Glueing Grids and Clouds together: A Service-Oriented Approach

Ashiq Anjum[1], Richard Hill[1], Richard McClatchey[2], Nik Bessis[1], Andrew Branson[2]

[1] School of Computing & Mathematics, University of Derby, Derby, United Kingdom
[2]CCCS Research Centre, FET Faculty, UWE Bristol, United Kingdom
[1] (a.anjum, r.hill, n.bessis)@derby.ac.uk, [2] (richard.mcclatchey, andrew.branson)@cern.ch

Abstract

Scientific communities are actively developing services to exploit the capabilities of service-oriented distributed systems. This exploitation requires services to be specified and developed for a range of activities such as querying, management and scheduling of workflows, image handling, provenance capture and management, amongst other activities. Most of these services are designed and developed for a particular community of scientific users. A key issue here is that the constraints imposed by architectures, interfaces or platforms can restrict or even prohibit the free interchange of services between disparate scientific communities who have a shared need for a particular service. Using the notion of “Platform as a Service” (PaaS), we propose an architectural approach, based on a so-called Glueing Platform, that addresses these limitations so that users can make use of a wider range of services without being concerned about the development of cross-platform middleware, wrappers or any need for bespoke applications. The proposed architecture shields the details of heterogeneous Grid and Cloud infrastructure within a brokering environment, thus enabling users to concentrate on the specification of higher level services that are more relevant to the intended application.

1. Introduction

Scientific communities are increasingly building high level distributed services such as analysis services, workflow and scheduling services, monitoring services and provenance services. These services or applications help scientific users to analyze their data, to manage, to monitor and optimize computational infrastructure, to extract knowledge and to share it with other users. Most of these services are designed and developed for a particular computing platform (such as the gLite Grid middleware [1] or the OpenStack Cloud platform [2]) and for a specific community of scientific users, for example molecular or biomedical analysis [3]. It is often difficult to share or re-use services from one community with another community due to architecture, interface or platform limitations. Similarly the services developed for one platform such as Microsoft

Azure [4] cannot be deployed in another computing environment such as Amazon Web Services [5] due to stack mismatch and interoperability issues between the service providers. This has become particularly critical in the case of distributed infrastructures such as Grids and Clouds that host a number of high level applications.

An application designed and developed for one Grid middleware, despite resource sharing, is often challenging to run on another platform thus limiting the wide scale adoption of this technology. This is particularly prevalent for Cloud providers, since there is a tendency to adopt proprietary standards for service implementation, which inhibits the generalization of service specifications, and leads to a consequent Cloud platform lock-in. As Grid technologies evolve onto Cloud platforms the situation worsens further since either the applications need to be redeveloped for the emerging platforms or significant changes are required to adapt the applications for the newer platforms. This is further compounded when Cloud stacks offer almost similar features, and services cannot cooperate due to technical or architectural limitations. Choosing a Cloud infrastructure or platform means a commitment for the whole application execution life. As such, an application is bound to a platform and users have limited choice if they intend to change an infrastructure provider in future. The potential wasted effort as well as the added investment required presents a significant barrier to the adoption of Cloud technologies. Since applications for Grids and Clouds are relatively tightly coupled to the underlying platform and we are presented with two significant issues. Firstly, typical Grid and Cloud services such as resource management, scheduling, monitoring and security policy management are specific to the platform and therefore users must acquire and maintain specific expertise related to the Grid or Cloud infrastructure used. Secondly, an application that itself exhibits heterogeneity before deployment to a Grid or Cloud requires a specific implementation that further increases the coupling of the application to a particular infrastructure. Grid and Cloud infrastructures and platforms are designed in such a way that an application has to be horizontally as well as vertically integrated with the underlying platform to achieve the desired performance guarantees, throughput and optimization. Isolation, encapsulation, ad-hoc communication and transparent contract negotiation between the platform and applications has generally not been considered in the design and development of the Cloud as well as Grid platforms.

As a result of these design practices, any changes in platform middleware, or a desire to move from one Grid or Cloud provider to another, will require a significant amount of application re-development [6]. Consequently users have a fundamental dilemma in terms of weighing-up the eventual value of the system against fundamental application re-development costs. Though most Grid and Cloud platforms claim to follow a service oriented paradigm, the current stacks only partially implement Service Oriented Architectures (SOA), and most of the time any benefits of SOA cannot be exploited due to the architectural choices that have been followed to implement the Cloud and Grid platforms. This clearly does not enable the benefits of a service-oriented approach to be realized, whereby users should be concentrating upon their core business, rather than the engineering requirements of their underlying infrastructure.

Whilst the marketing hype surrounding Grid and Cloud Computing often promises elasticity of resources, ubiquitous access and abstraction from heterogeneity [7], the reality is often somewhat different, and as long as application portability, compatibility and interoperability with underlying infrastructures remain a challenge [8], the true potential of the Grid and Clouds platforms cannot be realized. For instance, if an application's needs cannot be addressed by a local Grid/Cloud infrastructure, the ability to add capability on demand by calling upon third party Grid or Cloud services, irrespective of where and how they are hosted, is still problematic and far from the promise of 'utility computing'. As described earlier, service based infrastructures tend to be constrained by the suppliers of the platforms, thus acting as a disincentive towards the wider utilization of Grid and Cloud platforms. If an application is built upon a combination of local resources and commercial resources (such as RackSpace [9] or Amazon [10]), any future or emerging business requirement to adapt or transfer to other platforms will require application re-development, which does not faithfully follow a service-based model of computing. A purer form of PaaS and IaaS should enable the users to run their applications as a utility consumption, without being concerned with whoever is providing the infrastructure, what platform or protocol has been used or how to switch to a different computing infrastructure provider. The non-availability of such an environment is therefore a fundamental barrier towards the pervasive adoption of Grid and Cloud platforms demanding concrete steps to be taken to address the limitations.

This is far from ideal for users, who not only have preferences in using tools and technologies for their analysis, but they may resist the introduction of new interfaces or attempts to customize their applications. This takes up significant time and resources in understanding the platforms and technologies, which should be spent on solving the scientific or business challenge. Essentially, this trend drives users away from their core business and discourages them from exploiting Grid or Cloud resources. If a decision is made to exploit these alternative platforms, then users must invest considerable time to understand the platform, its interfaces, underlying protocols and technologies before high level applications to exploit the computer and data resources available can be written. There is no platform or service available that can enable users to abstract these details so that they only focus on the application level, with the underlying complex infrastructure details remaining hidden (unless, of course, it is important for increasing the performance of an application or for other users' needs). The problem is exacerbated by frequently changing platforms, technologies and interfaces, which force users to redesign and re-implement their applications. This restricts the adoption of computing platforms made available through Grid and Cloud technologies, which are anticipated as deployment platforms for better performance, scalability and serviceability of applications.

In this paper, we propose a platform that addresses the limitations stated above. We extend the notion of PaaS to address platform, interface and technology limitations, to ensure that users do not need to worry about the lower level details of a platform for cross-community applications. Using a service-oriented platform, a brokering environment is offered which shields the heterogeneity of distributed resources present in Grids and Clouds. This platform (referred to as the Glueing Platform (GP) thereafter) supports applications by allowing concurrent access to a number of platforms, by shielding the underlying infrastructure details from users and applications, thus enabling users to implement higher level services. The platform provides features such as workflow enactment, execution control and monitoring, provenance and tracking, concurrency management at the application level, scalability to support a number of applications and users, and security. The GP exposes the CDMI (Cloud Data Management Interface) [11], OCCI (Open Cloud Compute Interface) [12], OVF (Open Virtualization Format) [13] and SAGA (Simple API for Grid Applications) [14] implementations as a distributed service using the service-oriented paradigm. Briefly, the key advantages of the GP are:

- It provides (using OCCI and SAGA) a standard way of accessing Cloud and Grid services without tying down services and applications to a particular Cloud or Grid middleware.
- It enables access to any deployed Grid or Cloud middleware through a service-oriented brokering environment.
- It offers a solution that enhances reusability of existing services across disparate domains and applications.
- It facilitates development productivity by offering a service based approach to shield users from complex Grid and Cloud functionality.
- It offers an easy to use approach for enabling clients to port their applications to Grids or Clouds (or both), without installing and maintaining too many Grid and Cloud specific libraries.

The paper proceeds as follows. Section 2 discusses the GP architecture and describes the considerations that went into the design of the platform. Section 3 presents the role of Cloud standards in the Glueing Platform and Section 4 describes an exemplar case study about the usage of the platform. Section 5 describes the enactment and execution details of the platform. The services and components in the platform are then described in Section 6. These are the core APIs that enable the user to access the platform functionality. Section 7 concludes the paper and identifies opportunities for future work.

2. The Glueing Platform Architecture

A high level architecture for the Glueing Platform (GP) is shown in Figure 1. The GP sits between the infrastructure and the applications and provides the required independence from the underlying platforms. A user does not need to know about the lower level APIs or protocols, as components and services are provided to enable access to the underlying compute and data resources in a Grid or Cloud. The services in the platform include authentication and authorization services, scheduling and resource management services, monitoring and provenance services, data access and browsing services and enactment and execution services. To interact with heterogeneous middleware, an adapter based approach has been followed, that implements middleware for each supported Grid or Cloud in the GP. The GP enables applications to be platform agnostic by exposing suitable interfaces that can be accessed to run an application, to browse data, to monitor the execution details or to retrieve the provenance history. The user side interfaces remain the same irrespective of the underlying platform or infrastructure and the GP takes care of the evolving interfaces and components at the platform and infrastructure levels. The API details are described in the following sections.

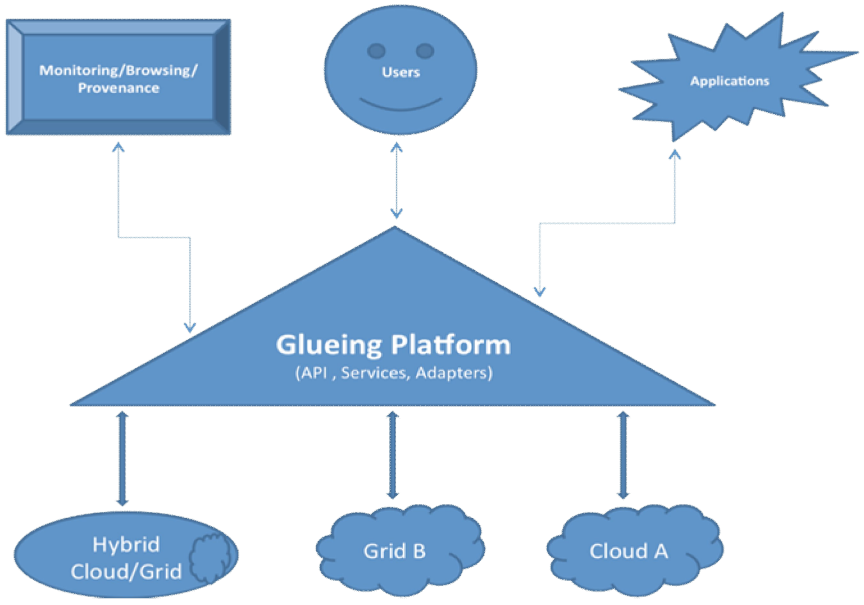

Figure 1: The Glueing Platform Architecture

The GP exposes the OCCI (Open Cloud Computing Interface) as a web service in order to achieve interoperability and portability across the Cloud platforms. The OCCI interface provides an API specification for remote management, monitoring, scaling and deployment of Cloud computing infrastructure. All aspects of virtualized service life cycle management have been considered in the OCCI specification and its implementations. The OCCI specification depends on the CDMI (Cloud Data Management Interface) standard, developed by SNIA (Storage Network Industry Association), to create, update, retrieve and delete data from Cloud. The CDMI manages all aspects of data movement and migration between storage services in Clouds through standard interfaces. There are some overlaps between CDMI and OCCI interfaces but in general the consensus is that data and storage operations come under CDMI and compute, security and infrastructure issues are dealt with OCCI. Further details on how OCCI and CDMI are integrated and how data and virtual machines are exported using OVF (Open virtualization Format) are discussed in section 3.

The GP exposes the SAGA (Simple API for Grid Applications) implementation as a web service to address the objectives described previously. SAGA is an open standard that is defined and maintained by the Open Grid Forum [15] (OGF), which describes a high-level interface for easy programming of Grid/Cloud applications. SAGA is not designed for middleware developers, rather it provides APIs for developing applications using different Grid or Cloud middleware. The SAGA API permits a job to be executed 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 middlewares, remove the need for the use of middleware specific APIs. For example, if an application intends to use three different middlewares then the application has to use a different API set 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, the GP, using SAGA, ‘glues’ a number of client applications together with different middleware using the relevant adaptors. The GP wraps the SAGA API and provides a WSDL binding for client applications. There are numerous advantages of this approach, compared to more traditional means whereby the client applications directly use the SAGA API implementation. The most important reasons for adapting a web service based approach to SAGA are now detailed.

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 a user who wants to submit a job to a Condor [16] cluster deployed as part of an OMII [17] middleware Grid. The client application must be deployed on a node, which has the OMII middleware client installed, and Condor installed and configured in submit only mode. If in 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 applications which make the life of common users quite difficult. 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 Platform can be used as a means to providing functionality, which is not currently provided by SAGA. In this scenario, the Glueing Platform 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.

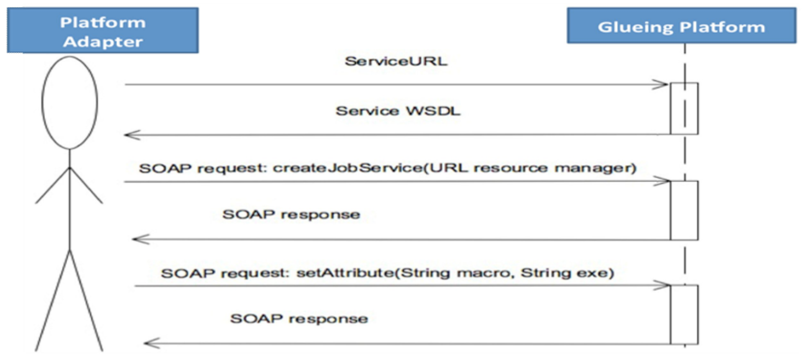
Platform Adapter
Glueing Platform
ServiceURL
Service WSDL
SOAP request: createJobService(URL resource manager)
SOAP response
SOAP request: setAttribute(String macro, String exe)
SOAP response

Figure 2: A Sample Use Case

The GP shields users from these highlighted difficulties, and deploys the adaptors and middleware at the location of the GP host. Clients interact with the GP 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 GP. In order to make the GP SAGA compliant, two separate WSDL descriptions have been provided, one of which points to the full SAGA implementation, and the other describes the extended functionality (which is not currently supported by SAGA). Both WSDL descriptions map to the GP. Additionally, the GP exposes SAGA API functions as web service methods and provides an one-to-one correspondence to the SAGA API functions. Client applications can transparently access the GP by using a SAGA SOAP adaptor, which is an implementation of the Adaptor API provided by SAGA. The clients 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 GP by the SAGA SOAP adaptor, which is responsible for all communication with the GP. The SAGA SOAP adaptor is a middleware between the client applications and the GP. The client applications define, create and submit jobs according to the standard specification of SAGA [18]. These instructions are then translated into SOAP requests by the SAGA SOAP adaptor. SOAP requests are used for communication with the Glueing Platform. The Glueing Platform actually executes the client instructions using SAGA APIs and middleware adaptors. The SAGA SOAP adaptor has methods to discover the GP and to 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 Platform sends back the 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. Figure 2 shows a scenario where a SAGA SOAP adaptor interacts with the GP. The SAGA SOAP adaptor passes the middleware and job information to the Glueing Platform using SOAP objects and the SOAP response is sent back to the SAGA SOAP adaptor from the GP.

3. Cloud Standards and the Glueing Platform

As stated earlier, the standard specifications have been developed to achieve interoperability, portability and to provide elegant communication and integration between heterogeneous components and services that access various features from different Cloud platforms. Cloud platforms are proprietary and vendors have implemented their own interfaces and access mechanisms, and without standard interfaces and specifications, it will be impossible to achieve infrastructure interoperability and application portability across various Cloud platforms. Sensing the gravity of the situation, organizations have worked together to develop industry and community standards for seamless communication, integration and interoperability of Cloud stacks and platforms. The three most important issues that have been addressed include the following:

- OCCI: Monitoring, management, scalability and deployment of Cloud infrastructure. This allows runtime management of compute infrastructure.
- CDMI: Creation, retrieval, removal and updating of data sets. This allows runtime management of storage infrastructures.
- OVF: Exchange of data and virtual machines in a standard machine readable format. This provides a means to package virtual infrastructures that can be imported or exported based on the service deployment requirements.

Figure 3 shows the communication that happens between the Glueing Platform and these standard implementations to achieve application portability and infrastructure interoperability.

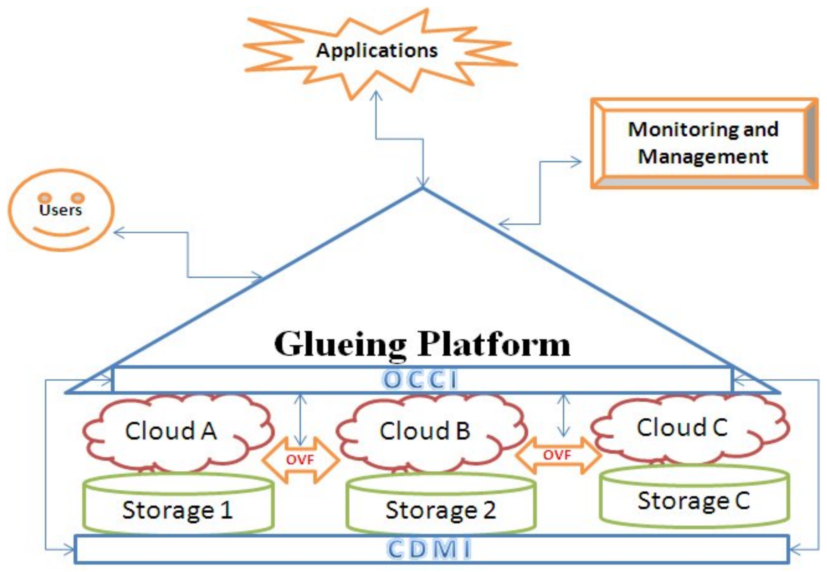

Figure 3: Clouds standards and the Glueing Platform

For example, a user may plan to deploy a Cloud service for a scientific analysis. The first thing the user should do is to create the service, test it and provide the necessary resources to make the service scalable so that it can cope with the user demand. Once the basic building blocks have been made available and the scalability issues have been resolved, the next important thing is to make the deployment exportable so that the same deployment can be made available in case of a failure or if more than one sites need to commissioned for fault tolerance or another infrastructure providers needs to be considered for cost savings.

The service deployment, availability, management and scalability are addressed by the OCCI. The Glueing Platform will trigger the OCCI interface to deploy the service and carry out the scalability and management operations. The storage management is a function of the CDMI and the Glueing Platform invokes the CDMI interface to store, retrieve, update and remove files from the Cloud storage devices. In case the service was already deployed and a new instance of the service needs to be created, the Glueing Platform will call the OVF interface to export or import the service instances. The virtual machine instances can be migrated between the resources within or between the data centres using the Distributed Management Task Force (DMTF) based specification and implementations of the OVF standard. The Glueing Platform, using the standards specifications and tools, offers platform and infrastructure portability. Users are free to move their applications and services across different Cloud platforms and infrastructures.

The Glueing Platform is the first point of contact for users and applications to deploy or use services and to manage and monitor the infrastructures. The applications do not have to access platform specific libraries or components to provide the service and are free to move to another infrastructure or a Cloud platform. The Glueing Platform takes care of all the low level details that deal with the standards interfaces and components. The Glueing Platform architecture is flexible enough to accommodate new requirements and standards such as audit and security from Cloud Security Alliance (CSA). The Glueing Platform brings all these into a single framework and users are provided a simple to access service and in return they get benefits from all the standards based implementations of tools and technologies for deploying, operating and managing the infrastructures. This approach makes the applications and services to be widely deployed on a number of platforms, avoids vendor lock-ins and promotes interoperability and application portability.

4. Case Study

The architecture diagram (figure 4) shows how an application contacts the GP. The pipeline service, shown in the figure, is one of the potential applications of the GP. This service uses the SAGA APIs and SAGA SOAP adaptors to communicate with the GP. The SAGA SOAP adaptor accesses different methods of the GP 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 GP by the SAGA SOAP adaptor. The GP, as shown in the diagram, can communicate with different Grid middleware such as OMII/GridSAM [19] or gLite etc. This allows client applications, such as pipeline service, to use Grid resources provided through different middleware. The GP uses the JavaGAT [20] middleware, which is a SAGA implementation, and OMII adapters to communicate with the OMII middleware.

GP client applications such as the Pipeline service in the neuGrid project [21], allow clinicians and neuroscientists to create neuro-imaging pipelines, in a user-friendly environment, for a series of automated transformations on 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 modified for their optimal execution over the Grid.

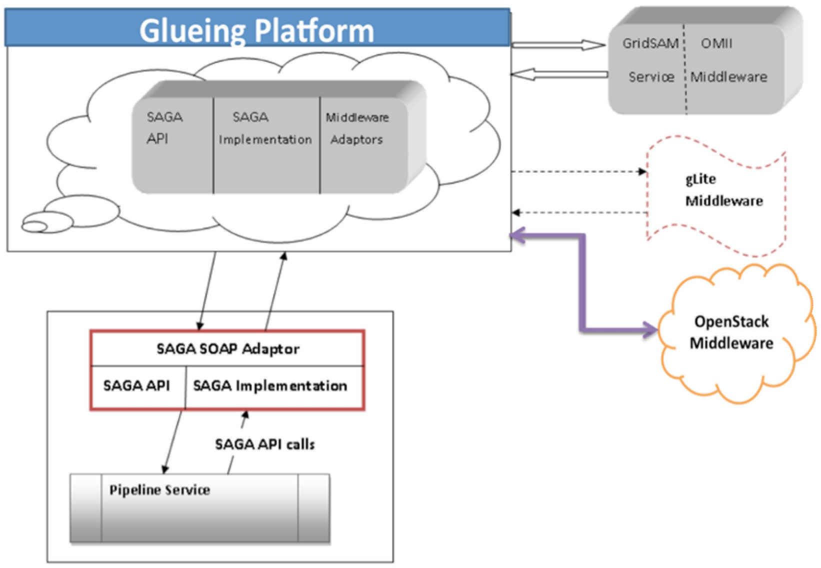

Figure 4: Glueing Platform and Grid Middleware

The neuro-imaging pipeline service, using the GP, submits the workflows to the Grid for their execution and the results of execution are returned to the client interface of pipeline service. A simplified design of the pipeline service is shown in figure 5.

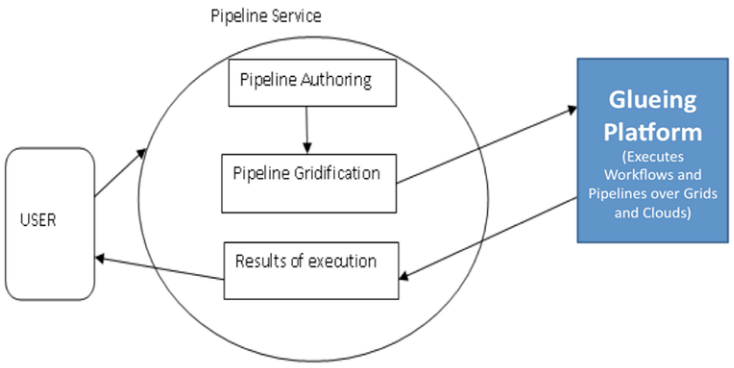

Figure 5: Pipeline Service

The neuro-imaging Pipeline Service passes workflows, or sequences of processes, to the Glueing Platform. The Glueing Platform exploits the SAGA system and executes the processes in the workflows on the Grid using appropriate Grid middleware. The results of execution are passed over to a provenance service and eventually reach back to the client.

5. Execution and Enactment in the Glueing Platform

The Glueing Platform offers an environment where workflows and applications can be executed in a platform neutral manner. The components of the Glueing Platform to achieve a platform-agnostic enactment are shown in figure 6. The Platform treats every user activity, whether it may be an application, user query or series of actions, as a workflow. This enables the platform to perform a series of actions on the queries or users' steps to plan and orchestrate the execution for a maximum throughput. This also enables the users to optimize their queries and workflows by performing a number of intelligent actions based on the concepts described in [22] during the planning phase to save costs and times.

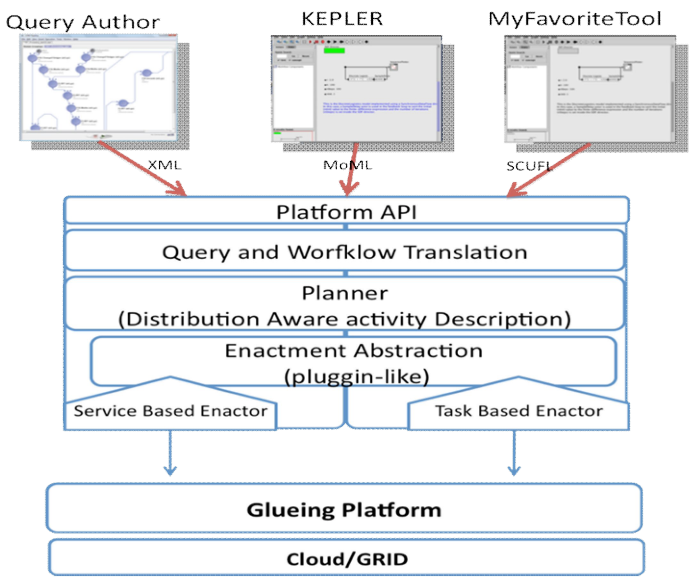

Figure 6: Enactment in the Glueing Platform

Every user query or job is charged by the Cloud providers; minimizing the number of jobs will save costs. Similarly, by planning their activities, users can remove redundant tasks, group similar tasks or schedule their activities on resources in Clouds which can provide users improved throughput. Considering the consequence of these actions, queries or workflows can do more in less time, and can lead to intelligent and adaptive practices in Clouds [23]. Users can select resources on which they can execute their workflows or jobs and they also have control on the choice of the authoring environment. They can author their queries or workflows in an environment such as Kepler, or they may use a simple command line tool, as their actions can be passed on to the planning component in a structured format. As shown in figure 6, the architecture of the Glueing Platform is flexible and any suitable authoring environment can be accommodated.

After authoring a query or workflow, the users invoke their submission. At this stage, several things can happen: firstly the authored query, process or workflow that is represented in a format which is native to the authoring environment can be translated into a common platform model. This model can be an object oriented model of the query or workflow or this can be translated into another format which is convenient to the planning engine. After workflow translation, architecturally the workflow should be planned for its execution. In the planning stage, a number of intelligent strategies can be evaluated for an optimal execution of the workflows or processes. The planned workflows or jobs can then be submitted to a Cloud or Grid for execution. The execution can be task based or there could be pre-deployed service instances to support the execution. In both cases, an appropriate enactor prepares the environment for execution by discovering suitable resources and scheduling the jobs/workflow onto these resources by performing matchmaking and mapping processes. Once appropriate resources have been shortlisted, the enactor component schedules the jobs/workflows for execution. A monitoring component keeps track of the execution process and a status update is reported to the user in intervals. The results, execution traces and their provenance logs are sent to the users to make it a transparent execution.

6. Platform Components and Functionality

The SOAPAdaptor as described in section 5 requires a Service Endpoint URL to communicate with the Glueing Platform. 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 SOAPAdaptor calls the published methods using SOAP requests and the Glueing Platform sends back SOAP responses to the SOAPAdaptor. The SOAPAdaptor then translates the SOAP response and returns the execution results to the client in the form of Java or SAGA specific objects.

Figure 7 shows a scenario where the SOAPAdaptor interacts with the Glueing Platform. The SOAPAdaptor passes the middleware and job information to the Glueing Platform using SOAP objects and the SOAP response is sent back to the SOAPAdaptor from the Glueing Platform. The exposed functions of the Glueing Platform are specifically discussed in detail in the following sections.

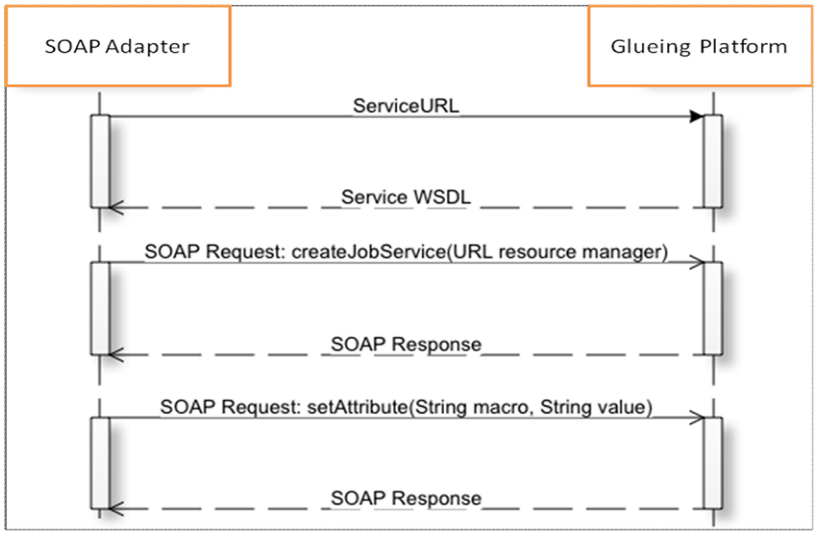
SOAP Adapter
Glueing Platform
ServiceURL
Service WSDL
SOAP Request: createJobService(URL resource manager)
SOAP Response
SOAP Request: setAttribute(String macro, String value)
SOAP Response

Figure 7: Information flow between SOAPAdapter and the Glueing Platform

The GP exposes SAGA APIs in the form of a web service (or web services) 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 Platform. The platform loads the appropriate middleware adaptor, based on the middleware information. For example if a client wants to run jobs or access data on an OMII based infrastructure, the client specifies an OMII/GridSAM middleware; the GP sets a system property "JobService.adaptor.name" equal to "javaGAT". This system property behaves like an environment variable for the SAGA engine to narrow down the selection of middleware adaptors. The GP then makes sure that JavaGAT loads a GridSAM adaptor by creating an appropriate session and context for SAGA. The 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 GP 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 GP creates a job, based on the job information sent by the client application. The GP 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; JobService, JobDescription, Job and JobSelf. The order in which the GP uses this job management and other APIs is as follows:

6.1 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 GP 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”).

6.2 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 GP, using JobDescription, defines a job as: <JobDescription> jd.setAttribute (JobDescription.EXECUTABLE, “/bin/echo”) and <JobDescription> jd.setVectorAttribute (JobDescription.ARGUMENTS, new String[] {“hello”}).

6.3 Data Stage-in/-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 GP and by passing it source and target URL parameters.

6.4 Job Creation

The GP 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 GP 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.

6.5 Asynchronous Job Execution

The GP 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().

6.6 Replica Management

The GP 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 GP 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")).

6.7 Job Monitoring

The GP monitors a job during its lifetime using callbacks, and it provides two methods of retrieving monitoring information. One is to use the higher level implementation, by which the client application can invoke the monitor (String jobId) function of the GP to get complete monitoring information, associated with a particular job id. In this implementation the GP stores the monitoring information of the job in a String and it is returned to the client when (s)he invokes the monitor. The other method of monitoring a job is to use a 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.

6.8 Data Querying

The GP 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 GP.

Example: The GP 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 GP fetches the list as: nsDir.find("*.jpg").

7. Conclusions and Future Directions

The Glueing Platform addresses some major requirements of users who wish to adopt a flexible approach to accessing the Grid and Cloud. The heterogeneity of distributed resources and details of Grid or Cloud middleware architectures will be obscured from users. The GP also hides complexities of interfacing with different Grid and Cloud middleware, which will allow access to Grid and Cloud resources through a set of high level functions. This is achieved by exposing CDMI, OCCI, OVF and SAGA APIs, all communication being achieved through middleware adaptors. This shields the low level middleware difficulties from the user and assists them in using resources with little knowledge. The design of the GP is based on service-oriented architecture principles, which will help different services and applications to use service functionalities through standardized interfaces. This will also allow other client applications to use the Platform features by inspecting its WSDL, available online at the service endpoint URL.

Future work includes the provision of support for resource discovery (building on the work reported in [24]) and improving it to a level so that it may be used for production quality needs. Support for workflow composition and enactment is another major issue that needs to be addressed [26]. The support for Cloud platforms such as OpenStack and EUCALYPTUS [25] also needs to be provided, along with support for application hosting, job distribution and concurrency. The infrastructure interoperability across disparate platforms also needs a solution for semantic interfaces in Clouds and Grids [27][28] and this is one of the services that we need to integrate in the Glueing Platform in future.

Author

References

[1] Aristotelis Kretsis, Panagiotis Kokkinos, Emmanouel Varvarigos, Developing Scheduling Policies in gLite Middleware, In Proceedings of the 20099th IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID '09), IEEE Computer Society, Washington, DC, USA, pp: 20-27

[2] OpenStack Open Source Cloud Computing Software, http://www.openstack.org/, Last accessed: December 5, 2011

[3] Martin Koehler, Matthias Ruckenbauer, Ivan Janciakm, Siegfried Benkner, Hans Lischka, Wilfried N. Gansterer, A Grid services cloud for molecular modelling, Workflows, International Journal of Web and Grid Services (IJWGS), 2010, Vol. 6, No.2, pp: 176-195

[4] Microsoft Windows Azure Platform, www.microsoft.com/WindowsAzure, Last accessed: December 5, 2011

[5] AWS, Amazon Web Services, http://aws.amazon.com/, Last accessed: December 5, 2011

[6] Minoru Uehara, Composite RAID for rapid prototyping data Grid, International Journal of Web and Grid Services (IJWGS), 2011, Vol. 7, No. 1, pp: 58-74

[7] Keith Jeffery, Burkhard Neidecker-Lutz, The Future of Cloud Computing, opportunities for European Cloud Computing beyond 2010, Expert Group Report public version 1.0, http://cordis.europa.eu/fp7/ict/ssai/docs/executivesummary-forweb_en.pdf

[8] Attila Kertész, Péter Kacsuk, Grid Interoperability Solutions in Grid Resource Management, IEEE Systems Journal, 2009, Vol. 3 Issue 1, pp: 131-141, ISSN: 1932-8184

[9] Rack Space Cloud Hosting, http://www.Rackspace.com, Last accessed: December 5, 2011

[10] AWS, Amazon Elastic Compute Cloud, http://aws.amazon.com/ec2/, Last accessed: December 5, 2011

[11] CDMI, Common Data Management Interface, SNIA Technical Position: Cloud Data Management Interface (CDMI) v1.0.1, http://www.snia.org/cdmi, Last accessed: December 5, 2011

[12] OCCI, Open Cloud Computing Interface, http://www.occi-wg.org/, Last accessed: December 5, 2011 and OGF working group, http://www.ogf.org/gf/group_info/ view.php ?group=occi-wg, Last accessed: December 5, 2011

[13] OVF, Open Virtualization Format, http://en.wikipedia.org/wiki/Open_Virtualization _Format, Last accessed: December 5, 2011

[14] Shantenu Jha, Hartmut Kaiser, Andre Merzky, and Ole Weidner, Grid Interoperability at the Application Level Using SAGA, In Proceedings of the Third IEEE International Conference on e-Science and Grid Computing (E-SCIENCE '07), IEEE Computer Society, Washington, DC, USA, pp: 584-591

[15] Open Grid Forum, http://www.Gridforum.org/, Last accessed: December 5, 2011

[16] Douglas Thain, Todd Tannenbaum, Miron Livny, Distributed Computing in Practice: The Condor Experience Concurrency and Computation: Practice and Experience, 2005, Vol. 17, No. 2-4, pp: 323-356

[17] E-Research Middleware: The Missing Link in Australia's e-Research Agenda, Discussion Whitepaper, The Commonwealth of Australia, Department of Education, Science and Training, National Research Infrastructure Taskforce, March 2004

[18] A Simple API for Grid Applications (SAGA), www.Gridforum.org/documents/GFD.90.pdf, Last accessed: December 5, 2011

[19] William Lee, A. Stephen Mcgough, John Darlington, Performance evaluation of the GridSAM job submission and monitoring system, In UK e-Science All Hands Meeting, 2005

[20] Rob V. van Nieuwpoort, Thilo Kielmann, Henri Bal, User-friendly and reliable Grid computing based on imperfect middleware, In Proceedings of the ACM/IEEE Conference on Supercomputing (SC'07), 2007, New York, NY, USA

[21] Alberto Redolfi, Richard McClatchey, Ashiq Anjum, Alex Zijdenbos, David Manset, Frederik Barkhof, Christian Spenger, Yannick Legré, Lars-Olof Wahlund, Chiara Barattieri, Giovanni B Frisoni, Grid infrastructures for computational neuroscience: the neuGRID example, Future Neurology, 2009, Vol. 4, No. 6, pp: 703-722

[22] Yongyun Cho, Jongbae Moon, Iksu Kim, Jongsun Choi, Hyun Yoe, Towards a smart service based on a context-aware workflow model in u-agriculture, International Journal of Web and Grid Services (IJWGS), 2011, Vol. 7, No. 2, pp: 117-133

[23] Qibo Sun, Shangguang Wang, Hua Zou, Fangchun YangQSSA, A QoS-aware Service Selection Approach, International Journal of Web and Grid Services (IJWGS), 2011, Vol. 7, No. 2, pp: 147-169

[24] Chia-Chuan Chuang, Shang-Juh Kao, Adjustable flooding-based discovery with multiple QoSs for cloud services acquisition, International Journal of Web and Grid Services (IJWGS), 2011, Vol. 7, No. 2, pp: 208-224

[25] Daniel Nurmi, Rich Wolski, Chris Grzegorczyk, Graziano Obertelli, Sunil Soman, Lamia Youseff, Dmitrii Zagorodnov, The Eucalyptus Open-Source Cloud-Computing System, In Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID '09), IEEE Computer Society, Washington, DC, USA, pp: 124-131

[26] Khawar Hasham, Antonio Delgado Peris, Ashiq Anjum, Dave Evans, Dirk Hufnage, Eduardo Huedo, Jose M. Hernandez, Richard McClatchey, Stephen Gowdy, Simon Metson, CMS Workflow Execution using Intelligent Job Scheduling and Data Access Strategie, IEEE Transactions on Nuclear Science (IEEE TNS), 2011, Vol. 58, Issue: 3, pp: 1221-1232 ISSN: 0018-9499

[27] Andrew Flahive, David Taniar, Wenny Rahayu, Ontology as a Service (OaaS): a case for sub-ontology merging on the cloud, The Journal of Supercomputing, 2011, pp: 1-32

[28] Andrew Flahive, David Taniar, Wenny Rahayu, Bernady O. Apduhan, Ontology tailoring in the Semantic Grid, Computer Standards and Interfaces, 2009, Vol. 31, No. 5, pp: 870-885