INTEGRATING CONTROL SYSTEMS TO BEAM DYNAMICS APPLICATIONS WITH CORBA

M. Böge, J. Chrin, PSI, Villigen, Switzerland

Abstract

High level beam dynamics applications typically require access to several distributed components, among which the hardware control system and an accelerator simulation model are crucial. A CORBA Application Program Interface (API) provides clients with the necessary objects with which to develop even the most complex of applications. This is exemplified by the global orbit feedback system at the SLS which is both a consumer to event generated data and a party to remote method invocations on a variety of servers. In particular, use is made of methods provided by the Portable Object Adapter (POA) to create and activate persistent objects, the Implementation Repository (IMR) for the automatic reactivation of servers and the Event Service for the propagation of controls and physics data.

INTRODUCTION

A considerable number of 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 (CORBA), an emerging standard for Distributed Object Computing (DOC), has been employed. Its use at the SLS has allowed to realize the potential benefits of distributed computing, and to simultaneously exploit features inherent to CORBA such as the interoperability between objects of different race (language) and creed (platform). Complex

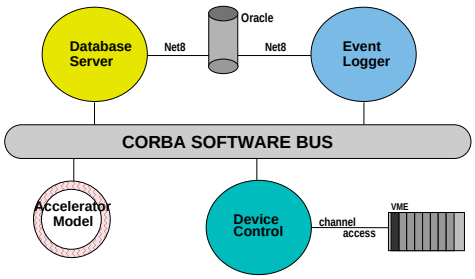

Figure 1: DOC components serving BD applications

tasks, such as the modeling of the SLS accelerators, can thus be handled by dedicated computers, and developed into reusable components that can be accessed through remote method invocations. Persevering with the notion of DOC and developing the entire suite of BD components as CORBA objects, further elevates the level at which applications are designed and implemented. Platforms hosting high-level software applications are no longer limited to the libraries and extensions available to the host operating system as the introduction of a CORBA middleware layer serves to *extend* the developers chosen programming language. BD application developers are, henceforth, able to focus on the specifics of the application at hand, such as determining user-friendly Graphical User Interfaces (GUIs), rather than struggle with the intricate internals of numerous Application Program Interfaces (APIs) and low-level communication protocols.

THE CORBA ARCHITECTURE

The most fundamental component of CORBA is the Object Request Broker (ORB) whose task is to facilitate communication between objects. Given an Interoperable Object Reference (IOR), the ORB is able to locate target objects and transmit data to and from remote method invocations. The interface to a CORBA object is specified using the CORBA Interface Definition Language (IDL). An IDL compiler translates the IDL definition into an application programming language, such as C++, Java or Tcl/Tk, generating IDL stubs and skeletons that respectively provide the framework for client-side and server-side proxy code. Compilation of applications incorporating IDL stubs provides a strongly-typed Static Invocation Interface (SII). Conversely, the Dynamic Invocation Interface (DII) and the Dynamic Skeleton Interface (DSI) allows objects to be created without *prior* knowledge of the IDL interface. Requests and responses between objects are delivered in a standard format defined by the Internet Inter-ORB Protocol (IIOP). Requests are marshaled in a platform independent format, by the client stub (or in the DII), and unmarshaled on the server-side into a platform specific format by the IDL skeleton (or in the DSI) and the object adapter, which serves as a mediator between an object's implementation, the servant, and its ORB, thereby decoupling user code from ORB processing. The Portable Object Adapter (POA) provides CORBA objects with a common set of methods for accessing ORB functions, ranging from user authentication to object activation and object persistence. It's most basic task, however, is to create object references and to dispatch ORB requests aimed at target objects to their respective servants. The characteristics of the POA are defined at creation time by a set of POA policies. A server can host any number of POAs, each with its own set of policies to govern the processing of requests. Among the more advanced features of the POA is the servant manager which assumes the role of reactivating server objects (servants) as they are required. It also provides a mechanism to save and restore an object's state. This, coupled with the use of the Implementation Repository (IMR), that handles

the automated start and restart of servers, realizes object persistency. 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. Fig. 2 shows the

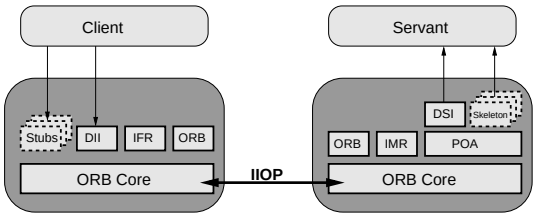

Figure 2: The CORBA client-server architecture

components of the CORBA architectural model. The ORB core is implemented as a runtime library linked into client-server applications.

Client and Server Perspectives

Despite the plethora of new terms and concepts introduced, CORBA, nevertheless, remains true to the DOC objective of providing developers with familiar object-oriented techniques with which to program distributed applications. Indeed, from the client perspective, once an IOR is obtained (typically from a Naming Service which maps names to object references) a remote method invocation essentially takes on the appearance of a local function call: `object->operation(arguments);` whilst the communication details of client-server programming are essentially hidden from the client, a more intimate involvement with the ORB is required when developing servers. In particular appropriate POA policies need to be chosen to configure object adapters that best fulfill the requirements of the server.

Exploiting the POA

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 transparently by restarting the server. In developing a DOC environment, the command to start a server may be replaced with a remote shell invocation and the next server instance run remotely, without the client being aware. Persistent objects also maintain their identify after a server crash. Whilst the POA supports and implements persistent objects, it does not handle the administrative aspects of server activations. This is managed by the IMR which stores an activation record for each server process; it is consulted automatically whenever a (re-)launch of a server is mandated. Thus, by virtue of the capabilities of the POA, and the activation techniques of the IMR, BD applications are never starved of the servers they require.

The Event Service

A reactive, event-based, form of programming is also 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 CORBA 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. The CORBA Event Service has been usefully employed in the monitoring of hardware devices and in the distribution of recalibrated data to client consumers.

THE SLS CORBA SERVERS

Server objects, typically of persistent type, have been developed using the CORBA 2.3 compliant product MICO [1]. The services which these objects provide are briefly reported here (for a more details see [2] and [3]):

Accelerator Model: A dedicated host ("Model Server" in Fig. 3) runs the servers ("TRACY Servers") that perform the modeling of transfer-lines, booster and storage ring. Procedures utilize selected routines from the TRACY accelerator physics library [4], enabling clients to employ accelerator optimization methods *online*.

Device Controls: The CDEV C++ class controls library provides the API to the EPICS-based accelerator device control system. The "CDEV Server" running on the host "CORBA Server" supplies functionality for both synchronous and asynchronous interactions with the control system. Monitored devices and recalibrated data are propagated to clients through CORBA event channels. Recently an interface to the EZCA C controls library ("EZCA Server") has been added for increased performance.

Database Access: A database server provides access to Oracle instances through the Oracle Template Library (OTL) and the Oracle Call Interface (OCI). Methods executing SQL statements that perform database retrieval and modification operations have been provided.

Logging Facility: A CORBA message server has been developed using the the UNIX syslog logging facility. Runtime messages are sent to the logger with various priority levels, the threshold for which can be adjusted dynamically for any given servant.

A COMPLEX CORBA APPLICATION

One of the most complex CORBA applications at the SLS is the implementation of a slow and the system integration of a fast global orbit feedback system. The system is based on 72 Beam Position Monitors (BPMs) and 72

correctors in the horizontal and vertical plane distributed around the storage ring. The corrector/BPM response matrix is "inverted" using SVD in case of a non quadratic response matrix taking into account *all* eigenvalues.

The realization of the global orbit feedback has been carried out in two steps. A Slow Orbit Feedback (SOFB) with relaxed requirements ($<$ 3 Hz correction rate) is in operation since August 2001 [5]. The experience gained with the various subsystems served as a basis for the implementation of a Fast Orbit Feedback (FOFB) (4 KHz orbit sampling rate) which is presently under commissioning [6].

The Slow Orbit Feedback (SOFB)

Since the beginning of SLS operation global orbit corrections have been successfully applied manually by the operators with the help of the Tcl/Tk CORBA GUI "oco Client". Due to the inherent modularity of the CORBA environment, thoroughly tested underlying CORBA components like the "TRACY Server" and the "CDEV Server" could be combined to implement the SOFB. In this case the operator is "replaced" by a C++ CORBA client ("Feedback Client") which initiates an orbit correction at a given rate (see Fig. 3). For the SOFB the digital BPM system [7]

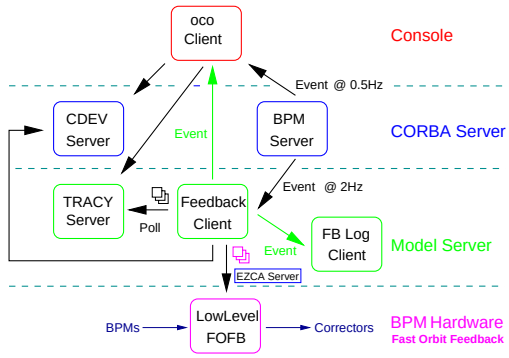

Figure 3: Schematic View of the SOFB: the "Feedback Client" on the "Model Server" level replaces the operator driven GUI "oco Client" on the "Console" level. It gets BPM data from the "BPM Server", asks the "TRACY Server" for the model predicted corrector pattern and applies the correction through the "CDEV Server".

is operated in an injection triggered mode which provides "stroboscopic" position readings averaged over 2 ms at a rate of 3 Hz with a resolution ≈ 0.3 μm. A "BPM Server" monitors, collects and sends the BPM data to the "Feedback Client" with 2 Hz. A low pass filter is applied to several successive BPM data sets. The data are then sent to the "TRACY Server" which predicts a corrector pattern to restore the "Golden Orbit" which is defined by the orbit centered in the quadrupoles. Additionally, local bumps at the location of the insertion devices are taken into account in order to steer the photon beam according to the demands of the experiments. Finally, the proposed correction is applied by toggling between the horizontal and the vertical plane resulting in a SOFB correction rate of ≈ 0.4 Hz.

The Fast Orbit Feedback (FOFB)

In the SLS storage ring a properly chosen BPM/corrector layout leads to an "inverted" response matrix where only the diagonal and their adjacent coefficients have non zero values. Thus, corrector settings are only determined by position readings from nearby BPMs [8]. This feature allows to run the FOFB decentralized, integrated in the 12 BPM stations of the storage ring where each of the stations handles 6 BPMs and correctors. The BPM data are transmitted over fiber optic links between adjacent stations. In order to provide well defined starting conditions for the FOFB the SOFB corrects the orbit to < 5 μm rms. The FOFB gets initialized and started through the SOFB which downloads the "Golden Orbit", 6×6 sub-matrices of the "inverted" 72×72 response matrix, 1×6 sub-matrices of the "inverted" 72×1 off-energy response matrix and the FOFB loop PID parameters to the BPM stations (see Fig. 3). The SOFB continues to run in a "watchdog" like passive mode without touching any corrector other than the RF frequency. But it keeps monitoring BPM and corrector values. Whenever a BPM is switched off, declared faulty or a corrector is close to saturation the FOFB is stopped and restarted with adapted settings. The off-energy content of the horizontal orbit is taken into account by the FOFB but corrected by the SOFB.

CONCLUSION

CORBA extends the capabilities of the programming languages used by BD application developers, thereby elevating the level at which complex applications exemplified by the orbit feedback system at the SLS are designed and implemented. The flexibility of the POA, coupled with the server activation records within the IMR, can be exploited to provide a robust and modular client-server framework.

REFERENCES

[1] A. Puder, K. Römer, "MICO, An Open Source CORBA Implementation", 3rd Edition, Pub: dpunkt.verlag, Heidelberg, December 1999.

[2] M. Böge et al., "Commissioning of the SLS using CORBA Based Beam Dynamics Applications", PAC'01, Chicago, June 2001.

[3] M. Böge, J. Chrin, "On the Use of CORBA in High Level Software Applications at the SLS", ICALEPCS 2001, San Jose, November 2001.

[4] J. Bengtsson, "TRACY-2 User's Manual", SLS Internal Document, February 1997; M. Böge, "Update on TRACY-2 Documentation", SLS-TME-TA-1999-0002, June 1999.

[5] M. Böge et al., "Orbit Control at the SLS Storage Ring", EPAC'02, Paris, June 2002.

[6] T. Schilcher et al., "Commissioning of the Fast Orbit Feedback at the SLS", Contribution to this Conference.

[7] V. Schlott et al., "Commissioning of the SLS Digital BPM System", PAC'01, Chicago, June 2001.

[8] M. Böge et al., "Fast Closed Orbit Control in the SLS Storage Ring", PAC'99, New York, March 1999.