



D3.4: Metadata Catalogue Web Service Design

Project Title: QCDGrid2

Document Title: D3.4: Metadata Catalogue Web Service Design

Document Identifier: QCDGRID_MDCWS_DESIGN

Document Filename: QCDGrid2-MDCWS-Design.doc

Distribution Classification: Commercial In Confidence

Authorship: Daragh J. Byrne

Approval List: QCDgrid Development Team

Distribution List: UKQCD Collaboration

Document History:

<i>Personnel</i>	<i>Date</i>	<i>Summary</i>	<i>Version</i>
DJB	31 st January 2006	First release	1.0

Contents

1	Introduction	2
1.1	The QCDgrid project.....	2
1.2	Document purpose	2
2	Design goals	3
3	Current QCDgrid components.....	3
3.1	Existing code.....	3
3.2	Existing technical architecture	4
3.3	Potentially useful re-factorings.....	4
4	Proposed architecture	4
4.1	Technical architecture.....	4
4.1.1	Metadata web service	4
4.1.2	Overall architecture	5
5	Low level design	5
5.1	Main interfaces and classes	6
5.2	MetadataCatalogueService class	6
5.2.1	Deployment within Axis	6
5.2.2	Configuration	7
5.2.3	Starting the service	7
5.2.4	doEnsembleQuery method.....	7
5.2.5	doEnsembleURIQuery method	7
5.2.6	doConfigurationQuery method	7
5.2.7	doConfigurationLFNQuery method	7
5.2.8	getConfigurationMetadata method.....	7
5.2.9	getEnsembleMetadata method	7
5.2.10	getMDCInfo method.....	7
5.2.11	executeQuery.....	8
5.3	Simple command line query client	8
	References	9

1 Introduction

1.1 The QCDgrid project

The UKQCD Collaboration aims to “procure and jointly exploit computing facilities for lattice field theory calculations, whose primary aim is to increase the predictive power of the Standard Model of elementary particle interactions through numerical simulation of Quantum Chromodynamics”. Such numerical simulations produce significant amounts of data in the form of binary files. The purpose of the QCDgrid project is to provide a software application and supporting infrastructure that simplifies the management, storage and manipulation of this data.

In the first three years of the project (2002 – 2004), software engineers at EPCC developed a software application called *QCDgrid* – a data management system that combines the distributed resources of the collaborators into a robust facility called the *UKQCD Grid*. The result is a multi-terabyte storage facility over six UK sites at: Edinburgh (including the University of Edinburgh Advanced Computing Facility), Liverpool, RAL, Southampton, and Swansea. Glasgow is also a member of the consortium.

The facility is based on commodity hardware and open-source software. The hardware consists primarily of high specification PC-based servers running the Linux operating system and managing large RAID storage arrays. On top of this infrastructure, the QCDgrid software (built with Globus Toolkit 2.4, EGEE, and an XML Database Server (XDS)) provides *Datagrid* management and user functionality – furnishing a simple and intuitive environment that hides the complexities of the underlying grid and presents a standard file system to the user. It incorporates a robustness metric that automatically disperses datasets across the grid, providing a resilience that ensures data is not affected by the loss of one (or possibly more) storage nodes.

QCDgrid allows the user to query and manipulate associated metadata using a *Metadata Catalogue Browser*. The software also provides a *Job Submission System* that allows a user to schedule computations on remote HPC systems, from the comfort of their desktop computer. Security is leveraged from the Globus Toolkit, based on digital certificates issued by the UK e-Science Certificate Authority. The result is a reliable, secure data management system.

Looking to the future, the collaboration aims to integrate the UKQCD Grid with similar activities in the International Lattice Data Grid (ILDG), allowing like-minded scientists around the world to share their data and benefit from the scientific progress of other groups. The ILDG has proposed an XML web-services based architecture, allowing each collaboration to expose their data management capabilities in a uniform manner.

1.2 Document purpose

One of the core components of the ILDG architecture is the metadata catalogue web service. This web service will allow ILDG members to search the metadata catalogue of each collaboration. The specification of this web service is addressed in [2].

The purpose of this document is to outline the design of the UKQCD implementation of the metadata catalogue web service. The document will discuss:

- Design goals that must be realised by the eventual design;
- Current UKQCD software components that will be of use;
- The infrastructure needed to implement and host the web service;
- The components necessary to implement the web service, and their interactions.

2 Design goals

The primary purpose of the metadata catalogue web service is to handle XPath queries received from the web via SOAP messages. The specification for the exact operations supported by the web service has been discussed elsewhere [2]. The queries must then be passed to an existing XML database server. The queries should be responded to in a timely manner.

The main goals of this design exercise are:

- To examine the best means of hosting the QCDgrid web service component;
- To explore how the web service layer and database server may best be integrated;
- To identify mechanisms to handle any web service layer configuration;
- To identify areas of design uncertainty and suggest ways of dealing with them.

3 Current QCDgrid components

3.1 Existing code

The architecture of the current QCDgrid system is illustrated in Figure 3.1.

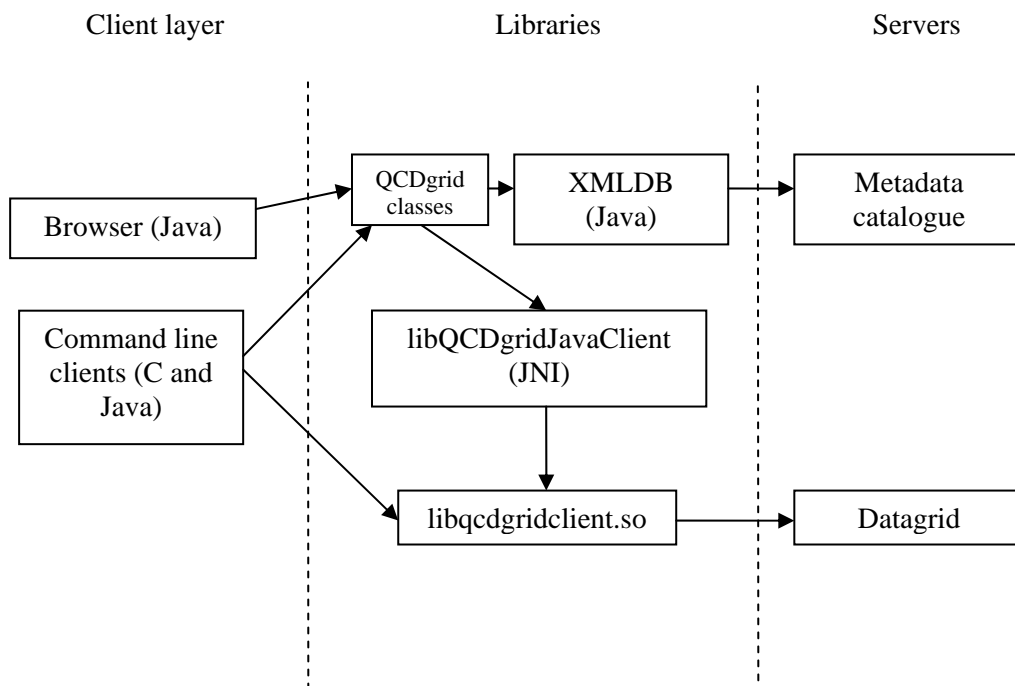


Figure 3.1: QCDgrid architecture, pre-Work Package 3.

One of the major components is a set of database access classes (QCDgrid classes) that were written for the QCDgrid metadata browser application. These classes are also used by the `put-file-on-qcdgrid` command line tool. These classes allow the database to be accessed, updated and searched. The most important classes are described below.

ExistDatabaseDriver

This class abstracts the XML database server. The most important method is `runQuery`, which accepts an XPath query as a string, returning the results as an instance of `QueryResults` (essentially an array of strings). It uses the XMLDB `XPathQueryService` class, supplied with `eXist` (the XML database that is used by UKQCD Grid to host the metadata catalogue), to query the database. The version of `eXist` presently used by the UKQCD Grid is Version 0.91. The database

server to be used can be specified by using the `init` method. This class is derived from the `DatabaseDriver` base class, which defines each of the methods mentioned here.

QCDgridMetadataClient

This class provides a more abstract API for working with the metadata catalogue. It contains methods for submitting metadata documents to the database, removing metadata files, querying whether particular files exist, and a few other utility methods. It uses `ExistDatabaseDriver` to communicate with the database. The database server to be used can be specified in the constructor.

EnsembleNameExtractor, QCDgridLogicalNameExtractor

These classes offer methods for extracting LFNs or ensemble URIs from metadata documents. It is likely that these will prove useful agents during the development of the web service layer.

QCDgridClient

This class provides Java access to the data grid using the QCDgrid C code and JNI, allowing files to be placed and retrieved from the data grid.

As can be seen from this analysis, the QCDgrid code exposes a domain-specific data access class (`QCDgridMetadataClient`) and a generic data access class (`ExistDatabaseDriver`).

3.2 Existing technical architecture

The eXist database runs as a web application within an instance of the Apache Tomcat web server. This currently runs on a machine called `edqcdgrid.epcc.ed.ac.uk`, an SMP machine running Red Hat Enterprise Linux (version 3).

3.3 Potentially useful re-factorings

The database access, GUI and metadata manipulation classes are just `.class` files at the moment. It would be useful to divide the classes into a set of JARs. Specifically, we could have a `QCDgridMetadata.jar` archive that contains the classes relevant to accessing the datagrid and metadata server, and a `QCDgridClient.jar` that contains all of the code used by the browser and command line tools. Alternatively, we could have a single `QCDgrid.jar`, suitable for use by both the web service and those using the client applications. On balance it is felt that having a single JAR is less troublesome than maintaining two separate JARs for maintenance reasons.

4 Proposed architecture

4.1 Technical architecture

Since we are already running an instance of Tomcat, and there is some experience within the project team of developing web services using Apache Axis, it is proposed that the metadata catalogue web service (and the other ILDG web services) be hosted under the existing Tomcat installation. This installation currently runs on the `edqcdgrid` machine; however, this machine is being replaced with a dedicated machine for running the database and the QCDgrid control node. The web services will run on this new machine when it is made available. The most recent version of Axis (version 1.3) will be used.

4.1.1 Metadata web service

The metadata web service will be SOAP based and will thus be based on Apache SOAP web services. A set of interfaces and types have been defined in Java by the ILDG (discussed below). ILDG will produce an up-to-date WSDL description file from these stubs which will form the contract for the service. Our metadata catalogue service will implement all of the operations defined by ILDG.

4.1.2 Overall architecture

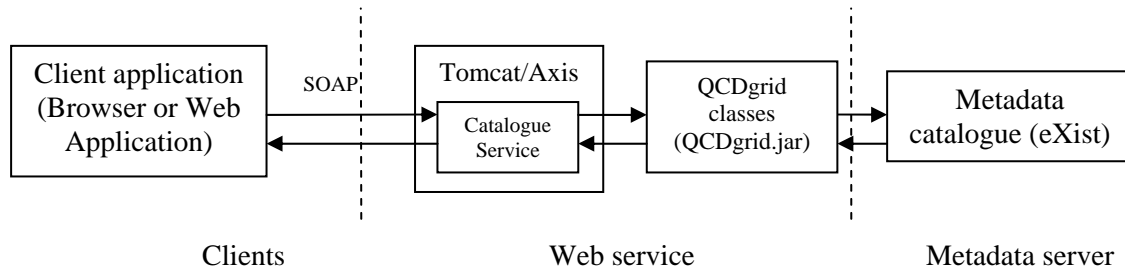


Figure 4.1: Proposed architecture for metadata catalogue service.

The proposed architecture for the metadata catalogue service is illustrated in Figure 4.1. Client applications (such as a new version of the metadata browser, or an ILDG web application) will communicate with the metadata catalogue web service via SOAP calls, using HTTP as the transport mechanism. The catalogue service is hosted in Tomcat using the Axis web application. Depending on which operation has been invoked, the service will issue a query to the metadata catalogue using the existing QCDgrid metadata access classes (either `ExistDatabaseDriver` or `QCDgridMetadataClient`). The results of the query will be parsed by the service if appropriate (for example, LFNs may be extracted). The service returns the parsed results to the client, using one of the classes defined by ILDG.

Instead of the catalogue service interacting directly with the QCDgrid classes, it may make sense for the service to interact with a query management layer. The query manager layer would be responsible for the following:

- Handling connection to the database service (configuration of the database URL etc could be done here);
- Caching query results – this could be added later if service performance becomes an issue, but initially will not be implemented.

An expansion of the above architecture to include the query manager would look like in Figure 4.2.

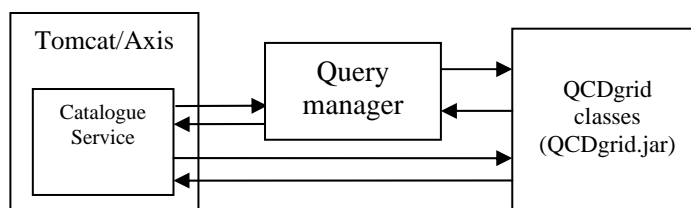


Figure 4.2: Extended architecture that incorporates a dedicated query manager component.

Of course there may still be times when the service needs to interact with the QCDgrid classes directly (for example, querying the existence of a file in the catalogue) – hence the corresponding arrows in Figure 4.2.

The Query Manager will not be implemented during this iteration of development, though is potentially significant in a subsequent phase.

5 Low level design

The ILDG have specified the operations to be implemented by the metadata catalogue web service. This section outlines the proposed design for our implementation of the web service.

5.1 Main interfaces and classes

The UML class diagram in Figure 5.1 illustrates the classes to be used and their relationships. The remainder of this section explains the classes in more detail.

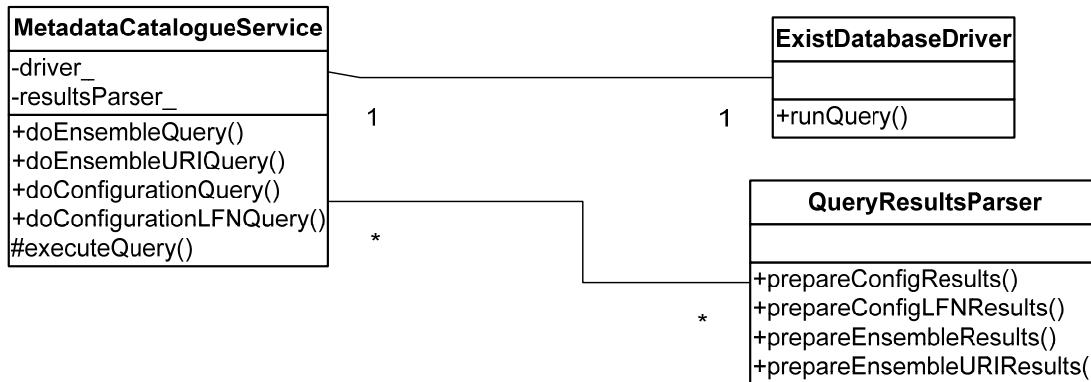


Figure 5.1: MDC classes and their relationship.

5.2 MetadataCatalogueService class

In Axis, any class may be deployed as a web service. This is normally achieved by editing the `server-config.wsdd` within the Axis web application, or by using a command line tool with a web service deployment descriptor file (an XML file containing configuration information about the service). Our metadata catalogue service will be implemented in the `MetadataCatalogueService` class. The responsibilities of this class are:

- Execute queries against the database (using an instance of `ExistDatabaseDriver`);
- Parse the results of the queries so that suitable data is returned to the client.

This class will interact with an instance of `ExistDatabaseDriver` to obtain query results from the database. It will also interact with an instance of `QueryResultsParser` (see below) to format the results correctly before they are returned to the client. In case a different database needs to be used in future, the type of the `driver_` reference will be the `DatabaseDriver` base class of `ExistDatabaseDriver`. The driver instance will be created using the `Class.forName` idiom. The instance will, in this case be of type `ExistDatabaseDriver`. The string indicating the class to be used will be indicated using the `<parameter>` Axis service parameter mechanism (see below).

5.2.1 Deployment within Axis

Axis allows web service classes to be deployed using one of three “scopes”. In request scope, a new instance of the object is created to service each and every request to the service (i.e. every SOAP message). In session scope, a new object is created for every HTTP session. In application scope, one object is created and used for the whole lifetime of the service. The scope chosen can affect the design of the object implementing the service, in particular with respect to how state is managed.

In request or session scope, any state used by the object must be loaded at the start of the request or the session, and stored at the end of the request or session if needed again. For example, any database connections will either have to be created or loaded from a connection pool during every request for a request scoped service. In application scope, a reference to the connection can be held in memory between requests.

The metadata catalogue web service is essentially stateless, as far as the client is concerned. The only allowed activities are lookups of data – no modification of the database is allowed. This can be achieved using a single instance of `ExistDatabaseDriver`. `ExistDatabaseDriver` uses the `eXist` class `XPathQueryService` to execute queries against the database. A review of the available

documentation for eXist would indicate no problem using a single instance of this class to execute multiple queries concurrently.

In the future, we may move to an architecture where all interaction with the metadata catalogue is carried out via the web service. Presumably, we would wish to add operations to write metadata to the database. These operations would have to be secured in some manner to prevent unauthorised writes to the database. The mechanism by which this could occur has not yet been examined in depth by the project team; however, a preliminary investigation indicates that it can be achieved in a manner that does not have implications on the design or deployment of this class.

5.2.2 Configuration

This class should be configurable with the URL of the database to be used. A potential mechanism for carrying out this configuration is by using the `<parameter>` element of the web service deployment descriptor that describes the web service.

5.2.3 Starting the service

The service will be started automatically when the Tomcat container that hosts it is started.

5.2.4 doEnsembleQuery method

This method executes an arbitrary XPath query against the collection of ensemble metadata documents (EMDs). The returned results are expected to be fragments of EMDs, returned as an array of strings.

5.2.5 doEnsembleURIQuery method

This returns the ensemble URIs of ensembles matching the query. There is little practical difference between this method and the one above except that the raw query results must be parsed to extract the ensemble URIs. These URIs are then returned as an array of strings. The `markovChainURI` element within the document currently contains this information.

5.2.6 doConfigurationQuery method

This operation returns the results of arbitrary queries against the collection of configuration metadata documents (CMDs). The returned results are expected to be fragments of CMDs, returned as an array of strings.

5.2.7 doConfigurationLFNQuery method

This method returns the logical file names (LFNs). Again, the query results must be parsed to extract the LFNs. Currently the LFNs are stored in the `dataLFN` element.

5.2.8 getConfigurationMetadata method

This method accepts a configuration LFN as a string and returns the complete configuration metadata document relating to that LFN from the database.

5.2.9 getEnsembleMetadata method

This method accepts an ensemble URI as a string and returns the complete ensemble metadata document relating to that URI from the database.

5.2.10 getMDCInfo method

This returns basic information about the metadata catalogue web service. The information is returned in a simple class (MDCInfo) having four public fields:

- String version; /* version supported in this MDC (currently 1.0)*/
- String queryTypes[]; /* supported query formats (currently "XPath")*/
- String groupName; /* group name running this MDC (currently "UKQCD")*/
- String groupURL; /* group info URL (unknown) */

5.2.11 executeQuery

This is a private method that simply executes the query and returns a `QueryResults` object. This method serves to avoid code duplication and to future proof against the way queries are executed (for example, the introduction of a query manager layer could happen in this method). In this implementation, it will simply execute the query using the private instance of `ExistDatabaseDriver` maintained by the class.

5.3 Simple command line query client

A simple command line client will be written to demonstrate and test the functionality of the service. The command line client will be implemented in the class `QCDgridMDWebServiceClient`. The client will be invoked as follows:

```
java uk.ac.ed.epcc.qcdgrid.metadata.webservice.client.QCDgridMDWebServiceClient  
<URL_of_service> <method_name> <XPath>
```

References

- [1] QCDgrid: Probing the building blocks of matter with the power of the Grid, QCDgrid Project homepage, available on-line at <http://www.gridpp.ac.uk/qcdgrid/>.
- [2] QCDgrid: [D3.3: Specification of the web service interface to the MDC \(Version 1\)](#) – QCDgrid Project report published on project web pages (November 2005).