EPJ Web of Conferences **214**, 04032 (2019) https://doi.org/10.1051/epjconf/201921404032
*CHEP 2018*

# A new mechanism to use the Conditions Database REST API to serve the ATLAS detector description

*Andrea* Formica[1,*], *Riccardo Maria* Bianchi[2,**], and *Alessandro* De Salvo[3], on behalf of the ATLAS Collaboration

[1]Université Paris-Saclay, IRFU/CEA,*FR*
[2]University of Pittsburgh, *US*
[3]Sapienza Università and INFN, Roma I, *IT*

**Abstract.** Efficient and fast access to the detector description of the ATLAS experiment is needed for many tasks, at different steps of the data chain: from detector development to reconstruction, from simulation to data visualization. Until now, the detector description was only accessible through dedicated services integrated into the experiment's software framework, or by the use of external applications. In this work, we explore the possibility of using a web access-based conditions database to store and serve the detector description, aiming at a simplification of the software architecture of the experiment and a reduction in the number of software packages to be maintained.

## 1 Introduction

Currently, the detector description of the ATLAS experiment [1] is built by fetching geometry data from an Oracle database through the experiment's framework, Athena [2]. On average, about 300 SQL queries are needed to build the complete geometry. Moreover, the geometry is built on-the-fly and stored in-memory only, thus the process needs to be repeated every time the detector description has to be accessed by a data-processing job.

Recently, new developments decoupled the geometry core software from the experiment's framework and created a mechanism to get a persistent copy of the detector description, together with adding new serialization formats [3].

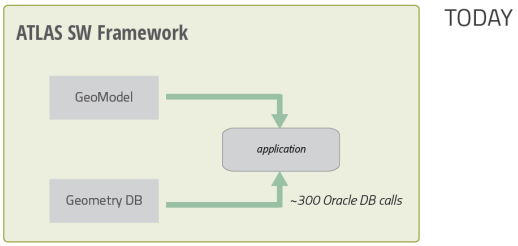
Leveraging these new developments, and with a view to further decoupling from the framework, a new mechanism is presented in this proceedings, which uses the new conditions database CREST [4, 5] to serve the detector description through a HTTP REST API.

The goals of the new prototype are the following: to provide a lighter software stack in the Athena framework to access geometry via a simple REST service and to reduce the amount of network operations to serve the geometry to clients.

## 2 Current access to the ATLAS Detector Description

The ATLAS experiment uses the GeoModel [6] library to describe the geometry tree: a set of nodes (shapes, containers, materials, ...) connected through relationships of different types.

*andrea.formica@cern.ch
**riccardo.maria.bianchi@cern.ch




(a)


(b)

Figure 1: a) the architecture currently implemented in ATLAS: the detector description is built on-the-fly, taking the definition of the GeoModel tree from C++ code and the geometry parameters from a dedicated Oracle database though a dedicated service; all operations are performed within the experiment's framework. b) The current actors and actions involved in the retrieval of geometry parameters from the Oracle DB. Today, about 300 SQL queries are needed to retrieve the full set of geometry parameters, performed by each of the jobs processing the data.

Parameters are used to further customize the base shapes/structure used in the GeoModel tree; they are stored in the Oracle-based GeometryDB [7]. Different sets of parameters can be saved in the GeometryDB, to store different configurations of the detector geometry.

When requested by a data-processing job (see Figure 1a), the GeoModel tree—defined in C++ code for each of the ATLAS subsystems—is traversed; then, the tree nodes—*i.e.*, the detector elements—are customized with the parameters extracted from the GeometryDB, and the geometry is built on-the-fly. All these operations are performed through the experiment software framework, Athena. The resulting detector description is then stored in-memory and used by the requesting job.

The GeometryDB database consists of a set of tables dedicated to parameters for individual detector elements and each set of parameters is identified via a dedicated tag. A set of valid tags at a given moment is identified via a unique parent tag, called *geometry tag*. The schema implementation is then based on a hierarchical versioning: a given *geometry tag* identifies a frozen version of the geometry parameters. Geometry parameters are then accessed via SQL queries , starting from the unique entry point provided by the *geometry tag*.

### 2.1 Access to geometry parameters from Athena

Today, the geometry parameters from the Oracle database are accessed through a set of SQL queries: as said, about 300 queries are needed to retrieve the full set of parameters for a given geometry tag. In addition, due to the fact that the detector geometry is built on-the-fly, those queries need to be performed by each job which needs the access to the detector's description.

A dedicated service acts as interface between the GeoModel and the Oracle DB (see Figure 1b). All queries are performed via Coral [8] (a C++ access layer to the database) and executed via Frontier/SQUID [9], in a context of distributed computing.

In HEP experiments, there is a distinction between *raw* and *readout* geometry; the GeoModel library describes the geometry of the single elements which compose the detector, without describing how those elements are assembled and their signal read out; that part, in fact, is defined and customized at a later stage by each sub-system of the experiment. In this paper, we only address the *raw* geometry, that is the collection of geometrical data which describe the bare detector elements.

## 3 A standalone geometry description

Being described in C++ code and built on-the-fly, the geometry was not queryable so far: the GeometryDB, in fact, stores properties and parameters; but all the relationships between nodes, which define the structure of the detector geometry, are stored inside the C++ description only. Moreover, the tight integration of the geometry mechanism with the ATLAS software framework prevented, so far, lightweight access to the experiment's geometry.

The recent decoupling of the GeoModel library from the framework, and the possibility of dumping the full experiment's geometry to file, made possible the exploration of alternative methods to access, read, explore, debug, and visualize the experiment's geometry, opening up many possibilities to further development. Two new exporters have been implemented: SQLite and JSON. Further information can be found in the above mentioned Ref. [3].

By using the new exporters (see Figure 2), we can dump the full detector description from memory into new data structures for any given *geometry tag*: the resulting file contains the customized version of the geometry defined by the properties defined by the tag. Of course, saving a full geometry description for each tag leads to redundant information; but it gives us the possibility of having a persistent copy of a geometry version, to be saved, shared, served, explored, and debugged without accessing the experiment framework. This is specially useful for data-processing jobs which need to perform read operations only.

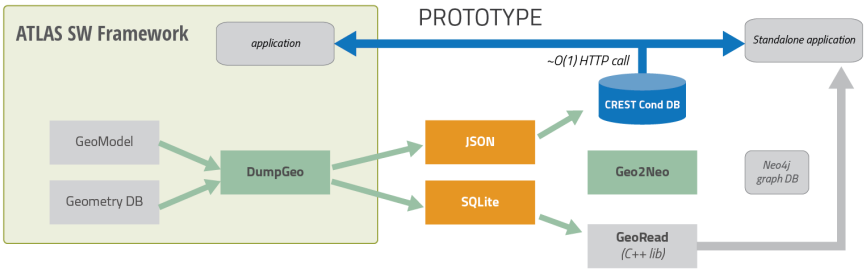



Figure 2: The new architecture proposed in this proceedings: once the detector description built, a persistent copy of it is saved to files, from which it can be read, served, and queried outside of the experiment's framework.

## 4 New ways of building and serving the detector geometry

The new standalone geometry representation lets applications easily read and store a persistent copy of the detector description, without the need of the experiment's framework. The

new mechanism presented here leverages this new feature, by using the HTTP REST API of the CREST [10] conditions database to efficiently serve the full detector geometry to clients which need read-only access to it, without the need of extra layers.

### 4.1 REST access to the geometry

The access through a HTTP REST API let applications running outside the experiment's framework access the information about the detector geometry (see Figure 2). However, it brings other benefits: it greatly simplifies the access to the detector description itself, for read-only operations, for both standalone and integrated applications. This has two main advantages: a reduced number of database queries, which can be reduced from $O(\sim 300)$ to $O(\sim 1)$; and the possibility of accessing the geometry through a REST API, removing the need to include specific SQL knowledge in the client code.

The new REST API helps to simplify the architecture of code running inside the experiment's framework, as well. In fact, while the Oracle-based GeometryDB has a table structure well suited for managing the storage and the versioning of all the different pieces of information about the detector's geometry, we do not need the same kind of complexity in read-only operations performed in a typical data-processing job running inside the Athena framework. Hence, the use of a REST API improves the development and the maintenance of applications, both Athena-based and standalone.

### 4.2 Storing geometry data in the Conditions database

For this prototype we have been using the proposed CREST database infrastructure that ATLAS is evaluating as a prototype for the future data-taking periods. However, similar conclusions do apply to the existing COOL [9] data model as well, and can thus be used in the upgrade of the current system.

The new standalone geometry mechanism, as said, is currently able to produce a persistent copy of the detector description in two formats: SQLite and JSON. The JSON version has been chosen for this prototype and the file is represented in the Conditions DB as a *blob* type, associated to a single, so-called *Interval of Validity* (IOV, that defines the time over which the data is declared "valid", and which spans from 0 to Infinity) and to a single *tag*, which could correspond to the previous mentioned *geometry tag*.

For these tests, the prototype of the CREST database deployed on Openshift [11] at CERN has been used, and Python client libraries have been used to interact with the CREST server via REST [12].

## 5 An overview of the new architecture

The new CREST REST API has been built using the Swagger OpenApi [13]—an interface description language (IDL), which lets users efficiently specify the specifications of a REST service. The actual code of the REST interface is then automatically generated based on the input specifications, either at server or client level, in several languages and frameworks (See Figure 3).

For this prototype [14], a C++ client library has been generated, based on the Qt framework [15], which can connect to the CREST server via HTTP using the REST API, without the need of additional components.

The new architecture using the Conditions database to serve the full details of a given *geometry tag* stored in a JSON file offers a number of advantages (See Figure 4).

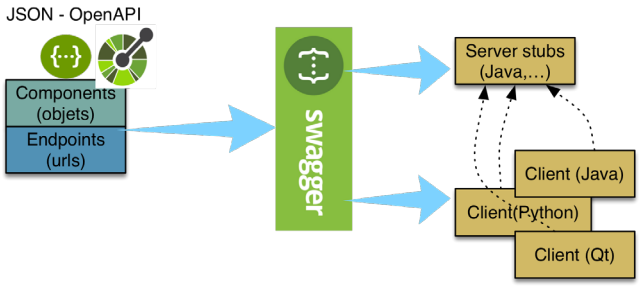



Figure 3: For the prototype, the Swagger [13] OpenAPI has been used to generate the server and client code.

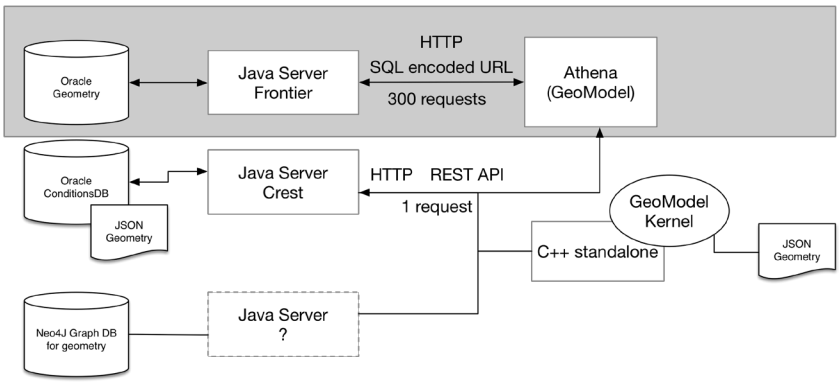



Figure 4: An overview of the new architecture integrating with the current one. In grey, the current system to retrieve the geometry data from the Oracle-based GeometryDB, through the experiment's framework. In the middle, the new architecture, which uses an HTTP REST API to retrieve geometry data from the CREST conditions database, where are stored as JSON *blob*. The same REST API could be used by standalone applications as well (on the right, in the drawing). At the bottom, a future extension of the architecture is foreseen, querying and filtering geometry data from a Neo4j, graph-database instance.

The information about the detector description is provided to clients with a single call to the REST API. Only a single request is necessary, as opposed to the $O(300)$ SQL queries required by the current system.

The new architecture also helps to simplify the structure of the ATLAS Athena services. For simple read-only operations, in fact, a single database can be used, the Conditions database, to retrieve both conditions and geometry data. This simplifies the software components maintenance, as well.

The new architecture also reduces the dependency on SQL libraries. By using a REST service to read the geometry data, instead of performing a set of SQL queries, the typical job, *de facto*, should not even be aware that is accessing a database. For a given *geometry tag* `ID`, in fact, clients can access the full geometry information with a single HTTP call—*e.g.*, `http://crest-undertow.web.cern.ch/crestapi/payloads/<ID>`.

The SQL layer will still exist, but would be used only for the management of the geometry parameters in the Oracle-based database (or other relational databases), which is a completely

different use-case from loading a geometry in memory, for example, for reconstruction or data visualization jobs.

## 6 Conclusions and plans

The new architecture explores the use of the Conditions database to efficiently store and serve the detector description. This, as seen, offers a number of advantages.

The first prototype of the new architecture has been developed and the first tests were successful. Now, in order to further develop and optimize the new system, a set of additional steps need to be performed. A set of connection tests will be run to measure the performance of the new access system compared to the one currently used in ATLAS. After that, based on the achieved results, the new system will be tested within the ATLAS framework, as a potential replacement of the existing mechanism, to load and build geometry data in jobs which only needs to perform read-only operations on the detector description.

After that, the REST approach will be further explored for other future geometry backends. One foreseen extension is the addition of a Neo4j [16, 17] backend, in order to offer advanced querying and filtering functionalities on detector description data.

## References

[1] ATLAS Collaboration, *The ATLAS Experiment at the CERN Large Hadron Collider*, JINST 3 (2008) S08003.

[2] Calafiura P *et al.*, 2004, *The ATHENA control framework in production, new developments and lessons learned*, *Interlaken 2004, Computing in high energy physics and nuclear physics*, CHEP 2004 Book of Abstracts, pp. 456–458

[3] S.A. Merkt *et al.*, *"Going standalone and platform-independent, an example from recent work on the ATLAS Detector Description and interactive data visualization"*, CHEP 2018, proceedings: https://cds.cern.ch/record/2649301 [accessed 2018-12-03], presentation: https://indico.cern.ch/event/587955/contributions/2937522/ [accessed 2018-12-03]

[4] *CREST Documentation*, https://twiki.cern.ch/twiki/pub/AtlasComputing/ConditionsRest/Crest_Project_Description_20171129.pdf [accessed 2018-12-03]

[5] Roland Sipos *et al.*, *"Functional tests of a prototype for the CMS-ATLAS common non-event data handling framework"*, 2017, J. Phys.: Conf. Ser. 898 042047, https://doi.org/10.1088/1742-6596/898/4/042047 [accessed 2018-12-03]

[6] Boudreau J, Tsulaia V, 2004, *"The GeoModel Toolkit for Detector Description"*, *Interlaken 2004, Computing in high energy physics and nuclear physics*, CHEP 2004 Book of Abstracts, pp. 353–356

[7] Boudreau J, Tsulaia V, Hawkings R, Valassi A, Schaffer A, *"Software Solutions for Variable ATLAS Detector Description"*, CHEP 2006 (2006)

[8] Trentadue R *et al*, *"LCG persistency framework (CORAL,COOL, POOL): status and outlook in 2012"*, J. Phys.: Conf. Ser. 396 (2012) 052067

[9] Valassi A *et al*, 2004, *"LCG conditions database project overview"*, *Interlaken 2004, Computing in high energy physics and nuclear physics*, CHEP 2004 Book of Abstracts, pp. 510–513

[10] *CREST prototype*, https://openshift.cern.ch/console/project/crest-undertow [accessed 2018-12-03]

[11] *"Openshift at CERN"*, https://cern.service-now.com/service-portal/article.do?n=KB0004358 [accessed 2018-12-03]

[12] L.Rinaldi *et al.*, *"Conditions evolution of an experiment in mid-life, without the crisis (in ATLAS)"*, CHEP 2018, https://indico.cern.ch/event/587955/contributions/2936820 [accessed 2018-12-03]

[13] *"Swagger and Swagger-codegen"* [Computer software], https://swagger.io [accessed 2018-12-03]

[14] *"CREST server and client code"*, https://gitlab.cern.ch/formica/swagger_crestdb [accessed 2018-12-03]

[15] The Qt Company, *Qt 5* [Computer software], https://www.qt.io/ [accessed 2018-12-03]

[16] R.M. Bianchi and I. Vukotic, *A scalable new mechanism to store and serve the ATLAS detector description through a REST web API* , ACAT 2017 Proceedings, (*submitted*), preprint: https://cds.cern.ch/record/2290833?ln=en [accessed 2018-12-03], presentation: https://indico.cern.ch/event/567550/contributions/2628864/ [accessed 2018-12-03]

[17] *Neo4j* [Computer software], https://github.com/neo4j/neo4j [accessed 2018-12-03]