Production experience with CORBA in the $BABAR$ experiment

S. Metzler[1]*, A. Adesanya*[2]*, S. Dasu*[3]*, T. Glanzman*[2]*, G. Grosdidier*[4]*, A. Samuel*[1] *and G. Zioulas*[2] *for the $BABAR$ Computing Group*

[1] California Institute of Technology, Pasadena, CA, USA
[2] Stanford Linear Accelerator Center, Menlo Park, CA, USA
[3] University of Wisconsin, Madison, WI, USA
[4] LAL-IN2P3-CNRS, Université Paris-Sud, Orsay, France (Currently at SLAC)

Abstract

CORBA is a collection of standards to address the apparent "distribution of objects" between processes and nodes, along with an extensive set of services. These standards are being updated and augmented frequently; implementations have struggled to keep up. The $BABAR$ experiment decided in 1998 to adopt the use of CORBA for a limited number of applications both to fulfill a functional need and to evaluate this emerging technology. This plan has been successfully implemented and numerous applications are now in production. This paper focuses upon our experience using CORBA. We briefly describe the scope of these applications, then discuss system performance, reliability and other operational issues. Practical aspects of code design and implementation will be mentioned. Finally, some conclusions about our experience and some recommendations about future use will be presented.

Keywords: CORBA, IPC, $BABAR$, ACE/TAO, Parallel Farms, Control Systems, Online

1 Introduction

From the earliest days of planning for an asymmetric B-factory project at SLAC, the computing planners embraced the concepts of object design and distributed processing. This has translated into the use of the C++ programming language, large numbers of nodes running Unix and a high-performance network to facilitate control and data communications. Part of the puzzle was clearly how to augment C++ and Unix system calls to achieve interprocess and interprocessor communication. For the most performance-demanding applications, such as event data acquisition, the TCP socket API was adopted. For other applications, however, we provisionally adopted in early 1998 the emerging technology known as the *Common Object Request Broker Architecture* (CORBA).

Initially, our use of CORBA was strictly limited to evaluations [1] and a small number of pilot projects. The success of these pilot projects opened the door for CORBA's more extensive use within $BABAR$. Today, numerous CORBA applications are successfully running in production. This paper briefly describes a selection of these applications and concludes with observations emphasizing the lessons learned and recommended directions for future CORBA usage in $BABAR$.

2 $BABAR$ Usage

CORBA is used in $BABAR$ to facilitate object-level, interprocess communication between C++ processes running on the same platform. It is also used to communicate between processes written in Java and C++ on different platforms. $BABAR$ chose the TAO [2] implementation of CORBA that works with the ACE system toolkit for Unix after reviewing the *Object Request Broker* (ORB) implementations of several vendors. All the applications discussed below use TAO for the C++ implementation of CORBA. Java implementations use the native CORBA implementation found

in the Java Development Kit (JDK). There are other $BABAR$ applications [3, 4] using CORBA that are not discussed in this paper.

2.1 Objectivity Browser

The $BABAR$ database group [5] is responsible for managing around 300 TB of physics events per year. An object database system, *Objectivity/DB* (OBJY), is used to store the data, which is distributed across several networked disk and tape servers. Support tools must be provided to allow users to view information and assist administrators in exporting databases to remote sites. While some tasks are well suited to batch operation, *graphical user interfaces* (GUIs) are of equal importance. CORBA addresses two key issues, object-level, interprocess communication and language interoperability, related to the development of the $BABAR$ GUI database browser [6].

Because of the size and structure of the physics data, a client/server framework is well suited for providing the foundation of a browser system. Servers reside close to the data sources while the front end was designed for wide-scale desktop deployment. Java was chosen for the client implementation because of its platform independence. OBJY provides several language APIs but, unfortunately, only C++ provides the database group with the required functionality. CORBA provides a high-level, platform-independent protocol for communication between Java and C++.

2.2 Distributed Histogramming

$BABAR$'s Online Event Processing (OEP) system uses 32 farm nodes that run the same Level-3 triggering program. Monitoring data quality in Level-3 is essential for quickly finding system problems. Shift crews must also have graphical access to the data summed over all nodes and from an individual node. The Distributed Histogram Package [7] provides a solution for these needs.

The original solution used native sockets to pass data and commands between processes. User-defined serialization and de-serialization methods were defined for the available data types. The writing of these methods was complicated by concerns for supporting multiple platforms and languages since we wished to use Java for locally and remotely displaying online data.

A new communication system was written using CORBA. This provided a very high-level tool for interprocess communication. Object serialization and de-serialization, referred to as "marshaling", was no longer an implementation or maintenance issue, whereas it was the most significant issue for communication using sockets. This made the CORBA solution significantly easier to extend to new object types.

A generalized Java browser, part of the Distributed Object Tree package [7], was added to the system to allow the display of any user-defined object through the use of a type registry that mapped known viewers to object types. The object definitions were shared through an IDL file, which allowed the separation and parallelization of writing the client and server.

2.3 Ambient Data Handling

Ambient data from the Detector Controls system [8] are accumulated by 27 archiving processes, running on two nodes. The data are periodically stored in a database [9]. The data are encapsulated in C++ objects, based on the source of their origin, e.g., high voltage power supplies, pressure systems, etc.... The ambient objects are stored in shared memory. A server is spawned for each archiver to monitor the data in real-time. The server uses CORBA to distribute the data over the network to Java clients for browsing and analysis. The ambient objects are translated into CORBA structs and are added to a Distributed Object Tree in a hierarchical structure. Browsers connect to individual servers using the CORBA name service. A user can navigate the Object Tree and select individual objects to display their time history. A similar server allows access to ambient data

stored in the database. The size of the ambient objects transported with CORBA is about 0.5–1.0 MB with a typical transfer time of 1–2 seconds.

2.4 Online Prompt Reconstruction

The logging of events received from the Level-3 trigger farm and the subsequent event distribution to the Online Prompt Reconstruction (OPR) farm nodes are handled by instances of the "logging manager". The logging manager for the OEP case collects events from 32 farm nodes at up to 100 Hz. For OPR, the logging manager distributes events to about 100-200 daemons running on worker nodes. The daemons receive events and drive offline framework jobs to reconstruct them.

The logging manager is configured to work in OEP or OPR mode by using CORBA. The actual event transport is handled using native TCP sockets for optimal use of resources. We tried using CORBA, but we found that it imposes a factor of 2 penalty in throughput and a factor of 5 in CPU resources. However, for monitoring and control, CORBA performance is adequate for almost all of our use cases. One exception found during commissioning was that, when 32 nodes start synchronously and try to reach the name service, there were often delays adding up to several tens of seconds. We were forced to find an alternative solution to satisfy the short timeout period imposed by the data aquisition system.

The status of the logging manager buffers and connections are monitored using CORBA. This live monitoring is essential to indicate the online data taking status and to measure the event-processing rate in OPR. We typically monitor the logging manager status every 10 seconds for the online case and every minute for the OPR case. Additional monitoring via CORBA is available from each of the OPR daemons.

OPR is also using CORBA to achieve interprocess control and monitoring over its widely distributed farm through yet another set of lightweight servers called "Global Farm Daemons". The ACE/TAO library, including the multi-threaded facilities available in ACE, was used for this implementation. The server has proven to be robust and reliable, while the client was shown to be scalable. It has been able to broadcast commands to up to 250 target nodes within seconds [10].

3 Observations and Conclusions

We discuss in this section the observed performance and reliability of CORBA using TAO. We then discuss operational issues and conclude with our recommendations for future use.

3.1 System Performance and Reliability

We have found that CORBA is much slower than native TCP, but is sufficient for graphical client/server applications. The data throughput rate is about half of TCP and the connection overhead seems to be much greater. This gives communication rate reductions of about a factor or 10 for small data transfers. This rate reduction is essentially unnoticeable for our graphical applications.

We have found ACE/TAO to be an invaluable tool for interprocess communication. When we worked with early beta releases, we often had problems and occasionally found bugs. The ACE/TAO group has been very responsive to problems and has promptly fixed reported bugs. We have found that the reliability of TAO has improved and we now consider it to be highly reliable.

3.2 Operational Issues and Lessons Learned

Software developers new to CORBA need to learn several pieces of the technology before they can be effective. We have found that enough can be learned by an experienced programmer within one week to be effective. We have also found it useful to wrap TAO within $B_A B_{AR}$ code. This provides

both insulation from changing APIs and configurations and an easier way to handle common functionality such as registering with the name service, un-registering and starting the ORB.

The use of the name service has helped with the organization and connection of the many processes that are used within the $BABAR$ online system. We have been able to avoid the development of our own name service or its equivalent. We have also avoided hard-wiring the host names and port numbers of servers.

IDL is very useful for separating the development of client and server applications. We have been able to build GUIs in parallel with server development by creating the IDL, implementing the skeletons with a very simple client and server, and allowing more sophisticated client and servers to be developed against the simple server and client, respectively.

IDL is an excellent way to avoid difficult migrations in serialization and de-serialization of objects. CORBA encapsulates all the details of marshaling and data passing. This issue is particularly important when multiple languages and multiple platforms are supported for communication.

For more demanding applications, multi-threading may be essential because of the high overhead of making remote calls. Systems that will involve communication with large numbers of identical processes should investigate threading issues with CORBA from the beginning. We are investigating multi-threading as a way to improve the performance of our applications.

3.3 Conclusions and Recommendations

CORBA is a very useful product for interprocess communication. We have found that the recent versions of TAO are reliable. Performance is inhibited compared with TCP, but is acceptable for many of our applications.

We have found that elementary CORBA is relatively easy to use. It provides services and abstractions that are useful for the long-term maintenance of a project. It is much easier and more robust than using TCP directly. There are many services available in CORBA that we have not yet explored. To become familiar with all the features of CORBA would take a considerable amount of time. However, we have found CORBA fruitful while only using a small fraction of the services.

We recommend the use of CORBA for applications that can afford to forego the better performance of TCP. We also recommend the use of TAO because of its reliability and user support.

References

1 S. Yang *et. al*, "CORBA evaluations for BaBar Online System", CHEP'98, Chicago, Autumn 1998.

2 D. C. Schmidt *et. al*, "The Design and Performance of Real-Time Object Request Brokers", Computer Communications, Elsevier Science, Volume 21, No. 4, April, 1998.

3 J. Perl *et. al*, "$BABAR$ WIRED - The BaBar Collaboration's Implementation of the WIRED Event Display", CHEP2000, Padova, Winter 2000.

4 S. Du *et. al*, "CORBA scripting in HEP and beyond", CHEP2000, Padova, Winter 2000.

5 D. Quarrie *et. al*, "Operational Experience with the BaBar Database", CHEP2000, Padova, Winter 2000.

6 A. Adesanya *et. al*, "An interactive browser for BaBar databases", CHEP2000, Padova, Winter 2000.

7 S. Metzler *et. al*, "Distributed Histogramming", CHEP'98, Chicago, Autumn 1998.

8 J. Olsen, "$BABAR$ Online Detector Control", CHEP2000, Padova, Winter 2000.

9 G. Zioulas, "The $BABAR$ Online Databases", CHEP2000, Padova, Winter 2000.

10 G. Grosdidier *et al.*, "Sending Commands and Managing Processes across the BABAR OPR Unix Farm through C++ and CORBA", CHEP2000, Padova, Winter 2000.