Evolution of the architecture of the ATLAS Metadata Interface (AMI)

J Odier¹, O Aidel², S Albrand¹, J Fulachier¹, F Lambert¹, on behalf of the ATLAS Collaboration.

¹Laboratoire de Physique Subatomique et de Cosmologie, Université Grenoble-Alpes, CNRS/IN2P3, 53 rue des Martyrs, 38026, Grenoble Cedex, FRANCE
²Centre de Calcul IN2P3/CNRS Domaine scientifique de La Doua, 43 bd du 11 Novembre 1918, 69622 Villeurbanne Cedex 69622, FRANCE

E-mail: jerome.odier@lpsc.in2p3.fr

Abstract. The ATLAS Metadata Interface (AMI) is now a mature application. Over the years, the number of users and the number of provided functions has dramatically increased. It is necessary to adapt the hardware infrastructure in a seamless way so that the quality of service remains high. We describe the AMI evolution since its beginning being served by a single MySQL backend database server to the current state having a cluster of virtual machines at French Tier1, an Oracle database at Lyon with complementary replication to the Oracle DB at CERN and AMI back-up server.

1. Introduction

ATLAS [1] is one of the two general-purpose detectors at the Large Hadron Collider (LHC). It investigates a wide range of physics, from the search for the Higgs boson to extra dimensions and to particles that could make up dark matter. The ATLAS Metadata Interface (AMI) is a framework used by ATLAS for two major applications. Firstly, it is used for the dataset search tool, acting as a mediator between the Production System and the acquisition of real data. Secondly, the framework is used by Tag Collector, the tool for software release management.

“Any fool can write code that a computer can understand” writes Martin Fowler in his book on refactoring [2]. It is certain that anyone who has been involved with computer programs over a long period of time has seen applications thrown away and replaced by complete rewrites. Software can be thrown away because it can no longer be adapted to the changing requirements of function or scalability, or simply because the original was written in such a way that it is incomprehensible and so impossible to refactorise, perhaps even for the original authors.

Certainly an application which must provide a service for a community over a long period should try to avoid drastic changes and aim for an adiabatic process. Therefore the initial design must allow evolution.
The architecture of AMI was described in detail in previous publications [3][4][5]. In this article we trace the history of what can now be considered to be a mature application. The article is organized by theme and by chronological order. We first describe the prototype and the establishment of the architecture. Then we show how the software has evolved over time, to adapt to changing requirements and to scale with the growing demands of the ATLAS experiment. Subsequently, we describe how the hardware we use has changed over the lifetime of the application. Evolution of both the servers and the database backend are discussed. Any long term project must establish methods of working efficiently. We discuss our adopted solutions.

2. The prototype of AMI
The first prototype of the AMI framework was developed in 2001. AMI has in two ancestors. The first one was a bookkeeping application for the ATLAS Liquid Argon sub-detector test beam acquisition. It was written in Java with a graphical interface. Data were uploaded asynchronously to a MySQL database. A web interface written in PHP was provided. The second ancestor was the first version of Tag Collector. This was a completely web-based application also written in PHP.

It was quickly realised that a large number of database applications needed the same functions, and therefore a good design would enable the two functions. At this time the AMI developers were asked to provide bookkeeping for the first ATLAS data challenges. This was one of the catalysts for the establishment of our design principles. The other was the rapid increase of demands for new functions in Tag Collector.

3. Definition of the Architecture
The basic architecture of AMI was established in the period 2002-2003. This phase of the development of AMI was extremely important. The importance of designing an application strong enough and flexible enough to withstand the inevitable evolutions in technology and requirements was recognized. We laid down some design principles. AMI should be independent of platform, operating system and database technology. It became clear that the software should be defined by 3 tiers.

The core packages, or lowest tier, manage the remote connection to the database, and the transmission of SQL commands. Connections are indirect, the code uses logical connections parameters, and the real physical parameters are kept in a central database, which can be mirrored. This allows a flexible mechanism for geographic distribution of databases.

Databases contain their own internal description for introspection; this means that code creating interfaces can be generic. The middle layers of the software provide generic classes for accessing the bookkeeping databases, using their internal description. This ensures support for schema evolution.

The top layers of the software are project specific. They contain all the business logic.

4. Initial Technology Choices
It was decided to adopt Java for both the server and client code. Web interfaces were produced in HTML generated by the Java server code. Java is platform independent, and widely used both for database applications and web applications.

The Java Database Connectivity (JDBC) interface is well adapted for use with multi-relational database engines. This choice has proved to be rational. The first deployment was on a single server at the LPSC, Grenoble. The database was a local MySQL installation. The evolution of the database infrastructure is described in section 6.

The core of the AMI server is a command engine over a high level database engine. All commands sent to the server either through the web interface or through the command line client produce output in XML. The XML is then transformed using XSLT for appropriate display.

The following sections show that it was possible to adapt the application while keeping the fundamental design intact.
5. Software Evolution
In the period 2003-2004 was the first major set of software changes. Partially they were triggered by re-write of the TagCollector using the AMI framework. Other changes were introduced over the last decade, following the needs of ATLAS, without ever jeopardizing the original 3-tier design.

5.1. User management
One of the features needed for Tag Collector is the management of user rights. A system with a high granularity was set up. Users are given a set of roles. Each role is associated with a set of commands which holders of the role can execute. Each role can be restricted to a set of AMI catalogues, and further qualification can be defined in a role validator class – which could for instance restrict command action to particular records in a database table. User rights in AMI are not connected with user rights defined at the database level.

In 2007, access to dataset metadata in ATLAS was confined to members of the collaboration therefore the management of users had to be extended beyond a simple username and password pair, to support the GRID Virtual Organization Membership Service (VOMS) mechanism [6]. Any member of the ATLAS VO can read information in AMI. Writing operations require a particular AMI role, or a VOMS role mapped to an AMI role. It is not possible for AMI to use the CERN single sign-on mechanism because the master site is not at CERN. However in 2014 a module to manage some Tag Collector rights on software packages was linked to CERN group mailing lists which define the rights on the ATLAS SVN repository.

5.2. Software for Database Operations
A database application with a large number of users must manage the database connections efficiently. A “home-made” connection pool was put into place in 2004. Since then, each AMI command is responsible for taking connections in the pool, and returning them at the end of the execution. Since AMI supports more than one database, a single command may involve update or insertion operations on more than one database. It was therefore necessary to extend the connection pool to manage transactions. All connections used by a command are held in a vector, and operations are committed (or not) when the connections are released.

The future version of AMI will replace the original connection pool by the Tomcat JDBC Connection Pool [7] designed for highly concurrent environments and multi-core/cpu systems.

5.3. The Web interface
As mentioned above, the first AMI web interfaces were written in PHP. Nevertheless this was rapidly abandoned for a more structured approach. A Java class hierarchy for the generation of HTML pages was written and this was used until 2013, when it was partially replaced by a structure based on JavaScript, making heavy use of the Twitter Bootstrap [8] CSS framework. This work was extended further in 2014. The web interface, which previously manipulated XML documents using XSL transformations, is now being migrated to Asynchronous JavaScript (AJAX).

Web development has been considerably simplified by the development of a framework for AMI based on JQuery [9] and Twitter Bootstrap. This work is described in detail in these proceedings [10].

5.4. The CLI and the API
The first AMI client was written in Java. It was not convenient because it was too dependent on server code changes, and clients had to update too frequently. In 2004, it was decided to move to a classic Simple Object Access Protocol (SOAP) “Web Service” approach [11]. This was actually an approach strongly supported by the EGEE Grid middleware at the time, and a workshop was held at CERN in June 2004 with contributions from the four LHC experiments [12]. The web service approach had many advantages in theory, since users can use utilities to build a client in any language they wish.
However in the ATLAS environment this feature was not exploited. ATLAS allows only a Python client.

The principle of a client which is as light as possible was retained, thus minimising client changes. The latest version of our client (pyAMI version 5) has dropped the SOAP web service in favour of HTTPS. The syntax has been changed, and the client has been split into modules. There is one basic client, and the other modules are experiment specific. This change of AMI was not backward compatible. It was rendered necessary when experiments other than ATLAS adopted AMI.

In the same way, a new C/C++ client based on HTTPS and OpenSSL is currently under development.

5.5. Filling the Databases
The first version of the AMI dataset bookkeeping used a "push" method of filling the databases. Executing jobs generated commands for the AMI clients, and the users executed the commands at remote sites. This did not work very well for several reasons. Once the ATLAS production system took over control of user jobs on the data grid, the "push" mechanism did not work at all, and the survival of AMI was in doubt. In 2006 an ATLAS internal review decided that AMI should be adopted for physics metadata bookkeeping, and accorded AMI reading rights on the necessary databases. AMI switched to a "pull" mode of filling the databases for ATLAS with two dedicated task servers.

Other users of AMI are free to push data into the catalogues, and client functions are provided for this.

6. Hardware Evolution
It was clear from the beginning of the project that MySQL was not a longstanding choice, for both technical and political reasons. Oracle is a much better choice for a large database, and it was available at both CERN and the French Tier 1 site of the Centre de Calcul IN2P3 (CC-IN2P3), Lyon.

AMI had rapidly outgrown the single web and database server at Grenoble. In 2004, it was time to change. After consultation with the ATLAS database coordination, ATLAS France and the CC-IN2P3, the move to Lyon became possible. Two servers were purchased and the database backend was extended to work with Oracle using the Oracle cluster. Part of the AMI application still runs on MySQL, but it was moved to the CC-IN2P3 managed cluster.

In 2008, an Oracle cluster of 4 nodes (dual processor, quad-core with 32GB RAM) was put in place at CC-IN2P3 for hosting the AMI database as well as the ATLAS database and two new web servers were bought and installed at CC-IN2P3 to cope with the high activity and load periods. At the beginning of AMI, the database size was only a few gigabytes (~12 GB) but with an activity continuously growing, the database size reached in 2015 more than 450 GB, see figure 1.

![AMI used space](image)

**Figure 1.** Space used by the AMI database at CC-IN2P3.
By an ATLAS request, since June 2005, a partial site failover was implemented at CERN, so that AMI users can switch to CERN in case of a CC-IN2P3 failure. An AMI Tomcat server is available at CERN with its own database replicated in real time from CC-IN2P3. Replication was done via Oracle Streams technology configured for one directional replica set read-only on the destination database at CERN. After the official announcement by Oracle in 2013 regarding the deprecation of the stream technology, it was decided jointly with the CC-IN2P3, AMI and the CERN IT to replace the stream technology by Oracle Golden Gate software.

AMI is very critical for ATLAS. The CC-IN2P3 implemented a data guard solution in 2014. Data Guard is the name for Oracle's standby database solution, used for disaster recovery and high availability. In case of crash or maintenance of the production database at CC-IN2P3, all users will be redirected to the standby CC-IN2P3 AMI database with a minimal interruption of service. In production, this solution allows the service downtime to be minimized to a few minutes during a major ORACLE upgrade. In 2015, Oracle servers come to the end of their warranty and should be replaced during the summer. This replacement will be a good opportunity for using ORACLE Data Guard.

In 2014 the CC-IN2P3 opened its OpenStack [13] cloud infrastructure to AMI, see figure 2. We are currently in the process of migrating to the virtual infrastructure, and the tasks to pull data to AMI will run on a virtual machine.

Figure 2. AMI infrastructure in 2014: 2 task servers, 7 Tomcat servers and 1 integration server.

7. Tools
During the period of AMI development the size of the developer team at Grenoble has doubled, and there were several short term collaborations from ATLAS colleagues based in other research institutes. Work methods had to be made more formal. Development collaboration needs a shared code repository and a tool to manage builds and integration. A bug tracker is also essential. A fairly standard set of tools was chosen: SVN for code versioning, Jenkins [14] for building and deployment, and Redmine [15] for tracking bugs. Redmine is about to be abandoned because ATLAS has decided to adopt JIRA [16] for bug tracking. It is planned to migrate to GitHub, which seems more suitable for the future developments, see section 8. Jenkins is an extremely satisfactory tool, and, in the near future, the only change will be to move from ANT [17] to Maven [18] for the Jenkins scripts.

8. Outlook
AMI is now a well-established and mature application, with about 2000 active users from the ATLAS collaboration. Since 2013 another experiment, nEDM [19], is using AMI for run bookkeeping. Agreement was reached with the SuperNEMO [20] collaboration for use of AMI, and at least one other experiment has expressed interest.
This expansion of AMI prompted some refactorisation of the core code, since the frontier with experiment specific code had become slightly fuzzy over time. The new preliminary core code is open-source and is available on GitHub [21]. We will work on making it much easier for experiments to set up AMI databases with a minimum of support from the core team.

Acknowledgements
We would like to thank all the staff of the CC-IN2P3 Lyon who have helped us with the AMI project, in particular Philippe Cheynet, Benoît Delaunay, Pierre-Etienne Macchi, Yannick Perret, Mattieu Puel and Jean-René Rouet.

References
[1] The ATLAS Collaboration 2008 The ATLAS Experiment at the CERN Large Hadron Collider JINST 3 S08003 doi:10.1088/1748-0221/3/08/S08003
[2] Refactoring - Improving the Design of Existing Code. M Fowler et al. Addison Wesley 1999
[3] J Fulachier, O Aidel, S Albrand and F Lambert 2013 Looking back on 10 years of the ATLAS Metadata Interface. Proceedings of the 20th International Conference on Computing in High Energy and Nuclear Physics (CHEP2013) J. Phys.: Conf. Ser. 513 042019 doi:10.1088/1742-6596/513/4/042019
[4] S Albrand, T Doherty, J Fulachier and F Lambert 2008 The ATLAS Metadata Interface J. Phys.: Conf. Ser. 119 072003 doi:10.1088/1742-6596/119/7/072003
[5] E Obreshkov et al., on behalf of the ATLAS Collaboration. Organization and Management of ATLAS Software Releases; Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment, Volume 584, Issue 1, 1 January 2008, Pages 244-251
[6] Virtual Organization Membership Service: http://italiangrid.github.io/voms/
[7] Tomcat JDBC Connection Pool: https://tomcat.apache.org/tomcat-8.0-doc/jdbc-pool.html
[8] Twitter Bootstrap: http://getbootstrap.com/
[9] Jquery: http://jquery.com/
[10] J Odier, S Albrand, J Fulachier, F Lambert 2015 Migration of the ATLAS Metadata Interface (AMI) to Web 2.0 and cloud. Proceedings of the 21st International Conference on Computing in High Energy and Nuclear Physics (CHEP2015) J. Phys.: Conf. Ser.
[11] SOAP: http://www.w3.org/TR/soap/
[12] http://indico.cern.ch/event/a042197/
[13] OpenStack: http://www.openstack.org/
[14] Jenkins: http://jenkins-ci.org/
[15] Redmine: http://www.redmine.org/
[16] JIRA: http://www.atlassian.com/software/jira/
[17] ANT: http://ant.apache.org/
[18] Maven: https://maven.apache.org/
[19] The nEDM Experiment: http://nedm.web.psi.ch/
[20] The SuperNemo Experiment: http://nemo.in2p3.fr/nemow3/
[21] AMI sources: https://github.com/ami-lpsc/