OBJECT ORIENTED PROGRAMMING INTERFACES FOR ACCELERATOR CONTROL[*]

L.T. Hoff
Brookhaven National Lab
Upton, NY, 11973-5000, USA

Abstract

A current trend in control system design is to provide an object oriented programming interface for application developers. This talk will discuss important aspects and features of object oriented APIs for accelerator control systems, and explore why such interfaces are becoming the norm.

1 INTRODUCTION

Several years ago, the AGS controls group was given the task of developing software for the RHIC accelerator. Like the AGS, the RHIC control system needs to control and monitor equipment distributed around a relatively large geographic area. A local area network connects this equipment to a collection of UNIX workstations in a central control room. Similar software had been developed for the AGS about a decade earlier, but isn't well suited for RHIC use for a number of reasons.

The AGS software was designed to work within AGS parameters. The AGS software expects data updates at AGS operating rates. This rate is typically every 3 or 4 seconds. The AGS software does not allow large (greater than about 40 Kbytes) data transfers. The AGS software enforces a very rigid format for grouping related data. In addition, the AGS software was written for a proprietary hardware platform. An aggressive porting effort was well underway to make the software usable on typical UNIX workstations. However, it seemed that this effort would not be complete in time.

More importantly, all software for AGS operations had been written by the controls group. A different paradigm was expected for RHIC software development. The RHIC Accelerator Physics group is composed of experienced programmers. Not only are these physicists fully capable of writing physics applications, they expect to do so. The controls group is expected to provide training and assistance in using control system software. This necessitates a succinct, well-defined application programming interface (API).

The AGS software represents accelerator equipment as collections of control points. These collections of control points are called “logical devices”. Other control systems focus more on individual control points, rather then on related groups of control points. The AGS approach contains the rudiments of an object oriented system. It is not truly object oriented, since the “logical devices” do not contain the methods for translating the control points into commands to accelerator equipment.

Rather than adapt the AGS software for RHIC use, the controls group opted to start with a clean slate. To develop software that would address the shortcomings of the AGS software, while preserving the useful features that evolved through years of use.

2 INFLUENCES

The ideas for the RHIC API were necessarily shaped by trends in industry. The network management protocol SNMP[1] was becoming more popular. Network management shares many characteristics with accelerator control. The simple, flexible interface of SNMP allows it to control and monitor a wide variety of network devices distributed around a large region. When Marshall T. Rose designed the SNMP interface, he commented “I wanted to make an interface so simple that no one could complain about it.”[2]

Accelerator control systems began mimicking industry trends. A new control system was developed for the European Synchrotron Radiation Facility (ESRF) [3]. This control system provides an interface even simpler than SNMP. SNMP has separate SET and GET functions for controlling and monitoring devices. The ESRF control system provides a single function, dev_putget(), for both operations. Like SNMP, the ESRF control system is object oriented. Accelerator equipment is represented as collections of related control points. These collections are called Device Servers. Each Device Server contained methods for translating the control points into commands to accelerator equipment.

Widespread application of object oriented programming techniques were becoming the norm. In particular, the C++ language was emerging as the most popular object oriented programming language. C++ had already become the standard programming language for application development at the AGS[4]. The RHIC Accelerator Physics group members were all well versed in object oriented analysis and design as well as object oriented programming.

Concepts such as abstract data types began appearing in class libraries. The Free Software Foundation made available a general purpose class library called libg++. This library contains classes to represent

[*] Work performed under the auspices of the U.S. Department of Energy

rational and complex numbers. These classes are designed so that they can be manipulated without the user knowing how the data is represented. Such classes contain methods to coerce their values into C++ native data types. This might be necessary for interfacing with legacy code, or for doing I/O.

3 GOALS

Combining the requirements for RHIC operations, lessons learned from prior AGS experience, and adopting useful industry trends, the following goals were set for RHIC software.

3.1 Flexible

The inflexibility of the AGS software limited its usefulness for RHIC. Conventional wisdom held that control system software requirements were such that accelerator-independent software was not feasible. However, both Vsystem[5] and EPICS[6] proved that this was not the case. Both of these systems are in use, or are expected to be put into use at a number of different sites.

3.2 Portable

The ongoing porting effort of the AGS software could not be ignored. To try to avoid this hardship for RHIC software, a decision was made to use popular industry standards whenever possible. While this strategy is not foolproof it does increase the chance that the software will run on a variety of hardware platforms.

3.3 Easily configurable

The AGS software uses a central database to describe the location and functionality of distributed software. This database defines the data size and type, and even where in memory to find the data in each remote system. This rigid design severely limits the ability to incrementally upgrade or deploy remote systems.

3.4 Succinct API

Since AGS software development was contained within the controls group, there was little incentive to simplify, or fully document the various programming interfaces. This information is transmitted informally to new members when they joined the group. There are no external customers for the API. If external customers are expected, the API must be succinct enough so that it can be easily documented, and so that training can be easily provided.

3. 5 Object Oriented

AGS applications programmers found object oriented programming techniques essential. Object oriented programming techniques often allowed for more easily supporting last minute changes without restructuring the application. An object-oriented API can be more succinct than a procedural equivalent. Since certain state information can be contained within an object, there is less need to clutter the API by passing state information back and forth across the interface. Functionality such as error handling and recovery can often be separated from the main functionality of the interface. This frees the application writer to concentrate on interfacing with equipment, not on dealing with control system or network weaknesses.

3. 6 Abstract data types

To work with data in the AGS control system, the application programmer needs to know the size and type of the data. Since this information resides in a database, this becomes somewhat awkward. Using database information, the data must be coerced to fit the data type the programmer wishes to use. To simplify this process, all data may be automatically coerced to a float type. This method is not without pitfalls. It forces the use of slower floating point arithmetic, and precludes data types such as character strings. In addition, the AGS software made assumptions about byte ordering and padding. These assumptions added to the difficulty of porting to new platforms.

Abstract data types, such as the ones in libg++ avoid these shortcomings. Data can be stored in convenient internal representations. The data can be coerced as needed to the most appropriate native data type. Abstract data types can be flexible enough to represent vectors or scalars, character strings, and even data structures. This can vastly simplify application code by avoiding the need for complex “switch” statements.

4 ESRF

The ESRF control system was written in C using a technique called Objects in C. This technique was chosen in lieu of using an object oriented programming language such as C++. At that time, C++ was not supported for the hardware platforms used at ESRF.

A single function, dev_putget(), handles monitoring and controlling accelerator equipment. A vector version handles commands to groups of devices, and an asynchronous version provides for a delayed response.

The ESRF control system does not provide a true abstract data object. Instead, there is a generic data pointer type, DevArgument, and separate type information, DevType. This already succinct API could be made even more succinct with a true abstract data object.

The ESRF directory service is not embedded within the dev_putget() function. There is a separate dev_import() function call which activates the directory service. This function returns a handle which must be used in subsequent dev_putget() calls. This function is roughly analogous to a C++ constructor. The returned handle is similar to a pointer to a C++ object. The directory service indicates whether the service provider is local or remote. If the service provider is remote, the directory service provides network addressing information. In this case, SUN RPC is used to handle parameter passing between the possibly different hardware platforms.

5 ADOIF

The RHIC control system API is called AdoIf[7]. This is an acronym for ADO interface. ADO is itself an acronym for Accelerator Device Object.

AdoIf was written entirely in C++. There is no other language binding for the API. Users are expected to be C++ programmers.

AdoIf provides three functions for controlling and monitoring accelerator equipment. Set() is used to change a current setting. Get() is used to retrieve a current setting or a current reading. GetAsync() is used to be notified of a future change to a setting or reading.

The name GetAsync() has been the source of some confusion. This may be replaced by the name subscribe() in the future. This name is consistent with the paradigm that new data is published to all subscribers.

Set() and Get() have vector equivalents for sending commands to groups of devices.

Data is represented by the C++ class Value. Value objects are abstract data objects. The Value class stores data internally in the most efficient manner. Users may coerce the data into native C++ data types as needed. The Value class also supports an ASCII dump method, which eases the task of writing generic utility programs. Such programs never need to know what type or size the data is. They can merely get the data, then display it in ASCII.

AdoIf has been used on a variety of UNIX workstations, using both RISC and CISC processors, and both Big-Endian and Little-Endian architectures.

The directory service is undergoing a transition from a file-based system to a server-based system. In either system, environment variables allow custom configurations. A custom configuration might combine real accelerator equipment with simulated accelerator equipment. In this way software can be tested even before the accelerator equipment is installed, or when the accelerator is not running.

6 CDEV

The Control Device API (CDEV)[8], is not a complete control system. Instead it is an abstract control system API which may use one or more underlying control systems. Issues such as data translation to different hardware platforms may be handled by the underlying control system.

CDEV was also written in C++. There is an effort underway to support portions of CDEV in Java, another object oriented language. Recently, a C language binding was added to CDEV. This was added at the request of some CDEV users, but does not change the object oriented nature of CDEV.

CDEV provides three messages which are analogous to AdoIf's Set(), Get(), and GetAsync() functions. They are "set", "get", and "monitorOn". All accelerator equipment is expected to respond to these messages.

Data is represented in the cdevData class. Like the Value class, this class provides methods for data coercion to native types, as well as an ASCII dump method.

The directory service is provided by the cdevDirectory class. This class directs requests to the appropriate underlying service provider. This may be a particular control system, a database, or other data acquisition software. By default, there is a single directory service. To provide custom configurations, users may register additional directory services with the CDEV system.

7 CONCLUSION

Several trends in API design are identified. The reasons for these trends are explored. All of these trends ease the job of application programming. Succinct APIs are easier to learn. Object oriented features make it easier to focus on the task on hand. Such features make it less necessary to focus on control system or network weaknesses, or data conversion. Portability allows control system software to run on a variety of hardware platforms. This allows users to choose hardware based on price and performance, rather than on compatibility grounds. Flexible directory services allow software to be written and tested using either real or simulated accelerator equipment. Identifying these common themes help applications programmers from different facilities to speak the same language, and perhaps even share software.

The underlying technology that shapes these trends continues to evolve. Emerging technologies, such as CORBA[9], may become more widely used. Such technologies will undoubtedly steer future trends in programming interface design.

REFERENCES

[1] RFC 1057 A Simple Network Management Protocol (SNMP), Case, Fedor, Schoffstall, and Davin, May 1990.

[2] The Simple Book: An Introduction to Management of TCP/IP-based Internets, Marshall T. Rose, Prentice-Hall, 1991. ISBN 0-13-812611-9

[3] A. Goetz, W.-D. Klotz, J. Meyer, Proc. Int. Conf. Accelerator and Large Experimental Physics Control Systems, Tsukuba, Japan, 1991, KEK Proc. 92-15 pp. 514-519.

[4] Joseph F. Skelly, Proc. Int. Conf. Accelerator and Large Experimental Physics Control Systems, Tsukuba, Japan, 1991, KEK Proc. 92-15 pp. 500-504.

[5] P. Clout, Nuclear Instruments and Methods in Physics Research A 352 (1994) pp. 442-446 North Holland.

[6] Dalesio, Kraimer, and Kozubal, Proc. Int. Conf. Accelerator and Large Experimental Physics Control Systems, Tsukuba, Japan, 1991, KEK Proc. 92-15 pp. 278-281.

[7] L.T.Hoff, J.F.Skelly, Nuclear Instruments and Methods in Physics Research A 352 (1994) pp. 185-188 North Holland.

[8] J. Chen et al, Proc. 1995 Int. Conf. Accelerator and Large Experimental Physics Control Systems, Chicago, Ill. USA, pp. 97-104.

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