GLAS-PPE/2005-19

UNIVERSITY of GLASGOW

Department of Physics & Astronomy
Experimental Particle Physics Group
Kelvin Building, University of Glasgow
Glasgow, G12 8QQ, Scotland
Telephone: ++44 (0)141 339 8855 Fax: +44 (0)141 330 5881

Development of Web Service Based Security Components for the ATLAS Metadata Interface

T.Doherty
Department of Physics and Astronomy
University of Glasgow, Scotland
Email: t.doherty@physics.gla.ac.uk

Abstract

This document provides a description of the development and implementation of five grid interfaces as a wrapper to the ATLAS Metadata Interface (AMI). It reveals the importance of metadata in the context of grid middleware; the concepts of datasets and logical files and how AMI allows access to and storage of this dataset metadata. Also included in this document is an analysis on how the existing AMI software was re-used to develop these grid interfaces and how these interfaces were made secure.

1 Introduction

'gLite' is the next generation middleware for grid computing. Born as part of the EGEE (Enabling Grids for E-science in Europe) Project it provides a framework for building grid applications, tapping into the power of distributed computing and storage resources across the Internet. The EGEE project aims to provide researchers in academia and industry with access to major computing resources, independent of their geographic location. The LHC Computing Grid (LCG) project has been selected to guide the implementation and certify the performance and functionality of this evolving infrastructure.

This document endeavours to summarise the implementation of the requirements contracted to AMI of five gLite metadata interfaces. These interfaces namely MetadataBase, MetadaCatalog, ServiceBase, FASBase and MetadaSchema deal with certain cases of use on dataset (and logical files) metadata (defined later) by particle physicists and project administrators working on the ATLAS experiment.

The implementation of the MetadataBase interface will allow AMI users to get, set, clear, list or query metadata for datasets. The methods in the MetadataCatalog allow for the creation or deletion of datasets and logical files. The ServiceBase interface allows for interface and schema versions to be retrieved. MetadataSchema is an interface that allows a user with administration privileges to add, remove, describe and list schemas to a database as well as adding and removing attributes (fields) to and from these schemas. FASBase is an interface that ensures that the privileges needed to use the metadataSchema methods can be checked, retrieved and set. There are three possibilities for clients that allow an AMI user interface with the AMI core software and these interfaces. From a Web Service client, from a browser using the AMI web search page and by installing the AMI core software on the client side. The development of a web service client for each of the implemented interfaces is also within the scope of this document along with securing these web services with the use of grid certificates.

For more details and background, see "Development of Web Service Based Security Components for the ATLAS Metadata Interface" [11].

2 Middleware, datasets, metadata and AMI

2.1 Middleware

The key to success of Grid computing is the development of the ‘middleware’, the software that organises and integrates the disparate computational facilities belonging to the Grid. Its main role is to automate all the machine-to-machine negotiations required to interlace the computing and storage resources and the network into a single, seamless computational fabric [4].

A key ingredient for the middleware is metadata. This is essentially “data about data”. Metadata play a crucial role as they contain all information about, for example, how, when and by whom a particular set of data was collected, where in the world it is stored and what it is useful for.

2.2 AMI

2.2.1 Introduction

In a Grid-like system the concept of a file (logical file), a block of data treated as a unit by file systems, need not concern a user. Instead, a user can consider collections of data, which are conceptually linked and unencumbered by restrictions from lower-level systems (on size, for example), while transparent middleware handles the files themselves. These collections are called 'datasets' [1]. They are related because a ‘dataset’ is a user-defined set of data gathered by a detector. It could be data sorted per project for example. A ‘logical file’ is a block of this data that is treated as a unit by a file system in a grid. A logical file can therefore be a subset of a dataset.
The ATLAS Metadata Interface is an application, as stated, which stores and allows access to this crucial dataset metadata for the ATLAS experiment.
It provides a set of generic tools for managing database applications. It has a three-tier architecture with a core that supports a connection to any RDBMS using JDBC and SQL. The middle layer assumes that the databases have an AMI compliant self-describing structure. It provides a generic web interface and a generic command line interface [2]. The principle use of AMI is the ATLAS data challenge dataset bookkeeping catalogues such as DC1 and DC2. Tag Collector is a tool for software release management and ExTra (External Traffic Analyser) is a system administration tool. Both are used in Grenoble where the core of AMI was developed.

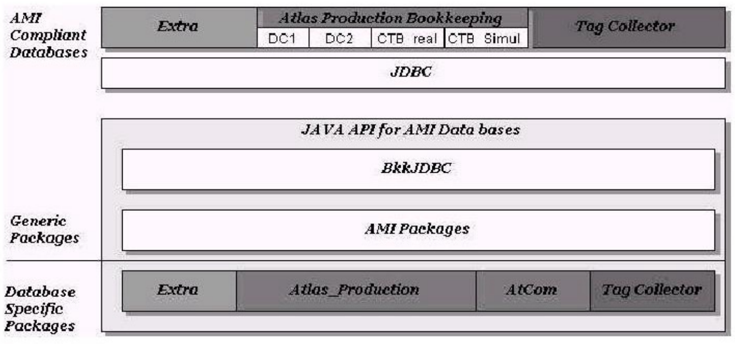

Figure 1: A Schematic View of the Software Architecture of AMI [2]. This diagram shows the AMI Compliant Databases as the top layer. This interfaces with the lowest software layer, which is JDBC. The middle layer BkkJDBC package allows for connection to both MySQL and Oracle. The generic packages contain command classes which are used in managing the databases. Application specific software in the outer layer can include the generic web search pages.

2.2.2 Three Layer Design

Figure 1 shows a schematic view of the architecture of AMI. The lowest software layer is JDBC, which handles the database connections. Only three databases are shown in the figure, but currently 6 projects use the AMI base classes. The middle layer contains two parts. BkkJDBC is a package that wraps the basic SQL requests. It contains light RDBMS specific plugin modules. The middle layer also contains packages that manage the AMI compliant databases in a generic way. The outer software layer has application specific software. The application specific software in most cases consists of a specialised web page, built on the generic functions [2].

2.2.3 The AMI Deployment Model

Figure 2 shows the deployment of the AMI compliant databases on different servers. The client software connects firstly to a router database, which is a mechanism for the redirection of client connections to an application database. No AMI software should ever assume that a particular application database will be stored on a particular server. Application databases may be distributed geographically, and may be running with different RDBMS. Client software only needs to be configured for a connection to the router database, and addresses the databases with an application semantic. In this way a database schema can be updated in a transparent way for the user. The client keeps open a connection to the router database, and a number of connections to different databases as required [2].

An AMI compliant database means a database that works with AMI must have a certain number of "information" tables. These tables are namely db_fields, db_elements and db_model. The table db_model is the most important as it describes the relations between the tables contained in the databases. The db_field table contains a list of all the fields of the AMI database it describes [3]. They play the part of the databases 'self describing' structure.

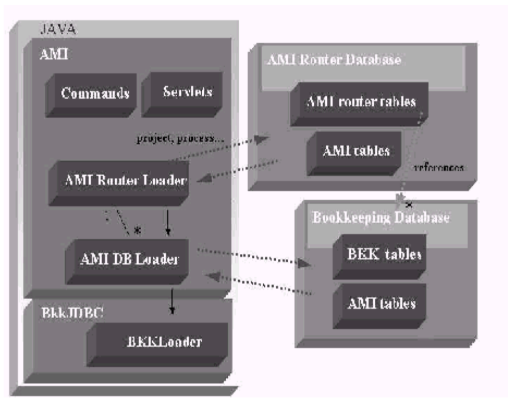

Figure 2: Redirection of Database Connections [2]. This diagram shows how the client connects first to a router database (through commands or servlets) and is then redirected to an application database. This means the client only needs to be configured for a connection to the router database.

2.2.4 AMI Clients

As previously mentioned the AMI core software can be used in a client server model as shown in Figure 3. There are three possibilities for the client:

- A Web Services client (SOAP) – a discussion on the development of this client is covered later.
- From a browser (HTTP) using the AMI web search page – currently in AMI this has been developed to the level where users can list datasets (or their component logical files) that fit a certain search criteria.
- ·By installing the AMI core software on the client side.

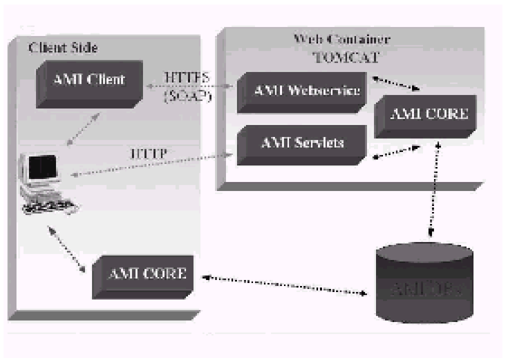

Figure 3: The AMI client possibilities [2]. This diagram illustrates how AMI can be accessed via SOAP from the client side if it is set up as a web service whilst deployed as

a web application in Tomcat. It can also be accessed via a browser using http and servlets. For this configuration it must also be deployed in Tomcat. Installing AMI on the client side is also possible. For each of these client server configurations AMI is also connected to a database.

3 Design

3.1 Requirements capture and Use Cases

There are many common use cases that are fundamental to the end-user (a physicist) when using Grid-like computing for experiments that imply requirements on the metadata catalogues used. These use cases can be divided into three categories and are summarised below:

- Dataset Handling –use cases related to the creation and access of data and metadata.
- Analysis –use cases dealing specifically with the needs of physics analysis.
- Job Handling – use cases for submitting, querying and controlling jobs submitted to a computing resource [1].

The gLite interfaces also act as a requirements specification document and on inspection indicate that the use cases can be categorised primarily under 'Dataset Handling'. The relevant use cases are:
(a) Specify a new dataset – which is equivalent to the createEntry method of the MetadataCatalog interface.
(b) Read metadata for datasets, update metadata for datasets and access data in a dataset – refer to the get/list, set/clear and query methods of the MetadataBase interface.

The system boundary is already clearly defined and is best visualised by looking at figure 3 which deals with AMI client possibilities. The actors to this system are project administrators with responsibility for the design and evolution of the project schema along with control over user privileges and particle physicists who will be using metadata to manage their project data.

The gLite interfaces, which provide the basis of the design for this implementation were supplied in Java interface code form. This code was reverse engineered into UML [9]using Rational Rose (a UML modeling toolkit) to better visualise the exact methods to be implemented and how each interface is related .

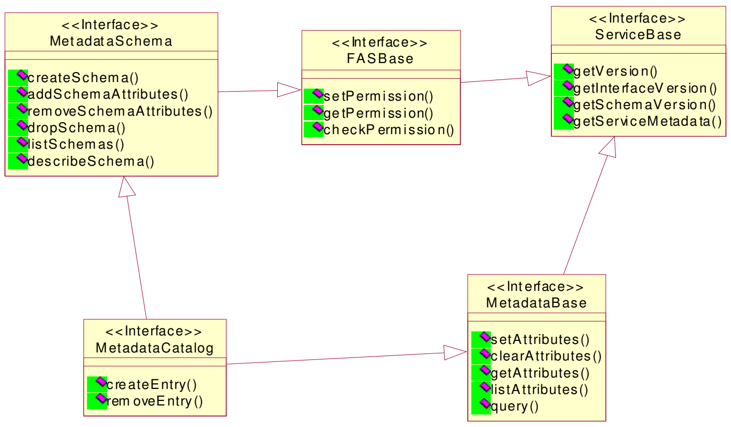

Figure 4: UML Class diagram of gLite Interfaces. Note how Metadatacatalog inherits from the MetadataBase, MetadataSchema and FASBase interfaces.

4 Implementation

4.1 Fundamental Architecture of AMI

Within AMI there are generic packages (see figure 1), which constitute the middle layer of its three-tier architecture. Command classes can be found within these packages. These classes are key to the implementation of the methods in each of the interfaces. The implemented gLite interfaces are therefore situated on the server side in this middle layer and directly interface with the client tier and the command classes in this middle layer. For this project the client is a web service client. The two other possible client are a generic web interface and a generic command line interface. The command line interface in particular has been implemented and each possible command follows a specified format. The first step in implementation was to become familiar with each of the AMI commands and their possible combinations of parameters.

Following from this it is possible to choose a corresponding AMI command that is equivalent to the basic requirements of each of the gLite Interface methods [6]. An example would be the setAttributes method of the metadataBase interface. UpdateDataset is the necessary command for this method as it allows a user to change or set attributes for a particular dataset.

The procedure used to further understand the structure necessary to implement the gLite methods was to observe how AMI is designed to absorb commands into its middle tier mechanism. This was achieved by mapping the delegation of methods through the relevant code and is best illustrated with the use of an UML sequence diagram.

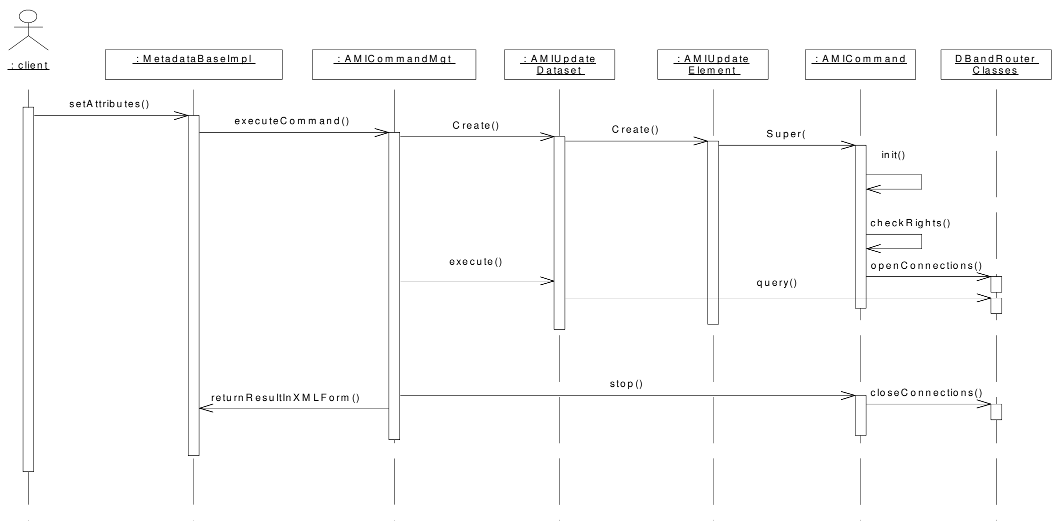

Figure 5: UML sequence diagram of basic workings of AMI.

From this diagram the following is apparent:

(1) There is a controller class which delegates what command class is invoked by extracting the name of the command from the list of parameters. It is called AMICommandMgt.

(2) The correct command class is instantiated with the remaining necessary parameters. In this case AMIUpdateDataset and therefore its parent class AMIUpdateElement.

(3) Every command is a subclass of AMICommand and calls its constructor (super) when invoked. This has a number of crucial features:

- The constructor calls an initialisation method, which firstly reads configuration details from a config file and overrides these parameters with whatever is found in the command line. These configuration parameters include driver names (connection to either a Oracle or MySQL database can be chosen here), username and password or an endPoint to be used with web services. This config file must be called at run time.
- It also deals with the core code that is specific to each RDBMS (Oracle and MySQL) and a router loader is instantiated to connect to the database.
- Each command ultimately returns XML output. This class also builds part of that XML output that is common to all commands including a timestamp and confirmation if the connection to the database was successful.
- It authorizes the user if a user grid certificate is being used by the checkRights method.

(4) The command management class then executes the correct command, if the parameters are correct and a successful connection to the database is possible. This will result in a query to the database and the result being converted to XML, to be returned with the rest of the generic XML output message.

Following from this investigation the structure of a gLite method will take the basic form of a method that:

(A) builds the parameter arguments as a HashMap (this being the parameter type accepted by the executeCommand method of AMICommandMgt);

(B) if the command is successfully executed the resulting XML can be displayed. If particular data is to be returned by the gLite method then it can be parsed and extracted from this XML;

(C) if the execution is unsuccessful then the correct exception must be thrown.

The FasBase interface does not follow this solution and is therefore dealt with separately below.

4.2.1 <u>FasBase</u>

<u>Introduction</u>

This interface is concerned with the (authorization) permission rights an AMI user has available to them. These rights were restricted to read and write access permissions. These permissions are of importance because certain users should only be allowed read access to metadata for example while a project administrator should be allowed to create or remove project schema or metadata.

To understand how this interface was implemented it is necessary to illustrate how user roles are defined within a database. This is achieved with the help of an ER diagram.

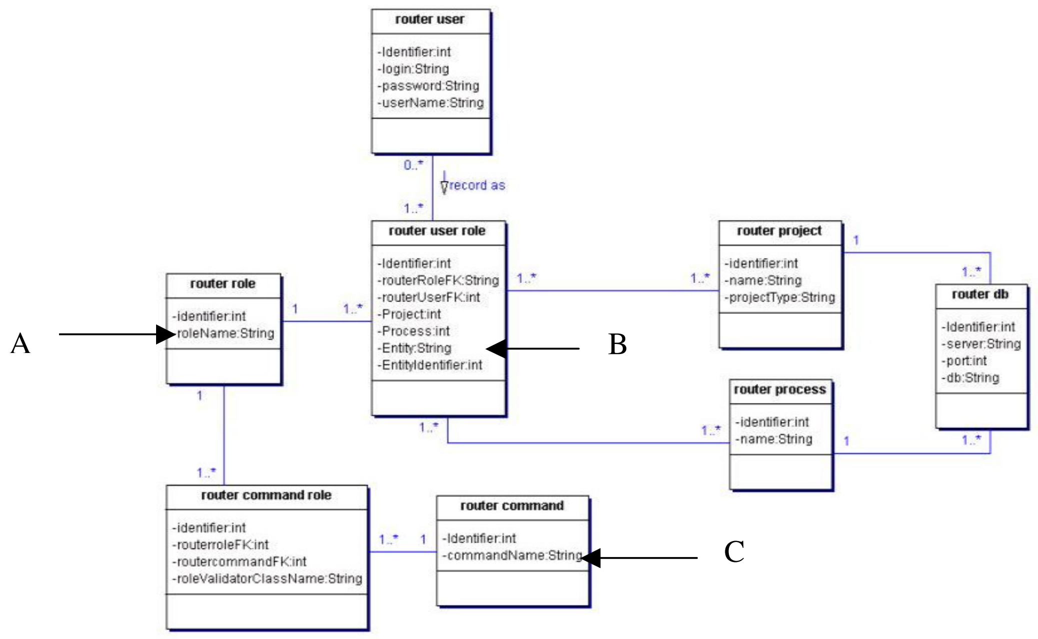

Figure 6: ER diagram of AMI router database role relationships [3]. A, B and C show tables(and specific fields) necessary for the role privilege mechanism.

All of the tables in the ER diagram are present in an AMI Router database. The table *router role* allows for different roles to be set up within the database. This could be an administrator role or a basic AMI user role. The roleName at (A) on the diagram is used to make this distinction. A users relationship with their role can also be set up in a *router user role* table. This table defines among other attributes what entities (dataset and logical file) a user is allowed to access. This is identified at point (B) on the diagram. Finally another table that is of use to permissions is *router command*, which defines the commands a certain role is allowed to use. Which can be seen at point (C). This information is key because if a role is allowed to use update or creation commands then it has write access and if it is only allowed to list then it has read-only rights.

AMI commands are available to access users role, entity rights and command rights. The AMI command listUserRole for example lists the roles a user has assigned to it. Each of them can be implemented using the generic structure for AMI commands Extracting a users role, entity rights and command rights means the three FASBase methods can then be implemented.

4.3 SAX Parsing

It was stated earlier (in section 4.1) that the structure of an implemented gLite method would not only include the setting and execution of command parameters but also the parsing of the XML output from the command if some form of information had to be returned by the method.

SAX was chosen in preference to DOM, as the parsing method because it can easily scan and parse the XML output from an AMI command without hitting resources. As different XML tags in the XML output are focused on for each command this event-based approach means they can be handled in a different way quite easily[8].

5 Deployment and Client Development

AMI is not a stand-alone application. It is a web application that exists in a three-tier environment. Set up as in figure 3. The deployment of AMI as a web application in a web container can take place when Tomcat is setup. This allows for further development with web services, which is discussed below.

5.1Web Service Clients

SOAP

SOAP is an XML-based communication protocol and encoding format for inter-application communication. SOAP is widely viewed as the backbone to a new generation of cross-platform cross-language distributed computing applications, termed Web Services [7].

<u>Axis</u>

Axis is essentially a SOAP engine --a framework for constructing SOAP processors such as clients, servers, gateways, etc.
But Axis isn't just a SOAP engine since it also includes:

- a server which plugs into servlet engines such as Tomcat.
- extensive support for the Web Service Description Language (WSDL).
- emitter tooling that generates Java classes from WSDL called WSDL2Java.
- a tool for monitoring TCP/IP packets called TCPMon which allows for the interception of SOAP messages by listening to the relevant port.

5.1.1 <u>Client set up</u>

To set up web services for AMI it is necessary to plug the Axis framework into Tomcat. Then with the use of WSDL and the axis tools that allow conversion from WSDL to Java client classes a Java web service client test class can be deployed. See appendices 1 for more detail.

5.1.2 <u>Securing web service clients</u>

When does a web service become a grid service? A grid service is just a sub-class of web services, but which give access to the sort of computing power that Grids enable. To gain this computing power then resources must be shared. A direct consequence of this is secure access. There is an issue of trust when it comes to sharing resources. This involves authentication and authorisation of users and machines. Authorisation was dealt with in the role-based mechanism for the FASBase implementation. Authentication is implemented by securing the web services using grid certificates. More generally in conforming to standards for the methods in this case the gLite 'standard' the methods are part of a standards hierarchy based on web services. See appendices 2 for more detail.

<u>6 Conclusion And Future Work</u>

This document ultimately provides a description of the task of implementing the gLite Interfaces for AMI. It explained how AMI was set up fully with these implementation classes interfacing with web service clients and how these clients were made secure with the aid of grid certificates.

To achieve this the initial step required an understanding as to why AMI actually exists. As AMI is an existing application its architecture and deployment was explored and understood. This resulted in an improved knowledge of the inner workings of AMI and the possible cases of use that a physicist or administrator would have for AMI as an application. It was then possible to see how the gLite interfaces could be used as solutions to the metadata use cases. By mapping the delegation of methods originating from a command control class within AMI it was possible to design a structure for the gLite implementation classes that interface with it. AMI was deployed as a web application using Tomcat (a web application container). Axis was plugged into Tomcat to be used as a SOAP engine to enable the use of web services and its tools helped develop the web service client (with automatically generated classes). Finally, investigating how to configure Tomcat to use secure connections and obtaining the relevant web (grid) certificates made the communication between the web service client and AMI secure.

AMI as an application will continue to undergo development. The following list outlines future areas of work in AMI. At the moment the web interface only allows physics metadata to be read at the 'Phyicist' role level. Future development will allow metadata to be added/updated as well as read at the 'Admin' role level. Such security mechanisms as servlet filters and axis handlers will enable this by restricting access to certain web resources.
Currently permissions in AMI are based on a local role system. An EGEE wide role system called Virtual Organizations Membership Service (VOMS) [10] is being developed. AMI would then have to be set up to read and understand VOMS attributes and grant permissions based on a user's role in ATLAS. Implementation of certificate delegation is also necessary which may also be handled in VOMS. This means AMI will be able to contact another service on behalf of a user. This will be important in the future because, in the current ATLAS computing model, AMI will handle datasets only with file information being held in a separate service. Delegation will allow AMI to contact this service to get information on the files constituting the dataset that the user is considering. File and dataset information will then be presented to users through one coherent interface.
More directly relevant to the gLite interface is the creation of a query language parser for performing cascaded searches through all projects and processes.

Bibliography

[1] Unlucky for some, The Thirteen Core Use Cases of HEP Metadata, *Steven Hanlon.*
[2] ATLAS Metadata Interfaces (AMI) and ATLAS Metadata Catalogs, *Solveig Albrand, Jerome Fulachier, LPSC Grenoble*
[3] The AMI Dev Wiki –
http://ATLASbkk1.in2p3.fr:8180/AMI_PUBLIC_WIKI/jsp/Wiki
[4] Middleware – http://gridcafe.web.cern.ch/gridcafe/gridatwork/middleware.html
[5] Developer's Guide for the gLite EGEE Middleware –
http://edms.cern.ch/document/468700
[6] ATLAS Metadata Interface User Guide, *Solveig Albrand, Jerome Fulachier, LPSC Grenoble.*
[7] Axis User's Guide – http://ws.apache.org/axis/axis/Java/user-guide.pdf
[8] XML Parsers; DOM and SAX Put to the Test –
http://www.devx.com/xml/Article/16922
[9] Unified Modelling Language – http://www.uml.org
-public/lhc_computing_challenge/lhc_computing_challenge_in_numbers.html
[10] VOMs – http://hep-project-grid-scg.web.cern.ch/hep-project-grid-scg/voms.html
[11] Development of Web Service Based Security Components for the ATLAS Metadata Interface – Masters of Science (Information Technology) Thesis, Glasgow University, 2005. - http://ppewww.ph.gla.ac.uk/~tdoherty/MScThesis/MScThesis.pdf

Appendices

1 Web Service Setup

Note: The following instructions assume AMI has been checked out and built into a directory named AMI.

- Place implementation classes in to AMI/WEB-INF/classes
- Edit AMI/WEB-INF/server-config.wsdd.
- First create a skeleton wsdd file for each service to be published. This will result in the implementation classes appearing on the axis page *'View the list of deployed Web services'*. There will be a link to the wsdl for this class which can be saved to file. This wsdl can be converted to Java files using wsdl2Java tool provided by axis. A deploy.wsdd file is created with these files which hold the XML typemappings (corresponding to serialiser/deserialisers for the complex types used in the methods. For example an attribute array) needed in server-config.wsdd for each service.
- With the server-config.wsdd complete – again use the wsdl2Java tool to create the proxies needed for the web service client. The client side classes that are created include an interface class, a soapbindingstub class, a service (interface) class, and a service locator class.
- To create a web service client and make a call to a method created in the web service. For example:

```
MetadataBaseImplService service = new MetadataBaseImplServiceLocator();
MetadataBaseImpl catalog;
catalog = service.getMetadataBaseImpl();
System.out.println(catalog.getInterfaceVersion());
```

Note how the method getInterfaceVersion is called. By instantiating a service locator object and using it to get the soapBindingStub instance for MetadataBaseImpl i.e. 'catalog' . This is then used to call the methods on the web service.

2 Grid certificate setup to secure web services

To secure web services we need implementations of point-to-point security using SSL based on Grid certificates, authentication using Grid certificates, and authorisation using the DN (Distinguished Name) on a user's certificate.
This is achieved as follows:

- Obtain a certificate by following the instructions on http://ca.grid-support.ac.uk and download it to export it from your browser. (note there is a different instruction set per browser)
- At http://www.grid-support.ac.uk/ca/openssl.htm find the instructions to use openssl to change the format of the .p12 file the browser gives.

A DN looks as follows:
`CN=thomasdoherty,L=Compserv,OU=Glasgow,O=eScience,C=UK`

In Java, The gLite TrustManager is based on the EDG Trustmanager. It is a replacement for the SSL implementations which are supplied with web containers and application servers.

- Make sure to change the service endpoint to https and port 8443. This can be set in the AMI configuration file.

- To be able to use the certificate in your web service client and secure it. This code must go into the client to set the Java vm system properties to point to the certificate and key files (converted earlier using openssl).

```
System.setProperty("axis.socketSecureFactory",
"org.glite.security.trustmanager.axis.AXISSocketFactory");
System.setProperty("sslCAFiles", "/etc/grid-security/certificates/*.0");
```

- Finally, The users DN must be set up in their AMI user account so that they can be authorised to use AMI by the AMI command AMIGetUserFromDN (used in listUserAttributesFromDN method on page 40) You need a column 'DN' added to the *router_user* table in your router database for this to work.

Note: Every time a user executes a command their rights are checked by the checkrights method (see figure 5). This also checks if the router database it is working with has the 'DN' column in the *router_user* table. If it doesn't, it can't authorise by certificate.