Migration of the ATLAS Metadata Interface (AMI) to Web 2.0 and cloud

2015 J. Phys.: Conf. Ser. 664 062044

(http://iopscience.iop.org/1742-6596/664/6/062044)

Journal of Physics: Conference Series **664** (2015) 062044 doi:10.1088/1742-6596/664/6/062044

Migration of the ATLAS Metadata Interface (AMI) to Web 2.0 and cloud

J Odier[1], S Albrand[1], J Fulachier[1], F Lambert[1], on behalf of the ATLAS Collaboration.

[1]Laboratoire de Physique Subatomique et de Cosmologie, Université Grenoble-Alpes, CNRS/IN2P3, 53 rue des Martyrs, 38026, Grenoble Cedex, FRANCE

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

Abstract. The ATLAS Metadata Interface (AMI), a mature application of more than 10 years of existence, is currently under adaptation to some recently available technologies. The web interfaces, which previously manipulated XML documents using XSL transformations, are being migrated to Asynchronous JavaScript (AJAX). Web development is considerably simplified by the introduction of a framework based on JQuery and Twitter Bootstrap. Finally, the AMI services are being migrated to an OpenStack cloud infrastructure.

1. Introduction

This paper is about the migration of the ATLAS Metadata Interface (AMI) to Web 2.0 and cloud technologies. Initially developed for the ATLAS experiment [1] at the CERN Large Hadron Collider (LHC), AMI is a generic framework for metadata cataloguing using relational databases. It is the basis of three important tools which are part of the offline software of the ATLAS experiment. AMI, the first and eponymous tool, is used principally to catalogue the datasets available for analysis. The second tool, Tag Collector, is a highly specialized application which is part of the management of the various releases of ATLAS software. The third tool, the AMI-Tags interface, is used to save the processing history of the datasets.

AMI can be considered to be a mature application, since its basic architecture has been maintained for over ten years. However, it is necessary to adapt both software and hardware infrastructure in a seamless way so that the quality of service remains high. The history of AMI, software side and hardware side, is described in detail in these proceedings [2].

In what follows we start by recalling the main characteristics of the AMI core. Then we describe the new AMI web framework based on Web 2.0 technologies used since 2014 in the new interfaces (Tag Collector 3, AMI-Tags, etc...). Finally we discuss the migration of AMI to a cloud environment, our technical choices and the consequences in terms of flexibility and scalability.

2. AMI server overview

The architecture of AMI was described in detail in previous publications [3][4][5]. In this section, the main characteristics of the AMI core are recalled.

2.1. Overview of the AMI server

The heart of the AMI server can be considered as a command engine over a high level database engine, see figure 1. It is written in Java because this language is well adapted for both web applications and database connectivity. It is divided into four main subsystems in a 3-tier architecture.

The first one is the driver subsystem. Its role is to wrap Java DataBase Connectivity (JDBC) drivers [6] in order to make them usable in AMI. An AMI driver allows to execute SQL without considering the underlying DataBase Management System's (DBMS) characteristics. Moreover, the driver subsystem implements very high level capabilities for introspection allowing to fully describe the database schema (tables, foreign keys, indices, etc...).

The next subsystem consists of the connection pool and the transaction pool. The connection pool is a cache of JDBC connections maintained so that the connections can be reused when new requests to the database are executed. This allows to dramatically enhance the performance. The current connection pool was specifically written for AMI but, in the next releases, it will be replaced by the Apache Tomcat JDBC Connection Pool [7] designed for highly concurrent environments and multi core/cpu systems. The transaction pool is meant to guarantee atomicity when a complex command is sent to AMI or for distributed transactions (a database transaction in which two or more network hosts are involved). The internal mechanism was described here [3].

The third subsystem allows to execute SQL or Metadata Query Language (MQL) queries with a generic syntax. MQL is a subset of SQL in which all joins between tables are automatically done thanks to the AMI internal knowledge of the foreign keys. It makes it possible, for physicists, to easily write queries without knowing all the database schema details. The SQL/MQL subsystem relies on the introspection capabilities of AMI as mentioned above.

The last subsystem is the command one, it allows to define sophisticated user commands. AMI supports authentication and roles in order to restrict both command and database access to authorized users. In parallel, support for a RESTful API is being implemented.

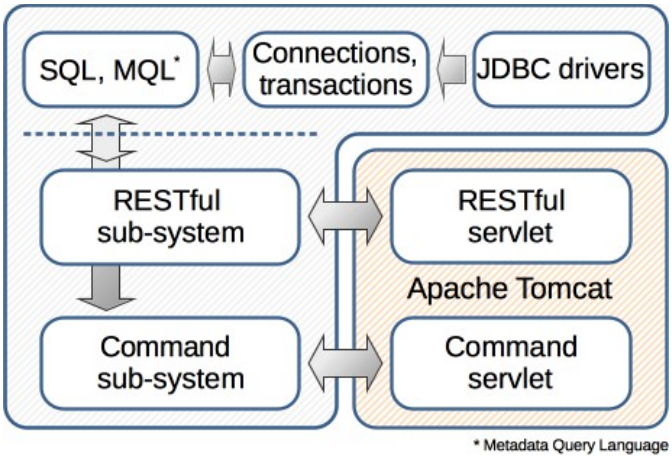

Figure 1. Overview of the AMI server.

2.2. Command and RESTful servlets

AMI can be used with Apache Tomcat [8], the open-source web server and servlet container. A "command servlet" is provided as a unique front-end for AMI. It manages HTTP/HTTPS sessions, credential or X509 certificate authentication and allows to send user commands to the command subsystem. The result, internally in XML, can be transformed using XSLT for appropriate display (JSON, CSV, text, etc...).

A very similar servlet is being implemented in order to access AMI via a RESTful API. This feature will be available soon.

2.3. The Python client

AMI comes with a powerful Python client called pyAMI, a JavaScript client (see section 3), and an experimental C/C++ client. In this sub-section, pyAMI is succinctly described.

pyAMI can run with both Python 2.6+ and Python 3.x. Based on HTTP/HTTPS, it allows credential and X509 certificate/proxy authentication. It is distributed as a Python library on the Python Package Index (PyPI) [9]. In the package, a Command-Line Interface (CLI) is also provided. It can be easily extended with experiment-specific features. All the information contained in AMI can be retrieved by pyAMI.

3. AMI web framework

The first AMI web interfaces were written in PHP, but this was rapidly abandoned for a more structured approach. A Java class hierarchy for the generation of HTML pages via eXtensible Stylesheet Language Transformations (XSLT) was written and this was used until 2013, when it was partially replaced by a structure based on JavaScript. This work was extended further in 2014. Web development was considerably simplified by the development of a framework for AMI based on JQuery [10] and Twitter Bootstrap [11]. This section is dedicated to this new AMI web framework.

3.1. Web framework capabilities

The AMI web framework has been developed in JavaScript (JS) and JQuery. It aims to separate the html (view) and the application itself (model). Each application relies on a core responsible for resource and event management (dynamic JS/CSS loading, html fragment loading, application loading/switching, common event dispatching, etc...). So one can consider that the AMI web framework follows a quasi Model-View-Controller (MVC) pattern.

In addition to the core subsystem, there is an AMI client (see section 2) and another subsystem, based on the client, responsible for authentication with credentials or X509 certificates. Commands are executed asynchronously using Asynchronous JavaScript and XML (AJAX) in order to increase both responsiveness and flexibility for applications. JSON [12] is used as data transfer format.

An AMI application is a JavaScript class containing a minimal set of methods: onReady, onExit, onLogin, onLogout and onSessionExpired. As mentioned in the previous paragraph, applications can be loaded/switched dynamically. By default, they automatically benefit from the Twitter Bootstrap CSS framework. This framework contains HTML/CSS-based design templates for typography, forms, navigation and other interface components. Consequently, the AMI applications are resolutely Web 2.0 oriented.

3.2. Web framework components

We have shown that the AMI web framework benefits from Web 2.0 functionalities. The technical complexity is hidden in some well-known libraries such as JQuery for JavaScript or Twitter Bootstrap for CSS and in the AMI web framework itself.

The interfaces are based on re-usable graphical components shared across applications. Two families can be distinguished: i) the core components which are part of the framework and are available from anywhere; ii) the dedicated components which are application specific.

An important core component is the "common login interface", see figure 2. Users get a homogeneous login system for all applications.

Another example of a core component is the "navigation tree", see figure 2. AMI applications are dealing with metadata and, very often, it is convenient to display results with a hierarchical structure. It is the reason why the D3.js javascript library [13] was included in the AMI web framework. Providing an appropriate JSON data, AMI applications are able to display information in highly customizable trees. As for authentication, users benefit from homogenous navigation tree components across all AMI applications.

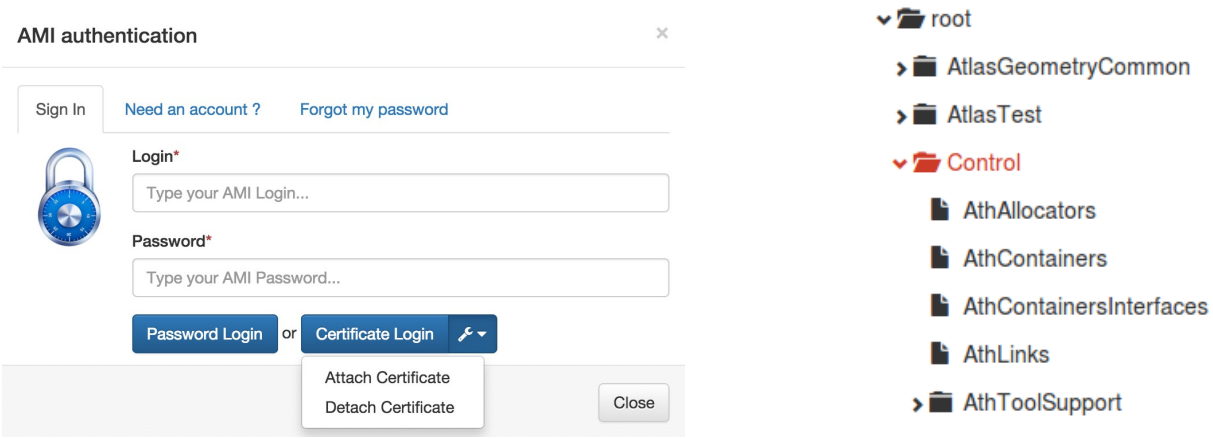

Figure 2. At left, the "common login interface" of the AMI web framework. At right, the "navigation tree" component. Both components are available for all applications.

The core components are asynchronously pre-loaded so there is no time loss when switching between applications.

Some components are dedicated to single AMI applications. For instance, in the Tag Collector 3 interface, users can manage their own packages in the ATLAS software releases with a dynamic custom dashboard. This component automatically adapts itself to the user rights and to the release context, see figure 3.

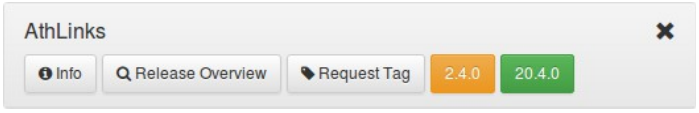

Figure 3. An example of custom reusable graphical component, a dynamic dashboard in Tag Collector 3.

To illustrate the full capabilities of the AMI web framework, figure 4 shows the AMI database modeler. This is one example among many others (hierarchical search engine, monitoring interface, configuration interface, basic Content Management System (CMS)).

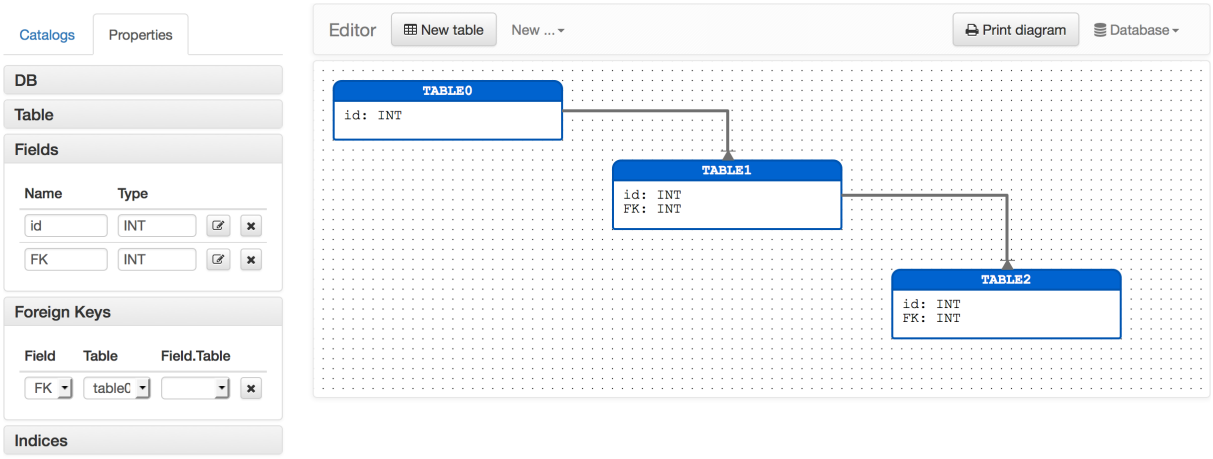

Figure 4. The database modeler of AMI, a good example to illustrate the capabilities of the AMI web framework.

4. Migration to the cloud

In this section, the migration of AMI from a cluster of physical machines to a cloud environment is described. In the short term, this experimental architecture will become the new production one.

4.1. Old architecture and limitations

The first AMI services were deployed at LPSC, Grenoble but they had rapidly outgrown the single web and database server. After 2004, they were migrated to the French Tier 1 site of the Centre de Calcul IN2P3 (CC-IN2P3), Lyon on a cluster of four physical machines. This architecture had some limitations: i) when a server reaches the end of its maintenance period, it is rapidly removed from the cluster and a new server has to be purchased; ii) the hardware tends to become inhomogeneous.

4.2. Advantages of the cloud

The limitations exposed in the previous paragraph are attenuated in a cloud environment. Using virtual machines guarantees uniformity and replacing a physical machine does not affect services. Furthermore, costs are lower for virtual machines than for physical machines, especially in a cloud environment. For these reasons, we have collaborated with the CC-IN2P3 in order to set up AMI as the first external service deployed on the cloud computing center.

4.3. OpenStack and JClouds

The cloud technology available at CC-IN2P3 is based on OpenStack [14], a free and open-source cloud computing software platform. It makes it possible to manage large pools of computers, storage and networking resources in a datacenter. OpenStack has a modular architecture with various components. For example, NOVA is the cloud computing fabric controller. It is the main part of the Infrastructure as a Service (IaaS) model. It is designed to automatically manage a pool of resources and is able to work with most virtualization technologies. To interact with the cloud, Apache JClouds [15], an open source multi-cloud toolkit for the Java platform, was chosen. It gives full control to the AMI framework.

4.4. The AMI infrastructure

It was decided to migrate both development and production servers to the cloud, see figure 5. AMI currently uses a pool of ten virtual machines that can be instantiated in a range of 30 fixed IP addresses. Each virtual machine is built from an image containing controlled versions of the operating system (SL6), of the required middleware and of AMI itself.

This infrastructure makes it possible for us to test upgrades of systems as well as upgrades of applications in an automatic way. Those operations are controlled by the integration server which is itself in the cloud using the JClouds based software.

Currently, there are four Tomcat/AMI nodes with load balancing, two AMI task servers (used to fill databases), one Tomcat/Jenkins integration server and two development servers. In case of spike of important load on the AMI task servers, new nodes can be deployed on demand.

An advanced monitoring service (CPU, memory, network) will be put in place in order to better size server flavors (CPU, memory) and the number of server instances.

5. Outlook

All the AMI web interfaces have to be migrated to the new AMI web framework. This migration is almost over for Tag Collector and the AMI-Tags interface. The core of AMI is being rewritten. There is no structural change compared to what is described in section 2. This rewriting can be seen as a synthesis of ten years of development. The new preliminary core code is open-source and is available on GitHub [16]. Since 2013 another experiment, nEDM [17], is using AMI for run bookkeeping. Agreement was reached with the SuperNEMO [18] collaboration for the use of AMI, and at least one other experiment has expressed interest. We will work on making AMI much easier for experiments to set up databases with a minimum of support from the core team.

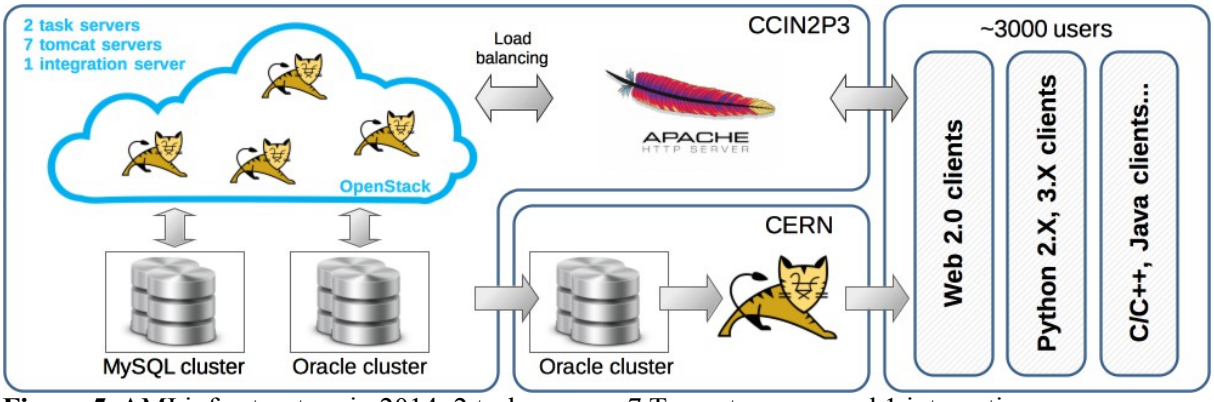

Figure 5. AMI infrastructure in 2014: 2 task servers, 7 Tomcat servers and 1 integration server.

Acknowledgements
Over the years we have been helped and supported by many people in the ATLAS collaboration, and at the CC-IN2P3. In particular Philippe Cheynet, Benoît Delaunay, Noel Dawe, Markus Elsing, Pierre-Etienne Macchi, Yannick Perret, Emil Obreshkov, Mattieu Puel, David Quarrie 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] J Odier, O Aidel, S Albrand, J Fulachier, F Lambert 2015 Evolution of the architecture of the ATLAS Metadata Interface (AMI). Proceedings of the 21st International Conference on Computing in High Energy and Nuclear Physics (CHEP2015) *J. Phys.: Conf. Ser.*
[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] JDBC: http://www.oracle.com/technetwork/java/overview-141217.html
[7] Tomcat JDBC Connection Pool: https://tomcat.apache.org/tomcat-8.0-doc/jdbc-pool.html
[8] Tomcat: https://tomcat.apache.org/
[9] pyAMI sources: https://pypi.python.org/pypi/pyAMI_core/
[10] JQuery: http://jquery.com/
[11] Twitter Bootstrap: http://getbootstrap.com/
[12] JSON: http://json.org/
[13] D3.js: http://d3js.org/
[14] OpenStack: http://www.openstack.org/
[15] Apache JClouds: http://jclouds.apache.org/
[16] AMI sources: https://github.com/ami-lpsc/
[17] The nEDM Experiment: http://nedm.web.psi.ch/
[18] The SuperNemo Experiment: http://nemo.in2p3.fr/nemow3/