LCG CONDITIONS DATABASE PROJECT OVERVIEW

A.Valassi, D. Düllmann, CERN, Geneva, Switzerland
A. Amorim, N. Barros, T. Franco, D. Klose, L. Pedro, FCUL, University of Lisbon, Portugal
S. A. Schmidt, Institut für Physik, University of Mainz, Germany
V. Tsulaia, University of Pittsburgh, USA

Abstract

The LCG Conditions Database project was launched in 2003 in the context of the LHC Computing Grid (LCG) Persistency Framework. The aim of the project is the implementation of a common persistency solution for the storage and management of the conditions data of the LHC experiments at CERN. Conditions data, such as calibration, alignment or slow control data, are non-event experiment data describing the state of the detector at the time of data taking, and are characterized by the fact that they vary in time and may have different versions. The LCG project draws on previous activities that have led to the definition of generic C++ APIs for conditions data access and their implementation using different storage technologies, such as Objectivity, MySQL or Oracle. The project is assigned the task to deliver a production release of the software including implementation libraries for several technologies and high-level tools for data management. This paper reviews the current status of the LCG common project at the time of the CHEP 2004 conference and the plans for its evolution.

INTRODUCTION

The LCG Conditions Database project [1] was launched in July 2003 in the context of the Persistency Framework [2] of the LHC Computing Grid [3] Applications Area [4]. The aim of the project is the implementation of a common persistency solution for the storage and management of the conditions data of the Large Hadron Collider (LHC [5]) experiments at CERN, which are scheduled to start operation in 2007.

Background and goals

The project draws on a rich background of previous activities in the area of conditions data for LHC and other experiments. This includes, in particular, the collaborative effort of some experiments (mainly LHCb, ATLAS, HARP and COMPASS) and the Database Group of the IT Department at CERN, between 2000 and 2001, to define a common C++ API [6] for conditions data access, and the successive implementations of this API using different database storage technologies, first Objectivity [7], later Oracle [8] and MySQL [9]. The experience of the BaBar experiment with Conditions Databases (see [10] for an updated status report) significantly influenced the definition of the original common API, and continues to be of great relevance to the LCG project.

Production quality conditions data from existing experiments have already been stored using all three of the above mentioned storage technologies. The Objectivity implementation was used by the HARP experiment to store its conditions data, which were later migrated to the Oracle implementation of the same original API [11]. The migration of the HARP conditions data and software was significantly simplified by the existence of the abstract API, which resulted in the decoupling of the user application from the details of the underlying storage technology. While the Objectivity and Oracle implementations share the same API, the MySQL-based Conditions Database software implements an extended API providing extra functionalities, which will be reviewed in a later section of this paper. This implementation, integrated into the ATLAS software framework [12] and supplemented by useful data browsing tools [13], was used to store the conditions data describing the experimental setup of the ATLAS test beams, first in 2003 and then also in 2004.

Given this background, the LCG project was set up in 2003 [14] with two tasks: first, that of integrating the existing activities and software implementations into the scope of the LCG Applications Area; second, that of coordinating the discussion about their evolution, and of promoting the agreement on a common programme of work that would allow the LHC experiments to share the C++ software and API to store and retrieve their conditions data, as well as the tools to manage these data in a distributed computing environment.

CONDITIONS DATA

In contrast to “event” data, which contain information about the response of the detectors to the passage of the particles generated in each interaction of the two collider beams (an “event”), “conditions” data record the state of the detector at the time when events are collected. Many different types of conditions data exist, such as the descriptions of the detector geometry and signal readout maps, the measurements of detector controls data such as temperatures or voltages, and also the alignment or calibration coefficients that may be computed by the processing of other sets of event and non-event data.

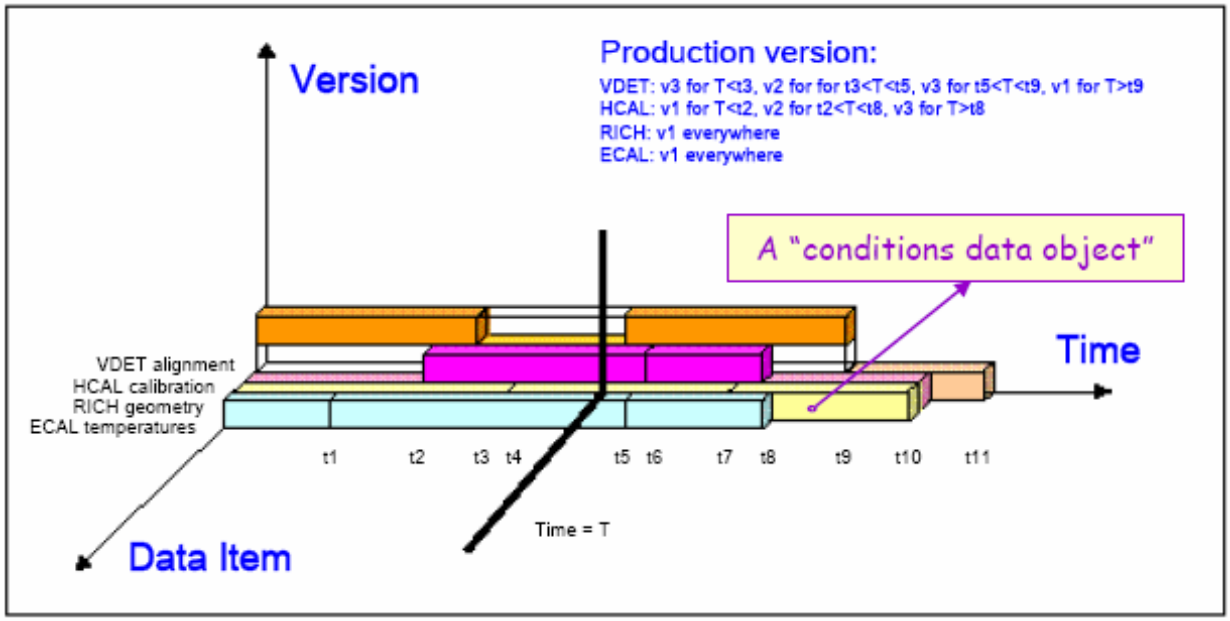

Figure 1 The three axes for uniquely identifying each "conditions data object" in the Conditions Database [6].

While their size is typically a few orders of magnitude smaller than that of event data, conditions data are extremely important because they are needed for the reconstruction and analysis of the events taken using the detector they describe. In addition, conditions data may also be useful to detector experts to identify and solve any malfunctioning of the apparatus for which they are responsible.

The main property of conditions data is that they vary with time. Each value or set of values of conditions data describes the state of the detector during a limited lapse of time, and should only be used for the analysis of the events collected during that "interval of validity" (IOV). The IOV duration of conditions data depends on their specific type, ranging from seconds or minutes for environmental data such as temperatures or pressures to months or years for detector geometry descriptions. For rapidly changing values, averages or interpolations may be used to reduce the amount of conditions data needed for offline event processing.

In addition to their time variation, certain items of conditions data may also exist in different versions: this is typically the case for those quantities that are themselves the result of a computational algorithm, such as calibration or alignment data. In this case, the most appropriate version of these data must be used for event processing. Conversely, conditions data produced by direct online measurements, such as temperatures or voltages, only exist in one version.

The original "common API"

The properties of conditions data outlined in the previous section are reflected in the data model displayed in Figure 1, on which the original common API is based. In this model, the set of values that are needed to represent a given conditions data "item" (such as the alignment or calibration of a specific subdetector) are encapsulated into a "conditions data object", the smallest atomic entity of conditions data that can be manipulated individually. For a given data item, each conditions data object has an interval of validity and may exist in more than one version. Three pieces of metadata are needed to lookup each conditions data object that is needed to analyse a physics event: the data item identifier, the time point for which a valid object is required, and a version number. Typically, a "tag" name is used for lookup instead of a version number: for each data item, tags are special subsets of all existing objects, such that only one object is valid in the tag for any given time. Note also that each object is stored in the database with an associated interval of validity (i.e. a pair of time values indicating the start and end of the IOV), but a single time point is needed for lookup.

In addition to its metadata, each object is also associated to its actual conditions data "payload", i.e. the set of values of the physical quantities describing the state of the detector (such as temperatures, or calibration coefficients). In the original "common API", the data payload associated to each object must be stored in the database as a byte stream, i.e. a string or a large binary object (BLOB). The idea behind this approach is that the experiment-specific software is responsible for encoding the relevant data structures into strings or BLOBs, either encapsulating the data directly into the byte streams stored in the Conditions Database, or using the string associated to each conditions data object to store references to data structures stored outside the Conditions Database (such as POOL object tokens [2] in their string representation, or external file names). As an example, both of these approaches were used in HARP, where conditions data objects were associated either to streamed vectors of numbers or to the names of calibration files.

The MySQL "extended API"

The main difference between the "common API" approach and that followed in the "extended API", used for the MySQL implementation, is that the latter allows end users to specify the payload associated to each conditions data object stored in the Conditions Database as a list of attributes of simple data types (such as floats, integers, Booleans or strings), or vectors of them. In the relational implementation of the extended API using MySQL, where each object is represented by a row of a relational table, the payload attributes of each object are naturally stored as columns of the table. In addition, the extended API offers the option to disable versioning for "online" data that are the results of direct measurements and only exist in a single version, allowing better optimization of storage and performance for these data.

The MySQL implementation was extensively used to store the conditions data collected at the ATLAS test beams. The overwhelming majority of the 10 GB of data in the database were stored using the extended API, most often using the "online" option to store detector controls data generated by the ATLAS PVSS system [12].

PROJECT STATUS AND OUTLOOK

In line with its goals, the LCG Conditions Database project has been active in two areas since it was launched over one year ago: first, in the integration of the existing Oracle and MySQL implementations into the LCG Application Area scope (support for the Objectivity implementation having been dropped in 2003 [11]); second, in the review of the two implementations and their APIs, and in the planning and discussion of a common program of work for their evolution into software and tools that may be shared by several LHC experiments.

So far, the progress of the project has been rather slow. One of the main reasons for this has been the lack of committed manpower available for the development of common software of tools, summing up to well below 2 FTEs averaged over the past 12 months. This count does not include the larger development effort within ATLAS [12-13] to maintain and extend the software and tools for the storage and management of the test beam data, as this activity mainly focused on experiment-specific needs and requirements. A second problem faced by the project has been the lack of a consistent approach between the APIs of the two existing implementations, which has significantly limited the possibility for fast development of further software components and tools along a unified direction. Discussions are underway and are expected to converge within a few weeks to a single unified API that could satisfy the different requirements of all interested groups in the LHC experiments. Additional manpower is also expected on the same time scale.

Software releases

The public software releases issued so far by the LCG Conditions Database project aimed at integrating the existing Oracle and MySQL implementation into the LCG development environment, while functionality enhancements are only foreseen for future releases. The software is maintained and built using the infrastructure services and tools provided by the LCG SPI project [15].

Release 0.1.0 (April 2004) consists of the two packages CondDBOracle and CondDBMySQL, resulting from the move of the existing Oracle and MySQL implementations from their previous development environments into the LCG CVS repository and SCRAM build system. Each package contains its own set of API header files and its own set of tests and examples. In release 0.1.1 (May 2004), support for the gcc3 compiler on Linux was added for the Oracle implementation, thanks to the release of the Oracle OCCI client library for this platform.

Release 0.2.0 (July 2004) introduced a common dependency of CondDBOracle and CondDBMySQL on a third package ConditionsDB, containing the header files for the common "BLOB" part of the two APIs. In order to achieve this, both APIs had to be slightly modified with respect to release 0.1.1, to remove some incompatibilities with the original common API that were introduced in the MySQL extended API. The MySQL support for relational data payload and "online" data is now properly encapsulated in additional header files that are only implemented in CondDBMySQL.

The current development work is focusing on two areas: first, on the development of common utilities, including tools to selectively extract and distribute "slices" of the conditions database, containing only specific data items and/or data valid in restricted time ranges; second, on prototyping the integration of the existing software with SEAL [16] and POOL, for instance by developing a component that would allow end-users to transparently retrieve the POOL objects corresponding to tokens stored (in their string representation) in the string payload of a conditions data object. The next release 0.3 will contain the work completed in these areas.

Limitations of the present software

As already mentioned, the main limitation in the current software provided by the Conditions Database LCG common project is the lack of a consistent approach between the two existing APIs and implementations, which is effectively delaying further development of software components and tools along a unified direction.

A particularly important side-effect of this problem is that it is presently very difficult to copy conditions data across the two back-ends. In the case of BLOBs, the difference between the relational table schemas chosen to implement the API using Oracle and MySQL implies that data can only be copied either at the level of the C++ API, or using SQL at the RDBMS level after developing dedicated scripts [17]. More importantly, no Oracle solution currently exists to store the MySQL conditions data collected using the extended API.

The question whether future developments should include an Oracle implementation of the MySQL extended API or a reimplementation of a new unified API

for both technologies is presently being discussed. In this context, it should also be noted that the performance of the present Oracle implementation of the original BLOB API is inadequate, as it is slowed down by the lack of bulk operations for the C++ access to the database. While well understood, this implies that significant development work is in any case needed on the Oracle implementation, providing additional support for the idea of more extensive redesign.

Future perspectives

Once agreement is reached on the new common API, an interesting possibility to solve both the problem of data distribution between Oracle and MySQL and that of the poor performance of the Oracle implementation may be that of implementing the agreed API using the POOL Relational Abstraction Layer (RAL) [18]. In this approach, which would decouple the Conditions Database implementation code from the persistent storage technology, the same relational schema could be used for both Oracle and MySQL. This would significantly simplify data distribution across the two storage technologies, as this could then be performed at the level of the persistent back-ends rather than at the level of the C++ application. In particular, any generic solutions developed in the context of the LCG 3D project [19-20] for the distribution of databases across the different computing tiers foreseen by the LHC experiments could be more easily applied to the case of their conditions databases too.

The use of RAL for a relational implementation of the Conditions Database software would also offer the benefit of minimizing the duplication of efforts between LCG projects. In this context, it should be noted that RAL is a central component of the relational implementation of POOL, which the Conditions Database "kick-off" Workshop held at CERN in December 2003 was instrumental in promoting to provide the relational functionality needed for conditions data. It may even be foreseen that a "POOL object relational handler" could be developed, analogous to the "POOL token handler" described in a previous section. Such a component should allow users to transparently access as POOL objects actual conditions data stored in rows of privately designed relational tables; in the context of the Conditions Database, the data payload of a conditions data object would in this case be a foreign key referencing a primary key of one such row in the external payload data table.

In addition to the SEAL, POOL and 3D projects, possible areas of cooperation with other projects are also under investigation. The collaboration with the Atlas Detector Description project, for instance, has already resulted in the proposal for the possible development of a shared component for "hierarchical versioning" of geometry or conditions data [21]. Another example is the definition of the most appropriate model for the storage of detector controls data from PVSS, which is likely to require input from the JCOP [22] project about the evaluation of the new PVSS Oracle archive.

REFERENCES

[1] The LCG Conditions Database Project, http://lcgapp.cern.ch/project/CondDB/

[2] POOL - The LCG Persistency Framework, http://pool.cern.ch/

[3] LCG - The LHC Computing Grid, http://lcg.web.cern.ch/

[4] LCG Applications Area, http://lcgapp.cern.ch/project/

[5] LHC – The Large Hadron Collider, http://www.cern.ch/lhc/

[6] P. Mato, "Conditions Database for LHCb - Interface Specification Proposal" (February 2000), http://lhcb-comp.web.cern.ch/lhcb-comp/Frameworks/DetCond/conddataspecs.pdf

[7] S. Paoli, Conditions DB Objectivity implementation, http://wwwdb.web.cern.ch/wwwdb/objectivity/docs/conditionsdb/

[8] E. Pilecki, "Conditions DB Oracle implementation – Overview and Current Status" (August 2002), https://edms.cern.ch/file/354076/1/RTAG_meeting_27082002.ppt

[9] A. Amorim et al., "Addressing the persistency patterns of the time evolving HEP data in the ATLAS/LCG MySQL Conditions Databases", these proceedings

[10] I. Gaponenko et al., "CDB - The distributed Conditions Database of the BaBar experiment", these proceedings

[11] A. Valassi et al., "HARP data and software migration from Objectivity to Oracle", these proceedings

[12] D. Klose et al., "Conditions Databases: the interfaces between the different ATLAS systems", these proceedings

[13] D. Klose et al., "The Web interface for the ATLAS/LCG MySQL Conditions Databases and performance constraints in the visualization of extensive scientific/technical data", these proceedings

[14] P. Mato, "Proposal to bring the Conditions Database into the Applications Area scope" (May 2003), http://lcgapp.cern.ch/project/mgmt/CONDB_Proposal.doc

[15] SPI - The LCG Software Process and Infrastructure, http://lcgapp.cern.ch/project/spi/

[16] SEAL - The LCG Core Libraries and Services, http://seal.web.cern.ch/seal/

[17] S. Eccher, "CondDBOracle and CondDBMySQL: converting data using perl scripting" (September 2004), http://lcg3d.cern.ch/evaluations/imex_docs.ps

[18] D. Düllmann, "POOL development status and plans", these proceedings

[19] 3D – Distributed Deployment of Databases for LCG, http://lcg3d.cern.ch/

[20] D. Düllmann, "On Distributed Database Deployment for the LHC experiments", these proceedings

[21] Atlas Geometry Database project, http://atlas.web.cern.ch/Atlas/GROUPS/DATABASE/project/geom/

[22] JCOP - Joint Controls Project, http://itco.web.cern.ch/itco/Projects-Services/JCOP/