SLAC-PUB-9177
September 2001

The Event Browser: An Intuitive Approach to Browsing BaBar Object Databases

Adeyemi Adesanya

Presented at the International Conference on Computing in High-Energy and Nuclear Physics (CHEP'01), 9/3/2001—9/7/2001, Beijing, China

Stanford Linear Accelerator Center, Stanford University, Stanford, CA 94309

Work supported by Department of Energy contract DE–AC03–76SF00515.

The Event Browser:
An intuitive approach to browsing BaBar Object Databases

Adeyemi Adesanya
Stanford Linear Accelerator Center, California, USA

Abstract
Providing efficient access to more than 300TB of experiment data is the responsibility of the BaBar[1] Databases Group. Unlike generic tools, The Event Browser presents users with an abstraction of the BaBar data model. Multithreaded CORBA[2] servers perform database operations using small transactions in an effort to avoid lock contention issues and provide adequate response times. The GUI client is implemented in Java and can be easily deployed throughout the community in the form of a web applet. The browser allows users to examine collections of related physics events and identify associations between the collections and the physical files in which they reside, helping administrators distribute data to other sites worldwide. This paper discusses the various aspects of the Event Browser including requirements, design challenges and key features of the current implementation.

Keywords: BaBar, CORBA, Java, C++

1 Introduction

1.1 The largest Object Database in the world?

There's little doubt that the BaBar physics detector experiment poses some major computing challenges. Since May 1999, more than 300TB of data have been stored in an ODBMS that is widely regarded as one of the largest in the world. The Databases Group who are based at SLAC, develop the software infrastructure allowing BaBar users to store and retrieve data in the form of persistent objects. Through an independent interface, physicists are shielded from the underlying ODBMS implementation (Objectivity/DB)[3]. In addition, information is arranged according to an experiment hierarchy that is recognized by the user community.

1.2 The Event Store

A major part of the BaBar persistent object interface is the Event Store. This sub-system manages persistent data representing the vast number of particle collisions recorded by the detector. Event collections refer to sets of related collisions or physics events and are the basis of user's data analysis tasks. The Event Store provides a mechanism for accessing collections through a tree structure similar to a modern filesystem. Collections are identified using pathnames. Although collections may be unique within a particular database system, events may be shared by many collections. Collections themselves may also be referenced from other collections. The term used for such a relationship is a Master Collection. Individual events may be accessed from a collection through iteration or direct reference. A 'tag' can be regarded as being a small summary of an event's state and provides an efficient means of querying collections for sub-sets of events with similar attributes. Physicists use a C++ API to manage collections via the Event Store but an interactive tool would provide a convenient method of browsing.

[1] Information about the BaBar experiment can be found at http://www.slac.stanford.edu/BFROOT
[2] Common Object Request Broker Architecture specification available from http://www.omg.org
[3] Objectivity/DB product information is available at http://www.objectivity.com

2 Key Requirements

2.1 Scalability

General-purpose database tools are completely unaware of the Event Store or any other BaBar structures. They can only interpret the physical architecture of the system. In the case of Objectivity/DB this means that access to objects can only take place by referencing database files, containers, pages, etc. which mean very little to the average physicist and does not scale to 300TB!

2.2 Discreet transactions

Interactive browsing of the Event Store demands adequate response times. However, browser related transactions should be discreet - designed to minimize the impact on any jobs that attempt to update databases. A database system must not grind to a halt because the browser is being heavily used. Administrators routinely import and export collections between different database systems. During such operations, files must be made quiescent in order to insure consistency.

2.3 Distributed processing

Roughly 600 BaBar collaborators are spread over 72 institutions. Some of these sites also maintain local database installations. A browser system will have to provide access to these sites. For optimum I/O performance, it's best to process persistent data close to where the associated filesystems reside. But we can also assume that nearly every potential user will have some form of desktop computing power.

3 Interoperability

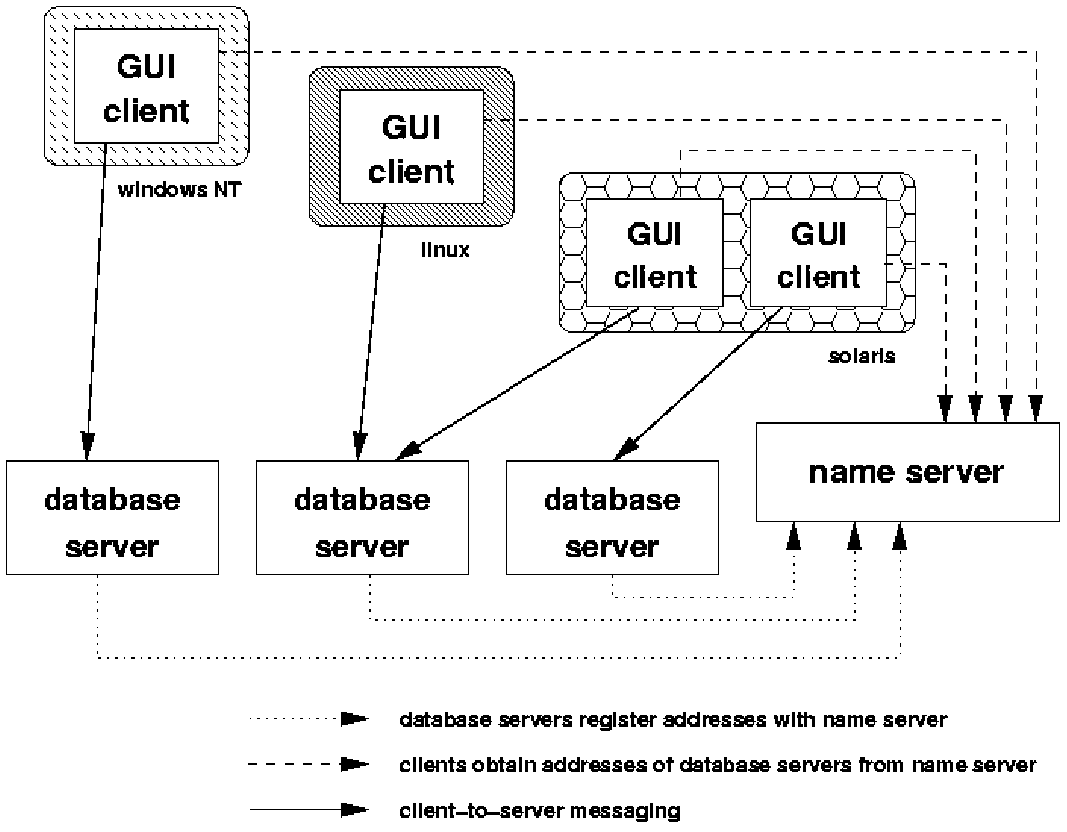
3.1 C++ vs. Java

The computing guidelines attempt to regulate the use of software tools and technology within BaBar. C++ has always been seen as the official programming language of the experiment and this extends to the database API. However, Java has been steadily gaining acceptance and support, especially in the area of graphics and visualization. Java's fully featured standard class library includes an extensive GUI toolkit. Theoretically, porting programs is a non-issue because compiled code runs on a platform independent virtual machine.

3.2 The CORBA architecture

Java is clearly suited to implementing user interfaces but it was guaranteed that database operations would be written in C++. A distributed object communications protocol was sought. The solution had to provide a mechanism for defining inter-object messaging interfaces as well as cross-platform interoperability. CORBA claimed to deliver both. CORBA services are described by defining data structures and methods using the IDL (Interface Definition Language) syntax. IDL Pre-processors read IDL files and generate client and server 'stubs' for a specific platform and programming language. Server stubs are compiled with the code containing the service object implementation. Client-side stubs are used to invoke methods on the service. A key component of the CORBA architecture is the ORB (Object Request Broker). ORBs handle all the steps involved in communication including resolving references to services, establishing connections and synchronizing requests. Several services are included in the standard's specification including a Naming Service that maps logical names to object references. This can be thought of as a directory for CORBA objects.

Each server in a browser system registers its address with a global Naming Service. The clients obtain the list of available servers from the Naming Service and the user can then select which database system to browse.


GUI client
windows NT
GUI client
linux
GUI client
GUI client
solaris
database server
database server
database server
name server
database servers register addresses with name server
clients obtain addresses of database servers from name server
client–to–server messaging

3.3 Proving the model

There were some initial doubts regarding the feasibility of the CORBA model. The process of converting data types to platform independent structures during distributed messaging is known as marshalling and can be significant overhead in certain situations. Work began on an implementation to discover the extent of the design's strengths and limitations. Choosing the right ORB can help to minimize the marshalling overhead. The TAO[4] C++ ORB was designed specifically for real-time environments that require robust, reliable and fast services. On the Java side, the natural choice was to select the ORB implementation provided in the standard environment. Avoiding 3rd party packages kept things simple. Past experience has taught us that lengthy installation requirements may discourage BaBar users from trying new software. Tests confirmed that the CORBA protocol was not a bottleneck in our model. Marshalling performance is also related to the size and complexity of the data types passed during messaging. The functionality of the prototype was extended and evolved into a complete package that was made available to the BaBar community.

[4] TAO real time CORBA documentation available at http://www.cs.wustl.edu/~schmidt/TAO.html

4 System Features

4.1 Viewing the Event Store tree

The central GUI component in the client is a tree widget that displays the Event Store in the style of a directory filesystem. Users can quickly locate event collections of interest by accessing folders that correspond to event collection path components. The response time is quick because the tree is constructed on the fly. When the user opens a folder, the information required to display its contents (a list of sub-folders and/or collections) is obtained from a browser server at that exact instant. The Database transactions associated with these operations are very short in duration to avoid lock conflicts with high-priority tasks.

4.2 Multithreaded capability

The browser servers are able to process client requests concurrently because both the ORB and Objectivity/DB support multithreading. Currently, browse servers can support either a thread pool or create a single thread per client connection. The thread pool model obviously consumes less CPU resources but the thread-per-client model dedicates a database context to each client thread. This results in an increased chance of 'hitting' the database cache when accessing persistent data.

4.3 Querying the Schema

The Database Schema contains definitions of all the persistent classes within a database system. The browser uses the Active Schema API to obtain these class definitions at run-time without any prior knowledge of a persistent object. This allows users to examine detailed raw event data. In fact this feature is generic and may be used to browse any persistent object.

4.4 Client Applet

Because the Client GUI is written in pure Java, it took little effort to produce an applet capable of running within most popular web browsers. Not only does this simplify deployment, launching the GUI may be as easy as selecting a hypertext link from a BaBar web page. By default, Java's security manager restricts the network activities of applets. To overcome this, the applet code was digitally "signed". Signing provides a way of authentication and preventing the code from being tampered with. Upon detecting a signed applet, the Java run-time will verify the signature and give the user the option of granting the applet permissions that include unrestricted network connections and local filesystem access.

5. Conclusion & Future Plans

The Event Browser succeeds in providing interactive access to the Event Store. Physicists are shielded from the underlying database system architecture and can navigate their way through more than 300TB of data. The CORBA model may not be the ideal solution for all applications but in this case, the benefits of a cross platform object protocol outweighed any performance overheads. As the number of users increases, we will learn more about the system's behavior under heavy loads but so far the signs are encouraging. User feedback will provide ideas for extending and re-using features. Administrators have already expressed their interest in using the browser to aid data distribution. Elements of the browser may also be used as a way of providing input to an event display or analysis systems.