The ATLAS Experience With Databases for Structured Data With Temporal and Hierarchical Order

Nuno Barros, Antonio Amorim, Richard Hawkings, Tiago Franco, Dinis Klose, and Luis Pedro

***Abstract*—Developing systems for the ATLAS (A Toroidal LHC ApparatuS) detector at LHC poses a new set of challenges like the storage of enormous amounts of data with different structures. This data is not only traditional Conditions Data which is slowly evolving data but also data from the online systems which cannot be stored with the event data but must be accessible from each event, like run information. Following the requirements from such systems we have developed a system which provides the facility to store structural data in a time order and with the possibility to classify hierarchically. We will also present a case study of use in the online systems of the ATLAS experiment where this system was used to store online data. Furthermore, we will show the different interfaces developed for this system and its use in different areas of application.**

***Index Terms*—A Toroidal LHC ApparatuS (ATLAS), conditions data, databases, Interval Of Validity (IOV).**

I. Introduction

RUNNING a detector like A Toroidal LHC ApparatuS (ATLAS) [1] poses many challenges in the requirements for storing different kinds of data. The information on the detector calibration and alignment, commonly named *Conditions*, must be classified in an hierarchical way and by having a time interval of validity (IOV). By looking at the applications that must store data in the database it became clear that one needed to extend the previous implementation [2] to create an extra layer where it would be possible to map data that could have temporal, hierarchical, and multi-version data. At the same time, this layer should also allow easy retrieval of structured data mapping the relational table into a transient C++ object. Furthermore, based on the experience acquired with the previous version, it was understood that the usage of serialization mechanisms in the user applications raised many issues that did not allow the sharing of data between applications that did not use the same serialization approach. Another disadvantage of this method was the inability of the central database management system (DBMS) to provide each user with enough information to process the requested data as from the DBMS point of view each object was nothing more than a binary large object (BLOB) without any kind of data structure. Following a case study in the ATLAS online environment, an approach was developed that could extend the relational model tables while keeping the object schema available from the central databases. To address these problems a solution based on a generic transient object was introduced, which could be interpreted and mapped into a relational table, keeping knowledge of its structure, contents and IOV. Being very generic, these new features made the interface more exportable to other areas of application where structured temporal data storage is needed. After a successful presentation to the users, a couple of interfaces were implemented to ease the usage of this technology and better integrate it into the overall infrastructure. The area of application became larger, allowing the basic interface to be used for different purposes on interacting with different technologies.

Manuscript received July 20, 2005; revised May 23, 2006. This work was supported by the Fundação da Ciência e Tecnologia by Grant PDCT/FP/FNU/50224/2003.

N. Barros is with the European Organization for Nuclear Research, CERN, CH-1211 Genève 23, Switzerland, and also with the Faculty of Sciences, University of Lisbon, P-1749-016 Lisbon, Portugal (e-mail: Nuno.Barros@cern.ch).

A. Amorim, T. Franco, D. Klose, and L. Pedro are with the Faculty of Sciences, University of Lisbon, P-1749-016 Lisbon, Portugal (e-mail: Antonio.Amorim@fisica.fc.ul.pt; Dinis.Klose@cern.ch; Tiago@cern.ch).

R. Hawkings is with the European Organization for Nuclear Research, CERN, CH-1211 Genève 23, Switzerland (e-mail: Richard.Hawkings@cern.ch).

Digital Object Identifier 10.1109/TNS.2006.879018

II. The ATLAS Database Environment

The ATLAS software system is composed of two usual subsystems, the Online and Offline systems. The Online System [3] is responsible for the software to read the detector Conditions and data and the Offline [4] system is related to the post-run tasks, like event reconstruction and data analysis. However, the Conditions Databases is a component that connects both systems, having thus a very different set of requirements imposed by each system. In the Online system, the Conditions Database is used mainly to store Slow Control Data through a SCADA system, PVSS [5]. Another Conditions Database usage is in the storage of Bookeeping data like run parameters. Run parameters data is all the data that define a particular run, like beam energy, beam type, run number and number of events in the run. Both these applications were intended to store objects with some structure that could be packed in the same folder, thus being a very interesting case study for a future implementation. In the Offline environment the Conditions Database is used both to read detector control system (DCS) data stored by the online system, the Run parameters data and to store data computed from the Offline system. As examples of data stored from the Offline system we can highlight the references to external objects that need to have an IOV, such as calibration and alignment constants.

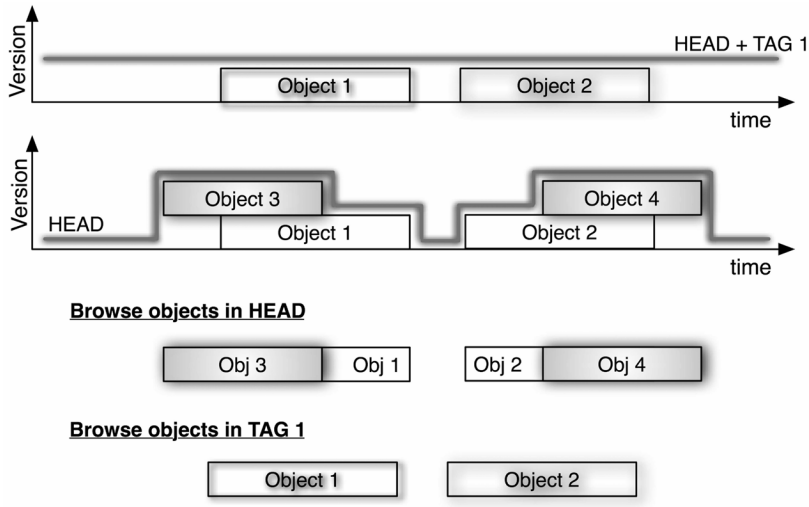

Fig. 1. Schema of the tag mechanism.

III. The Implementation Based on Byte Arrays

The first implementations of the ConditionsDB used either a separate commercial object database to manage the data [6] or provided the basic store of hierarchical data based on IOVs, storing the objects as BLOBS in the database [7]. These implementations allowed storage of temporally varying objects in a hierarchical structure using an internal mapping of folders. The mapping was transparent to the users, who just needed to provide the folder name at the moment of storage of the object. On the user side, the data seemed to be stored in a folder tree, while in the database itself, the data was stored in different tables, with at least one table per folder [8]. Another feature that was used mainly in the offline system, was the tagging mechanism. This feature allowed the users to insert objects with overlapping IOVs and apply a *tag* to each *version* of the object. Later, a user could provide one of these tags to specify which version of the object should be retrieved. This versioning system required a special database schema, specifically to map which objects were part of each tag. The user could only tag the latest version of the objects, which allowed the user to take a snapshot of the latest version of the objects in the folder (Fig. 1).

However, this implementation of the Conditions Database had the same limitation of other streamable solutions (see Section V), which was a great performance penalty for objects of big size and complexity. This was the main reason that leaded to the improvements described in this publication.

IV. Case Study of the Atlas Online System

In the ATLAS Online System, there was a need to store the detector control system (DCS) data into a persistency service that could handle the interval of validity (IOV) of the data stored. Thus, the Conditions Database was the natural solution.

Following the experience of the byte array based solution (see III) it became clear that storing the data as a BLOB imposed a big penalty both in performance and in functionality, as there was not a standard in the way the data was encoded. So, there was a need to find a different solution, that could solve the problems described.

Another kind of data to be stored was the Run Parameters and other similar objects published in the information service (IS). The IS is a component of the ATLAS Online Software Infrastructure, which allows information to be shared between all the ATLAS Online Software components.

The run parameters are data that classifies the current running state of the detector, and thus can be considered Conditions Data. Like DCS data, these objects have a structure, sometimes complex, and need to be stored keeping temporal information that would allow the users to later reconstruct the history of the state of the detector. The IS provides the facilities that allow the data to be published and read.

Having multiple partitions defined, as shall be the case during the data taking period, these objects can change at a high frequency, which imposes a great constraint in performance to the conditions DB. A partition is a logical representation of software and hardware resources that handle the ATLAS data acquisition.

Following the particular case of the DCS Data, we realised that the next natural step in the evolution of the Conditions database would be the creation of a storage mechanism that would allow to keep the object structure, getting rid of coding constraints, and making the objects more portable.

This new technology should not only be able to keep the data structured, but should also keep the same functionality of the legacy implementation. Due to the similarity in the characteristics of the objects to be stored from the ATLAS Online System, the previous arguments where taken as a base for the new implementation of the Conditions Database.

V. The Introduced Improvements

Resulting from very useful information that was collected from the use case analysis of the ATLAS online and offline system and their requirements, many improvements were introduced. The most significant was shifting the responsibility of encoding the objects from the user applications to the central database server.

As most data that is stored in the Conditions Databases results either from calculations or from specific monitoring hardware, the objects are generally larger and more complex than the ones

resulting from typical database user interfaces. Thus a new infrastructure was developed, the *CondDBTable*, along with associated managers, which implemented the ability to store structured data in a simple, straightforward way.

The new implemented infrastructure should provide the users the functionality to store structured data without imposing on them the responsibility for the object encoding. Thus, the best solution would be the creation of a generic object that could map, as directly as possible, into a relational table, or in particular into a relational row, and at the same time provide the information about the object schema.

Depending on the complexity and size of the objects stored, this solution proved to have advantages when compared with streamable solutions like POOL [9] and Root [10] files, and even the old BLOB based implementation, being more performant and thus more reliable for the needs of the online systems. The CondDBTable became a halfway house between the object oriented and relational models, combining the advantages of each technology.

The CondDBTable, was developed based on standard template library (STL) [11] containers that would hold the contents of the object. Using an internal system of enumerations, the objects keep also information about the object structure by column basis, having a strong relation with the structure of the object in the persistency.

Along with this object, a new set of updates were necessary in the core of the ConditionsDB implementation, allowing not only to store the legacy BLOBS but also the new table based objects. These changes lead to an evolution of both the database schema and the managers necessary to store and read this new kind of data.

Following the understanding of the online system needs, it became obvious that the ability to store the data mapped in a relational table combined with the hierarchical classification could lead to problems with the database size and number of tables. So, the CondDBTable was extended to support the storage of different objects with the same structure, which became known as *table with IDs*. This new feature allowed to manage several objects, sharing the same schema, with a single relational table and a single CondDBTable.

Each object was inserted into the CondDBTable, and a new column was created by default, which contained an identifier, specified by the user, that could distinguish from the other objects in the table. This was specially useful to store DCS channel data, where each channel had a unique identifier and thus avoiding the creation of a great number of tables in the database for each channel.

In order to extend the usability of this technology, the CondDBTable supported the storage of a larger range of native data types as well as strings. It was also extended to support arrays of the data types specified earlier in a single relational column. These types were mapped from a STL vector into a relational row. In Table I one can see the available data types and how they were mapped into the CondDBTable.

Being an *in memory* table, the CondDBTable is completely customisable by the users. The users are the responsible for defining the schema of a new object, using the provided methods to define both the schema and contents of the object. In Table II a list of the most used methods is presented. In the table, the typename T stands for templated methods. In this case an overloaded method for each supported data type is made available.

TABLE I
MAPPING FROM THE C++ DATA TYPE INTO THE CONDDBTABLE MAP

C++ data type	CondDBTable type
bool	cdbBool
long int	cdbInt
short int	cdbInt
long long	cdbLongLong
float	cdbFloat
double	cdbDouble
std::string	cdbString

TABLE II
SOME OF THE MOST USED METHODS AVAILABLE TO THE USERS TO HANDLE THE CONDDBTABLE

Method type	Method
Structure	int getNames(vector<string> &names) int setName(unsigned n_column, const string& name) int getTypes(vector<ICondDBTable::cdb_types> &types) int setType(unsigned n_column, cdb_types type) void resize(int num_rows) int setName(unsigned n_column, const string& name)
Read data	int getCell(unsigned n_column, unsigned n_row, T &ndata) int getRow(unsigned rowNumber, vector<string> &values) int getTillTime(const SimpleTime &till) int getSinceTime(const SimpleTime &since)
Write data	int setCell(unsigned n_column, unsigned n_row, T &ndata) int setTillTime(const SimpleTime &till) int setSinceTime(const SimpleTime &since)

There are still some other methods that allow the users to further manipulate the object, however, its use was not wide. Examples of such functionalities are the change of object data and IOV.

VI. THE SPECIFICALLY DEVELOPED INTERFACES

The implementation of the CondDBTable was successfully deployed and was used by many applications for each of the ATLAS subsystems. To ease the usage of this infrastructure by the different ATLAS subsystems, several specific interface applications were developed aiming to simplify both writing and reading data on the databases. A web browser was also developed, which allowed to look at the data without need to write any specific application. Some of the interfaces developed shall be described in more detail later.

A. The Online Conditions Database Interface

The interface to the Online Software, the conditions database interface (CDI) [12] was implemented in order to store the data objects collected by the trigger and data acquisition (TDAQ) IS [13] about the run parameters, and to store them in the ConditionsDB.

The IS is a system that allows several resources from the Online Software to share information between them. Amongst several other features, it is able to keep the structure of the data published, which made it a very good component to connect to the Conditions Database to store the data.

Thus, the CDI was implemented to connect to the IS and retrieve the data published by the resources. Using the structure information kept by IS, the CDI was able to translate the structure of the objects and map them into the CondDBTable.

The CDI connects to the IS and subscribes for a couple of objects that were defined by the user in order to specify the information sources (also IS objects) that will be the input information for the ConditionsDB. Using callbacks, when one of the subscribed objects was updated a signal was sent back to the CDI, which was responsible for retrieving and mapping the IS object into a CondDBTable, add the IOV and store into the database.

This was the main use of the CondDBTable in its basic form (without IDs). In fact, the CDI package was developed in a way, that, making use of the IS services, it was possible to subscribe and store into the ConditionsDB any object that could be published into the IS.

There was also some use for histogram storage into the ConditionsDB which was very simplified with the CondDBTable support for arrays.

B. The PVSS Data Manager

The main use of the CondDBTable derived from the storage of detector control system (DCS) data through the SCADA system PVSS. To provide support to this system a package was implemented, the *PVSS Data Manager* that acts as a PVSS standard plug-in, which linked both the PVSS libraries and the ConditionsDB.

Making use of the CondDBTable support for structured data with ID's, the different channel conditions acquired by the PVSS system were stored in the ConditionsDB into the same relational table. As the different channel information has the same structure, the ConditionsDB stores the different objects into the same relational tables, handling each object independently due to the additional index added internally to the object.

This system represented the largest source of usage of CondDBTable with the ID feature. The different channels represented objects with the same structure. Nonetheless these objects needed to be distinguished in some way since the IOV of each object could be different.

In this way, one could reduce the required number of tables and queries to the database, since the data with the same structure could be in the same place. The users could handle each object with different ID independently of the existing others, either to read or to store. This PVSS Data Manager was developed for both Linux and Win32 platforms, the later being extensively used by the ATLAS user community.

C. The Athena Conversion Service

Due to the success of the CondDBTable amongst the online community, the users demonstrated a great interest in reading the Conditions data into the offline reconstruction algorithms.

The offline system is composed by a modular framework, the *Athena Framework* [14], and is intended to be used by the scientific community to do the physics analysis in the ATLAS experiment.

The old byte array solution was already integrated into Athena, by means of the IOVDbSvc, which was mainly responsible for the objects stored as BLOBS with references for external objects. However, the new implementation, even if it kept the same functionality as the previous one, provided new features that would require a lot of modifications in the existing integration modules to provide the new functionalities. Thus it became clear that it would be best to make a distinction between the functionalities and implement a new module, responsible for the integration of the new CondDBTable technology.

In order to integrate the new improvements, both an Athena Conversion Service and a new object were created in the Athena framework [14]. The object, the *GenericDBTable*, is analogous to the CondDBTable and provides the same functionality. Like the CondDBTable, this object could easily map any object storable in the ConditionsDB, which means that the Conversion Service was not much different from a simple application to read data from the database.

With this interface the users could easily retrieve conditions data by issuing a request to the Athena StoreGate. The Conversion service knows how handle both the CondDBTable and the GenericDbTable, being thus able to convert data from one object to the other.

In Fig 2, one can see the structure and relation between the different Conditions Database related components.

D. The ConditionsDB Web Interface

For generic browsing of the conditions data, a Web Browser Interface was created. This tool was implemented in PHP and, due to the mapping of the objects in relational tables made by the ConditionsDB, it could show to the user the contents of the object, its metadata, allowing at the same time to make searches based both in the hierarchical order, IOV, tagging mechanism and even in the contests of the objects (for the case of objects stored with any "flavour" of the CondDBTable).

In order to be able to handle the C++ objects returned by the ConditionsDB, a bind was developed to extend the functionalities of PHP to read and handle the CondDBTable (Fig. 3). In this way the PHP browser was able to handle all the information available in the ConditionsDB, be it folder information, object structure or object data.

The tool was also developed in order to allow the users to have some basic information about the objects that were stored as BLOBS, however, due to the user side coding, it was not possible to usefully show the contents.

The browser allows the user to specify their own connection parameters, showing multiple information in several frames. In particular one can see an hierarchical folder tree which represents the hierarchical structure of the data in the database, and each leaf of the tree opens another frame where the user can make multiple queries, depending on the kind of object stored. For instance, the user can make searches by tag, by timestamp or interval of validity [15].

Fig. 4 depicts a representation of the contents of an object stored using the CondDBTable, as it is presented by the Web Browser. It also shows the web form that allows the user to refine his search based on the contents of the object and taking

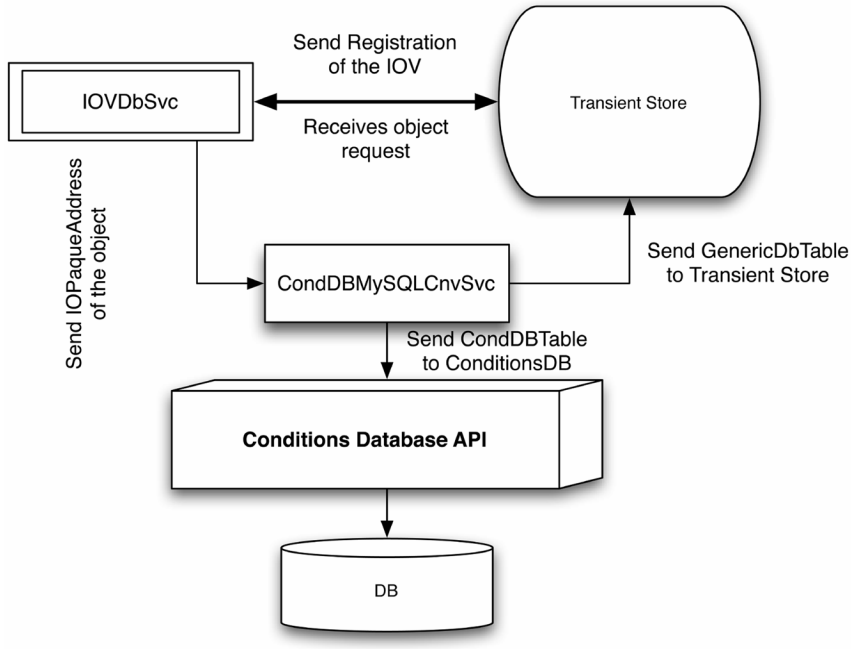

Fig. 2. Schema of the integration of the ConditionsDB into the Athena Framework.

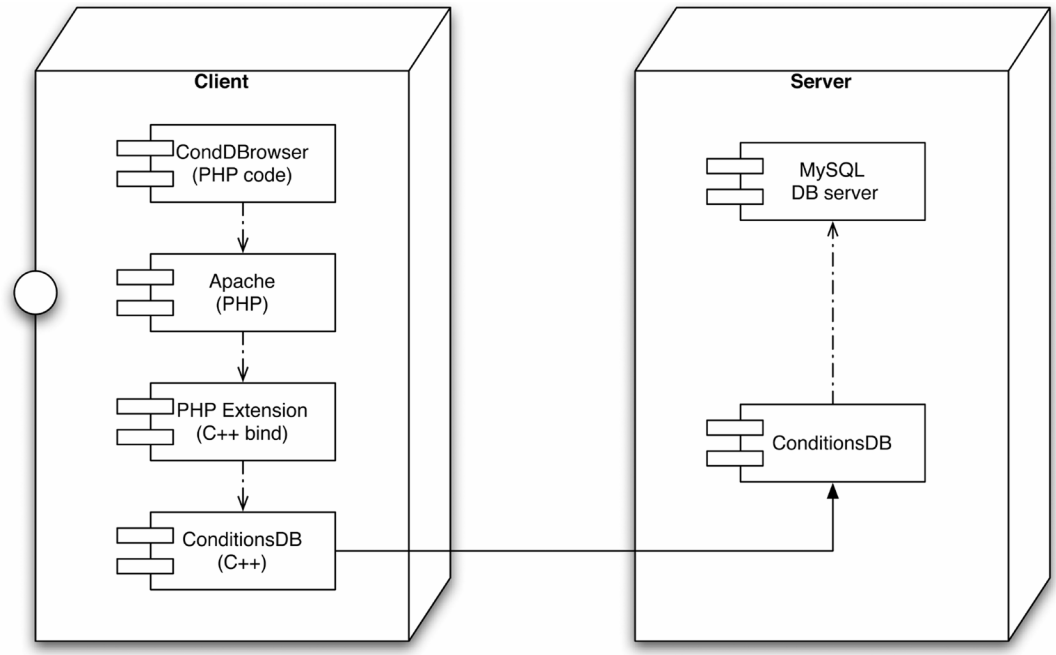

Fig. 3. Schema of the bind developed to interact with the ConditionsDB from a PHP web browser.

into account its structure: different column types allow particular search patterns.

In a later step the browser was extended to allow advanced searches, which allowed the users to make searches in particular values of the objects stored as CondDBTables, having already limited the amount of objects by choosing the folder and the IOV.

VII. Evaluations

This new evolution of the ConditionsDB was extensively tested and tuned leading to very promising results from its use in the ATLAS Combined Testbeam that occurred from May to November 2004. The overall performance of the new features surpassed the previous implementation providing faster storage and retrieval of objects from the database.

The DCS data was stored into the ConditionsDB using the PVSS Data Manager, leading to storage of the data as CodDBTables. In particular, the feature of tables with IDs (see V) was extensively used to store data points set up by the users. The byte array feature was still used, mainly to store references to external objects, where the integration of the byte array technology into the analysis framework played a major role. The

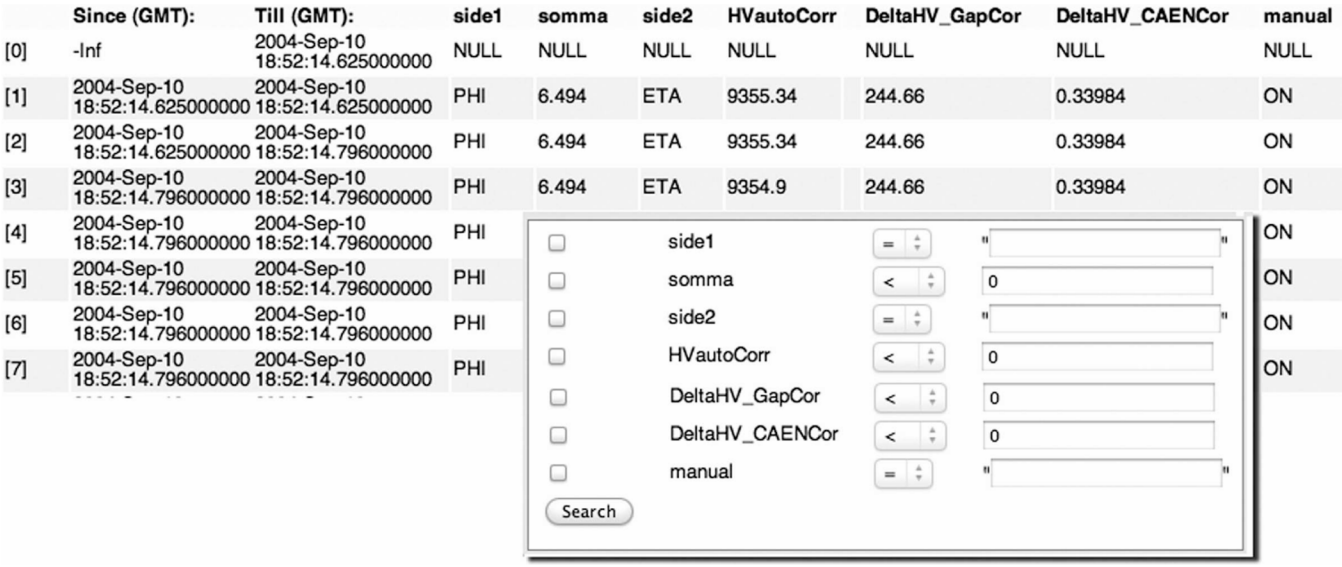

Fig. 4. Contents of the object as shown in the web browser.

data was computed with the user algorithms and stored into external objects, like POOL files. Then, a reference for this object was stored into the ConditionsDB, together with a IOV.

For the CondDBTable, the integration into Athena was extensively used to read data for the reconstruction jobs, particularly the DCS data stored using the PVSS Data Manager. Overall the ConditionsDB proved to be very reliable and performant, storing data in a simple, reliable way, that even now is being used to perform reconstruction on real data.

VIII. Conclusion

The new evolution of the ConditionsDB presents a solution based on the study of the requirements from the ATLAS systems, resulting in a very successful development which was widely tested and accepted among the user comunity.

Although this implementation was developed inside a particle physics experiment, and some of the extensions are very specific to this environment, the ConditionsDB use can easily be exported to other areas of application.

In fact, this solution is appropriate to every application that requires the storage of structured data that must have a temporal classification. Moreover, the hierarchical solution and versioning mechanism allows the ConditionsDB to be used in very different areas where the data stored can be very different in nature. The hierarchical solution allows to have a transparent separation of uncorrelated data inside the same database. The versioning mechanism proved to be very useful in areas where the data can change with external parameters and the need to have the different versions available is crucial. The small amount of external dependencies makes this technology very portable and easily configurable by any user.

As a result, the presented infrastructure is a reliable, easy to set up solution which can be easily applied in different areas, having already the successful background of the ATLAS experiment.

References

[1] *The ATLAS Experiment Web Site*, [Online]. Available: http://atlas.web.cern.ch/Atlas/

[2] A. Amorim, J. Lima, C. Oliveira, L. Pedro, and N. Barros, "Experience with the open source based implementation for atlas conditions data management system," *CoRR*, vol. cs.DB/0306006, 2003.

[3] *The ATLAS Online System Web Page*, [Online]. Available: http://atlas-onlsw.web.cern.ch/Atlas-onlsw/

[4] *The ATLAS offline Computing Web Page*, [Online]. Available: http://atlas.web.cern.ch/Atlas/GROUPS/SOFTWARE/OO/

[5] *The Athena Framework Web Page*, [Online]. Available: http://atlas.web.cern.ch/Atlas/GROUPS/SOFTWARE/OO/architecture/

[6] *The BaBAR Experiment Web Site*, [Online]. Available: http://www.slac.stanford.edu/BFROOT/

[7] *The CERN IT Conditions Database Project Web Site*, [Online]. Available: http://wwwdb.web.cern.ch/wwwdb/objectivity/docs/conditionsdb/

[8] A. Amorim, J. Lima, L. Pedro, D. Klose, C. Oliveira, and N. Barros, "An implementation for the ATLAS conditions data management based on relational DBMSs," *IEEE Trans. Nucl. Sci.*, vol. 51, Jul. 2004.

[9] *The POOL Web Site*, [Online]. Available: http://lcgapp.cern.ch/project/pool/

[10] *The ROOT Web Site*, [Online]. Available: http://root.cern.ch/

[11] *The Standard Template Library*, [Online]. Available: http://www.cppreference.com/cppstl.html

[12] A. Amorim, "Conditions databases: The interfaces between the different ATLAS systems," *CHEP*, 2004.

[13] *The Information Service Manual Web Page*, [Online]. Available: http://atlas-onlsw.web.cern.ch/

[14] *The Athena Framework Web Page*, [Online]. Available: http://atlas.web.cern.ch/Atlas/GROUPS/SOFTWARE/OO/architecture/

[15] A. Amorim, "The WEB interface for the AT-LAS/LCG MySQL conditions databases and performance constraints in the visualization of extensive scientific/technical data," in *Computing in High Energy Phys. Proc. CHEP*, 2004.