Broadcasting dynamic metadata content to external web pages using AMI (ATLAS Metadata Interface) embeddable components

F Lambert[1], J Odier[1], J Fulachier[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: fabian.lambert@lpsc.in2p3.fr

Abstract. AMI (ATLAS Metadata Interface) is a generic ecosystem for metadata aggregation, transformation and cataloguing. Often, it is interesting to share up-to-date metadata with other content services such as wikis. Here, the cross-domain solution implemented in the AMI Web Framework is described: a system of embeddable controls, communicating with the central AMI service and based on the AJAX and CORS technologies. The main available controls and their basic usage are also described.

ATL-SOFT-PROC-2018-036
26 November 2018

1. Introduction

This paper describes the AMI Web Framework (AWF), a JavaScript Web framework part of the ATLAS Metadata Interface (AMI) ecosystem, designed to develop custom, dynamic and embeddable Web application interfaces connected to the central AMI Web service. AMI [1][2] is an ecosystem designed for metadata bookkeeping and mainly used by the ATLAS experiment [3] at the Large Hadron Collider (LHC). Thousands of physicists use several AMI tools daily which are parts of the offline software of the ATLAS experiment, in order to display or manage their data from SQL or NoSQL databases. As an example, the AMI dataset discovery tool is used to catalog the official ATLAS datasets available for analysis. It has been the official dataset bookkeeping tool of the experiment since 2006. As another example, the AMI-Tags tool permits setting up the configuration parameters used by data processing and analysis jobs.

AMI is a mature ecosystem, since its basic architecture has been maintained for over fifteen years. The ecosystem continuously scales both hardware and software infrastructure so that the quality of service remains high [4]. Nevertheless, several requests made by the ATLAS physicist community, concerning either data sharing on various supports like wikis or totally new Web applications, required a complete redesign of the ecosystem interface subsystem.

The AWF provides mechanisms for building controls and applications within an object-oriented paradigm. In the following, the main components of the AMI ecosystem are recalled. Then, the AWF subsystems are described, especially the embeddable control mechanism. Finally the benefits of the technical choices are discussed in terms of flexibility and scalability.

2. The AMI ecosystem overview

The AMI architecture is described in detail in other publications [1][5][6]. In this section, the main characteristics of the AMI ecosystem are recalled and the new control paradigm for Web applications is described.

2.1. Overview of the ecosystem architecture

The first AMI ecosystem architecture principles were designed between 2002 and 2003. In order to withstand the inevitable evolutions in technology and requirements, a n-tier architecture (Figure 1) was chosen. The layers of the n-tier architecture are well decoupled, making AMI a scalable and flexible ecosystem for developing metadata-oriented applications.

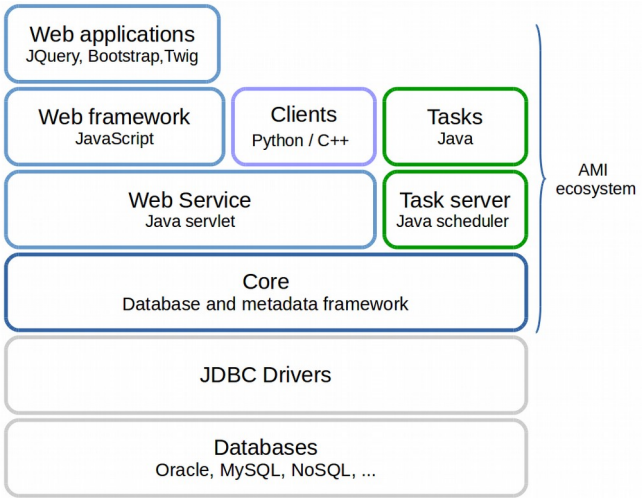

Figure 1. AMI multilayered architecture.

The Java AMI "Core" layer provides an API to interact with data sources. This layer provides a Metadata Querying Langage (MQL) to easily query the registered database catalogues. It uses the JDBC driver reflexion properties to provide end-applications a simplified view of the databases, hiding their internal structures.

Three categories of software use the AMI "Core" layer. The first category consists of the AMI task subsystem. It runs Java tasks and, for the ATLAS collaboration, is used to collect production metadata and to populate AMI catalogs. It has to be deployed along with the Java Core on our task servers. The second category is a set of lightweight clients which can be deployed independently of the Java core on the user computers, just having to be configured to point to an AMI Web Service endpoint. The last category consists of Web applications based on the AMI Web Framework. It is described in the following sections.

2.2. Evolution of Web applications in AMI

The first AMI web application prototype (Tag Collector) was written in 2001 with the PHP [7] server-side scripting language. However, due to PHP design patterns, the code for the business logic and for the HTML user interfaces were mixed together, making the whole system difficult to maintain. To address this concern, a first Java layered architecture was designed. It was composed of a Java "core" layer for the business logic and another one, using the XSL Transformation (XSLT) technology [8], for the HTML interface generation. Nevertheless, although the two layers were separated, making application evolution easier, they remained strongly coupled and were both processed server-side, affecting the whole system performance. To improve applications performance, from 2013, the XSLT layer was progressively replaced by modern Web 2.0 technologies.

2.3. AMI Web Framework

The AMI Web framework (AWF) layer is based on JavaScript, the Twitter Bootstrap [9] CSS framework, the JQuery library [10] and on a proprietary JavaScript implementation of the Twig template engine [11]. AWF permits loading and releasing arbitrary resources (like CSS, JS, JSON, …), AMI graphical controls and applications. Both controls and applications follow a Model-View-Controller (MVC) pattern (described section 3.3) in order to simplify developments by decoupling data model, visualisation and application specific code. AWF uses Asynchronous JavaScript And XML (AJAX) for interacting with the AMI server via a unique Java servlet in a secured and authenticated session. This servlet is configured to permit requests using cross-origin resource sharing (CORS) mechanism. The idea is to shift the processing of the HTML Web interfaces to the client, keeping only the business logic on the server. It gives AWF components the capability to access the AMI service from everywhere (external pages, personal pages, wikis, … see Figure 2).

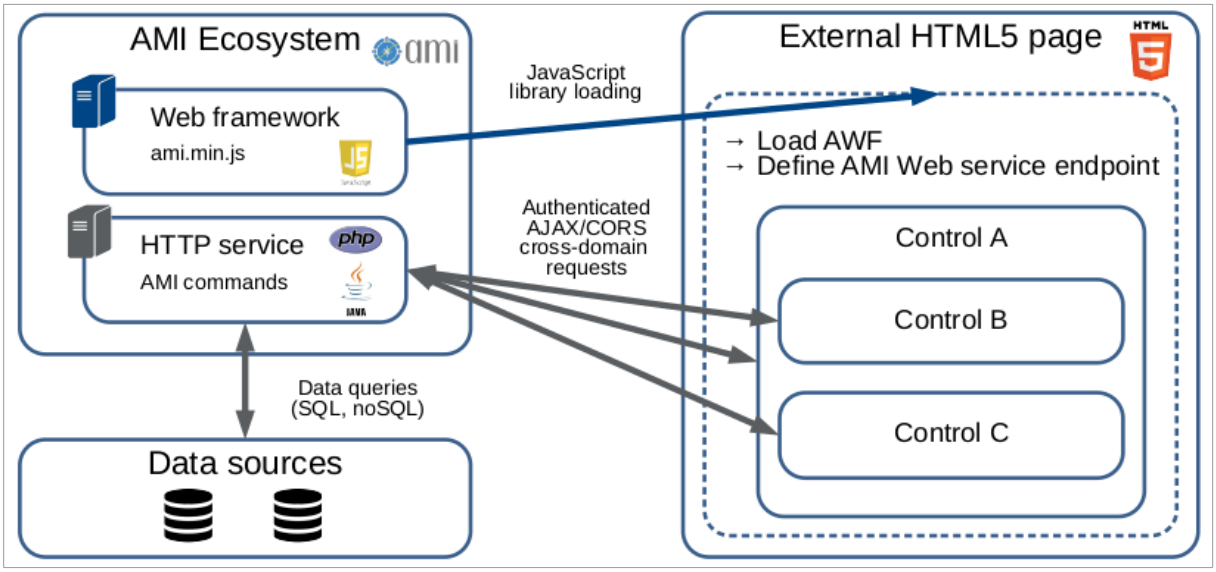

Figure 2. AWF controls embedded in external Web page

3. AWF controls

Controls are standalone components that can be associated to build a Web application or another composite control. The principles of the AWF control are described in this section.

3.1. Setting up AWF

To create a new Web application using AWF controls, a developer just has to pull the code from the official AWF GIT repository and execute few commands to generate basic control or sub-application skeletons.

```
# Setting up the AWF

curl https://rawgit.com/ami-team/AMIWebFramework/master/tools/build.xml > build.xml
ant setup-prod
ant make-home-page

# Creating a new control skeleton

ant make-control

# Creating a new sub-application skeleton

ant make-subapp
```

Figure 3. AWF environment setup

The commands, displayed in Figure 3, generate the following directories and files:

- *index.html*, the home page starting controls and/or applications
- *controls*, the directory containing the controls
- *subapps*, the directory containing the sub-applications
- *js/jquery.min.js*, the JQuery library
- *js/ami.min.js*, the AWF library

3.2. Embeddable control mechanism

There are two prerequisites to embed an AWF JavaScript control in an external Web page (for example a Wiki page). First, loading the JQuery library, the only dependency of AWF, then loading AWF library itself (Figure 4).

```
<head>
<script type="text/javascript" src="https://ami.in2p3.fr/app/js/jquery.min.js"></script>
<script type="text/javascript" src="https://ami.in2p3.fr/app/js/ami.min.js?embedded"></script>
…
</head>
```

Figure 4. Loading the AWF

Also, in the generated skeleton, two handler methods of the *amiWebApp* namespace object are overridden:

- *amiWebApp.onReady* is called as soon as AWF is initialized and ready to execute the user code. Within this method, miscellaneous resources (css, js, twig files, AMI controls, ...) needed by the main application are loaded using the AWF method *amiWebApp.loadResources* (Figure 5).

```
amiWebApp.onReady = function() {

    var result = new $.Deferred();

    amiWebApp.loadResources(['ctrl:table']).done(function(controls){
        myTable = new controls[0];
        result.resolve();

    }).fail(function(){
        result.reject();
    });

    return result
};
```

Figure 5. Loading AWF “table” graphical control

- *amiWebApp.onRefresh* is called each time at sign-in / sign-out or when user authorization changes. This method has one argument, *isAuth,* a boolean variable inherited from AWF that indicates whether the user is authenticated or not. Then specific code, usually for rendering one or more graphical components in an HTML division, can be executed (Figure 6).

```
amiWebApp.onRefresh = function(isAuth) {

    if(isAuth)
    {
        var result = new $.Deferred();

        myTable.render('#myDiv',{
            parameter1: paramvalue1,
            parameter2: paramValue2,
            ...}).always(function(){

                result.resolve();
            });

        return result;
    }
    else
    {
        $('#myDiv').empty();
    };
```

Figure 6. Rendering AWF "table" graphical control

Finally the method *amiWebApp.start* is called for starting the AWF (Figure 7) Web application. This function has an optional parameter to specify the endpoint URL of the AMI Web service.

```
amiWebApp.start({
    endpoint_url: 'https://ami.in2p3.fr/AMI/servlet/...'
});
```

Figure 7. Starting the AWF

3.3. Control pattern structure

AWF makes it possible to develop applications and controls with an MVC pattern.

3.3.1. Model

The *Model* is based on the CORS cross-domain AMI Web Service. It consists of a set of business objects called AMI commands, server-side. These commands permit adding/removing/updating (and much more) records in databases and can be called either by Web applications or other clients like the official AMI python client pyAMI [10]. The AMI Web Service returns the command results to the client in a JSON message that can be then processed by the *Controller*.

3.3.2. View

The *View* is based on the AMI-Twig template engine [12] and the Twitter Bootstrap 4 CSS framework. It renders data provided by the *Controller and* dynamically generates HTML 5 fragments using Twig templates (Figure. 8).

```
{% for row in linkedElementRowset %}
    <tr>
        <td class="text-center">
            {{row | json_jspath('..field{.@name==="entity"}.$')[0] | e}}
        </td>
        <td class="text-center">
            {{row | json_jspath('..field{.@name==="count"}.$')[0] | e}} records
        </td>
    </tr>
{% endfor %}
```

Figure 8. AMI-Twig and JSPath in action!

3.3.3. Controller

The *Controller,* written in JavaScript, is the central part of an AWF control. It uses standard technologies such as AJAX (for executing AMI commands) and external JavaScript libraries such as JQuery, JSPath [13], … It communicates with the central AMI Web Service, sending on-demand "AMI command" requests, and getting corresponding responses as JSON messages. It holds primitives to process these JSON data and transmits them to the *View* in order to update the applications.

3.4. Embedding controls in a web page

Benefiting from the CORS cross-domain AJAX technology, one can embed a JavaScript code in a Web page to interact with the central AMI servlet (Figure 8). AWF requires few HTML containers to be defined within the <*body*> (Figure 9).

- A <*nav*><*ul*> structure with the id *ami_login_content* where is loaded the AWF authentication control.
- A <*div*> with the id *ami_alert_content* to display error, warning or information messages issued from AWF.
- And at last, additional <*div*>s are needed to display user controls.

```
<head>
    <script>
        var myTable;
        …
            myTable.render('#myDiv',…
        …
    </script>
</head>
<body>
    <nav class="navbar navbar-expand-lg navbar-light bg-light">
        <ul class="navbar-nav ml-auto" id="ami_login_content"></ul>
    </nav>
    <div id="ami_alert_content"></div>
    <div id="myDiv"></div>
</body>
```

Figure 9. Embeding AWF "table" graphical control in a Web page

4. AMI controls in action

4.1. The ATLAS PMG View for MC dataset

"PMG View" control is currently used by ATLAS physicists, embedded in the ATLAS collaboration twiki pages. It provides physicists with a customized view of their Monte-Carlo data, corresponding to their needs. It is a composite control made essentially with three other controls provided by AWF: "Accordion", "Tab" and "Table" (Figure 10).

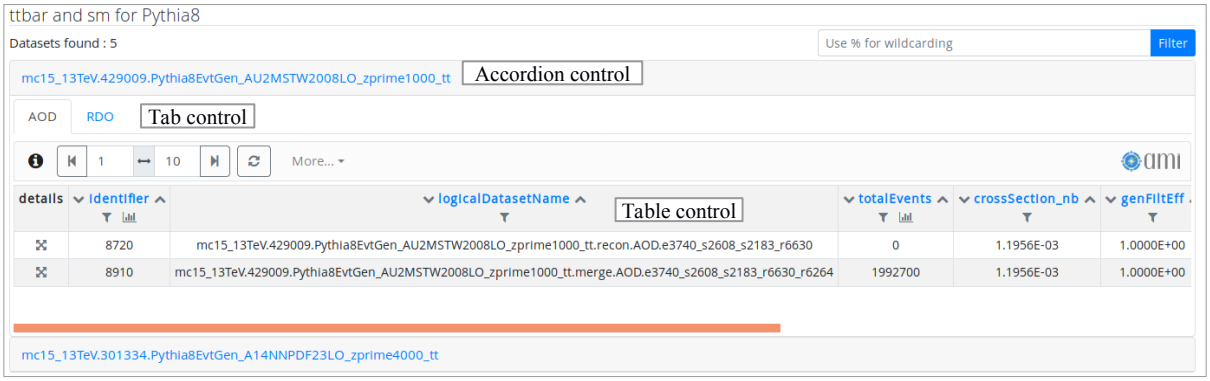

Figure 10. The "PMG view" composite control for ATLAS

4.2. The ATLAS AMI-Tags selection tool

Every day the ATLAS collaboration submit a huge number of analysis tasks on the worldwide LCG grid. AMI-Tags are used for storing software configuration parameters of these analysis and are an essential part of the ATLAS production system. The AMI-Tags selection tool is based on a generic search by criteria control. As a control, it can be embedded everywhere and permits, in this specific case, selecting an AMI-Tag (i.e. a set of configuration parameters) using a combination of dedicated search criteria (Figure 11).

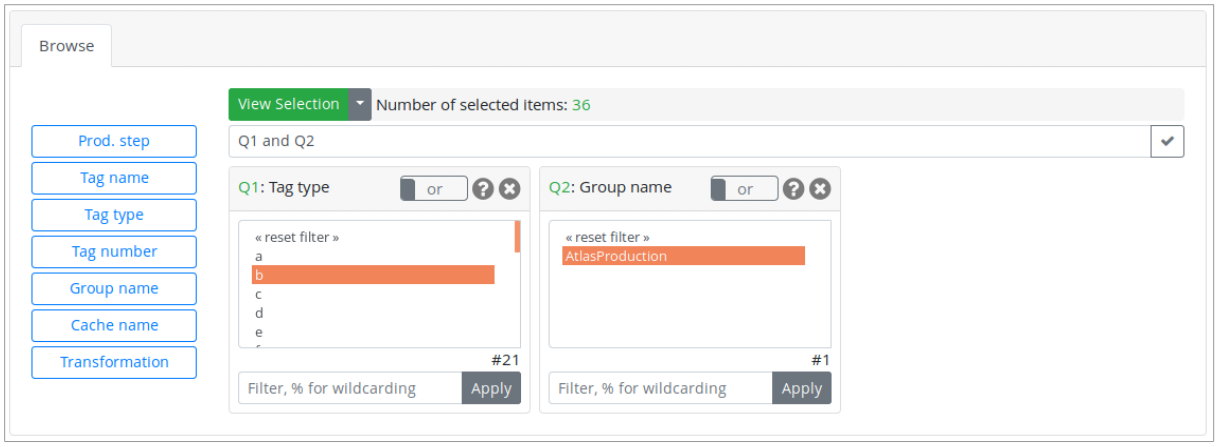

Figure 11."AMI-Tags selection tool" for ATLAS

4.3. The ATLAS dataset whiteboard

Metadata about official ATLAS production datasets, Monte-Carlo or Real Data, are stored in AMI catalogues. In 2017, physicists requested to be able to associate custom information to any dataset record, using a twitter-like hashtag system. The "ATLAS dataset whiteboard" is now integrated in the official ATLAS dataset selection tool, benefiting from the AWF control embeddable capability (Figure 12).

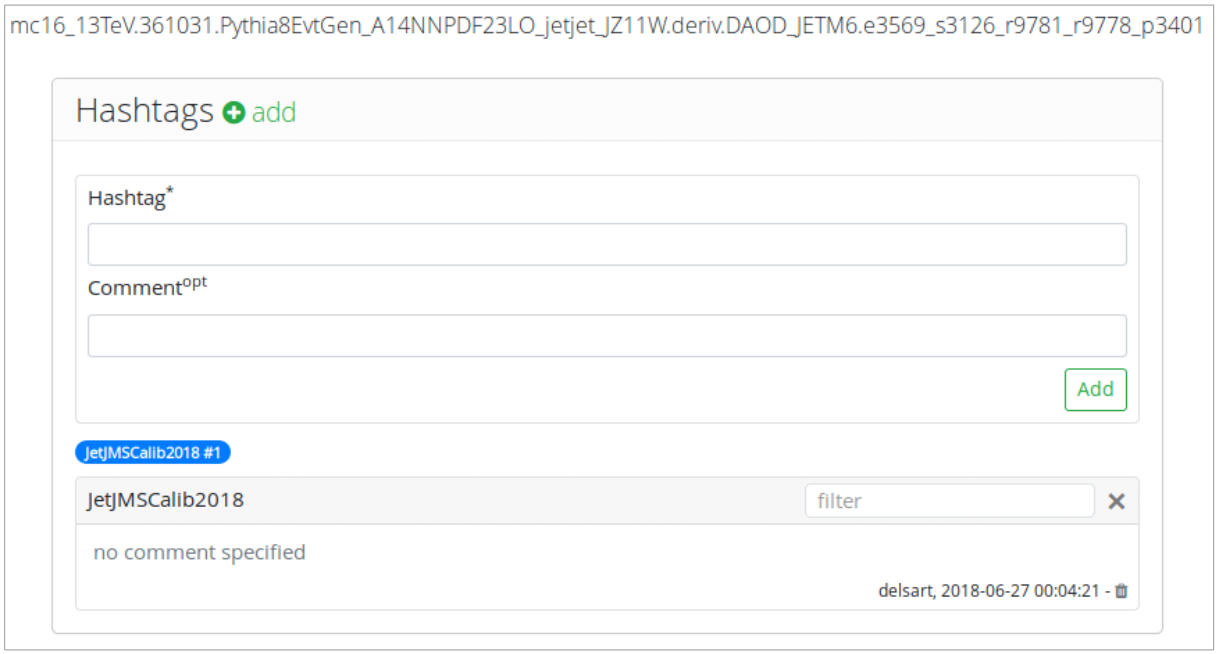

Figure 12."Dataset whiteboard" control for ATLAS

5. Outlook

AMI is a well-established and mature ecosystem, providing Web interfaces and applications to about 3000 active users of the ATLAS collaboration. All the AMI Web interfaces are being migrated to the new AMI Web Framework in order to benefit from its innovations. Although the rewriting of ATLAS production interfaces should be initiated carefully in order to ensure the continuity of service, it can be seen as a great improvement in terms of re-usability of graphical components. AWF is well documented [14] and can be used independently of the AMI core system. Since 2017, the Rosetta/Philae experiment [15][16], in the field of Astrophysics, has expressed interest in AWF, starting to develop their own controls with a minimum of support of the AMI Team. The team will still work on improving AWF and adding new generic controls, in order to provide a complete set of ready-to-use components to experiments.

Acknowledgements

Over the years the authors have been helped and supported by many people in the Rosetta/Philae collaboration, and at the CC-IN2P3, in particular Osman Aidel, Philippe Cheynet, Benoît Delaunay, Pierre-Etienne Macchi, Mattieu Puel, Yves Rogez, Mélodie Roudaud and Jean-René Rouet.

References

[1] Odier J, Fulachier J, Lambert F 2018 The AMI (ATLAS Metadata Interface) 2.0 metadata ecosystem: new design principles and features Proceedings of the 23rd International Conference on Computing in High Energy and Nuclear Physics (CHEP2018) *J. Phys.: Conf. Ser.*

[2] Odier J, Aidel O, Albrand S, Fulachier J, Lambert F 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.* **64** 042040 doi:10.1088/1742-6596/898/9/092001

[3] ATLAS Collaboration 2008 The ATLAS Experiment at the CERN Large Hadron Collider *JINST* **3** S08003 doi:10.1088/1748-0221/3/08/S08003

[4] Odier J, Aidel O, Albrand S, Fulachier J, Lambert F 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.* **664** 062044 doi:10.1088/1742-6596/664/6/062044

[5] Fulachier J, Aidel O, Albrand S, Lambert F 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

[6] Albrand S, Doherty T, Fulachier J, Lambert F 2008 The ATLAS Metadata Interface *J. Phys.: Conf. Ser.* **119** 072003 doi:10.1088/1742-6596/119/7/072003

[7] PHP : http://www.php.net

[8] XSLT: https://www.w3.org/TR/xslt/all/

[9] Twitter Bootstrap: http://getbootstrap.com/

[10] JQuery: http://jquery.com/

[11] pyAMI sources: https://pypi.python.org/pypi/pyAMI_core/

[12] AMI-Twig.js: http://ami.web.cern.ch/ami/twig/

[13] JSPath: https://github.com/dfilatov/jspath

[14] AWF API: https://ami.in2p3.fr/app/?subapp=document&userdata=api.html

[15] Rosetta/Philae experiment: http://rosetta.esa.int/

[16] Rosetta/Philae experiment: https://rosetta.jpl.nasa.gov/