A JEE RESTful service to access Conditions Data in ATLAS

This content has been downloaded from IOPscience. Please scroll down to see the full text.
(http://iopscience.iop.org/1742-6596/664/4/042016)

View the table of contents for this issue, or go to the journal homepage for more

Download details:

IP Address: 137.138.93.202
This content was downloaded on 09/03/2016 at 09:03

Please note that terms and conditions apply.
A JEE RESTful service to access Conditions Data in ATLAS

Andrea Formica\textsuperscript{1}, E J Gallas\textsuperscript{2}

on behalf of the ATLAS collaboration

\textsuperscript{1} CEA/Saclay IRFU/SEDI, 91191 Gif-sur-Yvette, France
\textsuperscript{2} Department of Physics, Oxford University, Denys Wilkinson Building, Keble Road, Oxford OX1 3RH, United Kingdom

E-mail: andrea.formica@cea.fr

Abstract. Usage of condition data in ATLAS is extensive for offline reconstruction and analysis (e.g. alignment, calibration, data quality). The system is based on the LCG Conditions Database infrastructure, with read and write access via an ad hoc C++ API (COOL), a system which was developed before Run 1 data taking began. The infrastructure dictates that the data is organized into separate schemas (assigned to subsystems/groups storing distinct and independent sets of conditions), making it difficult to access information from several schemas at the same time. We have thus created PL/SQL functions containing queries to provide content extraction at multi-schema level. The PL/SQL API has been exposed to external clients by means of a Java application providing DB access via REST services, deployed inside an application server (JBoss WildFly). The services allow navigation over multiple schemas via simple URLs. The data can be retrieved either in XML or JSON formats, via simple clients (like curl or Web browsers).

1. Conditions Database overview

The ATLAS experiment\textsuperscript{[1]} at the Large Hadron Collider (LHC) at CERN makes use of the LCG conditions database infrastructure, where the data are stored in an Oracle backend maintained by CERN IT and managed via COOL API\textsuperscript{[2]}.

ATLAS has recorded more than \(\sim 1.5\) TB of condition data in the first period of data taking (Run 1) of the LHC. Condition data from every system are stored in separate schemas inside the Oracle cluster (both for online and offline access). In each schema the content is organized in different tables with the following common metadata structure:

- NODES (folders): a table containing one or more related conditions data quantities (e.g.: alignment or calibration data) for a given sub system.
- TAGs : a table containing strings (versions of conditions) which map to coherent set of conditions during a unique set of Intervals of Validity (IOVs).
- IOVs (and CHANNELs): a given conditions payload data set is associated to an Interval of Validity (spanning a period in time or a range of run numbers). Every IOV is associated to one CHANNEL, allowing to split the NODE information into sub-groups of data entities. One database CHANNEL corresponds to a single or a group of hardware channels.
In any system, the tag name identifies a set of IOVs which are then retrieved depending on the events being processed, in order to access the appropriate set of condition data values (calibrations, alignments and so on) valid for the corresponding interval in time.

The condition data are accessed during the physics data processing step from ~10k reconstruction jobs simultaneously and are essential for physics data analysis. In order to access a coherent set of conditions, specific tags from different subsystems are linked to a single global tag. When a system improves its conditions in preparation for a reprocessing campaign, it can associate its new tag to the new global tag which is prepared for the upcoming processing. To browse metadata inside the COOL infrastructure, we need to connect to every schema in order to extract the relevant content. This is the case if we want, for example, to know which system tags are associated to a given global tag: we need to check in every schema, searching for the given global tag name and extracting the list of dependent tags in each folder, then we pass to the next schema and so on (see figure 1). These kind of operations are in general performed by global tag coordinators every time they need to prepare and verify the content of a global tag. More details about the global tags usage in ATLAS can be found in [3].

![Figure 1. COOL access to multiple schemas using standard C++ API.](image-url)

2. Gathering metadata from the Conditions Database
Using the COOL API to gather information on metadata related to a given global tag is cumbersome because this information is stored in separate tables in different schemas. It is necessary to loop in the client program over all the schemas containing COOL data. To overcome this limitation we have devised two separate and complementary approaches:

1) **Collect metadata in a dedicated relational database:** We set up a process for copying all the important metadata from the LCG conditions database schemas into a unique schema, inside a metadata database called COMA [4] which hosts other relevant information on data taking (trigger information, run numbers, luminosity over periods etc...). Using a dedicated set of tables inside COMA, we then have access to all content of COOL information about global tags, tags, nodes and payload metadata. These tables (see figure 2) are synchronised every 3 hours with the COOL schemas. Since these metadata are well organised in a relational table structure, we have really fast access to all information needed for global tag coordination.
during the preparation of reprocessing or simulation campaigns. Several diagnostic tools have also been used in order to flag unusual conditions database content, allowing a much better follow up of the system.

However, the main limitation of this tool is that we cannot reproduce the full intervals of validity, channels and payload content, because this would require the collection of a large volume of metadata which would be challenging to keep in sync with all evolving conditions in the conditions database. The number of IOVs from each system (in a single tag) can vary a lot, from a few intervals to 100K intervals for some systems, and the same is true for the channels content. Duplicating this amount information from the original database would result in a complete re-design of the Conditions Database itself.

However, in some cases, this information is extremely useful for the administration of global tags: for example, if we want to check the IOVs stored for a given tag in a given system node in order to verify that the conditions data coverage is correct for the type of processing that we want to perform. Performing these kind of tasks would require a large expansion in the scope of the COMA project.

![Figure 2. COMA tables, zoomed on Conditions metadata.](image)

2) Access all schemas via PL/SQL API: Using the Procedural Language extension to SQL, we have developed a set of queries inside a dedicated Oracle schema having read-only access to all COOL schemas. PL/SQL program units are stored and compiled in the database, run within the Oracle executable and inherit the robustness, security, and portability of the Oracle Database. Our package generates dynamically the appropriate SQL query to select information from tables in every schema, and present it as a unique table in the output. The dynamic SQL generation is done via a loop over all schemas concerned (selected by the client, as if it was a parametrised SQL VIEW). We have thus direct access to the whole set of COOL tables and their content, bypassing the COOL API itself and without any need for synchronisation; in addition, it is a flexible way to extend the COOL API by developing queries to gather statistics on the COOL tables, allowing holes in the tag coverage to be spotted and similar IOV related information to be analyzed. We will show in more details this kind of extension in the next paragraph.

Inside the same schema, another package contains PL/SQL functions to extract the same information from COMA tables, in order to make a comparison of content between COOL and COMA. This is useful to cross check the content of the two systems.
3. Extending COOL API queries via PL/SQL
The development of new queries inside the COOL API is a difficult and long process because of the dependencies that software releases have with respect to COOL libraries. Once new functionalities are developed, before being able to deploy the new API, we need to go through a long phase of testing. This process is not worthwhile for queries which are in any case not used by reconstruction programs. The development of this PL/SQL API allows us to perform several monitoring queries which would be difficult to develop directly in C++ (and sometimes slower to execute using direct Oracle access, because it would imply looping over schemas at client level): a simple python client using COOL API to resolve a global tag mapping tree shows a factor 10 drop in the execution time when using the PL/SQL API. These queries are relevant in the use cases of global tag coordination tasks, when checking the IOVs coverage for every tag is mandatory, and they help to easily spot missing IOVs in relevant periods for analysis and reconstruction.

4. Expose PL/SQL API via JEE RESTful services
Using the PL/SQL API from a client program requires knowledge of Oracle DB connection APIs. In addition, the maintenance of such a system (providing directly the PL/SQL code to the clients) would be harder, since it would be difficult to change the API itself without impacting the clients. For these reasons we chose to expose the PL/SQL API via REST (Representational State Transfer) services. A server based application in Java has been developed for this purpose. Data Access Objects (DAOs) implement the query PL/SQL query functions and expose them...
as RESTful services by mapping all functions to URLs. The output can be in JSON (JavaScript Object Notation) or XML format depending on the HTTP header of the request.

The application (see figure 3) is deployed inside a JBoss WildFly [5] application server, running in a dedicated ATLAS virtual machine at CERN. The server also hosts scheduled tasks, which can extract and store in dedicated tables daily information from COOL tables related to IOVs in order to speed up the monitoring capabilities of the system.

Another important advantage of using a middle-tier server for the DB access is related to security issues: the clients will not need to take care of hiding passwords, since the connection itself is managed at server level in a well protected machine.

4.1. Web and command line clients
The services delivered by the JEE (Java Platform, Enterprise Edition) application are accessible via any browser, and available via a python client based on PycURL. The client can be easily installed on any machine and gives the possibility to retrieve information from the JEE server. A web application GUI, which uses the AngularJS [6] framework is under development.

4.2. Usage of the server during Run 1 and Run 2
The set of services has been used by other systems in need of COOL data retrieval. The first system consists of a tool used essentially for tags management, which is described in [7]. A second system is used inside the TDAQ environment, to follow the activities related to start and end of run directly via a Web GUI and to monitor the detector TDAQ configurations during data taking. This monitoring application is installed inside Point 1 (ATLAS online environment), and is not accessible to clients outside the private network.

5. Summary
We have presented a set of developments to improve conditions metadata access using the information stored in the conditions database infrastructure in the ATLAS experiment. Firstly, the extraction of the metadata into a dedicated set of relational tables allows fast access to most of the information needed for global tag administration. Secondly, the development of a PL/SQL package to access COOL schema tables allows fast development of monitoring queries on the existing data without the need to add additional functionalities to the COOL API itself. All of these developments have been exposed for usage by clients via a set of RESTful services implemented in a JEE Java based application, deployed inside a JBoss WildFly application server. The services described have enabled a new level of monitoring tools to be developed with a view of the whole system which is extremely useful to global tag administrators. These tools help to easily spot problems with tag dependencies and IOV coverage. They have been invaluable in the preparation of conditions for Run 1 reprocessing campaigns and are now used in the preparation of every global conditions tag in ATLAS.

References