PAUL SCHERRER INSTITUT

SWISS LIGHT SOURCE

SLS-TME-TA-2005-0281
September 27, 2005

Developments to the SLS CORBA Framework for High Level Software Applications

M. Böge, J. Chrin

Paul Scherrer Institut, CH-5232 Villigen PSI, Switzerland

Abstract

A CORBA based client-server framework has been in active service since the start of SLS in 2001. It provides for an uniform interface to a variety of persistent objects required by beam dynamics applications in the domain of controls, accelerator modelling, database transactions and message logging. The framework underwent significant development during the course of the last two years motivated by advances in hardware capabilities, recent releases of CORBA packages and a renewed analysis of user requirements. Several additional software modules were developed and integrated into the updated framework to facilitate rapid application development. These include event processing agents which serve to aggregrate low-level hardware data to produce complex events which supply summarized data to event channels for distribution to registered clients. The recent updates to the framework are presented, together with an account of how complex events are created and processed for delivery to registered consumers through the CORBA Event Service.

Talk presented at the 10th International Conference on Accelerator and Large Experimental Control Systems (ICALEPCS'05), October 10-14, 2005, Geneva, Switzerland

DEVELOPMENTS TO THE SLS CORBA FRAMEWORK FOR HIGH LEVEL SOFTWARE APPLICATIONS

M. Böge, J. Chrin
Paul Scherrer Institut, 5232 Villigen PSI, Switzerland

ABSTRACT

A CORBA based client-server framework has been in active service since the start of SLS in 2001. It provides for an uniform interface to a variety of persistent objects required by beam dynamics applications in the domain of controls, accelerator modelling, database transactions and message logging. The framework underwent significant development during the course of the last two years motivated by advances in hardware capabilities, recent releases of CORBA packages and a renewed analysis of user requirements. Several additional software modules were developed and integrated into the updated framework to facilitate rapid application development. These include *event processing agents* which serve to aggregrate low-level hardware data to produce *complex* events which supply summarized data to event channels for distribution to registered clients. The recent updates to the framework are presented, together with an account of how complex events are created and processed for delivery to registered consumers through the CORBA Event Service.

INTRODUCTION

The initial interest in using CORBA as an high-level middleware in the development of beam dynamics applications dates from Summer 1999 [1]. Several generic tasks common to high-level applications were identified and developed into reusable components as CORBA objects, providing functionality for accelerator device control, data calibration and analysis, accelerator modelling, database operations and message logging. The use of CORBA further served to realize the potential benefits of distributed computing, an important consideration given the computer intensive accelerator model procedures, and provided for the interoperability between objects implemented in different programming languages. The broad scope of objects developed subsequently *extended* the developers preferred programming language, typically Java and Tcl/Tk. Application developers could henceforth focus on the specifics of the application at hand, such as developing user-friendly graphical interfaces, rather than be faced with the intricate details of the many application programming interfaces (APIs).

The CORBA framework underwent significant development during the course of the last two years. Modern computer hardware was commissioned to meet with the increasing demands imposed on the server hosting the CORBA objects, referred to as the 'Model Server', and recent versions of CORBA software packages were likewise installed. In addition, further software development served to consolidate the manner in which selected data from the low-level hardware are aggregrated by *intelligent agents*. These act to trigger *complex* events that are propagated to high-level applications through event channels supplied by the CORBA Event Service. Clients need only subscribe as a consumer to the appropriate event channel to passively receive updated values.

HARDWARE AND SOFTWARE PLATFORMS

Beam dynamics applications are constructed using a client-server framework [1, 2, 3, 4] wherein client applications typically operate on consoles located in the accelerator control room and connect to a dedicated 'Model Server' that negotiates access to the controls hardware, the accelerator model and the database on the client's behalf. With increased demands from high-level applications, e.g. the incorporation of the control of the fast orbit feedback [5] into the orbit correction application [6], the 'Model Server' hardware of old was replaced by a dual-processor Dell PowerEdge 2650 Server running Red Hat Linux v. 7.3. The server features two 2.8 GHz Xeon processors with Hyper-Threading Technology, 512K memory cache per CPU, 2 GByte of RAM, a 1 GBit/s Ethernet card and a RAID controller that duplicates data onto a second, hot-plug, 36 GByte SCSI hard drive providing additional data security. It is also equipped with two hot-plug power supplies. A second identical server is available to provide redundancy. The introduction of the high performance hardware was combined with the installation of recent releases of our CORBA products namely MICO (C++ mapping), ORBacus/JOB (Java) and Combat (Tcl). The new MICO release [7], our principal ORB, reflects a tighter implementation of the specifications formulated by the Object Management Group (OMG) [8]. In particular, a more robust Event Service was necessitated that now provides the transport mechanism by which logical sets of control and physics data are propagated to high-level applications.

Client applications run on Linux PC consoles, typically equipped with a single-processor Pentium 4 2.8 GHz CPU, 512K Cache and 2 GByte of RAM. The Red Hat Package Manager (RPM) is used for the distribution of the required software.

With the continual advancement of computer hardware capabilities and evolving Linux kernels, a preference towards the Scientific Linux operating system, a rapidly emerging Linux standard in the experimental physics community, is anticipated. In preparation for these developments we have also successfully implemented our CORBA framework on a quad-processor server machine hosting Scientific Linux v. 3.0.4. The server features four 2.2 GHz Pentium processors with 2 MByte memory cache per CPU, 4 GByte RAM, four 73 GByte SCSI hard drives and a RAID controller.

AN EVENT SERVICE FOR DATA PROGAGATION

The CORBA synchronous request/response exchange is the standard means of communication between a client and server. Following a reappraisal of user requirements, however, it became apparant that several situations exist where the accustomed two-way communication model is not the optimal means of data transfer. One example is when a group of related devices, of interest to many clients, changes value. Each client would be required to either poll the server repeatedly for updated values or establish a more involved callback procedure. In such cases, data propagation is better served through an alternative, *reactive*, form of programming wherein clients are notified *en masse* of updated values. Such a delivery mechanism has been specified by the Object Management Group's (OMG) Common Object Services (COS) Specification Volume [8] in the form of the Event Service and its extension, the Notification Service. The latter, being a recent addition to the original OMG COS Specification, is yet to be released in MICO. The OMG Event Service supports decoupled communication between multiple suppliers and consumers. The following describes how low-level hardware events are aggregated to produce *complex* events that are propagated through the Event Service providing applications with a more personalized view of a given component of a control system.

Complex Event Processing

Fig. 1 illustrates how control data are aggregrated by an *event processing agent* (EPA) to produce a *complex* event which is propagated to those applications subscribing as consumers to the designated event channel. A typical EPA uses the CDEV API [9] to establish a callback mechanism to the EPICS based control system [10], the communication protocol of which is channel access. In general terms, the EPA is a simple object that consists of *event pattern rules*, comprised of a *trigger* and a *body* of actions, and local variables whose values form its state. The EPA monitors its input to detect instances of the rule triggers. When a match is detected, the agent executes the action of the rule's body. Events that are output depend on the class of the EPA [11]. Three proven types are i) *filter* - reduces event executions to relevant subsets, ii) *map* - aggregrates and correlates events, and iii) *constraint* - detects proper and improper behaviour.

The *event pattern 'map'* is the class of most relevance here. Maps use *event pattern rules* to aggregate a *partially ordered set (poset)* of events into high-level events and, as such, are the basis for defining relationships between sets of system-level events and higher level abstraction events. Fig. 2 shows a generic template for a map agent that aggregates causal sequences of events in its input, and creates an event consisting of a sequence of values that summarize the aggregated data. With reference to Fig. 1, an invocation of the CDEV callback function causes the EPA to change its local state variable(s) and its output event. When the trigger is satisfied a *complex* event, i.e. an aggregration of the low-level hardware event, is formed and routed to its designated event channel.

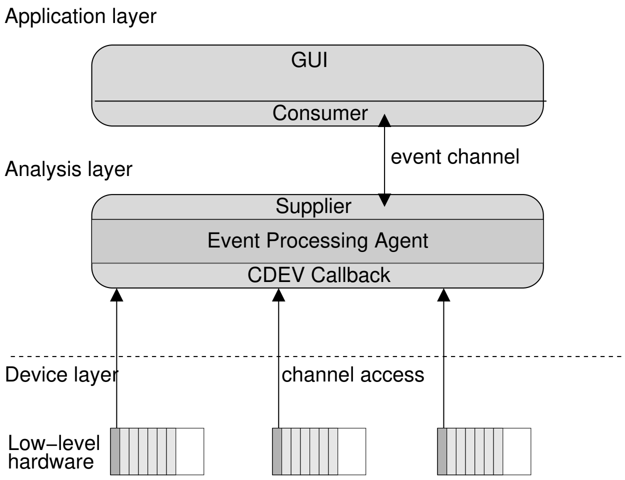

Figure 1: Data aggregation and propagation.

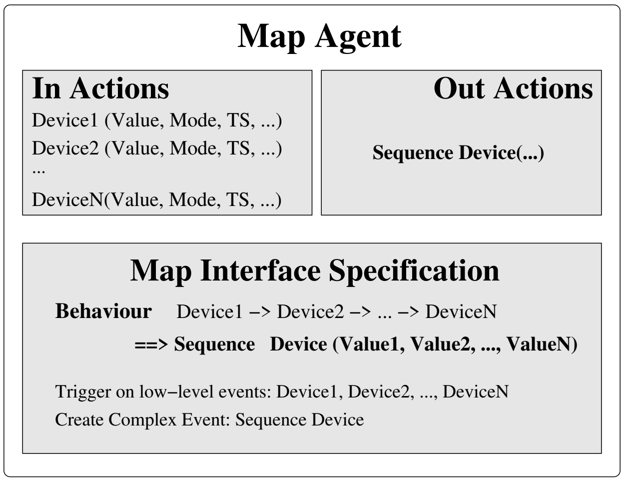

Figure 2: An event processing agent implementing the event pattern map.

EPAs exist for the aggregration of data from various types of hardware devices, including Beam Position Monitors (BPMs) and various magnet groups, such that the corresponding event data provide a personalized view of a given component of the control system. Two uses of EPAs, which exemplify their purpose, are briefly desribed.

EPAs for BPMs Separate EPAs exist for BPMs from the different accelerator facilities, namely the injectors, the booster and the storage ring. The principal task of these EPAs is to aggregate and analyze a set of BPM data, and to supply the summarized results to specific event channels serving various clients. These include the high-level Tcl/Tk based orbit correction application and the Java application responsible for the determination of the effects of the insertion devices on the closed orbit.

In addition, an EPA has been developed for the analysis of the entire contingent of BPM waveforms. For the storage ring, this constitutes 216 waveforms from 72 BPMs. Independent averages are calculated by the EPA, both over the complete waveform and over all waveforms for a given waveform element, to produce a complex event that is supplied to the specified event channel. This latter data is of relevance when the storage ring is operating in turn-by turn mode.

It is interesting to note that the viewing of these complex events at an high-level could be credited with the detection of anomalies that would otherwise not have been detected if only the low-level events were monitored. The onset of such an high-level anomaly would initiate a trace back procedure in order to locate the cause of the problem at the system-level.

The Tune EPA The computer-intensive calculation of the machine tune parameter has also been incorporated into an EPA. *Posets* from a dedicated 'Tune' BPM are aggregated and when the *event pattern rule* is triggered, through a data transfer complete acknowledgement, the tune calculation is performed. A 'physics' output event is subsequently created, the form of which is a tuple of data containing a sequence of the measured vertical and horizontal components of the tune values. The EPA additionally stores the full data complement, including that of the 'Tune' BPM waveforms, in virtual memory space, avoiding time consuming input/output operations. A dedicated CORBA 'Tune Server' provides methods that enable a client to both regulate input parameters to the tune calculation and to retrieve the full complex of results from the virtual data store.

Event Delivery Models

In the OMG Event Service model, suppliers produce events and consumers receive them. Events are propagated through an event channel which acts as a mediator between the consumer and supplier. Communication is anonymous in that the supplier does not have knowledge of the receiving consumers. Event channels support different models of event delivery, the type of which depends on the collaboration between suppliers and consumers. This is illustrated in Fig. 3 which highlights the push and pull mechanisms established between the event channel and the supplier/consumer. The various push-pull permutations lead to the four event delivery models shown.

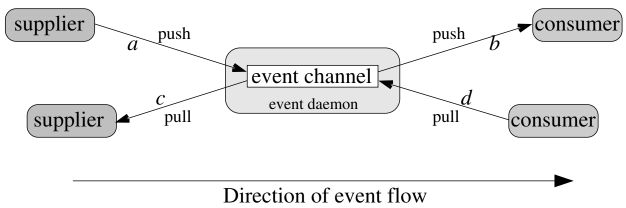

Figure 3: The event delivery models. Path $a \rightarrow b$ represents the canonical push model; path $c \rightarrow d$ the canonical pull model; path $a \rightarrow d$ the hybrid push-pull model; path $c \rightarrow b$ the hybrid pull-push model.

Fig. 3 further serves to emphasize that an event channel is able to fulfill all four roles simultaneously. Note that while the flow of events is always from supplier to consumer, the invocation of the method call, by which the event is transmitted, can be in either direction.

The task of creating event channels, which exist in the address space of the CORBA Event daemon, has been intentially kept separate from the EPAs which, by design, serve as the single data source to a given event channel. Rather, event channel creation is handled by a separate program that is initiated at server boot time. New event channels may also be added on the fly by adding entries to a configuration file. Typically event channel data is grouped according to accelerator components (e.g. storage ring, booster and injectors) and device types (e.g. magnets, BPMs).

The event channels are the primary source of information for users and are optimized to satisfy their reporting needs and so that no further data manipulation is required on the client side. Indeed, much of the data on display in beam dynamics applications are received in this way, with the canonical push event delivery mechanism being the most employed.

Migration To The Notification Service

The CORBA Event Service implements a publish/subscribe application paradigm that provides for a natural programming style in which pertinent data can be passively received by any interested client application. Certain drawbacks nevertheless exist including the necessity to propagate event data under the auspices of type *CORBA::Any*, the absense of event filtering and the lack of explicit quality of service (QoS) control. These limitations have been addressed by the OMG Group and are alleviated in the Notification Service largely through the introduction of the *structured event type*, which provides a well defined data structure into which different event types may be mapped. The Event Service is eventually to be superseded by the Notification Service such that CORBA releases that include the Notification Service no longer require an Event Service implementation to be compliant with OMG Specifications. A first examination of the Notification Service reveals many notable features, including the ability to filter out unwanted data and to further publish and transmit only those precise events for which there are interested clients. The efficiency and performance of the Notification Service would however first need to be appraised, an important consideration being the speed with which event filtering and delivery is accomplished. Since the Notification Service is a super-set of the Event Service, the present event delivery model could otherwise be preserved.

SUMMARY

An updated CORBA based software framework has been implemented on recently commissioned high-performance hardware offering improved response times and greater stability. The software developments have served to consolidate the CORBA Event Service as the standard event delivery mechanism for the propagation of aggregated low-level and other data to beam dynamics applications. Several *event processing agents* (EPAs) that act as data suppliers to the CORBA event channels have been integrated into the new framework. The EPAs are responsible for the capture of data from components of the low-level control system, their transformation according to predefined rules and their subsequent delivery to event channels. These event channels are the primary source of information for users and are optimized to satisfy their reporting needs. The updated CORBA based framework has proved to be both reliable and stable by the many applications deployed in the operation of the SLS.

REFERENCES

[1] M. Böge, J. Chrin, “A CORBA Based Client-Server Model for Beam Dynamics Applications at the SLS” ICALEPCS’99, Trieste, Italy, p. 555.

[2] M. Böge, J. Chrin, “CORBA Objects for SLS Subjects”, PCaPAC 2000, Hamburg, Germany.

[3] M. Böge, J. Chrin, “On the Use of CORBA in High Level Software Applications at the SLS”, ICALEPCS’01, San Jose, USA, p. 430.

[4] M. Böge, J. Chrin, “Integrating Control Systems to Beam Dynamics Applications with CORBA”, PAC’03, Portland, USA, p. 291.

[5] T. Schilcher *et al.*, “Commissioning and Operation of the SLS Fast Orbit Feedback”, EPAC’04, Lucerne, Switzerland, p. 2523.

[6] M. Böge, B. Keil, A. Lüdeke, T. Schilcher, “User Operation and Upgrades of The Fast Orbit Feedback at the SLS”, PAC’05, Knoxville, USA.

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

[8] OMG (CORBA), http://www.omg.org/

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

[10] S. Hunt *et al.*, “The Control and Data Acquisition System of the Swiss Light Source”, ICALEPCS’99, Trieste, Italy, p. 615.

[11] D. Luckham, “The Power of Events”, Pub: Addison-Wesley, 2002.