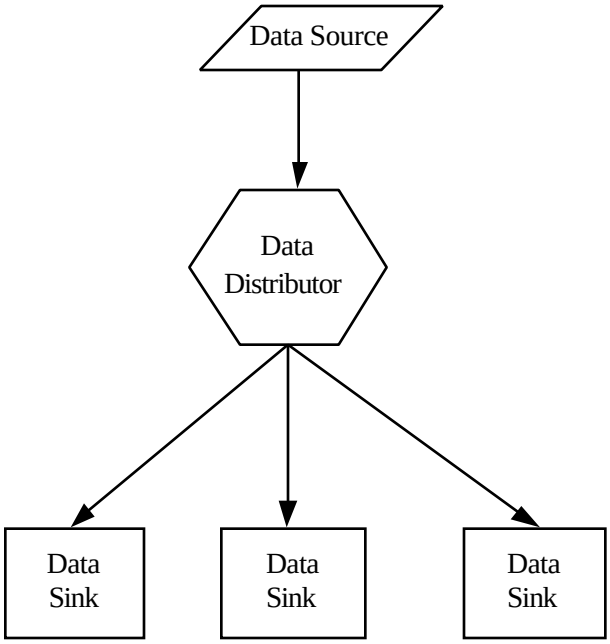

Figure 3 NSCL Data Flow

Active buffer managers push data through the system to consumer programs. Consumer programs are able to register interest in buffers of particular type as well as to indicate whether they receive only a sample of the data or the entire data set. Active buffer managers in this system allow the event recording process to be treated as just another data consumer program. The open nature of this

data flow allows the NSCL data acquisition system to adapt to additional experimental needs as well as to host data analysis tools which are used by outside collaborators, and collaborations.

Generalizations to the Data Flow

As Figure 3 shows, the NSCL data flow model assumes a single source of data. In[6], we described a system which supports multiple data sources in the UNIX environment. This system was easily ported to VMS.

While the multi-sourced active buffer manager of[6] could directly service consumers, and will in future versions of the NSCL data acquisition system, the application programming interfaces to it are substantially different from those of the original VMS buffer manager. Rather than require that programs be modified, recompiled and relinked for use with this system, we have instead, produced a variant of the original VMS buffer manager which used the buffer manager of [6] as its data source. This system retains compatibility with both old and new buffer managers.

Control System Buffer Injector.

To inject control system buffers, all that remained was to produce a control system buffer injector, as a second data source to buffer manager of [6]. The API to that buffer manager is a C language API. We wrapped the key concepts of that API into objects, and then produced application framework classes for both consumer and producer programs.

The Injector then incorporates the hierarchy of CORBA client classes produced to access data from the control system, and is able to periodically inject buffers into the data acquisition system.

Conclusions

We have described a project for incorporating control system data from the S800 control system into the data stream of the NSCL data acquisition system. This project is expected to be completed in summer of 1998.

Extensive use of software patterns[4], allows the expansion of this system to multiple incompatible control systems. Software pattern re-use allowed us to re-use previously proven designs of interacting classes and objects. This re-use is at a much higher level than previous work on component re-use.

Use of CORBA and parallel class hierarchies allow transparent distribution of this information across the network. The system we are developing will probably serve as a prototype and model for future similar requirements for coupled cyclotron physics experiments at the NSCL.

References

1. http://www.vista-control.com/vsystem.html
2. http://www.natinst.com/labview/
3. R.W. Goodwin, M.F. Shea, *Modern Control Techniques for Accelerators"* in 10'th International Conference on Cyclotrons and their Applications IEEE Press 1984 pp 547-558
4. E. Gamma et al. Design Patterns Elements of Reusable Object-Oriented Software Addison Wesley 1994
5. R. Fox et al. *Progress on the Data Acquisition System at NSCL* IEEE Trans. on Nucl. Sci NS-32 No. 4 1985

6. R. Fox et al. *Portable Data Flow in Unix* in Conference Record of the 8'th Conference on Real-Time Computer Applications in Nuclear, Particle and Plasma Physics TRIUMF TR-93-1 1993 pp 152-155
7. J. Vincent et al. *The NSCL Control System* in Procedings of the 1995 IEEE Conference on Real-Time Computer Applications in Nuclear Particle and Plasma Physics NSCL 1995 pp 297-302