CORBA EXPERIENCE AT THE SLS

C. Beny, M. Böge, J. Chrin, M. Grunder, M. Janousch, R. Krempaská, M. Muñoz, A. Streun
Paul Scherrer Institut, 5232 Villigen PSI, Switzerland

Abstract

Beam dynamics applications and components of the beamline experimental controls system at the Swiss Light Source (SLS) have benefitted from a distributed and heterogeneous computing environment in which the Common Object Request Broker Architecture (CORBA) forms the middleware layer and access point to essential software packages. Use is made of CORBA methods provided by the Portable Object Adapter (POA) for accessing ORB functions, such as object reactivation and object persistence, the Implementation Repository (IMR) for the automatic reactivation of servers, and the CORBA Event Service for the propagation of controls and physics data. An account of the experience gained, in the three years since development work began to the present time of first SLS operation, is presented.

1 MOTIVATION

The Swiss Light Source (SLS) is a synchrotron light source located at the Paul Scherrer Institute (PSI) in Switzerland. The SLS was successfully commissioned in August 2001 and has since been delivering light of high brilliance to beamlines occupied by experimenters from a variety of disciplines. Several high-level beam dynamics (BD) applications have been developed for the operation and monitoring of the SLS accelerator facilities. Fig. 1 captures typical components required by BD applications. Their number and demand on computer resources motivated, in part, a desire for a distributed computing environment. To this end, the Common Object Request Broker Architecture (CORBA) [1], now a 'de facto' standard for distributed object computing (DOC), has been employed. Similarly, in the beamline environment, experimenters, faced with the task of integrating distributed devices running incompatible controls systems, have turned to CORBA to provide the middleware to a common experimental controls user interface, as depicted in Fig. 2.

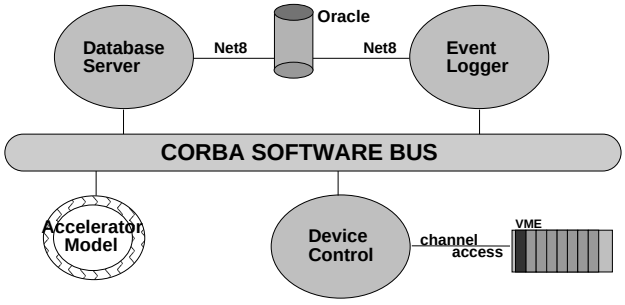

Figure 1: DOC components serving BD applications

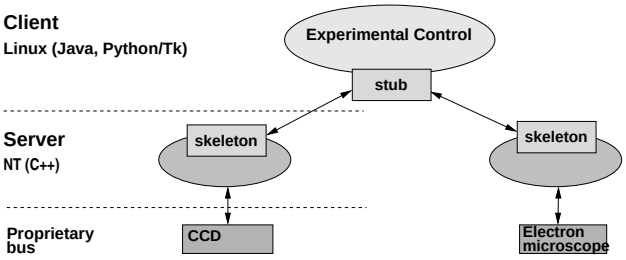

Figure 2: CORBA components of a beamline experimental controls system

The use of CORBA at the SLS has allowed us to realize the potential benefits of distributed computing and to simultaneously exploit inherent features such as the interoperability between objects implemented in various programming languages and on different operating systems. Further details of the CORBA software and hardware environment, as applied to beam dynamics applications and beamline experimental controls, appear in [2-5] and [6], respectively.

2 PRACTISING CORBA

First CORBA experience at the SLS dates from Summer 1999. The principal CORBA product employed was (and still is) MICO [7], available free of charge under the GNU public license terms. At the time, this was one of the most developed ORBs and, being CORBA 2.2 compliant (currently 2.3), offered the Portable Object Adapter (POA). MICO features IDL (Interface Definition Language) mapping to C++. In addition, and significantly for the SLS, a Tcl extension [8] provides CORBA client and server functionality through the Dynamic Invocation Interface (DII) and Dynamic Skeleton Interface (DSI), respectively. MICO does not, however, provide IDL to Java mapping. Several Java-based ORBs were examined, namely JavaORB [9], JacORB [10], before settling with ORBacus (version 4) [11], the only ORB of the three verified to operate fully with MICO.[1] A further ORB in use at the SLS is omniORB[12], chosen for its language binding to Python.

[1]Extensive interoperability tests between MICO and both JavaORB and JacORB revealed a misinterpretation of data due to a misalignment of the binary layout of certain IDL datatypes. The interoperability problems experienced may well have been rectified in the more recent releases of these Java-based ORBs

Experience with CORBA subcomponents in use at the SLS is presented. In particular, use is made of the Naming Service, functions provided by the POA, the Implementation Repository (IMR) and the Event Service.

2.1 The Naming Service

The Naming Service is used to map self-describing names to the obscure stringified object references. A further welcome feature, however, is the naming graph, which comprises a hierarchy of contexts and bindings. A name binding is the term given to a name-to-reference association, while a naming context refers to an object that stores name bindings. A naming graph has been effectively employed in implementing and managing the vast number of object references required for database application objects, where each object maps to a single row (or line) of a database table [13].

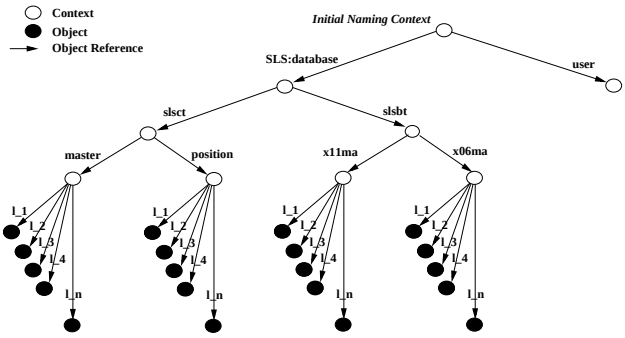

Figure 3: A naming graph for implementing database object references

Fig. 3 shows the naming graph in use for implementing a sample of database object references. Each context object implements a table that maps names to object references that point to either an application object (solid node) or to another context object (hollow node) in the Naming Service. Given a starting context, one can navigate to a target node by traversing a path from a starting context (e.g. SLS:database) through other contexts (e.g. database instant: slsct; table name: position) to the target node (line number: l_n). In this way, a sequence of bindings forms a pathname that uniquely identifies the target object.

2.2 The Portable Object Adapter

The most basic task of the Object Adapter (OA) is to create object references and to dispatch ORB requests aimed at target objects to their respective servants. In its now mandatory version, the POA further provides CORBA objects with a common set of methods for accessing ORB functions, ranging from user authentication to object activatiion and object persistence. The characteristics of the POA are defined at creation time by a set of POA polices. A server can host any number of POAs, each with its own set of policies to govern the processing of requests.

Transient and persistent objects are two categories of objects that relate to the lifespan policies of the POA. A transient object is short-lived with a lifetime that is bounded by the POA in which it was created. A persistent object, on the other hand, is long-lived with a lifetime that is unbounded. It can consequently outlive the very server process wherein it was created. This has several advantages. A server may be shutdown whenever it is not needed to save resources; server updates can be implemented transparantly by restarting the server; in the event of a server crash, persistent objects are able to maintain their identity.

Among the more advanced POA features is the servant manager which assumes the role of reactivating servants as they are required, diligently providing a mechanism to save and restore an object's state. Requests for server reactivation can, alternatively, be delegated to a single default servant which provides implementations for many objects, thereby increasing the scalability for CORBA servers. Appropriate POA policies need therefore be chosen to configure OAs that best fulfill the requirements of the server.

Server objects at the SLS are typically of persistent type and are handled through a servant manager, the servant activator, which provides the incarnate and etherealize operations to save and restore an object's state. Servers that implement a single default servant are also in use. The database server is such an example [13]. As illustrated in Fig. 3, each row of a database table is identified by its own object reference. However, by adopting a single default servant to incarnate each database application object, the number of active objects remains constant, thereby demonstrating the scalability of such CORBA servers.

2.3 The Implementation Repository

While the POA supports and implements persistent objects, it does not handle the administrative aspects of server activations. This is managed by the Implementation Repository (IMR) which stores an activation record for each server process; it is consulted automatically whenever a (re-)launch of a server is mandated. The first instance of the server is started by an IMR administrative process and object references, pointing to the POA Mediator within the ORB daemon process, are exported to the Naming Service. The POA Mediator thus intercepts initial client requests, (re-)activates the server if so required, and forwards the actual server location to the client for all subsequent operations. Thus, by virtue of the activation techniques of the IMR, coupled with the capabilities of the POA, clients are never starved of the servers they require - a tremendous response to reviewers who may doubt the reliability of such a DOC environment. The importance of the IMR cannot therefore be over-emphasized! It is thus perhaps surprising that the actual interface to the IMR is not defined by the Object Management Group (OMG) group [1], which mandates only its existence. Consequently, each ORB hosts its own specific implementation.

2.4 The Event Service

A reactive, event-based, form of programming is supported by the CORBA Event Service which provides services for the creation and management of CORBA event channels. These may be used by CORBA supplier/consumer clients to propagate events asynchronously on a push or pull basis. Event channels are created and registered with the Naming Service allowing clients to obtain object references in the usual manner. Communication is anonymous in that the supplier does not require knowledge of the receiving consumers. Publicized inadequacies of the Event Service are a lack of explicit quality of service (QoS) control, the necessity (for most ORBs) of propagating event data with type *CORBA::any*, and the absence of event filtering. Nevertheless, by applying a few simple design techniques, these limitations can be largely circumvented and the Event Service has been usefully employed in the monitoring of hardware devices and in the distribution of recalibrated data to client consumers. The CORBA Event Service is ultimately to be replaced by the CORBA Notification Service which systematically addresses the shortcomings of the Event Service. A significant improvement is the introduction of the "structured event type", making it possible to distinguish between different event types.

3 PRESENT DIRECTION

MICO is our "backbone" CORBA product. When software development work began in 1999, MICO was already an advanced and well developed ORB, allowing us to consolidate quickly and begin with implementing our client-server model. We were thus able to use features (e.g. the POA) that only later became available in other ORBs. Our present CORBA framework will thus keep us in good stead for the years to come, allowing application developers to continue to make use of the current CORBA framework. Nevertheless, some niggling problems, which have to some extent been circumvented through some simple design considerations, need to be addressed in the near future. We experienced memory leaks in servers that use the PortableServer::Current interface (specifically, the "get_object_id" method) and in C++ clients when catching CORBA exceptions. Tcl client applications that are consumers to an event channel have been known to crash the event daemon when terminated, ungracefully, while in the process of receiving an event.

ORBacus, once a free ORB, now demands license fees! Our use of Java is client based and, as such, our frozen version (ORBacus 4) is sufficiently capable of serving our needs for the near future. However, as ORBs develop and comply to newer OMG standards, it is hoped, and can be expected, that other non-commercial ORBs will provide a viable match to the commercial products.

MICO supports objects by value semantics, wherein a value type can declare both state members and operations/attributes. Such value types could be gainfully employed in the realm of database application objects allowing, for example, a convenient "result set", similar to that returned from database queries within the JDBC API, to be realized [13].

4 CONCLUSION

The CORBA middleware has been used to interface several software packages required by beam dynamics applications and beamline experimental control. The use of CORBA has allowed for the development of a heterogeneous distributed system in which objects, implementented in various languages (C, C++, Java, Tcl/Tk and Python) and on different operating systems, are able to interoperate. Use has been made of the Naming Service, the Event Service, several POA functions and the IMR. In particular, the power and flexibility of the POA, coupled with the activation records stored within the IMR, has been exploited to provide a robust and modular CORBA based client-server framework. The framework has been proved to be both reliable and stable by the many CORBA based applications deployed in the first operation of the SLS.

5 REFERENCES

[1] OMG, CORBA, http://www.omg.org/

[2] M. Böge, J. Chrin, "On The Use of CORBA in High Level Software Applications at the SLS", Proc. 8th Int. Conf. on Acc. and Large Experimental Physics Control Systems (ICALEPCS'01), 27-30 Nov. 2001, San Jose, USA, p. 430

[3] M. Böge, J. Chrin, M. Muñoz, A. Streun, "Commissioning of the SLS using CORBA Based Beam Dynamics Applications", Proc. 2001 Particle Acc. Conf. (PAC 2001), 18-22 June 2001, Chicago, USA, p. 292

[4] M. Böge, J. Chrin, "CORBA Objects for SLS Subjects", Paper ID: 054, Proc. 3rd Int. Work. on Personal Computers and Particle Acc. Controls (PCaPAC 2000), Oct. 9-12, 2000, DESY, Hamburg, Germany; http://desyntwww.desy.de/pcapac/Proceedings/

[5] M. Böge, J. Chrin, M. Muñoz, A. Streun, "Development of Beam Dynamics Applications Within a CORBA Framework at the SLS", Proc. 7th European Particle Acc. Conf. (EPAC 2000), 26-30 June 2000, Vienna, Austria, p. 1354

[6] J. Krempaský et al., "The SLS Beamlines Data Acquisition and Control System", Proc. 8th Int. Conf. on Acc. and Large Experimental Physics Control Systems (ICALEPCS'01), 27-30 Nov. 2001, San Jose, USA, p. 24

[7] MICO, http://www.mico.org/

[8] F. Pilhofer, "Combat, CORBA Scripting with Tcl", http://www.informatik.uni-frankfurt.de/~fp/Tcl/Combat/

[9] JavaORB, http://dog.team.free.fr/index_javaorb.html

[10] JacORB, http://www.jacorb.org/

[11] ORBacus, http://www.ooc.com/

[12] omniORB, http://omniorb.sourceforge.net/

[13] M. Böge, J. Chrin, "Making a Statement with CORBA", Paper ID: TU-02, to be presented at the 4th Int. Work. on Personal Computers and Particle Acc. Controls (PCaPAC 2002), Oct. 14-17, 2002, Frascati, Italy; http://www.lnf.infn.it/conference/pcapac2002/