COOL DEVELOPMENT AND DEPLOYMENT: STATUS AND PLANS

A.Valassi, CERN IT-PSS, Geneva, Switzerland
S. A. Schmidt, Institut für Physik, University of Mainz, Germany
M. Clemencic, CERN PH-LBC, Geneva, Switzerland
D. Front, Weizmann Institute, Israel and CERN IT-PSS, Geneva, Switzerland
U. Moosbrugger, Institut für Physik, University of Mainz, Germany

Abstract

Since October 2004, the LCG Conditions Database Project has focused on the development of COOL, a new software product for the handling of the conditions data of the LHC experiments. The COOL software merges and extends the functionalities of the two previous software packages developed in the context of the LCG common project, which were based on Oracle and MySQL. COOL is designed to minimise the duplication of effort whenever possible by developing a single implementation to support persistency for several relational technologies (Oracle, MySQL and SQLite), based on the LCG Common Relational Abstraction Layer (CORAL) and on the SEAL libraries. The same user code may be used to store data into any one of these backends, as COOL functionalities are encapsulated by a technology-neutral C++ API. After several production releases of the COOL software, the project is now moving into the deployment phase in Atlas and LHCb, the two experiments that are developing the software in collaboration with the CERN IT department. This paper reviews the status and plans for COOL development and deployment in April 2006, shortly after the CHEP 2006 conference.

INTRODUCTION

The LCG Conditions Database project [1] was launched in July 2003 with the goal of implementing a common persistency solution for the storage and management of the conditions data of the Large Hadron Collider (LHC) experiments at CERN, which are scheduled to start operation in 2007. The project, which is part of the Persistency Framework of the LHC Computing Grid (LCG) Applications Area, draws on a rich background of previous common activities in the area of conditions data for LHC and other experiments. This includes, in particular, the collaborative effort of some experiments and the IT Department at CERN to define a common C++ API for conditions data access and the successive implementations of this API using different storage technologies, first the Objectivity [2] object database, later the Oracle [3] and MySQL [4] relational databases. The experience of the BaBar experiment with Conditions Databases [5] significantly influenced the definition of the original data model and API.

From July 2003 until October 2004, as reviewed at the last CHEP conference in September 2004 [6], the project was active in the two areas corresponding to the tasks assigned to it in its mandate: first, in the integration of the existing Oracle and MySQL implementations into the LCG Applications Area (AA) scope; second, in the review of the two packages and their APIs, with the goal of discussing and planning their evolution into software products that may satisfy the common requirements of several LHC experiments.

In this initial phase, the progress of the project in the actual development of common software and tools was slowed down by two problems. First, the lack of committed manpower available for development did not allow much more than the integration of the two existing packages into the LCG Application Area, without any functionality enhancements. A larger development effort was carried out within ATLAS to maintain and extend the software and tools for the storage and management of the experiment test beam conditions data, but this activity mainly focused on ATLAS-specific needs rather than on requirements shared by several experiments. Secondly, the lack of a consistent approach between the APIs of the two existing implementations significantly limited the possibility for fast development of further software components and tools along a unified direction.

Birth of the COOL

The situation significantly improved at the end of 2004, thanks to the success of the project in its parallel task of reviewing the existing packages to propose a common program of work for their evolution, such that more than one experiment could benefit from it and would commit resources to implement it. Following the many discussions during this period, in particular that held at the LCG AA Meeting in October 2004 [7-8], the decision was taken to start the development of a new software product, merging and enhancing the functionalities of the two existing packages along a unified direction. The development of COOL (Conditions Objects for LCG) was thus started in November 2004, initially by a team of two persons, from CERN IT and Atlas. Following an aggressive development schedule, the first production version of the software COOL 1.0.0, providing the same user-level functionality as the two previous Oracle and MySQL implementations, was released in April 2005.

Over time, additional contributors from LHCb, Atlas and CERN IT have joined the core team, allowing the progressive functional enhancement of the software, in addition to its bug fixing, performance optimization and porting to new platforms and external package versions. A comprehensive test suite has been developed in parallel to the implementation code, allowing its consolidation over

successive release cycles. In total, seventeen versions of the software have been released so far by the core COOL team (averaging approximately 2.5 FTE active on development since the start of the project), the latest being the 1.3.0 release at the time of writing in April 2006. As requested by the experiments, development has consistently focused on relational database technologies, providing support initially for Oracle and MySQL in COOL 1.0.0 and soon after (COOL 1.2.1 in July 2005) for the SQLite [9] file-based SQL database engine.

In parallel to the core development activities, the integration and testing of COOL in the software frameworks of the two experiments participating in its development has progressed significantly, as reported in the relevant presentations by Atlas [10-11] and LHCb [12] at this conference. Additional tests of the software simulating its production deployment and use in a distributed environment have also been performed [13], in collaboration between the COOL team, the experiments and the LCG 3D project [14].

COOL DESIGN OVERVIEW

The basic data model of COOL is essentially the same as that used by the two previous LCG implementations. The set of values that are needed to represent a given conditions "data item" (such as the calibration of a given 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" (IOV) and may exist in more than one version. Three pieces of metadata are needed to lookup a conditions data object: a data item identifier, the time point for which a valid object is required, and a version number or a "tag" name. 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 a set of calibration parameters). In COOL, the data payload of a conditions data object is represented as an instance of the CORAL AttributeList class [15], i.e. as a list of attributes of simple data types (such as numbers or strings).

Taking into account the limited resources available for its development, as well as the problems caused in the past phase of the project mainly by the divergence of the two previous packages [6], the design of the COOL software has been driven from the start by the need to avoid all duplication of effort, both internally amongst the different provided functionalities and supported persistent technologies, and more widely in the context of the LCG.

Single relational implementation using CORAL

These two goals were first and foremost achieved by the choice to base the development of the relational implementation of COOL on the CORAL Common Relational Access Layer [15-16] (previously known as RAL, the POOL Relational Access Layer [6]). To start with, the use of CORAL has made it possible to develop a single implementation for all supported relational backends, where only a limited number of lines of code are needed to handle the special case of data stored using a specific technology. At the same time, delegating complex and largely backend-specific tasks, such as the handling of SQL statements and their integration with the relevant C++ client libraries, to the CORAL component has significantly reduced the load on the COOL team. The collaboration with the friendly CORAL team has been very easy and mutually beneficial, resulting in faster bug fixes and more focused enhancements of both packages.

Within COOL, the CORAL software acts as an abstraction and insulation layer that decouples the COOL implementation code from the choice of the underlying relational storage technology. Almost all COOL implementation code has been written without the a-priori knowledge whether it would be used for Oracle, MySQL or another backend. Indeed, while COOL 1.0.0 could only manage data stored using one of these two backends, the later addition of support for SQLite required only minor efforts, as only a few lines of COOL code had to be added to solve issues specific to the SQLite backend.

Single relational schema for all backends

This huge simplification of the development and maintenance effort, however, was only possible because of another independent design choice: that of using exactly the same relational schema for all technologies supported via CORAL. While this comes at the expense of not being able to use features that are not supported by all backends (such as views or partitioning), schema differences and other backend-specific optimizations can always be implemented at a later stage, if needed.

The choice of using the same schema for different relational backends was also motivated [6] by the need to ease the replication of conditions data in the LCG distributed computing environment, where the current deployment models foresee the use of Oracle at Tiers 0 and 1, and MySQL and SQLite at higher Tiers [14]. While data extraction and copy tools based on the C++ API have been included in COOL since release 1.2.6 (November 2005), application-independent cross-vendor replication of relational data stored using the same logical schema is also possible at the level of the persistent backends. Tools for such cross-vendor replication, for instance between Oracle at Tier 0 and MySQL at Tier 2, are being developed in the context of the LCG 3D project [14]. Tools for the relatively simpler case of same-vendor replication at the database level, for instance between Oracle at Tier 0 and Oracle at Tier 1 via the Oracle Streams technology [3], are also being developed and tested within the 3D project.

Implementation of abstract C++ interfaces

As it was the case for the two previous LCG conditions database packages, the COOL software is based on the implementation of abstract C++ interfaces that do not expose any backend-dependent features. From a user level perspective, this means that the same user code can be used with almost no change for the many supported technologies. At the same time, this allows the internal implementation to be changed without any impact on the

users. As the interfaces do not even assume that its implementation must be based on a relational database management system, a non-relational implementation of the same C++ API may also be envisaged if ever required.

Reuse of LCG software and infrastructure

The collaborations of COOL with the CORAL and 3D projects are just two examples of the more general COOL design choice to reuse as much as possible any software, tools and expertise already available in the LCG context, to avoid any duplication of effort. This is also the wider direction of the LCG Application Area as a whole [17].

At the software level, COOL heavily depends (either directly or through CORAL) on the SEAL core libraries and services, in areas such as the component model, messaging, runtime configuration and dynamic plugin loading, as well as for the handling of data types such as 64-bit precision integers or time classes. While the C++ implementation of COOL does not depend on reflection and dictionaries, the Reflex package [18] is needed by the PyCool component, which allows fast interactive access to the COOL functionalities from a Python shell through a "Python-ized" version of the C++ API. Since the COOL 1.2.8 release (January 2006), this has introduced a dependency of COOL on the ROOT framework [19] because of the recent merger of the SEAL and ROOT projects [17]. The dependency of COOL on ROOT is presently limited to the use of the ROOT Reflex component in PyCool: new dependencies may arise as the merger of SEAL and ROOT progresses.

Finally, COOL also owes to other LCG AA projects for much of its configuration and development infrastructure, mainly to the SPI project, but also to CORAL and POOL.

Clearly delimited software scope

More generally speaking, avoiding duplication of effort has been possible by clearly delimiting COOL as a software component with a well-defined scope: the management of the time variation and versioning of the conditions data of a generic LHC experiment, the non-event data describing the state of the detector at the time of data taking. As previously discussed, COOL delegates to other LCG projects many tasks which can be performed in a more generic way, such as the generic C++ access to relational data, handled by CORAL, or the generic deployment and distribution of relational data, handled within the 3D project. At the same time, COOL does not attempt to solve the problems specific to a given experiment, focusing instead on providing flexible software solutions that different groups of users in the various experiments may configure to solve their needs.

So far, the scope of the COOL software has been further restricted to focus primarily on the conditions data requirements for event reconstruction and analysis, where the main use case is the "direct" lookup of a set of conditions data objects in a given time range and for a given tagged version. In particular, COOL presently provides no special functionality to address the "inverse" lookup of the time ranges during which given values of data payload were observed, for instance, temperatures higher than the allowed maximum. This is a very different use case, relevant to detector experts who need to identify and solve any malfunctioning of the apparatus for which they are responsible. Specific solutions optimizing the performance for this use case via server-side database queries may be provided in a future COOL release, but this issue is still being discussed. For the moment, this use case can only be addressed in COOL through the same "client full scan" methodology that is generally used for event analysis, i.e. by retrieving all conditions data observed during a larger time span and looking for any payload measurements in the region of interest.

Consistent metadata model for several use cases

To provide the flexibility required by the users, the COOL API offers many hooks to customize the storage of conditions according to the data model most appropriate to each user. For instance, two basic modes of operations are foreseen: a "single-version" mode, optimized for online data such as temperatures, which vary in time but only exist in the single version that is the result of a direct measurement; and a "multi-version" mode, allowing the versioning and tagging of offline data such as calibration parameters, which can be recomputed according to several different algorithms. At the same time, thanks to the use of the AttributeList, some users may describe the data payload of their conditions data objects encoded as a string in a long character object (CLOB), while other users may represent it as an actual list of floats or integers. Finally, COOL allows users to store different conditions data items using independent conditions data "folders" (i.e. different relational tables, for instance because they require different payload schemas), but also as different "channels" in the same folder (i.e. within the same relational table, but identified by different values of a channel number column).

In the LCG conditions database project, these three issues were all first addressed by the "extended API" of the MySQL implementation [6]. COOL also implements solutions to provide the flexibility required in all these areas: differently from the previous MySQL package, however, COOL treats these issues as minor variations of the same basic metadata model for conditions data. In practice, whenever possible it is strictly the same COOL implementation code that is executed to handle conditions data of different types, whether associated to CLOB or multi-column payloads, relative to single-channel or multi-channel folders, single or multi version. The use of this single consistent approach has been one of the key reasons why these functionalities could be provided by COOL as early as in its first release 1.0.0.

Emphasis on performance

Keeping in mind the large data volumes and especially large data rates expected for the LHC experiment conditions data, as well as the inadequate performance of the previous Oracle conditions database implementation, data insertion and retrieval performance has been taken into account in the design of the COOL API and implementation right from the start. In particular, COOL

provides mechanisms to store and retrieve several conditions data objects at the same time, through the use of bulk relational updates with bind variables, and of row pre-fetching in relational queries. Particular attention is also paid to the design of the relational schema, to ensure that the appropriate execution plan is used for queries and updates thanks to the presence of all relevant indexes.

While many significant optimizations are still needed, both in the internal C++ management of data buffers on the client side and in the client-server and server-side handling of SQL queries, the performance measured so far is satisfactory. A simplified version of the important Atlas "first-pass event reconstruction" use case, in particular, has been successfully validated [13]: the expected sustained data rates of 20 MB/s, representing 20k conditions data objects per second, have been met for retrieval from an Oracle RAC cluster database.

STATUS AND PERSPECTIVES

One year and a half after the start of its development, as the LHC start-up gets closer, the focus of the COOL project is rapidly moving from functional enhancements to deployment issues in Atlas and LHCb. Its integration within the software frameworks of the two experiments is already well advanced, as reported in other presentations at this conference [10-12]. The relevant database services are being deployed and tested in the context of the 3D project, in collaboration with the CERN IT-PSS team at CERN and the local database experts at the other Tiers. In Atlas, the phasing out of the old MySQL implementation will be completed during 2006, with the migration to COOL of the conditions data from the 2005 test beams.

As these activities advance, the COOL development team is progressively concentrating on deployment-related issues. The integration into COOL of the CORAL advanced connection management features [16,20], such as database replica lookup and connection retrial, has been completed in the latest COOL release 1.3.0 (April 2006), while the future integration of the CORAL client-side database monitoring is also foreseen.

Several performance optimizations are also still needed, as previously observed. As the software provides the flexibility to address rather different use cases, such as the single-version and multi-version data models, separate performance tests are required for the many supported modes of operation, sometimes leading to separate optimizations of the relational queries and updates. In particular, optimizations for the bulk insertion of conditions data objects into separate channels of the same folder will be added in one of the next COOL releases.

Data replication is also one of the highest priorities for the near future. Support for the FroNtier [21] multi-tier data access mechanism will soon be prototyped, allowing the retrieval of conditions data stored in a remote Oracle server as http pages which can be cached in middle tier Squid proxy caches. As previously observed for SQLite, this task will be simplified by the fact that FroNtier is already supported by CORAL. Tools for the "dynamic" replication of COOL databases at the C++ level have also been requested, to copy to the target database only data inserted into the source database after the last replication.

Finally, even if the emphasis has shifted towards deployment issues, the development of new functional enhancements in COOL is still far from finished. In the latest release 1.3.0, for instance, support has been added for the "hierarchical versioning" (HVS) of conditions data trees, along the same design that was first proposed in collaboration between the LCG conditions database and the Atlas detector description projects, and which has long been implemented in production by the latter [21].

REFERENCES

[1] COOL - The LCG Conditions Database Project, http://lcgapp.cern.ch/project/CondDB/
[2] Objectivity Database, http://www.objectivity.com
[3] Oracle Database, http://www.oracle.com
[4] MySQL Database, http://www.mysql.com/
[5] I. Gaponenko et al., "Using Multiple Persistent Technologies in the Conditions Database of BABAR", these proceedings
[6] A. Valassi et al., "LCG Conditions Database Project Overview", proceedings of CHEP04, Interlaken (September 2004) and references therein
[7] A. Valassi, "Conditions Database Project Status and Direction", LCG AA meeting (October 2004)
[8] A. Amorim, "Time and Storage Patterns in Conditions: Old Extensions and New Proposals", LCG AA meeting (October 2004)
[9] SQLite Database, http://www.sqlite.org/
[10] A. Vaniachine et al., "Database Access Patterns in the ATLAS Computing Model", these proceedings
[11] M. Verducci et al., "Conditions Database and Calibration Software Framework for ATLAS Monitored Drift Tube Chambers", these proceedings
[12] M. Clemencic et al., "The LHCb Conditions Database Framework", these proceedings
[13] A. Valassi et al., "COOL Performance and Distribution Tests", these proceedings
[14] D. Duellmann et al., "LCG 3D Project Status and Production Plans", these proceedings
[15] CORAL – Common Relational Abstraction Layer, http://pool.cern.ch/coral/
[16] I. Papadopoulos et al., "CORAL, a Software System for Vendor-neutral Access to Relational Databases", these proceedings
[17] P. Mato, "Common Application Software for the LHC Experiments", these proceedings
[18] S. Roiser et al., "Reflex, Reflection for C++", these proceedings
[19] ROOT – An Object-Oriented Data Analysis Framework, http://root.cern.ch/
[20] A. Vaniachine et al., "ATLAS Distributed Database Services Client Library", these proceedings
[21] L. Lueking et al, "FroNtier: High Performance Database Access Using Standard Web Components in a Scalable Multi-tier Architecture", proceedings of CHEP04, Interlaken (September 2004)
[22] V. Tsulaia et al., "Software Solutions for a Variable ATLAS Detector Description", these proceedings