An Overview of the *BABAR* Conditions Database

I. Gaponenko, D. Brown, D. Quarrie[1], E. Frank[2], S. Gowdy[1,3]

[1] Lawrence Berkeley National Laboratory, USA
[2] University of Pennsylvania, UK
[3] University of Edinburgh, UK

Abstract

This paper presents special database software developed in the context of the *BABAR* experiment used to keep the detector alignments, calibrations constants, as well as other time-dependent records of the conditions under which the experimental events are taken..

keywords *BABAR*, database, Objectivity

1. Introduction

The *BABAR* [1] experiment at the Stanford Linear Accelerator Center studies CP violation in decays of neutral B mesons produced in electron-positron collisions at the PEP-II asymmetric B-factory. The experiment began its operation in May 1999, and will take data over the period of approximately 10 years.

A special database is being used to track detector alignments, calibrations constants, and other time-dependent records of the conditions under which physics events are taken. This layered software complex, providing the mechanism for storing, retrieving and managing these data is referred to as the Conditions database. The *BABAR* Conditions database software makes use of Objectivity/DB [2], a commercial Object Oriented database with C++ language interface, as the underlying storage technology.

In this paper we present a wide spectrum of aspects associated with the current design and implementation of the Conditions database software, ranging from its architecture, the internal structure of the database, to the experience gained during the first months of the detector's operation. This discussion is concluded by trends in foreseen future development of the Conditions database software.

2 The design of the Conditions database

At the highest level of abstraction the Conditions database may be viewed as a collection of so-called *conditions functions* providing the values (represented as persistent objects) for the conditions parameterized by the *validity time*, which serves as the primary access key. A 64-bit unsigned integer is used to specify the validity time, although access granularity is determined by the uppermost 32-bits, corresponding to a resolution of one second.

Each conditions function covers the whole history of the corresponding condition ranging from a logical *minus infinity* (starting from January 1, 1900, UTC) through a logical *plus infinity*, thus covering the period of 2^{32} seconds. Internally this history is split onto a sequence of intervals, each covering the period where the value of the condition is constant.

The possibility of multiple versions of conditions for the same validity time is also available in the Conditions database. This requires the use of a secondary key for the conditions function specified above. This vertical dimension is called the *revision*.

Any modifications to a conditions function are also recorded in a special persistent journal, which is specific to each function. This provides for better control over modifications of the functions as well as providing for the possibility of rolling back accidental modifications when required.

2.1 Major components of the Conditions database

The following figure (Figure 1) represents the major building blocks of the database software.

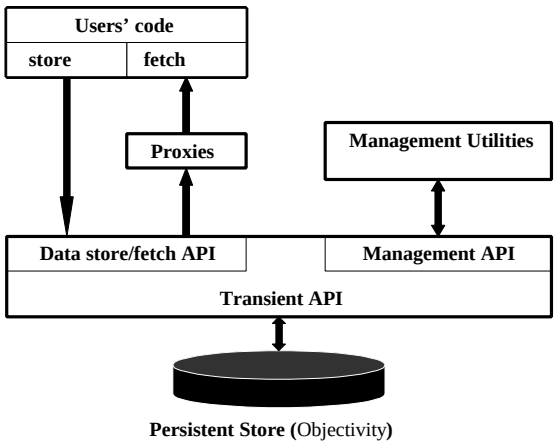

Figure 1: The structural components of the Conditions database software

It includes the following components:

- The specially structured persistent store (based on the Objectivity/DB) as a data store;
- The core code of the Conditions database software implemented as C++ and DDL classes grouped into three software packages;
- A layer of proxy classes between the application and core interfaces of the database;
- A number of management tools (utilities).

Generally there are three different types of applications:

- Applications loading data into the database. These work directly through the core API.
- Applications fetching conditions from the database during event reconstruction or detector calibration. This code typically accesses data through proxies;
- Special management utilities managing the contents of the database. A special management API is provide for these.

Proxies are responsible for a number of ancillary tasks, including transaction management, conversion of conditions objects from a persistent to transient representation, and dealing with schema compatibility issues. The core packages provide just a base class for proxies, leaving the specific implementations of them up to the developers of final (detector-specific) proxies.

In order to improve the overall performance of the database two levels of caching have been implemented in the Conditions database software. The first level, built into the core classes, caches the most recently accessed persistent handles pointing to the database object, container object and a condition object itself. This cache is used by applications fetching the data from the database. Another level of caching deals with the transient representations of conditions objects inside proxies.

All of this effectively reduces the number of the "real" I/O operations for the most frequent data access patterns - when the condition values are accessed sequentially at the validity time.

2.2 Database API Functionality

The main functions provided through the Conditions database API are:

- A two-layered name space for the condition functions;
- The "store" interface to store new Conditions data into the database. This interface can store both a single value for a condition or a vectors of values, thus increasing the performance of applications;
- The "fetch" interface allowing stored data (values of condition functions) to be retrieved from the database. It's possible to iterate in both dimensions in the history of a specific condition;
- The revision (vertical dimension in the condition function) interface. This interface supports the creation and management of revisions;
- An extensive management API intended to manage the contents of the database.

2.2.1 The name space for the conditions functions

All the conditions functions have symbolic names in the database. According to the adopted convention these names must be the names of the corresponding persistent (DDL) classes. The functions are grouped into detectors, forming two layered name space (see Figure 2), made from detectors and function names. Each detector group has independent set of names.

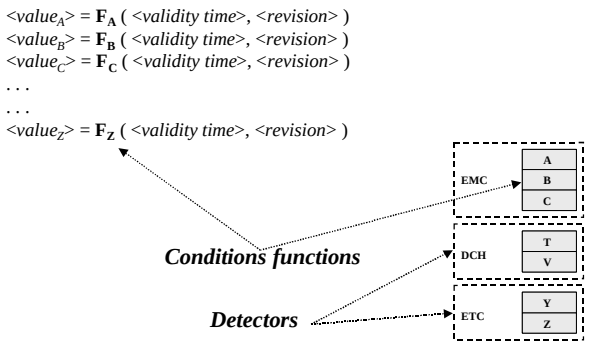

Figure 2: Two-layered name space for conditions functions

2.3 The internal structure of the Conditions database

Internally, in the persistent store of an Objectivity federation, the conditions data are spread between three types of databases (see Figure 3): *link*, *index* and *object*.

The *link* databases serve as catalogues (namespaces) for the conditions functions. There is one *link* database per detector group. Each named container in these databases corresponds to a single condition function. This container has a symbolic link to a persistent container in the index database.

The *index* databases maintain the time and revision history or shape of the condition functions, including all bookkeeping information for each function. As in case of the link databases, there is one-to-one correspondence between a condition function and a persistent container. There may be

more then one such database per detector, each corresponding to a different *origin* (a federation where the corresponding condition function was created and is managed).

The third type of databases is used for storage of the condition objects themselves, providing the specific conditions information. The objects are referenced through the index databases.

Such a multi-layered architecture provides more efficient clustering of the Conditions data, and subsequently faster access to those data. This separation for example makes possible browsing the history of a specific condition (the shape of the condition function) without loading huge amount of data from the object databases when it's not needed.

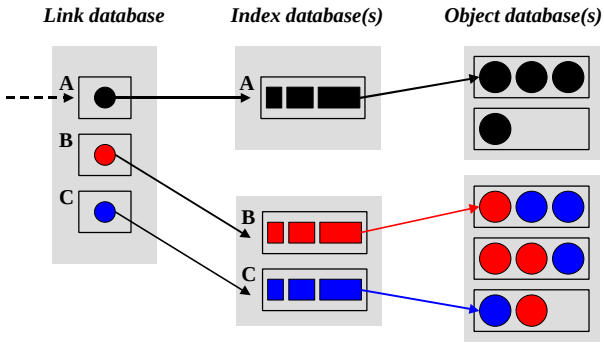

Figure 3: The navigational path between databases in a federation

3 Conclusions

The Conditions database has been in production use since the turn on of the *BABAR* experiment for physics in May 1999. Since that date it has gone through a number of performance improvements and extensions in its functionality.

Currently the total amount of data stored in the Conditions database has reached 2 GB. The database includes about 250 different condition functions, grouped into 10 detector groups (7 of them represent the real sub-detectors of the *BABAR*).

While both the performance of the database and its functionality satisfy the initial requirements, including the non-functional ones, such as reliability and performance, there are several significant improvements in the implementation of the software that are foreseen. Among them are: developing better management tools and Java-based GUI tools for browsing (and possibly managing) the contents of the database.

We are also considering extending the current implementation of the database to serve conditions data using the distributed CORBA [3] technology in order to decouple most of the database clients (those ones reading the conditions data only) from a specific federation. This would also essentially reduce the amount of relevant management work both at SLAC and at the remote collaborators' sites.

References

[1] D. Boutigny at. al. "BaBar Technical Design Report", SLAC-R-95-457
[2] http://www.objectivity.com/
[3] http://www.omg.org/