Grant agreement no.211714

neuGrid

A GRID-BASED e-INFRASTRUCTURE FOR DATA ARCHIVING/ COMMUNICATION AND COMPUTATIONALLY INTENSIVE APPLICATIONS IN THE MEDICAL SCIENCES

Combination of Collaborative Project and Coordination and Support Action

Objective INFRA-2007-1.2.2 - Deployment of e-Infrastructures for scientific communities

Deliverable reference number and title: **D6.1 Distributed Medical Services Provision (Querying Service)**

Due date of deliverable: **Month 12**

Actual submission date: **31st January 2009**

Start date of project: **February 1st 2008** Duration: **36 months**

Organisation name of lead contractor for this deliverable: **University of the West of England, Bristol UK**

Revision: Version **1**

Project co-funded by the European Commission within the Seventh Framework Programme (2007-2013)		
Dissemination Level		
PU	Public	X
PP	Restricted to other programme participants (including the Commission Services)	
RE	Restricted to a group specified by the consortium (including the Commission Services)	
CO	Confidential, only for members of the consortium (including the Commission Services)	

Table of Contents

Intended Recipients

The WP6 workpackage entitled **"Distributed Medical Services Provision"** aims to design a group of *generic* services that can be used in a number of related medical applications. These will then be implemented in order to fulfil the neuGrid specific project requirements. The services will be built according to the design philosophy presented in the WP6 deliverable. This will help to enhance and promote their re-usability in other related applications.

This deliverable document presents a design philosophy that the generic services will follow, maps user requirements against suitable services and briefly presents a list of the services. An initial implementation of the services and their detailed API descriptions will be delivered in the year 2 deliverable.

The WP leaders, technical users and neuGrid developers are the intended recipients of this document. To a lesser extent, since indirectly concerned (through the natural abstraction of Workflow/ Pipeline authoring environments such as the ones proposed in WP6), neuro-scientists and prospective users (e.g. Pharmaceutical companies) as well as internal and external reviewers of the project activities, are anticipated as potential readers of this document.

9 The Querying Service

9.1 Introduction

The neuGrid project is centred around the provision of a research infrastructure for the analysis of complex medical data. Heterogeneous sources of relatively complex data are clearly present within clinical research studies and are therefore difficult to integrate. Integration is needed to ensure that researchers get the most relevant and subsequently, the best information for their research. The querying service will provide methods by which heterogeneous data can most efficiently be queried within neuGrid and hence assist users in their research. The primary emphasis, being on allowing users to query the data successfully, in the future this could progress into developing ways to assist the user i.e. adding semantics to give the querying service a level of intelligence. The data, despite being heterogeneous in nature could also be in many different formats. Examples of these include images, flat files, relational databases and XML to name just a few that the querying platform should be flexible enough to handle. This service will provide a choice of ways in which the user can query the data held in neuGrid.

The primary issue with this service is combining the heterogeneous sources of data together so that they can be queried as a single resource. Once this has been accomplished, work will be focused into including some semantics in order to enrich the querying offered to researchers to browse and access the distributed data resources. The fundamental way in which semantics are currently applied to aid existing systems' intelligence will be analyzed in the evaluation process. Current thoughts suggest that semantics will be employed to provide a very flexible query service offering lots of types of query mechanisms for the users.

The neuGrid project design philosophy identifies service orientated architecture as being the best choice for addressing the user requirements. It is clear that querying will play a central role in the system by providing mechanisms for accessing the distributed data resources which are inside the grid environment. This service will research and develop a platform independent and developer friendly querying service which can be customised and extended in the future. Local data can often be specific to individual institutions and therefore structured in different ways. Such data often comes in a variety of formats and it is challenging to query and integrate it. Rather than having a specific data-adaptor approach for each type of resource, it would be beneficial to use a service oriented approach to provide the level of abstraction and scalability that is necessary.

Ultimately, the goal is to create a synergy between the distributed querying service and semantic information. This is relevant to the service because in some cases researchers will type a query which in itself will return very specific results from heterogeneous data resources. The aim is to bridge gaps between distributed data resources using information contained within the querying service itself and to make inferences as the software is running thus adding to the services own knowledge base. It is this very bridging of data within separate data resources which could be of interest because that is something not easily done by a human with limited time.

It is important to remember the possible impact of such a service being implemented, it is reasoned that researchers may be less explicit in their searches and still receive relevant search results. This service should help to close the gap between distributed querying and semantic integration in order to provide richer results.

The following objectives of the querying service have been identified:

- Create a querying service which can query disparate data resources.
- Craft a solution which is platform independent and service oriented.
- Create a synergy between the querying of heterogeneous data resources and semantics, something which based on initial research, there are very few systems in existence accomplishing this.
- Look at semantics fundamentals and how they can be applied to aid the querying service and bridge gaps between data resources within the service to yield more useful results.

9.2 Requirements

To set the scene in terms of the user requirements, the main group of actors that have been identified for the service are clinicians (medical researchers). Clinicians are looking to analyse a cross section of MRI scans, but don't always have local access to the range of scans that are necessary to make a statistically meaningful analysis. Therefore, they need to interact with a querying service in order to identify a suitable sample from the data. This data may be distributed throughout the grid and held locally and must be matched with criteria supplied. Upon composition of a study set, the clinician will pass it through one or more image processing pipelines (referred to as workflows). It is at this stage that complex algorithms are executed on the study set thus performing the analysis.

The requirements are presented in two sections. In this first section, the user requirements are presented along with how a decision has been made that they should be a requirement. In the second section, some high level requirements are presented more in terms of the architecture and the software which will be produced along with justification. These querying service requirements are technical requirements which are derived from the user requirements.

9.2.1 User Requirements

In order to derive the following list of requirements, the user community has been actively consulted. This process has been vital in order to get an overview of how the querying model that will be developed will be used by the various actors that are present in the neuGrid project. The unified modeling language provides a use case diagram for displaying user requirements and in particular for identifying what should be functional requirements of the service and this is shown in figure. 53. [53]

There are two components within the service which should be explained at this stage:

- **Data** - The data as a whole. I.e. The heterogeneous data stored in the grid and local data which can be queried.
- **Data Set** - A subset of the data containing candidates of interest which may at some point be put through one or more workflows.

All of the requirements shown are essential unless otherwise stated.

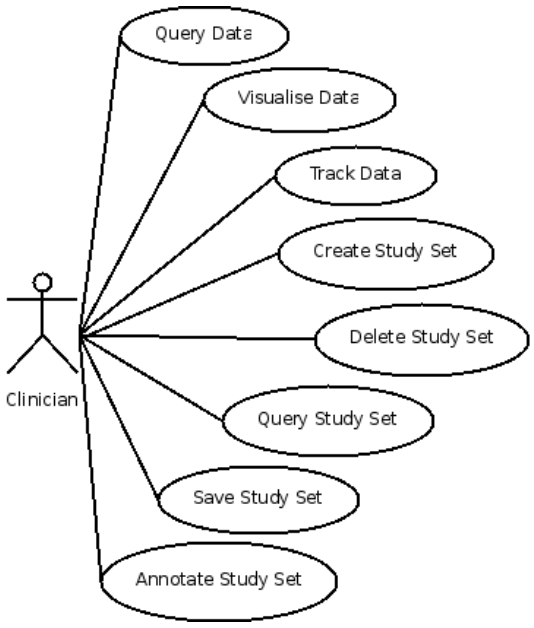

Figure 53. The Use Case Diagram

9.2.1.1 Query Heterogeneous Data

The user must be able to query heterogeneous data stored within the grid and locally by providing some search criteria. The heterogeneous nature of this data is what presents the problem. Such a service is necessary because browsing through the vast number of data records held in the grid could become cumbersome and waste researchers time. This requirement represents the essence of the service and could be considered the most important. The querying of data is not a problem if it is static and in one physical location, the nature of this data however is dynamic and heterogeneous which presents a problem. Therefore the design of the query service must take this into account.

9.2.1.2 Visualise Data (Desired Requirement)

It would be desirable, although not essential for users to be able to visualise data and data sets using visualisation tools integrated into the querying platform. Visualising data conjures up an image of graphs and charts, it need not be this complex. It could be something as simple as checking the data resources and providing users with the ability to select data from drop down boxes. This would provide interactive queries and mean that the user has to think less about interacting with the service through a specific querying language and more about the actual data they are looking to extract.

9.2.1.3 Track Data

The tracking of data is very useful and it will be necessary to hold provenance information. E.g. Changes to data sets. Information such as which workflows have been executed on which data sets and when will be very useful. This is especially important in the case that a researcher wishes to recreate a query. At present it is believed that a database will exist to contain this provenance information and the querying service will have to incorporate this data as and when it becomes

available. This data will hold lots of information containing users specific steps which could be traced at any point in order to recreate or to clarify the steps taken to produce a result.

9.2.1.4 Create Study Set

A required feature of the service will be for users to create study sets. These study sets will eventually be executed through one or more workflows. It will be important to examine the way in which these study sets are composed, since the service will need to interact with the workflow services. The interface must be standardised in order for the querying service to work together in harmony with the workflow services.

9.2.1.5 Delete Study Set

If a study set can be created, it should also be possible to delete it. At any particular time a researcher will want to focus on very specific data and it may not be useful to store lots of historical study sets within the main querying service (this will be the job of the provenance service).

9.2.1.6 Query Study Set

Sometimes it will be necessary for a user to query data held within a particular study set, therefore the feature to filter a search to show just results contained within one or more study sets will be a useful one.

9.2.1.7 Save Study Set (Optional Requirement)

The ability to save study sets could be very useful, particularly as a study set is being built. This requirement is related to allowing tracking of the data.

9.2.1.8 Annotate Study Set

The ability to annotate a study set is essential since users may wish to comment on individual pieces of stored data or data sets as a whole. This is necessary to provide useful information for future researchers considering some data for their own study set.

Based on the user requirements, the next two sections provide the results of initial research into providing a service which fulfils all of the requirements and is as maintainable and scalable as possible.

9.2.2 Functional Requirements

9.2.2.1 Corrupt Data Set Prevention

Occasionally within data sets, a piece of data in the grid may be deleted. In this case the study set may become corrupted. It will be necessary to provide some tools or a mechanism which acts when a scenario such as this occurs and attempts to recover the study set.

9.2.2.2 Access Control Strategies

The nature of the content held in neuGrid is personal and therefore very sensitive and access must be controlled to make sure that only authorised users can access the service and make changes to data stored within it.

9.2.2.3 Visualisation Tools

The integration of visualisation tools into the querying interface is desired functionality of the querying service.

9.2.2.4 Study Set Annotation Privileges

Researchers in one institute may wish to allow other researchers to annotate their study set to offer their valuable contribution. Similarly researchers may wish to prevent other people from commenting on their data set and in fact limit it to a number of specified individuals.

9.2.3 Non-Functional Requirements

9.2.3.1 Non-Technical Standardised Interface

The assumption is made that not all clinicians are Computer Scientists and as such the interface should be operable by anybody. Although the interface is not the central focus of this work, it will still be important to consider the interface because end-users will eventually interact with the service through such facilities.

9.2.3.2 Metadata for Images

Due to the graphical nature of some of the data being queried, metadata will have to be added and exploited to provide an efficient querying mechanism. Depending on the types of data and whether meta-data for that resource already exists this may be a large task or a fairly small one. This meta-data will provide the fuel that will drive the semantic element of the querying service, since it will thrive as more information describing data resources it is provided to it.

9.2.3.3 Metadata for Study Sets

Meta-data will need to be present for the study sets so that they can be queried as is set out in the user requirements. This could be implemented with something so simple as researchers being able to tag their study sets for easy retrieval at a later date.

9.2.3.4 Service Oriented Nature

The delivered software should be of a service oriented nature and available on the Internet as a web service so that the querying functionality is accessible through a standardised interface. This requirement is in line with the project being as scalable as possible. The aim is to provide a

"single composite service" encompassing various applications beneath. [54] This is a classic example of a application demanding a SOA type implementation since it provides the possibility to be accessible from anywhere by any type of client supporting the standardised interface.

9.2.3.5 Semantic Query Enrichment

The search should be made "intelligent" with the use of semantics adding meaning to the underlying data and allowing software to better understand the content. This is probably not something a user would ask for, but it is a way in which the service could be made to be very efficient which is something that a user would request.

The question of whether this is needed is an important one and of course given any searching mechanism, the less explicit a search needs to be and the more results it can yield, the better it is. It is thought that some knowledge of how to process queries can be held in the form of rules which could access a semantic knowledge base. It is hoped that resources can be bridged in this way and that more useful results are returned to the client.

9.2.3.6 Provide a Coherent Interface to the Heterogeneous Data

The data is split up and held in different physical databases in neuGrid as is the nature of grids but also could be stored locally. It is most useful and efficient for a layer of abstraction to be inserted so that the querying service can be built up without having to worry about how to interface each different data source. This will not only make it easy at this stage but it will make it much easier in future since developers will already have access to all of the data.

9.2.3.7 High Accessibility

The service must be accessible from various institutions around the globe. This will have to be taken into consideration when deploying the interface to the platform. This once again points towards the implementation of service oriented architecture.

9.2.3.8 High Availability

An inaccessible service due to hardware failure is not uncommon when a single instance of a server exists. It is hoped that an architecture can be chosen based on the possibility of scaling it up to provide a highly available service. This would introduce the issue of load balancing and the notion of a query cluster of servers.

9.3 Justification

9.3.1 Service Oriented Architecture

The implementation of the querying service in a SOA is justified in this section. Although the design philosophy has been referenced as detailing a SOA for the services provided. The argument can be further strengthened for the querying service in particular.

The SOA will provide an intermediary between the underlying data resources and the user. This intermediary is necessary to manipulate the query before it reaches the

underlying data resources. Whilst using a SOA it means that we remain agnostic with respect to the abstraction used to access these resources. This is also beneficial since the querying logic remains enveloped and separate within the service and not just placed within the data access abstraction. For this reason, the querying service should be scalable since the querying logic placed in the service can evolve alongside newer versions of the data access abstraction.

9.3.2 Intelligent Querying Platform

The justification for making semantics an integral part of this querying services architecture centres around several key points. The opening point is that clinicians will use this querying service from different cultures and backgrounds. The use of semantics can bridge these cultural differences in uniting different terminology to mean the same thing in the context of the query. Furthermore, semantics will provide heightened flexibility within the service. It becomes possible for the service to accept queries in different formats with different data. Semantics, in this case would be used to determine what data has been submitted and how it should be processed.

The service can be personalised when using semantics, for example, a knowledge base of the domain can be re-configured depending on the user and their specific needs. Offering such a customised service means that users can broaden or tighten query results as they wish. An example of this is if a clinician is referencing a cortex of the brain, which returned few results. A clinician could broaden the search to include surrounding cortices. Similarly, if a clinician searched for several cortices and they received a wealth of results, they could tighten it to include just one cortex. Holding this knowledge of the domain means that the service can be expanded in future by adding new logic in the web service to utilise the knowledge base in differing ways.

9.4 Architectural Models

Three possible models have been derived from the requirements and are analysed in this section. At this point, it is important to point out various terms which are used in the images which follow:

- DQS (Distributed Querying Service)

This represents the abstraction layer providing access to the underlying data resources. This is necessary to fulfil the requirement which says that the data should be accessed in a coherent way.

- DR (Data Resource)

Data resources already residing inside the grid or in fact local files of various types. At any time these resources could be removed and new resources could be added, this is something which needs to be considered when designing the service.

- Query Portal

Any client to the DQS and a way into the knowledge for example, a web interface. The word "portal" has become more of a computing term recently but its generic definition provides a good start. It is defined as "a doorway, gate, or gateway, especially a large and imposing one", the

portal which will be provided as part of the querying platform is a "gateway" to the data in the underlying resources. Essentially, the portal represents a standard interface through which all clients will access the querying service through due to the distributed nature of the data.

- Portlets

Very specific query application of which there would be several used to access the service. These can be positioned together within a query portal each fulfilling specific entry points to the service. Eventually these will map to the user requirements, with each portlet providing a cohesive entry point to the service.

- SM (Semantic Meta-data)

Some representation of knowledge and a way of reasoning within the querying logic in order to yield richer and more relevant results.

Three models are proposed:

- Centralised Meta-data
- Replicated Meta-data
- De-centralised Meta-data

9.4.1 Centralised Meta-data

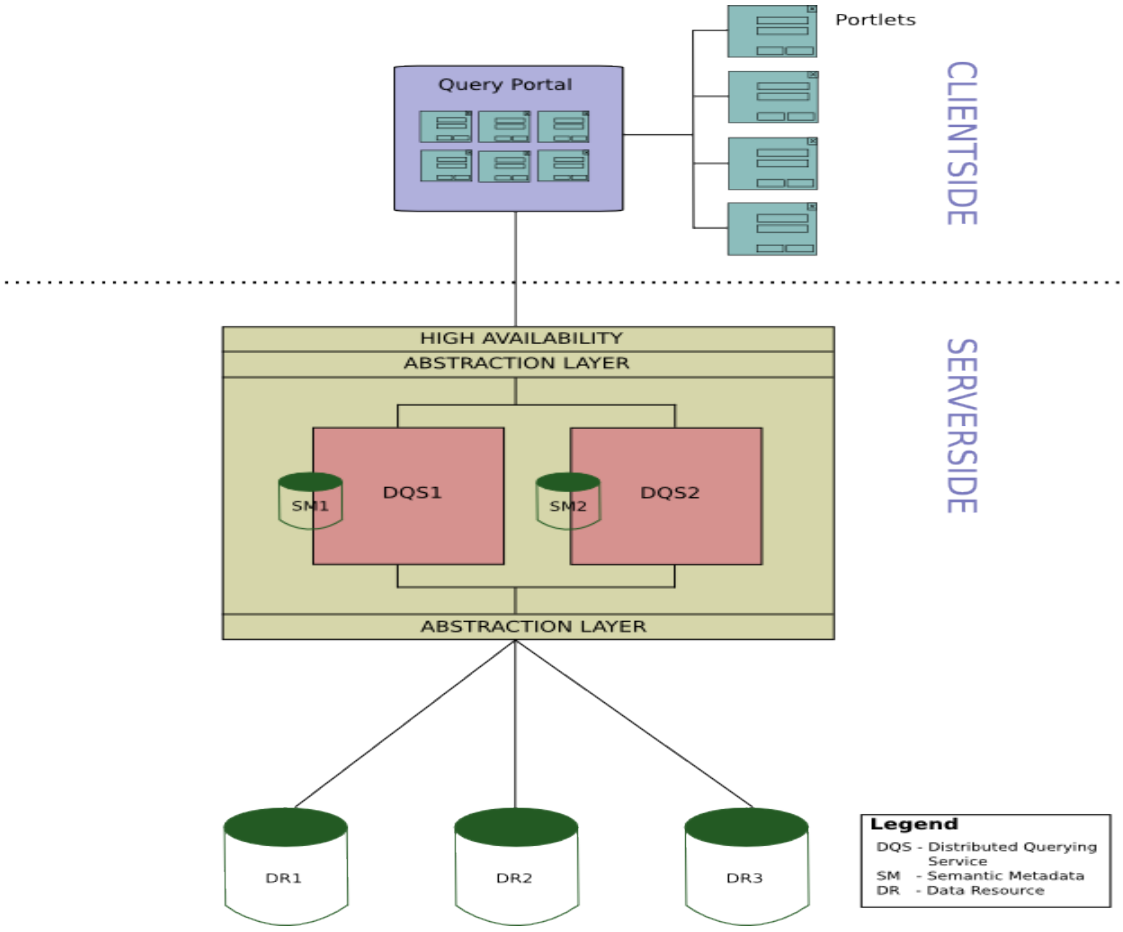

Figure 54: Centralised Meta-data

The model shown in figure 54 represents one that is highly available. Several instances are deployed simultaneously; this means that if one of the instances fails to respond initially, the client could automatically select a different DQS. The quality of service would be unaffected. The distributed querying service can be instantiated multiple times and each time it is deployed a local instance of the meta-data database is created. Queries could be more efficient with this architecture since each DQS holds a local instance of the meta-data. Backup would be fairly trivial in the case of this model since the data is instantiated in several places at any one time and a master copy could be kept somewhere and sequentially updated.

Scalability is a requirement that this model well fulfils. If querying is extremely popular and the quality of service falls below expectations, the DQS complete with an instance of the semantics could be deployed on another server. Therefore, making the data more available. The model can be modified to provide only one instance of the DQS and a single instance of the meta-data. This would make maintenance easier but the high availability would be compromised and the model would not be as scalable to allow more traffic. Load balancing could be implemented as a layer

above the querying service instances, with each service providing information to the load balancer based on their load and speed. The client could use this information to select the best choice (nearest querying service with the lowest load). Every time a query is submitted, it would first pass via the load balancer which would select the optimum querying service. The most simple way of implementing such a load balancer would likely contain a single point of failure. Clients could however, select a default querying service which is known to be available as a fall-back in the case that the load balancer is down, thus eliminating this issue.

In summary, the model offers the following advantages and poses the following issues.

Advantages:

- Well suited to an SOA design.
- Scalable.
- Straightforward to backup.
- High Availability

Disadvantages:

- Raises the issue of keeping the DQS and semantic instances up-to-date and consistent.
- Bandwidth is a valued resource with not all institutions having a great deal and la significant amount would be needed in this approach. This is because the data that is available at a particular institution is only available at that institution and no other, since the data is truly distributed in this architecture. Institutions suffering from low bandwidth would be queried with each query that enters the querying service.

9.4.2 Replicated Meta-data

In the replicated meta-data model that is represented in figure 55, there is multiple instances of the DQS server. This means that the high-level of availability required is delivered. Each individual data resource acts as a client to the DQS server with the meta-data being stored at the data resource level where a master/slave type approach has been adopted. Backup would be more straightforward with such an approach. This could be performed by taking snapshots of the Master meta-data store at intervals which could be rolled out and synchronised with each of the slave meta-data stores. The downside of this might be that there would be increased network traffic as databases update and synchronise with one another. Requests would come in from the server to the DQS client adaptor which would be situated at the same level as the data resource. The meta-data would be attached directly to this to enable queries to be translated using meta-data before being passed on to the actual data resource. This does introduce the question of maintenance and this approach certainly isn't the most scalable model represented in this section. If a new data resource of a different type is introduced into the service, it would be necessary to make some changes to the DQS client. This may include adding support for communication to and from the data resource and integrating this with the associated meta-data. The DQS client is used since this ultimately offers a layer of abstraction so that the server can communicate with all data resources in the same way. It brings with it several advantages; one of these is that the querying load is well balanced since the meta-data is accessed locally at the data resource. This could provide a quicker and ultimately better quality of service overall.

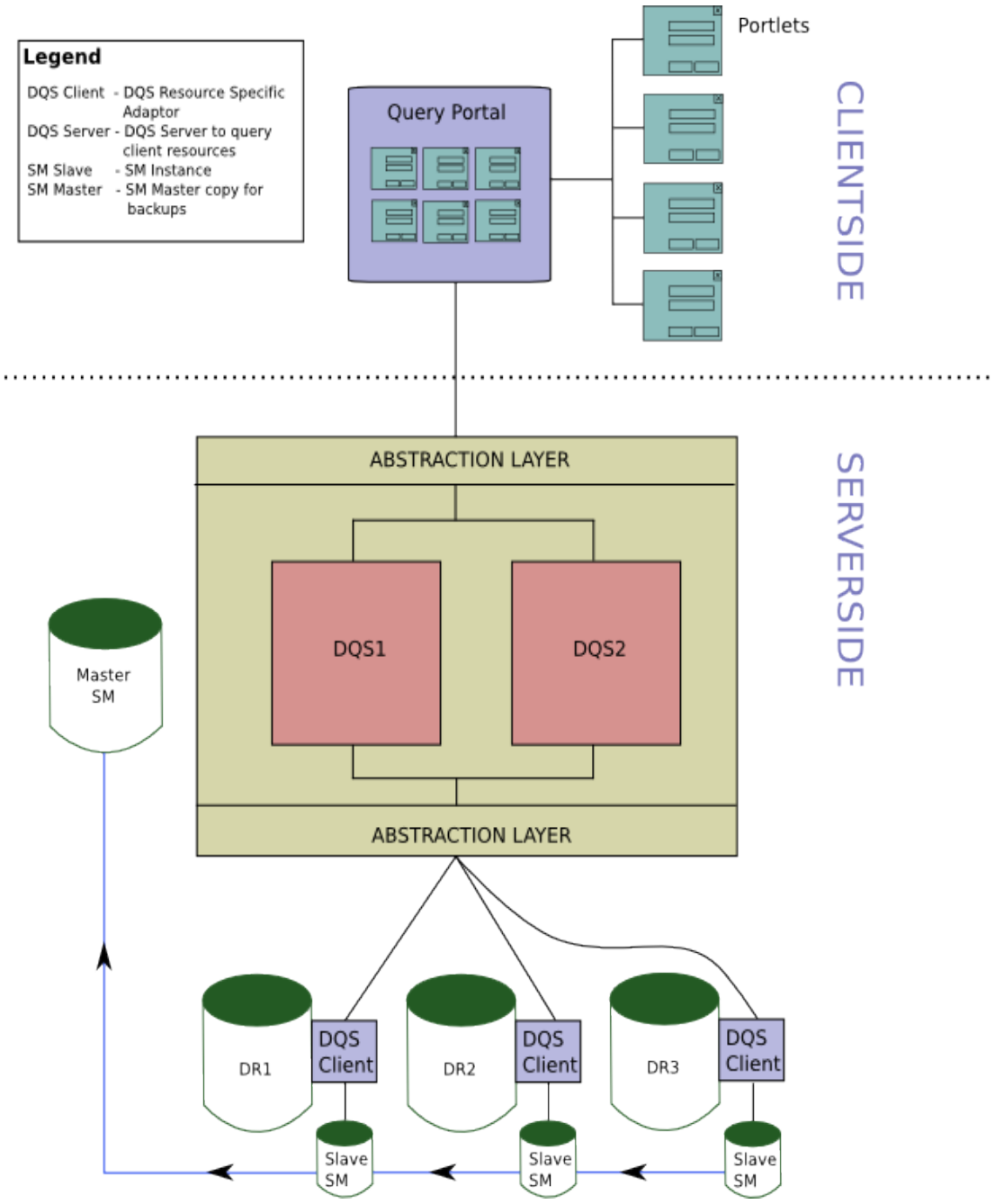

Figure 55. Replicated Meta-data

In summary the model offers the following advantages and poses the following issues:

Advantages:

- Multiple instance approach means the model is scalable, decentralised and highly available.

- Local semantics mean better quality of service.
- Easy to backup.

Disadvantages:

- Increased network traffic as databases synchronise.
- Scalability is hindered by the "adaptor" approach.

9.4.3 De-Centralised Semantics

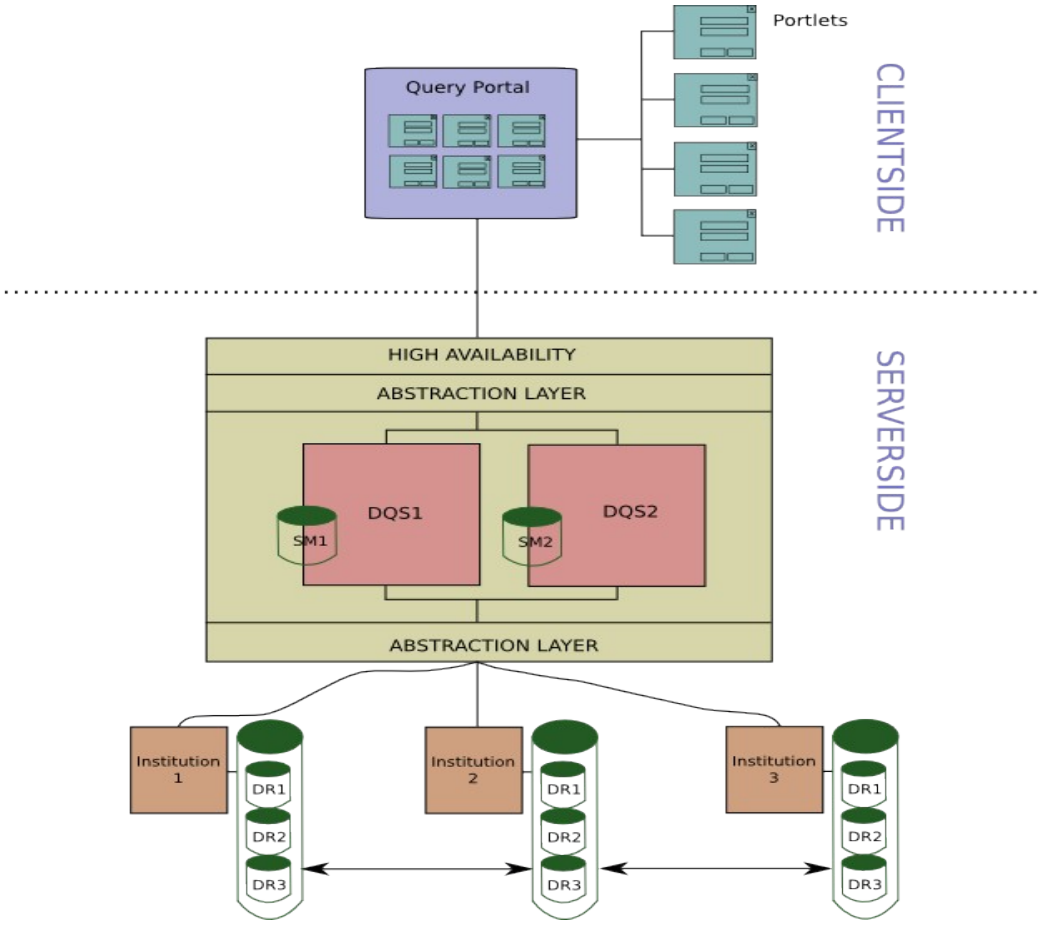

Figure 56. De-centralised Semantics

The de-centralised semantics or "peer-to-peer" model shown in figure 56 offers both high availability and the level of abstraction required. The main downside being that the "peer-to-peer" enabling software would have to be developed and this could be a lengthy and difficult task. Although, this is probably the largest barrier to adopting such a model for the architecture.

There are several persuasive arguments that can be made regarding the positive and negative effects of adopting a de-centralised architecture. It would certainly require a large amount of storage at all of the institutions, it is arguable as to whether this is redundant or not since it would provide some benefit in terms of a better quality of service.

All of the institutions have the same data available to them, there would be some mechanism in place to allow automatic updating of the data between institutions. In the model that is presented, the peers are connected through the DQS and something could reside at this level to facilitate this. Network traffic would be increased in this model but it would be of little importance since all peers have the same data and the queries are performed locally. It is only the results being passed up to the server and returned to the client. It is thought likely that the quality of service of this model would be good and that excellent speeds may be achieved. The model represents one that is highly available with a multiple instance approach of the DQS.

In summary, the model offers the following advantages and poses the following issues:

Advantages:

- Excellent speed (all peers have all data locally).
- Highly Scalable.
- Easy to backup.
- High Availability (multiple instance approach).

Disadvantages:

- Peer-to-peer software to be developed.
- Requires lots of storage at each institution.
- Bandwidth becomes an issue with this approach since the peer nodes are constantly sharing data to make sure that they are synchronised.
- The issue of security arises since all institutions hold all sensitive data from not only their institution but all other institutions.

9.5 Description & Justification of the Proposed Architecture

The de-centralised model is technically a good option but it remains the most complex to deploy, carrying with it the physical deployment of peer-to-peer software at each institution. The security and bandwidth issues which arise cannot be justified where such sensitive data is concerned. The data is replicated and if some data becomes corrupted in one of the nodes, this will spread to others rapidly. This could cause problems with the research of clinicians in the meantime which is unacceptable. In addition to the mention of the disadvantages in adopting such an architecture, we have the issue of maintenance and providing support to users at each institution.

The replicated meta-data model, although perhaps the second best contender for the architecture is not chosen because of the nature of the heterogeneous data resources. They are so geographically separated from one another. This model would require visiting each site and installing the DQS clients alongside the local databases. This would further complicate the deployment of the replicated meta-data architecture. The way in which the chosen architecture fulfils the requirements posed earlier is set out in this section.

The centralised meta-data model is the best contender for the architecture. The grounds for this statement are that it represents a truly abstract querying service which is designed with heterogeneous data resources in mind. The standard interface opens up the querying service to other services and clients via its own API. The architecture is also designed with the neuGrid design philosophy very much in mind. It lends itself to service oriented architecture with a standardised interface. Meta-data enrichment is easily incorporated into this architecture; however it is not so ingrained into the architecture in such a way that might make it difficult to get the fundamental querying service developed initially.

The centralised meta-data model is examined in the following sections with regards to the aforementioned benefits which are expanded:

9.5.1 Abstraction

Abstraction is present in two areas of the architecture. In the first instance, between the query service and the requesting client. This is necessary because there will be standardised communication between the query service and the client. The second area of the architecture which suggests abstraction is between the query service and the underlying heterogeneous data resources.

Abstraction is believed to be present in the right places because the only element underneath the DQS is the heterogeneous data resources and not meta-data. Owing to this, the service can handle changes at the underlying data level, providing scalability and minimal disruption of services throughout. The use of abstraction between the portal and the querying service means that the service contains a very standardised interface. This means that an API will emerge in order to communicate with the service under development. This API will be used with all clients including other services such as workflow and provenance services to access the querying service.

9.5.2 Querying of Heterogeneous Data

This relates back to the very first requirement which states that the user should be able to query heterogeneous data. This architecture addresses this issue by providing the layer of abstraction between the querying service and the underlying data resources as discussed. The standardised interface that the query service offers will allow clients to query all of the underlying resources as if they were one resource.

9.5.3 Service Oriented Architecture

A SOA provides a standardised interface which any client can use. For this reason, this kind of service is developer friendly and invites developers of other services to easily utilise this service. Something which is very useful since it has already been established that humans alone won't make up the main user base. The neuGrid design philosophy document sets out a SOA as being the architecture to use for these very reasons. [55]

Thus far, a standardised interface provided by a SOA has been discussed. This is based around SOAP (Simple Object Access Protocol) [56] which is XML transmitted across the Internet. The querying logic contained within this application will be such that it will incorporate semantic enrichment by using meta-data and interacting with data resources indirectly. Instead of each client containing the meta-data and communicating with data resources directly. It makes sense to provide a service which can incorporate all of this functionality and develop public methods which can be invoked by any potential client. This leads to a querying API as has been discussed briefly. Using a service-based architecture will be extremely useful if a multiple instance approach is used. Should we wish to adopt a querying cluster, the querying service complete with

the meta-data will represent a package which could be easily replicated. This could be deployed as needed to form cluster nodes.

9.5.4 Semantic Enrichment of Queries

The choice of architecture provides a benefit in that no commitment must be made to the semantics that will be used to enrich the service. Research points towards a rule-based reasoning approach used in collaboration with a knowledge base in order to process queries but this is still subject to more research and discussion.

Since it is required that this query service remain as generic as possible, no limitations should be made as to what data should be supplied as a query. For this reason, it is proposed that a knowledge base and rules are used to determine what data has been sent to the query service and exactly how it should be processed. This would add to the flexibility of the service, making the addition of query types a simple process. The query service should make the distinction between an SQL statement to be executed and keywords as examples and both queries would be treated differently.

In order to enrich the data supplied, it is thought that a meta-thesaurus could be used which would consult synonyms in cases where more than one term can be used to refer to something. This is often the case in the field of biomedicine, and so it is justified as something which would enrich queries.

9.5.5 Coherent Interface Possibilities

The development of a query service API has already been discussed. In reality, the user interface to this querying service will be a web portal with a stub created to interact with the API of this service. A step such as this serves to make the service more into a platform which could be interacted with as a complete component by external services and applications. This component coexists with other services and is part of a much larger range of services which make up neuGrid.

The stub and portal approach to user interfaces seems to fulfil this requirement of delivering coherent interfaces. Portals can be customised and portlets maintain a similar style. This means that for new features, users still see a familiar interface as the portlet inherits the style of the portal.

9.5.6 Availability / Accessibility of the Platform

The querying service and meta-data sit together and they could make a package which could be deployed on as many different servers in as many different locations as necessary to make the querying cluster as highly available as needed. Tools could easily be created to facilitate this and the fundamentals behind such an approach are effective yet not technically challenging.

The high availability presented in this model represents another reason as to why it was chosen. The ability to mould the multiple instance approach of the querying service into a querying cluster with each instance representing a node within the cluster is highly attractive to fulfil the some of the requirements. Although each instance of the querying service would most likely be distributed it would represent a clustering type approach with each instance of the querying service sitting inside this theoretical cluster. Instances could be distributed globally to where they are most needed to address quality of service issues. The service should run faster as a result and a load balancing mechanism can be introduced to distribute queries. In the case of a querying cluster, nodes of the cluster would most likely map to instances of the querying service.

9.6 Components, Standards & Interfaces

In this section a closer look will be taken at the key components of the chosen architecture and the way in which they interact. The key components that have been identified from the chosen architecture are:

- Client Interaction with Querying Service.
- Querying Service interaction with Knowledge.
- Querying Service interaction with Heterogeneous Data Resources.

In the following diagram, three distinct layers can be observed; the interface layer, the querying logic layer and the storage layer.

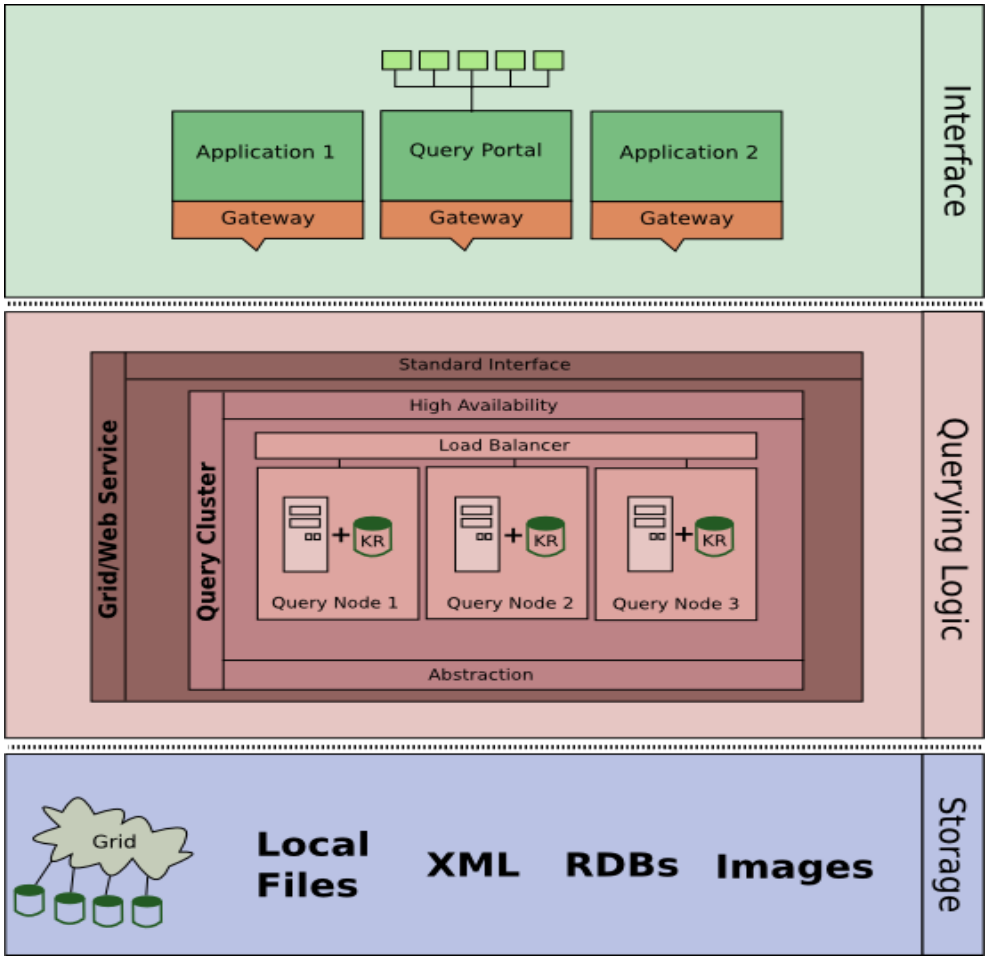

Figure 57: Three tier centralised architecture

9.6.1 Client Interaction with the Querying Service

Each portlet provides a customised way into the service. The portal acts as a placeholder which maintains a consistent look and feel for the portlets. The user types in their input into the portlet and then the portlet delivers it to the querying service. The querying service and semantics have been grouped together for the time being for simplicity but more about the communication between the DQS and the semantics can be found in coming sections. This interface will be standardised to provide support to any client supporting the standard interface.

9.6.2 Querying Service Interaction with Knowledge

Figure. 58 represent one instance of the brain of the querying platform. There would most likely be more than one of these instances running at any one time and on different servers. The client simply must know what services are available so that if an instance is not responding it can resort to trying another. The DQS and Semantics have been grouped up until now but this diagram shows the actual communication between the two components.

Upon the querying service receiving a query, it first fires rules derived from a semantic knowledge base to determine attributes of the submitted data to select the appropriate query type. Once this is known, the querying service then knows how to go about executing that query and returning the results.

In the case of the semantically enriched query, the search criteria will first be combined with the semantics for translation via a metathesaurus as used in the UMLS [57], the result will be returned to the user.

9.6.3 Querying Service Interaction with Heterogeneous Data Resources

The querying service must be able to interact with heterogeneous data resources in a coherent way such that a client can submit a query and the service takes care of the abstraction. Again, a standardised interface will be implemented here. Ways in which this can be performed are considered in the next section where we look at specific technologies and mechanisms for standardised distributed data access.

9.7 Technology Evaluation

Upon closer inspection of the chosen architecture, it could be split into a clearly defined three tier architecture before assessing technologies. It is clear that that there is an:

- 9.7.1 Interface Layer
- 9.7.2 Querying Logic Layer
- 9.7.3 Storage Layer

A detailed analysis will take place to identify and map the requirements and desired functionality of each layer to specific technologies with reference to research already performed in the field in the next section. The primary aim of this section is to simply list the options of what could be implemented and where it could fit in to our chosen architecture and how it complements the list of requirements.

In a previous figure a very simplistic view of the architecture can be observed(communication between layers is not shown since it will be discussed in this section).

The querying logic layer is the most complex layer as it will contain the business logic of the service including any enrichment. Key choices will focus on this layer in replacing generic components with actual technologies that will be implemented. Since there is a standard interface to the service, there are few decisions to be made around the interface. The same goes for the storage layer since this is a highly dynamic layer including grid resources and the service must operate within the constraints of these existing resources.

Semantic Integration Issue

In an article from the SINBAD research group at the Technical University of Madrid [58], a detailed architectural model that features semantic integration into a web service is explained. This article mainly addresses the dilemma of integrating semantics into such a model. The key way in which the proposed model differs from the model presented above is that it introduces a fourth subsystem which acts as a coordinator sitting between the application layer and the grid middle-ware. This subsystem would interact with the semantics and the distributed querying service and in the case of this service the middle-ware would be the heterogeneous data abstraction. [59] Whilst this article provides sound theoretical material with regards to the design of such a model, it lacks specific technological specification. At this stage of the service design,

finding technologies that can work together in harmony are a huge benefit.

Load Balancing Issue

Load balancing is another key issue with this architecture. Any single point of failure in the architecture has been avoided thus far and it is not intended to introduce one with load balancing. To maintain high availability and efficiency it will be necessary to balance the workload in the query cluster. This is not something to dwell upon within the service design, if usage did get sufficiently high, load balancing wouldn't be difficult to implement with the chosen architecture, even if this was simply making clients aware of what services are available on what servers and if one didn't respond within a given time frame submitting the query to another default node.

9.7.1 Interface Layer

The interface layer will simply contain front-ends (clients) and a gateway to access the querying logic which will make up the most complicated layer. As part of this service it is only intended to create one such application which will be the portal containing portlets. The main area of focus in this layer is how the application layer will communicate with the querying service. The gateway that is represented in the diagram is the API which will be used to communicate with the querying service i.e. Send requests and receive results.

When this service is fully implemented, a stub will be created to allow portlets to communicate with it, the portlet, being inside a web portal. For the purpose of the demonstration of the querying service layer it will not be of great importance to develop a rich interface. Since this will eventually be layered above the querying service to provide access to the functional querying service beneath. For that reason, there are several options of simple interfaces which could be used:

- CLI (Command Line Interface)
- Simple Web Based Form (Perl, Python, PHP or any other web scripting language capable of communicating via SOAP and processing forms).
- Mobile clients such as applications for Windows mobile devices or Apple's iPhone.

9.7.2 Querying Logic

The querying logic layer represents the following aims, these are carried all the way from the requirements stage of what the platform should support:

- Abstraction
- Load Balancing (in the case of a query cluster)
- High Availability
- Interoperability

9.7.2.1 Distributed Data Access

- EGEE (Enabling Grids for E-sciencE) AMGA (ARDA Metadata Grid Application) [60]

- Nordugrid ARC (Advanced Resource Connector) [61]

- OGSA-DAI Server [62]

Of the three distributed data access solutions it is apparent that only one, OGSA-DAI is truly generic in the sense that it isn't tightly coupled with any grid middle-ware product. In fact it is implemented as a web service thus ruling out the need for any grid middle-ware. This is a very large advantage over other solutions since in such a generic querying service, it is important not to be tied to any one technology. AMGA is very tightly coupled with gLite middle-ware and this would mean that should developers working on neuGrid decide to use Globus middle-ware later, they would not be able to. This kind of limitation is extremely bad in such a project and should be avoided.

In terms of published information surrounding the various products, there is an overwhelming number of documents and tutorials on OGSA-DAI. This gives the impression that this software has matured over some years and has been crafted according to the needs of the some 50 large projects which use it. This is very much in contrast to AMGA which appears to have a smaller developer base and ARC which tends to be very Scandinavian-centric in terms of its development.

The features of all of the products are very similar in that they all provide some standard access to grid resources. For this reason it is important to look at those products that have reached heightened maturity in terms of their development and those with the largest developer following. In addition, the more independent this component can be remembering that a grid middle-ware agnostic service is desired [55], the better.

9.7.2.2 Semantic Technologies

When working with semantics it is necessary to examine exactly what format the data will be held in i.e. an ontology or some other form of knowledge base. It is also important to define how that knowledge will be utilised i.e. by reasoning which could be implemented using a rule engine. In using this kind of approach, it is apparent that the rule engine would form the processing part of the semantic element and the knowledge base encapsulates the business logic which remains re-configurable as a separate component of the service.

There are many methods of knowledge representation and some of the most popular are shortlisted below.

- RDF [63]

RDF uses triples to describe resources, it is very simple and can easily be read by many API's in Java (for example JENA) making it a very good option to integrate with web services. Due to the simplicity of RDF there are limitations in its expressiveness. This must be carefully considered when deciding what it is hoped to achieve by using semantics.

- OWL-DL [64]

OWL-DL is much more descriptive than RDF but essentially has the same goals of describing resources. As such, it suffers from more difficult integration with Java code, the developer being presented with the dilemma of reading and making use of the ontology within their Java code.

- Relational Database

There is no reason why one should not use a relational database to convey knowledge which some may be tempted to put into an ontology. It could be argued that in order to provide flexibility, a database doesn't quite provide that level of flexibility that an ontology does. The benefit of an ontology is that it provides a configurable component that can be utilised by the rules. The ontology can be modified as the service runs in order to re-configure the way the service operates without any Java code re-compilation.

When discussing the various ways of making good use of the knowledge and rule engines, the following products were found:

- Drools [65]

Drools is a forward chaining rule engine, it is comparatively young, first being developed in 2001 and is tightly coupled with JBoss application server. There seems to be very little documentation and developer resources for Drools.

- JESS [66]

JESS features a simple to use Java API for creating rules 'on the fly'. It seems to be a very mature option and presents a language which forms the notion of 'rule based programming'. This was first developed in 1995 so it has stood the test of time and has been widely used in servlets and applets which are similar implementations to what we are hoping to do with the querying service. There are many published papers which talk of Jess and tutorials which go into great detail about how to get the most from the advanced features of Jess.

9.7.2.3 Web Service Enabling Technologies

Three enabling technologies have been identified as necessary to provide a web service to perform what is required:

- Java Web Service Implementation Code
- Application Server
- SOAP Server

The Java web service code is standard Java application code which is compiled and run within a SOAP server which in turn is deployed within the application server. It is at this point that abstraction is achieved since the SOAP protocol communicates via HTTP and the raw data is similar to XML thus insuring high interoperability.

The technological choices which must be made are based on which application server to use and which SOAP server to use. For example, Apache Tomcat [68] could be used as the Java application server; naturally this would lead to the decision to use Apache's answer to the SOAP server, Apache Axis [67] since it is logical that they should integrate well. There are many application servers from many vendors; the functionality is similar in that they all can host a servlet containing Java code of which a SOAP server would most likely comprise.

There are a vast number of application servers available and it is important at this stage for a candidate application server to emerge with the functionality we require. Since such a large

number of application servers exist all offering standard functionality of providing what is essentially a container to hold one or more servlets (applications). We must look to how well an application server could integrate with the rest of the system and how robust it is, other features such as how widely supported it is by developer communities is also important. Apache Tomcat now looks to be an attractive solution, with wide user support by the way of forums and online discussions [68]. In addition to this if the OGSA-DAI middle-ware is chosen, all of the documentation assumes a deployment into a Tomcat Server since OGSA-DAI is based around Apache Axis SOAP server.

The argument for using a service oriented architecture has already been extensively discussed in previous sections. It is important now to relate how these technologies fulfil the requirements. In using a SOA, the functionality we are trying to achieve is interoperability, high availability and abstraction. Since these represent a large proportion of the required functionality, these technologies are essential to deliver the querying service.

9.7.3 Storage Layer

The storage layer will contain the data resources, it is important to note that this could be both grid data and local data. The system should be as flexible as possible so that it isn't limited to just grid data resources in the future. This layer is included for completeness, although the layer is out of the scope of the project it is important to know what kinds of data need to be interfaced by the query cluster whilst looking at technologies. To semantically enrich this data it is essential to define the type of data queried by the service in conjunction with the user requirements.

9.8 Technological Choices & Justification

The technological choices for this platform are based around combining the following technologies. Since it is necessary to provide some test conditions at this stage in order to be able to assess the success of implementing the service, some solid decisions must be made as to what exactly will be demonstrated.

9.8.1 Interface Layer: CLI & Simple Web Interface

Interface Layer: Simple Web Based or Command Line Interface.
The justification for implementing such a simple interface is because it is simply an interface and is subsequently not in the scope of this service. The importance, complexity and interest of this service is in the querying logic. All of the services that make up neuGrid will be accessed through a stub which will exist as a portlet in a web portal in the future. In order to focus efforts on the querying service, portals and portlets will not be covered in depth because they are covered in another chapter. Indeed, a simple web based or command line interface will suffice in order to demonstrate the querying logic layer working to deliver enriched query results.

The demonstration of the querying logic in action over a simple command line interface and a web interface will serve to highlight the interoperability, and high accessibility of the platform. The most important aspect of this layer for it to work with the querying logic layer is that it must support the SOAP protocol. Fortunately, many technologies support communication via SOAP since it has become a standardised protocol.

9.8.2 Querying Logic: Querying Service

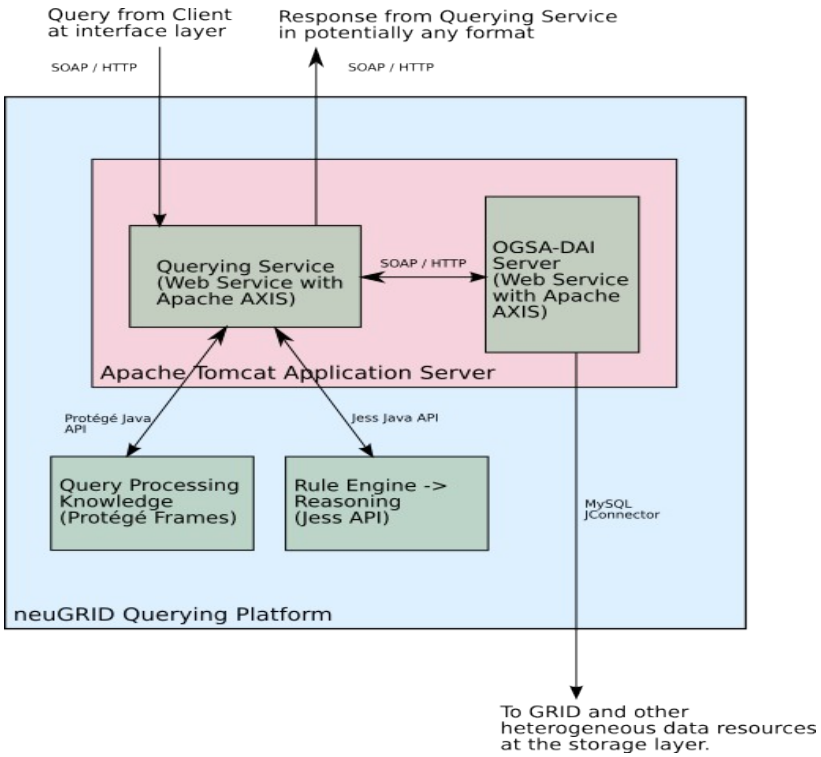

Figure 58: Detailed Querying Logic Layer with Chosen technologies

Querying Logic: Querying Service (Standard Java including Jess & Protege to provide reasoning and the semantic element)

The querying logic will exist as a service so that it can be instantiated as many times as necessary within the querying cluster. This will utilise JESS to reason with rules which will lead to how we identify a particular type of query and enrich it. The querying logic layer will create rules on the fly as it responds to queries and protege will be used to contain the ontology and subsequently the knowledge of the querying service. The protocol with which the client communicates with the querying logic layer is via SOAP over HTTP and thus the Internet. It is envisaged that the two web services will communicate via SOAP with each other, they will both be installed in the same Application Server (Tomcat) which will be instantiated as many times as necessary to make up the querying cluster.

OGSA-DAI communication with data resources is another interesting area of this diagram. OGSA-DAI was chosen since it is already heavily utilised when it comes to querying heterogeneous data resources but it includes no application specific semantics. These are therefore added by the way of a rule engine (Jess) and the knowledge (Protege). Both Jess and Protege offer their own Java API's so that they can be utilised in the code of the querying service which will contain most of the application code which will be written for this project. Once the decision has been made to use OGSA-DAI, we are subsequently agreeing to use Apache Axis or GT4 as the SOAP server to make it operate as a web service to accept requests and return responses. Upon closer inspection of the two versions of OGSA-DAI, the Apache AXIS version will be chosen as it offers exactly the functionality required. The Globus Toolkit 4 version encompasses more technologies and bloats what should essentially be a very simple case of fetching data and transporting it via SOAP. OGSA-DAI and our web service which will be developed requires installation in an application server. It is not necessary that they are on the same machine either.

In the diagram shown in previous sections, OGSA-DAI uses MySQL JConnector to query MySQL databases. This choice to use the JConnector driver is part of the abstraction which is handled by OGSA-DAI and in fact is of little concern within the scope of the querying service. The nature of the data resources means that they will not always be a MySQL database, in fact, OGSA-DAI supports a wide range of data resources. This is just a demonstration of the level of abstraction that OGSA-DAI provides by choosing the correct driver to access a particular resource which links well back to the requirements of studying heterogeneous data resources of different types.

9.8.3 Storage Layer: MySQL (LORIS DB)

Storage Layer: MySQL Data Resource (Containing LORIS) or other data resource type.
LORIS will be one of the databases used containing sample data at this stage, this contains the kind of data which will be processed by the service eventually, therefore it represents a good choice. It will be very useful to look at ways in which this data can be best queried with the use of semantics. Loris provides an open source database schema complete with interface to collect and manage imaging as well as non-imaging data for research studies.

9.8.4 Interaction: Protocols and Layers

1. The user generates a query via some mechanism, possibly through some drop down menus, the output of which is an SQL query. This is submitted via SOAP to the querying service.

2. The querying service analyses the query using its reasoning engine and the knowledge it has to try to enrich the results. The finalised query is submitted to OGSA-DAI via SOAP, but, communication may take place several times as the query is narrowed down and relevant data resources are bridged.

3. Return the query back to the client via SOAP.

The corresponding pseudo-code which follow helps to demonstrate the interaction between the different components.

```
while(query.waiting())
{
   query.determineType();         // Determines how we will enrich the query
&amp;nbsp;&amp;nbsp; &amp;nbsp;query.initEnrich();           // Perform initial query enrichment
   while(!query.fullyEnriched()); // Flag set when query fully enriched
  query.enrich()                  // Attempt further enrichment with results.
      query.submit()             // Submit query to OGSA-DAI
   }
   result = new Result(query.result)
   return result;
}
```

9.9 Implementation Plan

A small number of broad yet comprehensive milestones have been identified in order to implement the querying service, which complements the full list of user requirements.

9.9.1 Obtain Data & Deploy OGSA-DAI with a Simple Test Web Interface

This should be the initial priority. Once an instance of OGSA-DAI is running on a server, the fundamental querying service exists. It just needs to be enhanced with semantics so that intelligent tasks can be performed by the service.

9.9.2 Semantic Enrichment Component of Querying Service

The requirements must be reassessed at this stage in order to enrich the querying service using semantics in the most beneficial way to the users as possible. These must be evaluated and then implemented to provide the semantic enrichment component of the querying service. The first thing to do is to determine what additional knowledge we can use. Then, fundamental ways of representing knowledge within software will be studied, progressing onto ways in which this knowledge can be utilised.

9.9.3 Test, Evaluate and Document the Implementation

Full testing of the service should be engaged at this stage and an evaluation is made as to how it solves the initial problem and requirements. Documentation will need to take place at this point, including writing a user guide, and system report.

9.10 Conclusion

To conclude, a service has been designed which is in-line with the neuGrid design philosophy. The user requirements have been analysed and a web service will be implemented to allow flexible access to neuGrid data. The proposed interface has been discussed along with the communication down through the layers to the underlying data resources. It became apparent early on that adopting a web service alone lacked the semantic capabilities which would be required to make the service as flexible as is desired. Methods of enrichment via

semantics have been discussed along with the need for the presence of such features within the querying service. It is thought that these methods of semantic enrichment will keep the querying service as flexible as possible whilst providing the user with an array of querying options.

An API will be developed forming a standard interface to the querying service. This will make the service as accessible to human users as it will be to other services. The vision being that, a user provides a query and the service cooperates with a rule engine and an ontology to determine what the users intentions are and if there are possibilities for query enrichment.

9.11 References

[53] Fowler, M., "UML Distilled Third Edition", Addison-Wesley, 2004.

[54] Papazoglou, M. P. & Georgakopoulos, D., "Service Oriented Computing", Communication of the ACM, pp. 25-28, Vol. 56, Oct. 2003.

[55] NeuGrid Design Philosophy

[56] http://www.w3.org/TR/soap/

[57] Unified Medical Language System (National Library of Medicine) Website, http://www.nlm.nih.gov/research/umls/

[58] Miguel, J.L.; Calleja, A.; Costilla, C.; Garcia, M., Web services for a semantic Web integrated architecture, Services Computing, 2005 IEEE International Conference on Volume 2, Issue , 11-15 July 2005 Page(s): 239 - 240 vol.2

[59] Oscar Corcho, Pinar Alper, Ioannis Kotsiopoulus, Paolo Missier, Sean Bechhofer and Carole Goble, An overview of S-OGSA: a Reference Semantic Grid Architecture

[60] http://eu-egee.org/

[61] http://www.nordugrid.org/

[62] http://www.ogsadai.org.uk/

[63] http://www.w3.org/RDF/

[64] http://www.w3.org/TR/owl-guide/

[65] http://jboss.org/drools/

[66] http://herzberg.ca.sandia.gov/

[67] http://ws.apache.org/axis2/

[68] http://tomcat.apache.org/