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 (Glueing 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

.....2

Intended Recipients 3

7 The Glueing Service 4

7.1 Introduction 4

7.2 Architecture 5

7.2.1 Service Interface 7

7.3 Functionality 8

7.3.1 Selecting a Resource Manager 9

7.3.2 Job Definition 9

7.3.3 Data Stagin- out 9

7.3.4 Job Creation 10

7.3.5 Asynchronous Job Execution 10

7.3.6 The Discovery Service 10

7.3.7 Replica Management 10

7.3.8 Job Monitoring 11

7.3.9 Data Querying 11

7.4 Glueing Service Client Applications.....11

7.4.1 Overview 11

7.4.2 Pipeline Service Requirements 12

7.5. Limitations and Issues.....14

7.5.1 Workflow Support 14

7.5.2 The Discovery Service 14

7.5.3 The gLite Middleware Adaptor 15

7.6 Plan of Implementation 15

7.7 Conclusion.....15

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.

7 The Glueing Service

7.1 Introduction

The medical domain is increasingly building high-level distributed services such as querying, workflow creation, scheduling services, image handling and repository services. Such services help medical users to analyse their data, extract knowledge from it and share the results with other users. Most of these services are designed and developed for a particular community of medical users. The services built by one community often cannot be shared or re-used in other medical domains due to architecture, interface or platform limitations. The Glueing Service addresses the aforementioned issues by proposing an architecture which shields the heterogeneity of distributed resources and enables a set of generic services to access Grid resources without getting into the details of underlying Grid middleware or architecture. The Glueing Service is a constituent service of the generic middleware services layer in neuGrid, which aims to provide:

- A standard way of accessing Grid services without tying services and applications to a particular Grid middleware
- A mechanism to access any deployed Grid middleware through an easy-to-use service.
- A solution which extends and enhances the reusability of already developed services across domains and applications
- A service-based approach to shield users and applications from writing complex Grid specific functionality. The user requires a minimum set of Grid specific APIs and the rest of the functionalities are managed by the service.
- A simplified approach for enabling clients to Gridify their applications without installing and maintaining too many Grid specific libraries.

The Glueing service will expose the SAGA [37] (Simple API for Grid Applications) implementation using a web service to meet the aims mentioned previously. SAGA is an open standard that is defined and maintained by the Open Grid Forum [35] (OGF), which describes a high-level interface for effortless programming of Grid applications. SAGA is not designed for middleware developers, rather it provides APIs for developing applications using different Grid middleware. The SAGA API allows running a job on a particular middleware by the runtime loading of a middleware adaptor. SAGA middleware adaptors pass jobs to the middleware as its clients and are responsible for low-level communication with it.

The high-level interfaces, provided by SAGA, for communicating with different middleware removes the need for the use of middleware specific APIs. For example, if an application intends to use three different middleware then the application has to use a different API set, provided by the middleware, for interacting with each middleware. SAGA, on the other hand, provides a single generic API to interact with every middleware whose adaptor is loaded in the application. Thus, SAGA facilitates the Glueing service to glue a number of client applications with different middleware using the respective middleware adaptors.

The Glueing service wraps the SAGA API and provides a WSDL binding for client applications. There are numerous advantages in this approach, compared to the traditional approach where the client applications directly use the SAGA API implementation. The two most important reasons for adapting a web service based approach to SAGA include:

- SAGA Implementations come packaged in two parts: Middleware adaptors and the API. The client uses the API to communicate to the grid middleware via the appropriate adaptor. In order for the adaptor to communicate with the Grid middleware the client application, SAGA API and the adaptor must be deployed on a system which has the core middleware packages installed, deployed and configured. For instance, consider the user wants to submit a job to a condor cluster deployed as part of an OMII middleware grid. The client application must be deployed on a node, which has the OMII middleware client installed, as well condor installed and configured in submit only mode. If in the future the application is ported to another middleware, the client application must be deployed on a machine which has the new middleware installed. This increases porting costs and introduces complex installation and deployment procedures for SAGA based applications.

The Glueing Service aims to shield the users from these highlighted difficulties, and deploys the adaptors and middleware at the location of the Glueing Service host. The clients interact with the Glueing Service via a standard web service based infrastructure. In this scenario, if the middleware needs to be changed, all that the clients require is a new endpoint to an appropriately deployed Glueing Service.

- SAGA is an evolving standard. Currently there are numerous things, which are not provided by the SAGA API. Workflows are one of them and discovery is another. The Glueing Service can be used as means to provide functionality, which is not provided currently by SAGA. In this scenario, the Glueing Service would expose all of SAGA's capabilities. In addition it would also expose higher-level functions which SAGA does not provide, however they deal with the Grid middleware and are required by client services.

In order to make the Glueing service SAGA compliant, two separate WSDL's will be provided, one of which points to the full SAGA implementation, and another one describes the extended functionality, which is currently not supported by SAGA. Both WSDL's would map to the Glueing Service.

7.2 Architecture

The Glueing service exposes SAGA API functions as web service methods and provides one to one correspondence to the SAGA API functions. The client applications can transparently access the Glueing service by using a SAGA SOAP adaptor. It is an implementation of Adaptor API provided by SAGA. The client can include the SAGA SOAP adaptor and can write applications using the standard SAGA API classes. The SAGA API calls, generated on the client side, are passed to the Glueing service by the SAGA SOAP adaptor, which is responsible for communicating with the Glueing service.

The SAGA SOAP adaptor is a middleware between the client applications and the Glueing service. The client applications define, create and submit jobs according to the standard specification of SAGA [36]. These instructions are then translated into SOAP requests by the SAGA SOAP adaptor. SOAP requests are used for communication with the Glueing service. The Glueing service actually executes the client instructions using SAGA APIs and middleware adaptors.

The SAGA SOAP adaptor has methods to discover the Glueing service and retrieve its endpoint URL. The endpoint URL is used to access the service WSDL, which is then used for service invocation. The WSDL describes the definition of all the exposed methods. The SAGA SOAP adaptor calls the published methods using SOAP requests and the Glueing service sends back SOAP response to the SAGA SOAP adaptor. The SAGA SOAP adaptor then translates the SOAP response and returns the execution results to the client in the form of Java or SAGA specific objects.

The following diagram shows a scenario where a SAGA SOAP adaptor interacts with the Glueing service. The SAGA SOAP adaptor passes the middleware and job information to the

Glueing service using SOAP objects and the SOAP response is sent back to the SAGA SOAP adaptor from the Glueing service. The exposed functions of the Glueing service are discussed in detail in section 7.3.

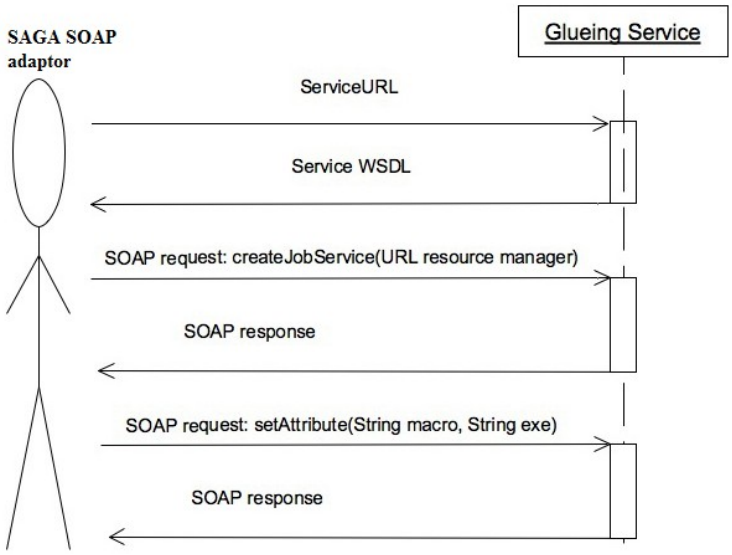

Figure 35: A sample use case

The architecture diagram (Figure 36) shows how an application contacts the Glueing service. The pipeline service, shown in the figure, is one of the potential applications of the Glueing service. This service uses the SAGA APIs and SAGA SOAP adaptors to communicate with the Glueing service. The SAGA SOAP adaptor accesses different methods of the Glueing service by getting its WSDL. The methods, published in the WSDL execute the actual instructions which are generated on the client side. Thus, the Pipeline Service initiates a grid activity which is then forwarded to the Glueing service by the SAGA SOAP adaptor.

The Glueing service, as shown in the diagram, can communicate with different Grid middleware such as OMII/GridSAM [38] or gLite [39] etc. This allows client applications, such as pipeline service, to use Grid resources provided through different middleware. The Glueing service uses JavaGAT middleware, which is a SAGA implementation, and OMII adapters to communicate with the OMII middleware. We are investigating the adaptors which have been written to access the gLite middleware, but most of these gLite adapters are either immature or do not provide sufficient functionality. We are expecting this situation to change in next couple of weeks, when some groups may release more mature gLite adaptors for SAGA. Figure 36 shows how the Glueing service glues the client applications with the underlying gird middleware.

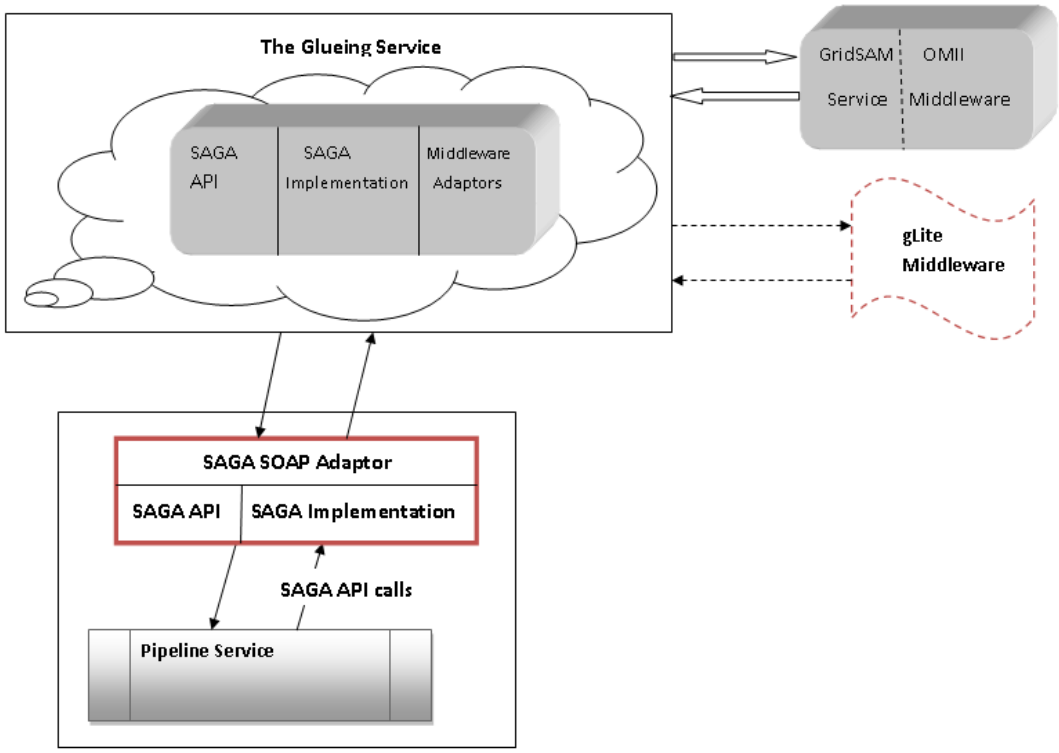

Figure 36: Glueing Service Architecture

7.2.1 Service Interface

The web service interface of the Glueing service is a set of SAGA-like methods which are exposed to clients. As the Glueing service uses a Java implementation of SAGA, the service interface is also developed in Java. The signatures of some of the exposed methods are given below:

- JobService createJobService();// This function creates an instance of JobService
- JobDescription createJobDescription(URL resourceManager);// This function creates an instance of JobDescription using the URL endpoint of the resource manager.
- Job createJob(JobService js, JobDescription jd);// This function creates an instace of Job identified by the JobService and JobDescription
- void setAttribute(JobDescription jd, String appMacro, String applicationExe); // This function sets Job attributes
- void setVectorAttribute(JobDescription jd, String parameterMacro, String[] applicationParameters);
- void run(Job job);// This function runs the job identified by the Job instance.
- Directory openDirectory(Directory dir, URL url);// This function creates an instance of SAGA Directory.
- File openFile(Directory dir, URL url);//This function creates a new File instance at the specified URL.
- int read(File file, Buffer buffer);// This function reads up to the buffer's size from the file into the buffer.
- int readP(File file, String pattern, Buffer buffer); // This function performs a pattern based read.
- int write(File file, Buffer buffer);// This function writes up to the buffer's size bytes from the buffer to the file at the current file position.
- JobDescription getJobDescription(Job job);// This function retrieves the job

description that was used to submit job.

- void migrate(Job job, JobDescription jd);// This function asks the resource manage to migrate the job
- InputStream getStdout(Job job);// This function returns the output stream of the job, which can be read.
- void suspend(Job job);// This function asks the resource manage to perform suspend on the running job.
- List find(NSDirectory nsDir, String pattern);// This function finds entries in the directory and below that match the specified pattern.
- List list(NSDirectory nsDir);// This function lists the entries in the directory namespace.
- void makeDir(NSDirectory nsDir, URL target);// This function creates a new directory at the target URL.
- void move (NSDirectory nsDir, URL source, URL targer);// This function renames the specified source to the specified target, or move the specified source to the target if the target is a another directory.
- void addLocation(LogicalFile lf, URL url);// This function add replica location to the replica set.
- List listLocations(LogicalFile lf);// This function lists the locations in this location set.
- void removeLocation(LogicalFile lf, URL url);// This function removes replica location from the replica set.
- void replicate (LogicalFile lf, URL url);// This function replicates a file from any of the known locations to a new location.
- void updateLocation (LogicalFile lf, URL nameOld, URL nameNew);// This function updates the replica location in the replica set.
- String listMetrics(Job job);// This function lists the metrics associated with the Job.
- Metric getMetric(Job job, String metricName);// This function returns a metric instance, identified by the metric name.
- State getState(Job job);// This function gets the state of the job.
- service_description[] list_services(String jobFilter, String dataFilter, String voFilter);// This function discovers different services based on the specified filters.
- void get_url(service_description sd);// This function gets the URL of a particular service which was discovered.
- String[] getAttributes(service_description sd);// This function gets all the attributes associated with the instance of service_description.
- String getId(Job job);
- void monitor(String jobId);

7.3 Functionality

The Glueing service exposes SAGA APIs in the form of a web service which allows client applications to create, run and monitor jobs transparently from the underlying Grid middleware. The client applications, using the SAGA SOAP adaptor, pass job and middleware information to the Glueing service. This service loads the appropriate middleware adaptor, based on the middleware information. For example if a client wants to run jobs or access data on OMII based infrastructure, the client specifies an OMII/GridSAM middleware,

the Glueing service sets a system property "**JobService.adaptor.name" equals to "javaGAT"**. This system property behaves like an environment variable for the SAGA engine to narrow down the selection of middleware adaptor. The Glueing service then makes sure that JavaGAT loads a GridSAM adaptor by creating an appropriate session and context for SAGA. A context is a specific piece of information that is shared with a particular session. An application may associate different contexts with a particular session in order to perform different functions in that session.

The Glueing service creates a preferences context for JavaGAT to make sure that it loads the GridSAM adaptor. This is done by setting the context property "**ResourceBroker.adaptor.name" equal to "gridsam"**. This context is then added to the application session for further reference during the life-time of the application. Once the session is created the Glueing service creates a job, based on the job information sent by the client application. The Glueing service uses SAGA job management APIs for creating a job, submitting it to the available Grid resources and retrieving its status. The SAGA job management API covers four classes; these are JobService, JobDescription, Job and JobSelf. The order in which the Glueing service uses this job management and other APIs is as follows:

7.3.1 Selecting a Resource Manager

The JobService API is used to select a resource manager. An instance of JobService represents a resource manager backend. A resource manager is an endpoint where the job is submitted by the client application. This resource manager can also be an execution service if it executes the job.

Input parameter: To create an instance of JobService class an endpoint URL of resource manager is required as input parameter to *createJobService* method.

Example: The Glueing service uses an endpoint URL for submitting the job to resource manager. For example if resource manager is GridSAM then an instance of JobService is created as: *<JobService> js = JobFactory.createJobService (new URL ("https://HOST:PORT/Gridsam/services/gridsam")*.

7.3.2 Job Definition

The *JobDescription* API defines the job using a well defined set of attributes such as the application executable and associate arguments. The *JobDefinition* attributes behave like tags in JSDL/JDL and thus these attributes mimic JSDL for the middleware and are passed to it internally by SAGA.

Input parameters: The JobDescription API needs two essential parameters in order to define the job i.e. the application executable path and the exact application parameters.

Example: If the application execution is "/bin/echo" which takes a string as input parameter i.e. "hello" then the Glueing service, using JobDescription, defines a job as: *<JobDescription> jd.setAttribute (JobDescription.EXECUTABLE, "/bin/echo")* and *<JobDescription> jd.setVectorAttribute (JobDescription.ARGUMENTS, new String[] {"hello"})*.

7.3.3 Data Stagin- out

The job execution results can be stored in an output file as: *<JobDescription> jd.setAttribute (JobDescription.OUTPUT, "outputFile")*. The JobDescription class also defines resources for data staging. For example the results of execution can be staged out at a remote location using FTP as: *<JobDescription> jd.setAttribute (JobDescription.FILETRANSFER, "file://HOST:PORT/outputFile < outputFile")*. The execution results are staged, once execution is complete. The results can be copied by calling the *copy* function of the Glueing service and by passing it source and target URL parameters.

7.3.4 Job Creation

The Glueing service creates an instance of the Job class using *createJob* function of JobService, which takes an instance of the JobDescription class as input parameter. Instances of both JobService and JobDescription classes are a pre-requisite for creating a Job.

Example: A job is created as: *<Job> j = <JobService> js.createJob (<JobDescription> jd)*. The Glueing service executes jobs by *<Job>j.run()* and an implementation of *CallBack* interface allows it to get monitoring information of job during the course of execution.

7.3.5 Asynchronous Job Execution

The Glueing service also allows the client applications to run different independent jobs at the same time. The feature of running jobs asynchronously is provided by the SAGA *TaskContainer* API. An instance of *TaskContainer* may contain multiple tasks. A task is a SAGA API call, whereas a job is a remotely running application. The SAGA *Job* class extends *Task* interface therefore a *TaskContainer* can also contain multiple jobs. The jobs are executed asynchronously when the instance of *TaskContainer* calls its *run()* method.

Example: If there are two different jobs, j1 and j2 and both copies a file from one URL to another then the *TaskContainer* will add both jobs as: *<TaskContainer> T.add(j1); <TaskContainer) T.add(j2)*. The task container will execute these jobs asynchronously as: *T.run()*.

7.3.6 The Discovery Service

The *JobService* API, provided by the standard SAGA implementation, allows the application to select only one resource manager. The client however sometimes wants to select the best resource from those that are available and this requires a discovery service. It is proposed that the Service Discovery APIs developed by RAL (Rutherford Appelton Laboratory) are used for the discovery of the grid resources. The RAL APIs provide a *discoverer* class and a method named *list_services,* which lists the available services. This process is based on the service, data and VO (virtual organisation) filters. All these filters are of a string type and use SQL like statements to filter out the best resource. The *list_services* method returns an array of different services encapsulated in *service_description* class. The client can then call *get_url()* for a specific instance of *service_description* to get the endpoint URL of the selected service.

Example: If a client wants to filter out SAGA job services with the name *TestJobs* owned by a VO, named UWE and for which the RunningJobs parameter is less than 10, then the client would get a list of services based on the aforementioned filters as: *(array) <service_description> sd = <discoverer>d.list_services("type = 'org.ogf.saga.service.job' AND name = 'TestJobs'", "RunningJobs < 10", "VO = 'UWE'").* Now the client, based on the attributes list, can select a particular service as: *sd.get_url()*. The API for the discovery service is not available for the Java implementation of SAGA, therefore the WSDL of the related functions could not be generated (see section 7.5), but we will do it as soon as the implementation becomes available.

7.3.7 Replica Management

The Glueing service also allows client applications to replicate files over Grid resources using FTP or GridFTP. If client wants to replicate a file to a Grid resource using GridFTP, he can do this by calling *replicate* function of the Glueing service by providing it the location of GridFTP server and the source file.

Example: If the location of the source file is */home/test.txt,* the replica is generated by first creating an instance of for the source file as: *LogicalFile lf = new LogicalFile("/home/test.txt")* and then this file is replicated to a remote server at the server URL as: *lf.replicate(new URL("ftp://server:port/test.txt")).*

7.3.8 Job Monitoring

The Glueing service monitors a job during its lifetime using call backs; it provides two ways of retrieving monitoring information. One way is to use the higher level implementation, by which the client application can invoke *monitor (String jobId)* function of the Glueing service to get the complete monitoring information, associated with a particular job id. In this implementation the Glueing service stores the monitoring information of the job in a *String* and is returned to the client when he invokes the *monitor*. The other way of monitoring a job is to use the SAGA API function. In this implementation the client first gets the job metric and it can then be used to get the metric name and value.

Example: A job metric can be obtained by using a *Job* instance and the *metricName*. If *j* is an instance of *Job* as created in section 7.3.4 and *metricName* is Job.JOB_MEMORYUSE then the job metric can be obtained as: *Metric m = getMetric(j, Job.JOB_MEMORYUSE)*. This job metric contains the information of memory used by the particular job *j*, while it is in execution. The metric value can be obtained by *m.value.*

7.3.9 Data Querying

The Glueing service also provides a mechanism for querying the data, staged out or replicated at the remote locations. The staging endpoints may contain a number of execution results in the form of files. A client may want to query a particular pattern to see the list of files present at the remote location. This can be achieved by calling the *find (NSDirectory nsDir, String pattern)* function, exposed by the Glueing service.

Example: The Glueing service queries a particular pattern using the instance of NSDirectory. The instance of NSDirectory points to a remote directory. If the client queries a pattern "*.jpg" to list all the jpg files, staged out at the remote directory then the Glueing service fetches the list as: nsDir.find("*.jpg").

7.4 Glueing Service Client Applications

7.4.1 Overview

The Glueing Service client applications such as a pipeline service, allow clinicians and neuroscientists to create neuro-imaging pipelines, in a user friendly environment, for a series of automated transformations on the brain images. Once the pipelines are constructed, the neuro-imaging Pipeline Service parallelizes them. The processes defined in the pipelines are usually very compute-intensive and deal with large amounts of data. Therefore, in the next step the workflows are Gridified for their optimal execution over the Grid. The neuro-imaging pipeline service, using the Glueing service, submits the workflows over the Grid for an execution and the results of execution are returned to the client interface of pipeline service. A simplified design of the Pipeline Service would look like:

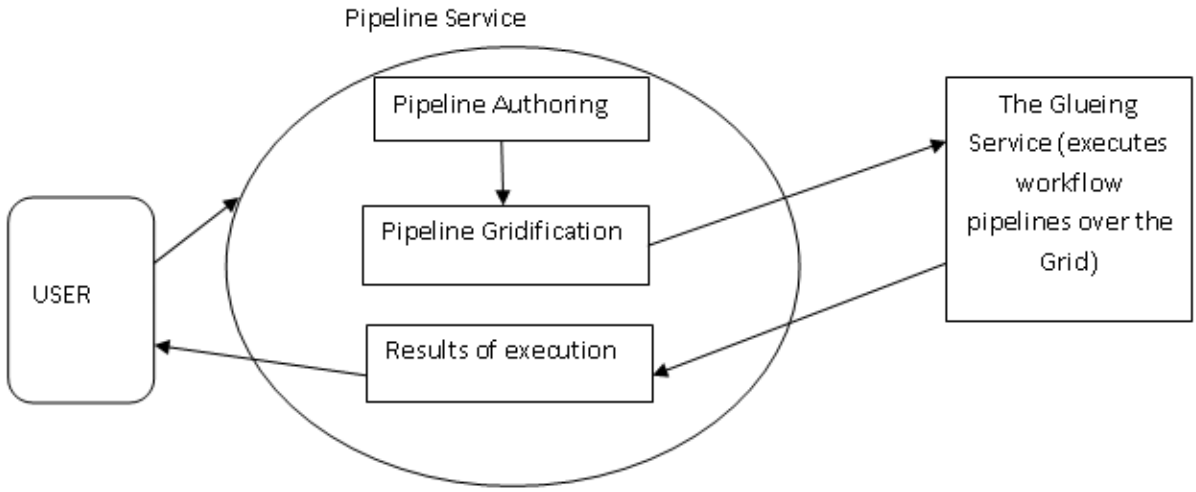

Figure 37: Pipeline Service

The Neuro-Imaging Pipeline Service passes workflows, or a sequence of processes, to the Glueing service. The Glueing service exploits the SAGA system and executes the processes, in the workflows, over the Grid using appropriate Grid middleware. The results of execution are passed to the provenance service and eventually reach the client. In the following sections, some of the requirements of the neuro-imaging pipeline and other services are described that should be addressed by the Glueing service. These requirements have been discussed to evaluate the capabilities and functionality of the Glueing service (and thereby of SAGA implementations). An attempt will be made to support all of these requirements in the Glueing service and any limitations will be reported where necessary. Since most of other services in WP6 will be invoked in the Pipeline Service to carry out a brain imaging analysis, we list here the Pipeline Service requirements only. These requirements will represent most of the requirements of client application of the Glueing service.

7.4.2 Pipeline Service Requirements

The Pipeline Service requires a set of functions from the Glueing service. The diagram shown below explains these requirements.

From the diagram we can see that there are some common requirements amongst the services, the following is a brief discussion of each function which Pipeline Service requires from the Glueing service.

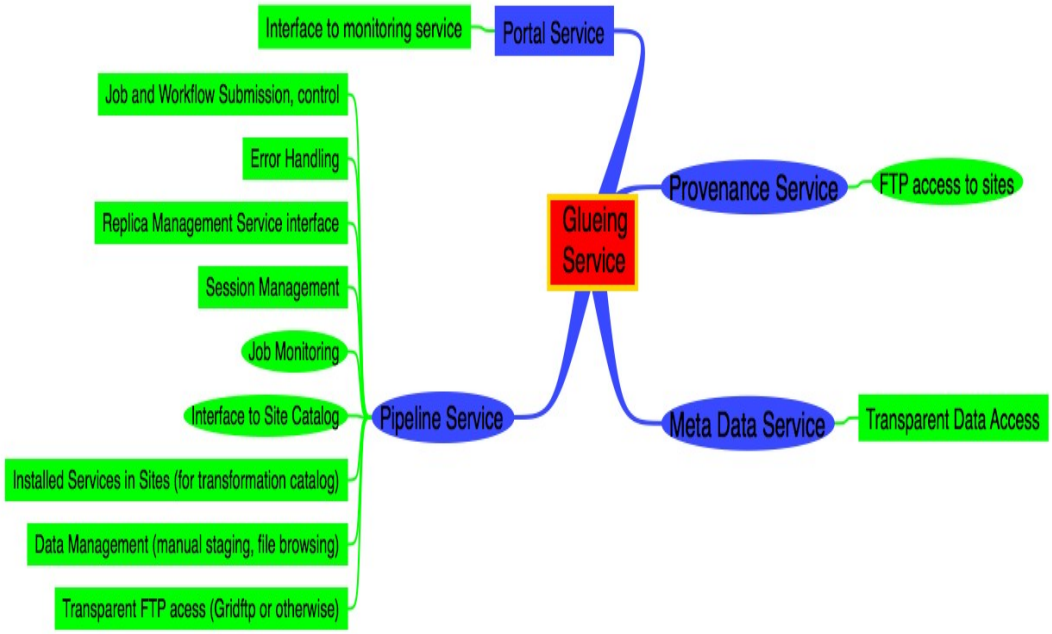

Figure 38: Pipeline Service requirements

7.4.2.1 Job and Workflow Management

The Pipeline Service is designed to be infrastructure agnostic, it is not a necessary requirement that a Grid has to be used as the fabric infrastructure; hence support for simple job management is required. This simple job management will encompass the submission of a single job to a cluster management service and handle job control and error reporting. Workflow management is more relevant to Grids. Workflows will be passed to the Glueing service in standardized formats like JSDL, and the Glueing service will translate it to a Grid meta-scheduler format (e.g. JDL for gLite WMS) and forward it to the Grid metascheduler concerned. The functionality to monitor the progress of a workflow, the data generated and any errors that occur will also be provided.

7.4.2.2 Error Handling

The Glueing service will report errors occurring in the Grid middleware, or fabric infrastructure. Errors may encompass unavailable sites, incomplete or incorrect workflow specification or missing actors in the workflow etc.

7.4.2.3 The Replica Management Interface

This is an essential requirement for the pipeline service. The pipeline service's Grid-enabling mechanism needs to detect preinstalled workflow actors, and if the data created in the workflow stages already exists or not.

7.4.2.4 Session Management

Session management encompasses the whole activity of workflow execution. When a user wants to submit a workflow, a session is created for the user, all the data generated including error or general information is related to the specific session ID. The user will be able to track workflow information via the session ID.

7.4.2.5 Job Monitoring

This requirement is related to the need for session, job and workflow management. The primary role of this function is to provide a means of access to low level site execution data,

to retrieve execution logs, error logs and the outputs that are created.

7.4.2.6 The Site catalogue Interface

This is another primary functionality required for Grid-enabling a pipeline and partitioning them to available sites. The Site catalogue is usually maintained by the monitoring service.

7.4.2.7 Access to Grid Information service

This functionality allows the Pipeline Service to query the information service to retrieve installed actors in disparate sites and allow users to use them as actors in their pipelines. This is also required by the Grid-enabling mechanism to determine where the executables are located to efficiently partition a workflow.

7.4.2.8 Data Management and Transparent FTP Access to Sites

This area of functionality deals with the following:

- The ability to search and locate specific files and data.
- The ability to discover used data sources in specific sessions.
- The ability to transfer files or folders between sites, for file staging purposes.
- The ability to retrieve via FTP output data from site level data sources.

7.5. Limitations and Issues

The Glueing service exposes SAGA APIs; therefore it can only provide those functions which are supported by SAGA API. The requirements of the Pipeline Service are not fully addressed in the Glueing service because of the lack of support, for those requirements, in the current SAGA implementation. Some important limitations in the implementation are discussed below:

7.5.1 Workflow Support

As the Pipeline Service generates pipelines or workflows to be executed over the Grid, it needs an enactment engine which will break the workflow into its constituent parts/jobs. It also needs to resolve job dependencies and then execute the sequence of jobs efficiently. The current release of SAGA can only submit one job at a time through its JavaGAT adaptors, to a submission system such as GridSAM. Thus it does not have support for workload management and scheduling a series/sequence of jobs as per the requirements of workflow. This lack of workflow enactment in SAGA limits the scope of the Glueing service. Thus, the Pipeline Service requirements, described in section 7.4.2 which deals with workflows cannot be fully supported by the Glueing service. We expect however that workflow support will shortly be added into the SAGA specification.

7.5.1.1 Enabling Workflow Support for specific workflows

The Glueing service, as described in section 7.3.5, can support asynchronous execution of jobs. Therefore the Glueing service can use *TaskContainer*API for executing workflows of a specific type. A workflow is a set of executables and these executables can be defined as different jobs using the SAGA API and then a task container can execute them asynchronously. This can only be done if the executables defined in a workflow are independent of each other, i.e. one job does not wait for the other one to complete. Thus, enabling workflow support with the help of *TaskContainer* allows the Glueing service to meet different Pipeline Service requirements. These include session management, data staging, data replication, monitoring, error handling, discovery service and catalogue management. The requirements are covered in the functionalities section of the Glueing service.

7.5.2 The Discovery Service

The Glueing service uses a Java implementation of SAGA (1.0 rc1 release) and the discovery

service is not implemented for this release, therefore this also limits the scope of the Glueing service. But an implementation of the discovery service has been done at the Rutherford Appleton Laboratory and we will investigate its functionality for possible use in the Glueing service.

7.5.3 The gLite Middleware Adaptor

The Glueing service promises to glue different client services to different Grid middleware and vice versa. It uses JavaGAT adaptors as the intermediary connectors between the client service and the Grid middleware. The potential middleware, for testing the Glueing service, are OMII and gLite. The Glueing service works fine with the GridSAM JavaGAT adaptor to submit independent jobs to the OMII middleware. The gLite adaptor however, provided by the JavaGAT, is not complete and is not included in the recent release of the SAGA-JavaGAT implementation. This issue with the gLite adaptors also limits the scope of the Glueing service by restricting it to use the stable list of adaptors provided by JavaGAT like OMII/GridSAM. Consequently, the Pipeline Service cannot be exposed to Grid resources using gLite middleware. We are constantly looking into the availability of the gLite adapters and the Glueing service will provide the support once the required adapter becomes available.

7.6 Plan of Implementation

The implementation plan of the Glueing service will be a three phase process as shown in the following diagram.

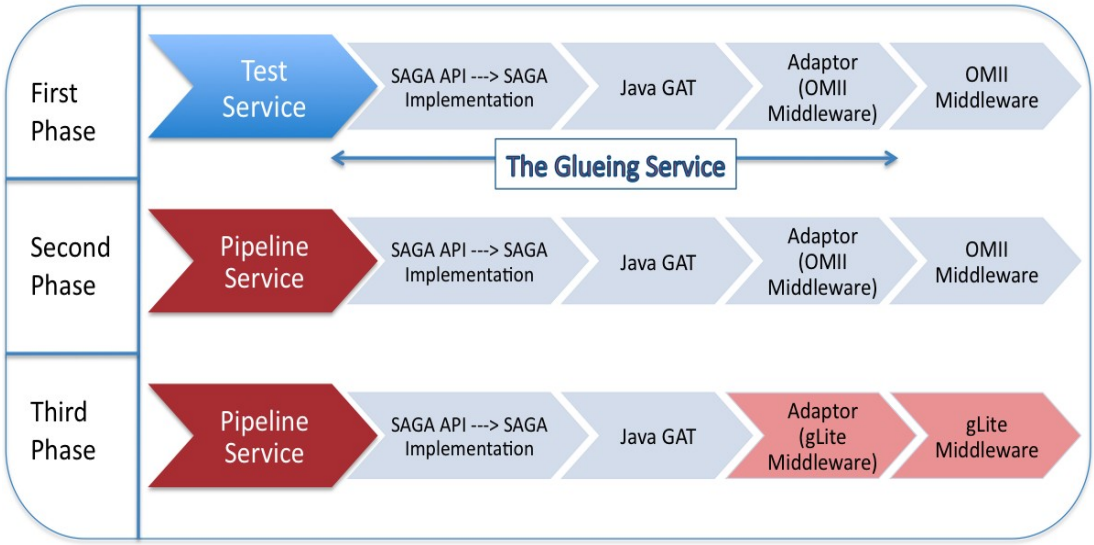

Figure 39: Implementation Plan

The first phase of the Glueing service development is to test its functionality with a test service (client application) and OMII middleware. The second and third phases are replacements/additions for some components on the both sides of Glueing Service in order to make it generic. In the second phase the test service will be replaced with the Pipeline Service, being developed in UWE, Bristol. This replacement is to test the generic behaviour of the Glueing service. In the third phase the Glueing Service will be tested with the gLite middleware, which ensures that changing the middleware does not cause any change in the application and service interfaces.

7.7 Conclusion

The Glueing service addresses some major requirements of the neuGrid project and will provide a generic framework for accessing resources over the Grid. The heterogeneity of

distributed resources and details of grid middleware architectures will be transparent from users. The Glueing service also hides complexities of interfacing with different grid middleware, which will allow accessing grid resources through a set of high-level functions. The service exposes SAGA APIs and can communicate with different middleware through their middleware adaptors. Details of middleware interactions are kept hidden from users enabling them to seamlessly use grid functionalities. This shields the low-level middleware difficulties from the user and will encourage him to use them with little or no knowledge. The design of the Glueing service is based on SOA principles, which will help different services in neuGrid to use service functionalities through standardized interfaces. This will also allow other client applications to use service features by inspecting its WSDL, available online on the service endpoint URL. SOA-based architecture of the Glueing service is in line with the project requirements and will provide a gateway to all WP6 service to access grid resources.

7.8 References

[35] http:/www.ogf.org/documents/GFD.90.pdf

[36] http://www.ogf.org/Public_Comment_/Docs/Documents/2008-05/saga_sd-v1.0rc1.pdf

[37] http://saga.cct.lsu.edu

[38] Frank, Daniel; Soddemann, Thomas, "Interoperable job submission and management with GridSAM, JMEA, and UNICORE", German e-Science Conference 2007.

[39] http://www.gLite.org