The Implementation of an OO Control System API with CORBA

S. Hunt[1], B. Jeram[2], M. Plesko, C. Watson[3],
[1] Paul Scherrer Institute, Villigen, Switzerland
[2] J. Stefan Institute, Ljubljana, Slovenia
[3]Thomas Jefferson National Accelerator Laboratory, Newport News, USA

Abstract

The trend in software development for accelerator control systems, for both the client and server layers, is toward object oriented programming. Commercial developments such as CORBA are now providing object oriented environments for distributed systems which have significant advantages over older procedural libraries.

Object oriented client-server systems using C++ to C++, Java to C++ and Java to Java communications over CORBA have been tested and compared to Java's remote method invocation (RMI) mechanism. Benchmark performance measurements show that CORBA can be of comparable performance to many RPC or raw TCP systems.

An object orient application programming interface (API) has been implemented in a prototype of the control system for ANKA, a 2.5 GeV synchrotron radiation light source being built in Karlsruhe, Germany. This system is based on TACO, a control system architecture with object oriented servers developed at the ESRF and now used in a wider collaboration. Use was made of CORBA and its IDL to define first class objects both on the client and server side. By using CORBA remote methods in both directions, asynchronous processing of events and call-backs are used to provide efficient use of network resources. Static data such as calibration and other data that are stored in databases are accessed directly through object serialization or synchronous client-server methods. Each object that is exported by CORBA has specified methods for its atomic actions, such as get(), set(), getStatus(), on(), off(), etc. Alarm objects are always asynchronously sent to the alarm handler application.

1. Background

ANKA and the SLS are two examples of the new generation of compact, high performance synchrotron light sources that are at present under development. The key parameters for the control system for these and other modern machines are Performance, Functionality, and Cost. Performance is needed to provide responsiveness to the operator, such that no delay is noticed between the setting and readback through the control system of accelerator parameters, and needed for programs that must manipulate a large number of parameters (acquire beam orbit at each step of a tuning algorithm). Functionality includes the provision of now 'standard' features such as archiving, synoptic displays and alarm handling. Cost is becoming an even more important factor in determining the technology to use for the implementation of a control system, and can involve both the cost of control system hardware as well as the cost of developing custom software. To address these concerns, the ANKA control system is based on TACO[1], an object-oriented control system tool-kit that was developed at ESRF. TACO has been upgraded to offer an object oriented API at the client level, as well as at the real-time server level, and is migrating from RPC to CORBA.

2. The evolution of accelerator control systems

The evolution of accelerator control systems has occurred in clearly identifiable phases. From the mainframe era, the Mini-computer (VAX) era, the Workstation era, to the present PC era, some common trends can be discerned: from home-built to commercial tools and standards; from being 'computer scientists' at the leading edge of technology to becoming systems integrators; from expensive (specialized) to cheap, mass market components; and from working in isolation to working in collaboration.

Developers of accelerator control systems have often been slow to admit that their requirements might be met by standard industrial products, services, and protocols. For example, from building network hardware, one started to use Ethernet, but with home-built protocols. Slowly the use of UDP and TCP became common. Later still, higher level protocols such as RPC have become widely used in accelerators. And now, distributed component technology, such as DCOM and CORBA have appeared in control systems. Often in these cases, there is a trade-off between functionality and performance. But the trend is clear: as the performance of commercial products improves, and the cost of providing the desired functionality 'in house' increases, relying on home built systems becomes even less attractive.

3. What CORBA provides

In common with RPC, CORBA[2] provides the ability to build distributed applications, running on heterogeneous systems, without knowledge of the underlying network. By linking with 'stub' procedures that have the same signature as the procedures on the remote system, the programmer can be made (almost) unaware that some of his program is executing on a remote machine. However CORBA goes much further than RPC; it defines an object-oriented, rather than a procedural interface. It also defines a very rich language (IDL) for describing the interface and behavior of

remote objects. CORBA also allows late binding, the discovery of objects at run-time, allowing CORBA objects to be self describing. CORBA objects can be serialized (archived) to and from a variety of persistent storage devices, including files and databases. CORBA defines many of the services needed in building distributed systems such as accelerator control systems. CORBA services include PERSISTENCE, CONCURENCY, NAMING, EVENTS, and SECURITY.

4. Arguments made against CORBA

4.1 CORBA is slow?

CORBA is a standard for implementing distributed systems, it is not a product. The CORBA standards are defined by the Object Management Group (OMG), an industry consortium including the leading computer companies. As such, CORBA is not inherently slow or fast, and one has to examine implementations of the standard. There exists a large spread in performance between different CORBA implementations. This is partly due to products focusing on different target markets. One of the markets for CORBA, embedded systems, clearly identifies real-time performance as a goal. Although some features of CORBA, such as dynamic invocation interface, may cause a performance penalty, one is not forced to use those features where they are inappropriate. Tests using one CORBA implementation (DOME[5]) on low-end personal computers, has shown sub-millisecond response time for synchronous calls.

4.2 CORBA object resolution takes too long ?

CORBA defines a naming service, but it also supports third party naming services. As in existing control system architectures, in CORBA one must choose between a centralized naming service, a totally distributed naming service, or a hybrid offering redundancy but without the overhead of total distribution. Note that a CORBA accelerator object can represent a device, such as a power supply or beam position monitor, which can include many low-level inputs and outputs. Therefore a CORBA system has to bind to many fewer named items than a system that deals with low-level channels.

In one test, CORBA has been shown to resolve references to 1,000 objects within 4 ms per object, which may be considered an acceptable overhead when starting an application program. If this is too slow (and is not brought down fast enough by evolution of hardware and software), there is still the possibility of a custom naming service and batching of name resolution (a technique used in a number of control systems).

4.3 CORBA is too complex ?

Many users of control systems find the change from a procedural to an object oriented program paradigm difficult. Some of the reports that CORBA is complex result from this general difficulty, and do not represent a problem with CORBA itself. However, the language bindings, particularly those that are non-standard, do seem unnecessarily obtuse. One reasonable approach to handle this is to hide the CORBA related calls from the end-user in a well-designed control system API. CDEV is one framework which could easily encapsulate CORBA calls (especially the difficult dynamic invocation calls) within an easier to use API.

4.4 CORBA does not run on real-time systems ?

Vendors offer CORBA on a number of real-time platforms, including OS-9, VxWorks, and Psos. CORBA extensions for real-time have started the standardization process. Some of the issues for CORBA on embedded real-time systems, such as the suitability of dynamic invocation, are being studied.

4.5 CORBA has no asynchronous calls and call-backs ?

OMG has not defined a standardized way to implement call-backs, or asynchronous calls. However this does not mean that these features are not available. As often is the case, some vendors are ahead of the standardization process, and some behind it. Some CORBA vendors offer non-blocking (asynchronous) calls and callbacks as a proprietary extension, while users are always free to implement callbacks themselves.

4.6 CORBA does not scale to massive systems ?

OMG designed CORBA to encompass global networks, with thousands of servers and millions of objects. Although many existing implementations do not support the services, such as efficient name resolution, to support this, there are no inherent bottlenecks to global CORBA networks. Systems such as accelerators, with perhaps hundreds of Orbs and tens of thousands of objects can use existing CORBA implementations. As CORBA does not enforce a single naming service, a 'real-time' naming service, perhaps one already used for accelerator control, might be appropriate for these larger systems.

4.7 CORBA is not supported by Microsoft ?

DCOM, the distributed component architecture from Microsoft is a very strong contender for use in accelerator control systems. It may be that DCOM becomes the de-facto standard, with CORBA occupying only a niche market. If one decides to use only Windows as an operating system, DCOM might be the best choice. However DCOM is not yet as mature as CORBA, it is lacking many of the CORBA services, and many developers are not willing to run only Windows. If DCOM does come to dominate, it might become necessary to port existing CORBA applications. Given the architectural similarities, this should not be too daunting a task. Alternatively it might be better to rely on CORBA-DCOM inter-working products, which are becoming available.

4.8 RMI will replace CORBA ?

Java Remote Method Invocation protocol (RMI)

provides a more 'natural' environment for distributed applications written using only Java. However, Java does not yet seem suitable for embedding in real-time systems. This is mainly due to the much discussed issues involved in garbage collection. At the moment there are examples Java clients talking to servers on real time systems written in C or C++, in which case one cannot use RMI.

4.9 CORBA is not easy to integrate with existing systems?

Many laboratories have a large installed base of software which is not CORBA based, and must be preserved. Two approaches allow integration with CORBA. First, a gateway could be provided between the CORBA and non-CORBA parts of the system. Second, an object oriented framework such as CDEV[3] could be used to integrate the systems at a higher, wrapper layer.

5. CORBA performance

5.1 Visigenic performance using Java

Visigenic offers the most common CORBA ORB - not surprisingly as every new copy of Netscape, said to be the most popular computer application program ever written, ships with a copy of the Visigenic Orb. Visigenic offers Orbs with a native C++ binding for a number of systems, including popular UNIX platforms and Windows. Visigenic also offers an ORB with a Java binding, itself written fully in Java, making it suitable for any platform offering a Java Virtual Machine.

Table 1. Performance comparison of RMI and CORBA.

	RMI	CORBA (Visigenic)
first bind	70 ms	24 ms
remote redraw fillCircle	5.8 - 6.3 ms	4.0 - 4.4 ms

Visigenic however has a relatively high cost for a development licence, and does not run on real-time platforms.

5.2 DOME performance using C++

DOME, from Object Oriented Technologies, is a real-time embeddable ORB. It is available for a number of platforms, including NT, Win95, Linux, Solaris, HPUX, OS-9, and Psos. DOME supports callbacks and asynchronous calls, and has a small footprint, but a CORBA 2.0 compliant version is presently only available under beta release. DOME achieves performance at the cost of some incompatibilities with other Orbs, but this is configurable in the new version, allowing either IIOP or proprietary transport. It is even possible to bypass byte order conversion when systems are compatible, which further increases performance.

Table 2. Performance of DOME

Num. Of Objects	Average Time for Bind (ms)	Time for 1st Object access (ms)	Time for last object access (ms)
1	1.01	0.85	0.85
10	1.40	0.86	0.88
100	1.54	0.88	1.14
1000	2.75	0.89	3.40
10000	14.32	0.88	32.71

These results were achieved between two 133 MHz Pentium PCs running Linux 2.0. As can be seen, DOME does not behave as well when dealing with very large numbers of objects on one server. This is because it uses a linear search algorithm to obtain the object reference. OOT has announced that the new release of Dome has addressed this problem. Dome licenses are free for educational users, and they also offer free licensing for personal use on Linux.

5.3 Other ORBS

A number of other Orbs are worth considering. OMI-Broker is also very fast, is free for non-commercial use, and source code is available. OmniORB is a fully threaded Orb that is available under GNU licensing.

6. Features of a CORBA OO - API

In order for common accelerator applications to be used at more than one laboratory, a common API and basic accelerator class hierarchy are needed. Devices such as Power Supply (or Magnet), Position Monitor, and Vacuum Gauge have to use a minimum set of common attributes. Common attributes include: access control; engineering units; device description; alarm limits; and time-stamp.

6.1 Wide or Narrow API

Accelerator Control system API's have traditionally employed a 'narrow' API - defining a few well-known calls, which take device or channel name and value or command as arguments. Such calls typically are get(), set(), etc.

- get(Booster_PS01,Status,result);

Object oriented languages more typically use a 'wide' API - with each class of devices having its methods, and using an instance of its class per device.

- result = BoosterPS01.Status;

Narrow interfaces are preferable when writing general purpose applications such as save/restore or a synoptic display program. Wide interfaces are more natural for accelerator physics applications which know what attributes they will be dealing with. A control system using CORBA could be implemented with either approach. ANKA has chosen a wide interface, within a defined accelerator class hierarchy.

6.2 Databases and serialization

Default device properties (such as engineering unit conversions), Operational parameters (such as power supply current), and archive data, can be kept local to the server or in a database. CORBA serialization can make the type of storage used transparent to the programmer.

6.3 Organization of Classes

Classes should be independent of the language used (for instance, Java). Therefore there should be no reference to the visualization or GUI for a given object or attribute. Also there is a clear difference between accelerator control objects (power supply, insertion device) and physics objects (orbit). These differences should result in two class hierarchies that share a common root class.

6.4 Root class methods

Some root class methods would be defined in the root class, providing default behavior. Some methods would be virtual, forcing each device class to implement that functionality. All devices should support callbacks and monitors (send data only when changes have exceeded limits). Methods should also be defined to get a list of all devices of a specific type. Locks (mutexes) should be available to the programmer to ensure exclusive write access to resources.

6.5 Standard Accelerator Objects

Standard classes should be defined for common accelerator objects. These should include:

- Power Supply
- RF
- Vacuum-Pump
- Valve
- Kicker
- Septum
- Position monitor
- Profile monitor
- Current monitor

6.6 Introspection

In order to be able to write generic applications, there should be a method to find all attributes of an object. This could be either part of the root class, or use CORBA dynamic binding, or a set of helper objects as is done in CDEV.

6.7 Aggregation and Groups of Devices

Devices of the same or even different classes should be able to be grouped together in logical arrangements (for example SR group, or RF group, or orbit correction group). Methods, including serialization, should operate on groups as well as on individual devices. Another type of aggregation used in CDEV allows a logical device to span multiple servers, for example to have dynamic data in a real-time front end and static data in a commercial database. The exact location of a piece of data is hidden from the user, easing use and maintainability.

6.8 Object Naming

Although a device naming convention is outside the scope of this paper, the OO API should not put unnecessary restrictions on the device naming scheme chosen. Many existing accelerator naming conventions are a mixture of location and type (e.g. SR/PS/Q1 - SR and 1 refer to the location, PS and Q refer to the type of object). Software sharing arguments (and the impossibility of having an international naming convention) implies that such information should be held as an object attribute, or retrieved from a database.

7. Conclusions

CORBA is a rapidly maturing technology, that can provide many of the components needed for building accelerator control systems. However in many laboratories, legacy considerations will delay the introduction of component software technologies. Furthermore, many applications are available for existing API's such as Channel Access, CDEV, and the existing TACO interface. CDEV in particular offers a strong object-oriented environment in which to share software at the application level, independent of the underlying control system and able to integrate with CORBA and non-CORBA API. It is hoped that these existing accelerator software bus technologies, rather than try to compete with commercial component software technologies, will migrate in a controlled fashion to make use of the new opportunities offered.

References

[1]Taco, http://www.esrf.fr/computing/cs/taco/taco.html

[2] CORBA: Architecture and specification, OMG, 1996.

[3] CDEV, http://www.jlab.org/cdev/

[4] Client Server programming with Java and CORBA, R. Orfali. & D. Harkey, Wiley computer publishing.

[5] DOME, http://www.oot.co.uk