Grid Enabling Legacy Applications through a Standard Job Submission Interface

A. Stephen McGough, William Lee and Shikta Das
Imperial College London
London, UK
{asm,wwhl,shd}@doc.ic.ac.uk

Abstract

The potential of the Grid for users, software developers and resource owners is well appreciated. Users of the Grid are abstracted from its complexity, while software developers are provided with tooling which allows them to develop applications. Resource owners are able to expose their resources potentially for financial reward. However, as most of the applications which currently exist pre-date the concept of the Grid, they lack the appropriate functionality to exploit it. In this paper we propose the use of a standards based job submission system which is capable of deploying legacy applications onto existing resources within the Grid along with the use of web-based portals to expose these applications to the end users. We further propose that applications which comprise a number of legacy tasks can be handled through the use of a workflow enactment service submitting each of these tasks through the standards based job submission system. We exemplify our approach through the e-Protein project by taking their existing proteome annotation software and exposing it as a Grid service.

1. Introduction

The Grid [19] provides a powerful abstraction for the use of heterogeneous computational resources. For the user this means that the implementation used and the resources selected can be determined automatically without the user ever becoming aware of their existence and complexity. Moreover, the user can be presented with a simplified view of the infrastructure that resembles the experience of using a personal computer. A web-based portal interface is increasingly becoming a de-facto choice for delivering software applications. For the resource owner it provides an environment in which they can encapsulate complex software and computational provision, potentially for financial reward [16], to be delivered through a browser. For the software developer, it provides a forum in which their software can be exposed without the need for users (or the software developers) to own the hardware necessary to execute the application [20]. This removes the requirement for users to own resources, software developers to support complex installations and allows services on the Grid to determine the best place to deploy software – to provide an appropriate quality of service to the user.

Thus it would seem that the software developer may make their application available as a web based service accessible through a web portal, a resource provider provide a hosting environment environment for these services and the end user access the application through their web browser.

This utopian view is somewhat marred by the fact that much of the existing software developed pre-dates the existence of the Grid. The majority of existing legacy applications are based around executables – commonly referred to as jobs – which are often platform specific. Legacy applications are often designed to be executed from the command line on an interactive resource which is easily accessible by the user. Without significant effort these executables cannot be converted into web based services.

The Grid provides an environment where jobs are submitted to remote resources without the ability to interact with them. The configuration of the remote resource is also often unknown. The availability of libraries and other elements that are required for execution are often difficult to ascertain. In most cases, these executables can work in batch mode where the inputs to the executable can be predetermined and sent to a resource along with the request to run the executable. However, in some cases the executable will require some form of interactivity during its execution. It may be possible in some cases to alter the original executable so that it can be run in a batch manner. However, this is not always possible due to the unavailability of the source code.

In many cases these executables can be wrapped inside of an "environment container" which maps the interactive nature and executable requirements into that of a batch job

and ensures that the environment in which the executables run matches its requirements. Thus in general we are able to use most existing software without the need to alter it in any manner.

A portal interface can encapsulate the functionalities provided by an executable that is anchored on a particular resource. However, this prevents the software from being made available through the Grid without tying it to a particular hardware or location. Distributed Resource Managers (DRMs) are mechanisms for exposing many Grid resources and allowing arbitrary executables to be enacted upon them.

There are many Distributed Resource Managers in existence capable of executing legacy code; such as Condor [29], Globus [18], Grid Engine [4] LSF [6]. However, each of these systems provides its own proprietary job submission interface using a proprietary job description language. Thus submitting jobs to these underlying Grid fabrics require large amounts of knowledge to use. This homogeneity can be hidden from the end user by writing Web portals capable of understanding each of the different underlying DRM submission interfaces. However, this would lead to highly complicated Web portal services requiring significant work each time a DRM system updates, changes or adds a submission interface.

In this paper we propose the use of a common (standardised) job submission interface for deploying jobs onto the Grid. This allows the Web portal implementer to concentrate on the usability and development of a portal service independently of the complexity of the job submission process. They need only be aware of one (standard) job submission interface thus allowing simpler portals which are independent of changes in the underlying fabric of the Grid.

1.1. Application Workflows

Many applications comprise of a number of sub-jobs, often referred to as components, which are performed to achieve the overall application. These components may be composed together within some form of control structure (a control script) and submitted to the underlying Grid Fabric. This will allow the whole application to be executed. However, it reduces the ability for services on the Grid to best match the quality of service requests imposed by the user, as the Grid will have to deploy the entire application to a single resource. However, if we allow the set of components to be exposed to the Grid independently along with a workflow document describing how these components interact it is then possible for a Grid brokering service to determine the best place for each component to be executed. This will potentially allow more efficient execution of the workflow. Parallelism within the workflow may be exploited and resilience added in the cases where resources fail.

In the next section we outline the architecture for our approach. In section 3 we describe GridSAM our standards compliant job submission interface. We describe the use of the Grid brokering service in Section 4 followed by a discussion of the workflow service in Section 5. The approach of wrapping jobs within an "environment container" is discussed in Section 6. We show how we are using this architecture within the e-Protein project to perform proteome annotation in Section 7 before concluding within Section 8.

2. Architecture

In previous work [24, 25] we have proposed an entire Grid workflow architecture in which elements of the architecture can be selected in an à-la-carte fashion. We illustrate here a combination of these elements which can be combined to provide a system capable of efficiently executing legacy applications on the Grid. Figure 1 illustrates how these elements can be combined together. We briefly describe these services below before expanding on them more fully later in the paper.

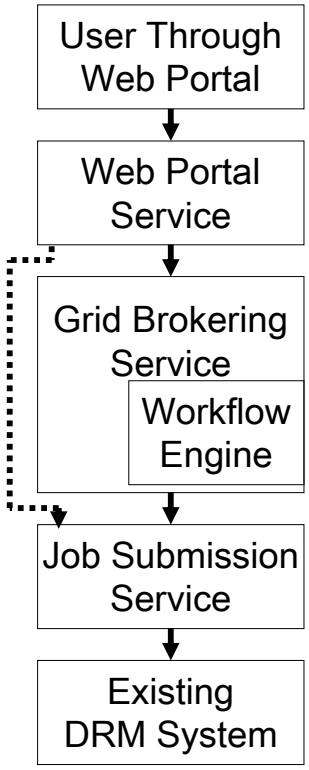

Figure 1. Architecture

- **Web Portal**. The user interface for interacting with an application. This can be delivered through a standard Web Browser.

- **Web Portal Service**. This portal should be designed with end-user usability in mind. This element will be specific to the application that is to be deployed. To expose an application the developer needs to develop a portal service which uses a standard job submission language to submit the whole application to a Grid Brokering Service or if the use of Brokering is not required the job can be submitted directly to a Job Submission system. Alternatively if the application consists of a number of components along with a workflow description document, these can be submitted to the Brokering service and enacted through a workflow Engine. We will illustrate the Portal service for the e-Protein project in section 7.

- **Grid Brokering Service**. The Brokering service is tasked with determining the "best" place to deploy each job into the Grid. Jobs are then dispatched to the appropriate Job Submission system for execution. The Workflow service can be combined with the Brokering service (as in our exemplar). The workflow aware Brokering service is capable of selecting not only the "best" place for each component in the workflow but also ensure that the combination of these places is good for the entire workflow to complete successfully. We discuss the Brokering service further in Section 4.

- **Workflow Engine**. The Workflow engine takes the workflow description document along with the individual components of which it is composed. The workflow engine is then responsible for dispatching job submissions for the individual components according to the sequencing constructs to ensure that the entire workflow is enacted as required. We discuss the use of the workflow engine in section 5.

- **Job Submission System**. In order to abstract away from the heterogeneity of the underlying Grid fabric we are developing a standards based Grid Submission and Monitoring (GridSAM) [22, 5] service. GridSAM is designed to be situated between existing DRM systems providing a standard job submission interface into these DRM systems. The power of GridSAM comes from its use of the emerging standards in job submission along with its modular architecture which makes it easy to extend GridSAM to new DRM systems. We describe GridSAM in Section 3.

- **Existing DRM System**. There are many DRM systems which manage submission of jobs to the underlying resources. We seek to exploit as many of these as possible without making changes to them. As we see these as the underlying fabric of the Grid we do not discuss them any further in this paper.

- **Wrapping Executables**. In order to execute an application on a resource in the Grid it is often required to mimic the environment which is expected by the application. This may be as simple as ensuring that libraries required by the application are available or more complicated, such as providing a whole environment which is able to interact with the application in such a way that the application believes that it is operating in its normal environment. The wrapper is discussed further in Section 6.

3. GridSAM

There are many Distributed Resource Management systems in existence for launching jobs efficiently onto computational resources. However, most systems adopt proprietary languages and interfaces to describe and interact with the job launching process. This leads to the requirement that a user needs to learn a large number of job description languages and deployment mechanisms to exploit a wide variety of Grid resources.

Web Services have been recognised as the preferred technology to build distributed services in the Grid context. It provides an inter-operable approach to message-oriented machine-to-machine interaction. Commercial adoption of Web Services has catalysed the development of industrial-strength platforms for deploying high-performance Web Services. Moreover, the Grid community has gathered pace in recent years to define standards for core functionality of Grid Computing. This is required to achieve interoperability across organisational and technical boundaries in order to attain the vision of a global computational network. In particular, the Job Submission Description Language (JSDL) [14] standardised through the Global Grid Forum (GGF) [10] and the evolving Open Grid Services Architecture Basic Execution Service (OGSA-BES)[7] interface from the GGF are essential standards to promote interoperability among DRMs.

Here we present a standards-based job submission system, GridSAM[1], undertaken as part of the Open Middleware Infrastructure Institute (OMII) Managed Programme[13], as one of the first systems adopting JSDL and Web Services for job description and interaction. GridSAM utilises Web Services interfaces, and conforms with the emerging OGSA-BES specification along with the JSDL job description language. GridSAM provides a transparent and efficient bridge between users and existing DRM systems, such as *Condor* [29], *Globus* [18] and *Grid Engine* [4]. The encapsulation allows the functions to be used through the GridSAM Web Service as well as utilising it as an independent library.

[1] Downloadable from `http://www.lesc.ic.ac.uk/gridsam`

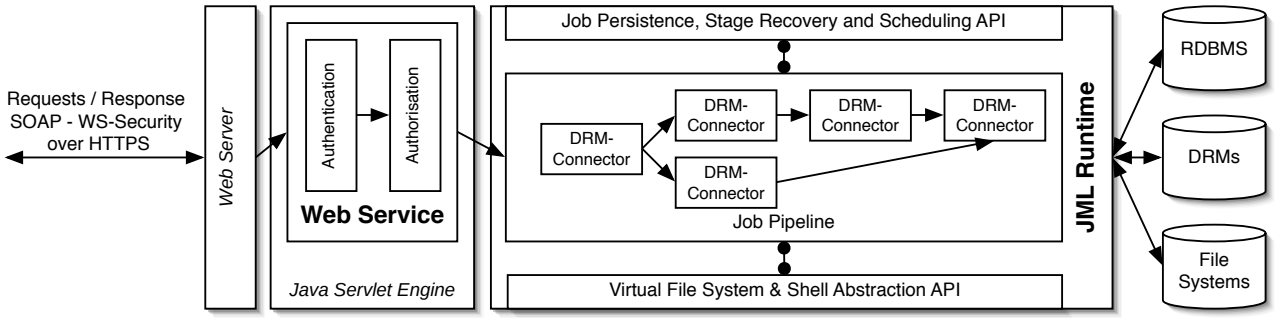

Figure 2. GridSAM System Architecture

3.1. GridSAM Architecture

Int this section we present the architecture for GridSAM as illustrated in Figure 2. Requests for execution of jobs arrive through the Web Service interface in JSDL documents while the underlying DRM system is interacted with through a collection of DRMConnectors representing different functionalities of the DRM.

The objective of GridSAM is to let users execute applications through existing distributed resource managers transparently. The Transparency is achieved through the use of a common job description language, JSDL, and a uniform networked access interface, Web Services, OGSA-BES. The core function of GridSAM is to translate the submission instructions specified in a JSDL document to a set of resource specific actions to stage, launch and monitor a job. This function is encapsulated in the GridSAM *Job Management Library (JML)*.

The role of the *JML* is to orchestrate the execution of a set of *DRMConnector*s - reusable components encapsulating job management actions. These components are composed by the deployer into *a network of stages* resembling a job launching pipeline. The *JML* runtime alleviates system engineers from programming common tasks, such as persistence, failure recovery and concurrency management by exposing these through the *JobManager* API.

The GridSAM Web Service makes available the *JML* through a Web Service interface. It is implemented as a Java JAX-RPC-compliant Web Service deployable in any Java Servlet compliant container. This opens up the choice of deployment platforms depending on the scalability requirements. The Web Service interface makes use of HTTPS transport security and the OMII WS-Security framework to protect message exchange as well as authenticating and authorising users. The Web Service interface demonstrates the use of the *JML* as a networked multi-user service. It is envisaged that the *JML* can be embedded in other frameworks (e.g. portal, Grid applications) offering different modes of interaction.

Below we discuss further some of the advanced features of GridSAM which help in abstracting both the user of GridSAM from the underlying fabric of the Grid and also the developer of DRMConnectors for GridSAM thus making it easier for new DRM systems to be exposed.

Submission pipeline as a network of stages. The GridSAM pipeline is constructed as a *network of stages* connected by event queues. This design is inspired by the *staged event-driven architecture* (SEDA) [23]. Instead of treating each job submission request as a single submission action, it is decomposed into robust stages that may be individually conditioned to load by thread-holding or filtering its event queue. A number of exemplar systems (e.g. *Haboob* web server) have demonstrated the use of this principle to deliver robustness over huge variations in load.

Figure 3 depicts a pipeline that launches jobs onto a *Condor* pool. Each stage in the pipeline is an implementation of the *DRMConnector* event handler interface. *DRMConnector* instances encapsulate a specific functionality that is triggered by an incoming event (e.g. stage-in event) associated with a job. Once the *DRMConnector* has completed its operation, it may enqueue events onto another stage. It effectively passes control to the next stage in the pipeline asynchronously. Long-running stages that perform blocking operations (e.g. reading files) can potentially be broken down further into sub-stages by using non-blocking I/O libraries.

The explicit control boundary, introduced by the queue between stages, improves overall parallelism in the system. The simple message-oriented event-based interface allows system engineers to focus on the DRM-specific logic, rather than the details of concurrency and resource management. Moreover, the event-based interface echoes the *Command* design pattern[17] that encourages component reuse and action encapsulation. For example, the *Forking* and *Secure*

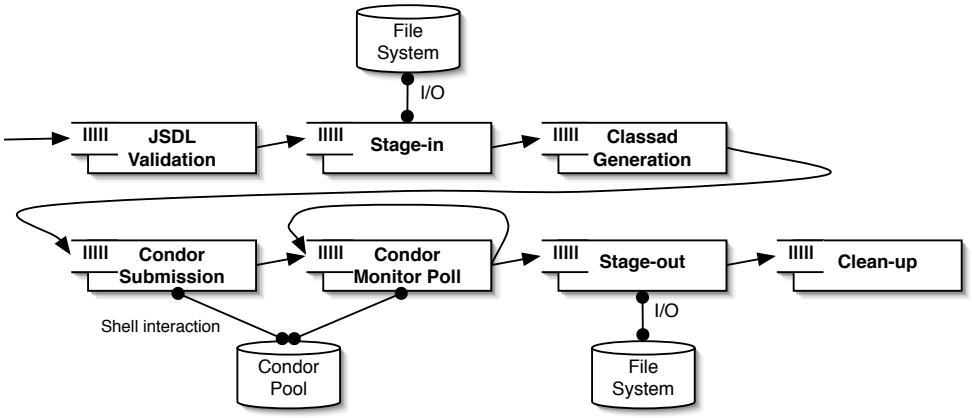

Figure 3. Submission pipeline for Condor job in GridSAM

Shell pipeline share most of the pipeline components apart from the launching stage. Representation of state is completely encapsulated in the incoming event message, the *DRMConnector* can be distributed easily across a cluster without complex state management.

Fault recovery. The adoption of an event-based architecture allows individual stages to be restarted upon failure by persisting the event queues and the information associated with each job instance. The GridSAM *JML* provides the *JobInstanceStore* API for persisting per-job information that needs to be carried between stages and inspected for monitoring purpose; GridSAM uses the *Hibernate* [26] toolkit to provide transactional object-to-relational mapping. *DRMConnector* implementations are agnostic to the underlying persistence mechanism (e.g. in-memory replication, RDBMS persistence). *JobInstance* objects are stored in JDBC compliant RDBMS databases along with the event queues by default.

When the *JML* is initialised the event queues and the scheduler are reinstated. A previously failed job pipeline will be restarted from the beginning of the failed stage instead of the beginning of the pipeline. A stage can perform the necessary recovery action by undoing the failed resource-specific actions or perform the idempotent operation again.

Concurrency management. Each stage in the pipeline is served by a pool of threads that consume events from the stage queue and invoke the stage-specific *DRMConnector*. GridSAM builds on the *Quartz* framework [27] to schedule stages and allocate threads. Welsh et al [23] described in the original SEDA proposal the role of an application controller to dynamically self-tune resource management parameters based on run-time demands and performance targets. For example, the number of threads allocated to a stage can be determined automatically without *a priori* knowledge of job arrival rate and perceived concurrency demands. Although *Quartz* currently lacks the dynamic load adaptation support, it provides advanced date-based scheduling, fault recovery and clustering support that is unavailable in other frameworks. This can be accommodated in the future with the extensible *JML* framework using the *Java Management Extension* as an instrumentation tool.

4. Grid Brokering Service

GridSAM provides a mechanism to transparently execute a job on a remote resource within the Grid. However, it provides no mechanism to select which resources to use. Although many of the DRMs that GridSAM sits on-top of may be able to perform this for their local set of resources. Thus to make the best use of the Grid a brokering mechanism is required which is capable of selecting the best resources (and hence the best GridSAM instance) for a particular job.

In previous work [24, 31, 30] we have proposed an architecture for a brokering (scheduling) system for the Grid. We leverage this work for this paper. We discuss here how to handle a workflow though a single job submission can be considered as a workflow with only one component without loss of generality.

The broker takes an abstract workflow and translates this into a concrete workflow in which the software selected for each component along with the resource on which to execute the component have been determined. This process may be carried out in a one pass process [31] or through a constantly updating process [30]. In either case the elements of the worklfow which have been made concrete can be sent to a workflow service for execution. The broker monitors the workflow to determine if it performs as expected. If the workflow does not perform as expected or the

broker becomes aware of new functionality within the Grid then it may choose to re-evaluate the workflow [24]. Many of the workflow optimisation stages presented in this paper can be applied here in the brokering service.

It should be noted that the brokering service is tasked with providing the "best" results for all users of the Grid. It must balance all user requirements against the resource owners desires and the software providers desires. It is a hard task to balance all of these requirements and desires and is an area of ongoing research.

We have developed a Grid brokering framework for the Imperial College e-Science Networked Infrastructure (ICENI II) [24] which we are using for this work. The functionality of this service and the brokering algorithms that are available for it is an area of open research. We are tracking the standardisation work going on within the GGF in the areas of brokering through both the Open Grid Services Architecture (OGSA) [9] and OGSA-Resource Selection Service (OGSA-RSS) [8] working groups.

5. Workflow Service

Often a user will make use of a number of separate applications. The user may choose to execute each of these applications in turn by making separate job submissions to the Grid. This may result in hundreds if not thousands of calls. It is therefore more convenient to use a workflow service which can orchestrate all of these job submissions. The workflow service is tasked with taking the set of these components and a workflow description document and ensuring that these items are enacted on the appropriate resources.

Our approach here is to use off the shelf commodity solutions wherever possible. As such we are adopting the Business Process Execution Language (BPEL) [15] workflow description language and are working with the ActiveBPEL [1] workflow engine.

Although the use of workflows may be the most appropriate paradigm for dealing with multiple job submissions this may not be the best medium to present to the end user. The end user should be protected from the need to learn workflow languages or theory. Instead the Web Portal service should be designed in a manner that provides an interface to the end user which matches in with the knowledge of the user and then translates this into a workflow document.

6. Wrapping Executables

Legacy applications often need a specific environment in which they can run. This may require (though is not limited to): libraries; specific files in pre-defined locations; access to databases; along with the handling of interaction with the executable. Instead of submitting the actual legacy application to a Grid resource an environment container, or wrapper, is deployed. The wrapper is responsible for generating the environment which is required for the legacy application, dealing with interactions with the application and cleaning up after the execution.

The complexity of the wrapper depends on the requirements of the application to be wrapped. The closer an application is to a batch executable the less the wrapper is required to provide. In general the actions provided by the wrapper are as follows:

- **Generate file environment**. The wrapper needs to place relevant files on the resource in locations where the application expects to find them. This may be to uncompress a set of libraries and files into a predetermined location.
- **Set up environment variables**. In many cases environment variables are required for the application to operate correctly. These need to be set up at this stage.
- **Database access**. If the application requires access to a database this access needs to be provided at this stage. This may be achieved through the use of OGSA-DAI [12] or through a SSH tunnel [28].
- **Enacting the application**. Starting up the application.
- **Application interaction**. This can often be achieved through streaming the standard input and output for the application through the wrapper with the wrapper providing the correct input as appropriate.
- **Collecting files for staging back**. This may be to compress a large number of files into a single archive which can be staged back from the resource.
- **Cleaning up**. Once the application has finished all changes and temporary files need to be removed.

7. Proteome annotation in e-Protein

While various tools are available that solve the larger problems of protein structure prediction, it is necessary to build software to integrate them together to perform protein annotation. The e-Protein [3] project is a pilot initiative funded by the Biotechnology and Bioinformatics Sciences Research Council (BBSRC), to combine structural and functional genome annotation databases from Imperial College London, University College London and European Bioinformatics Institute, through a single interface using the Distributed Annotation System (DAS) [2] and utilising Grid middleware technology. This is an on-going, open source initiative allowing a bioinformatician to functionally annotate their proteins sequences.

We have taken a bioinformatics workflow developed by the Structural Bioinformatics Group at Imperial College

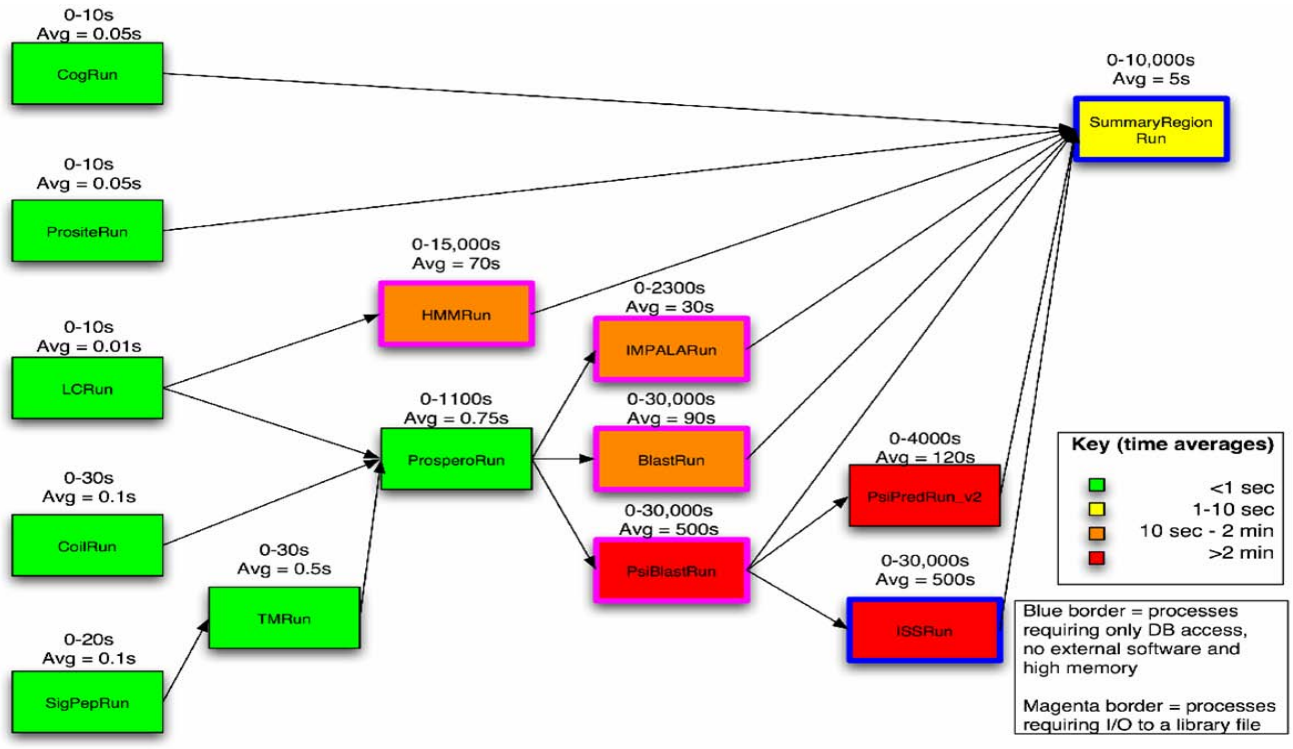

Figure 4. The components within the e-Protein workflow

London. This provides a structural and functional database, called 3D-GENOMICS [21] along with an annotation workflow. We have developed a Web Portal for running the annotation workflow over the Grid, called 3D-ANNOTATE. Allowing the user to select a wide range of analysis tools and automatically apply them to each proteome of their requirement.

3D-GENOMICS: The 3D-GENOMICS database provides a range of structural and functional information for protein sequences from sequenced genomes. It allows queries to find genes within selected organisms that encode proteins with particular three-dimensional folds. At present there are 261 genomes available in the database, out of which homology models are available for the protein sequences of 13 genomes. The protein annotation for each organism is loaded into tables within a relational database.

The annotation workflow comprises of a number of Perl programs which perform the following tasks: gather genetic information from the 3D-GENOMICS database; process this data which may include using an existing bioinformatics tool sets; submit the results from this work back into the 3D-GENOMICS database. These Perl programs form the components of the workflow depicted in Figure 4.

Originally this workflow was realised through a hard-coded Perl program which calls each of the components as required. Note that not all components in the workflow will be used for each annotation. The bioinformatician can select those which are needed to perform the task required.

3D-ANNOTATE: In order to convert this application to the Grid we have replaced the Perl workflow with a portal service (as depicted in Figure 5) called 3D-ANNOTATE. This portal allows the user to select the appropriate components (referred to as filters – thus matching the users knowledge and environment) along with the proteome to annotate. This abstracts the user from the understanding of the architecture or the workflow requirements for performing the annotation.

Once the bioinformatician has selected the components required for the current run of the application a workflow description is generated and submitted to the Brokering service.The components of the workflow can then be submitted through GridSAM.

7.1 The 3D-ANNOTATE Architecture

Our system is primarily designed to support annotation of proteomes via a workflow system. The architecture can be defined as the series of stages from the generation of the goal description, workflow, execution, to the point where results are collated and staged to the location required by the user [24]. Here the goal description stage is our Web Portal. A typical workflow includes: requirements, optimization of

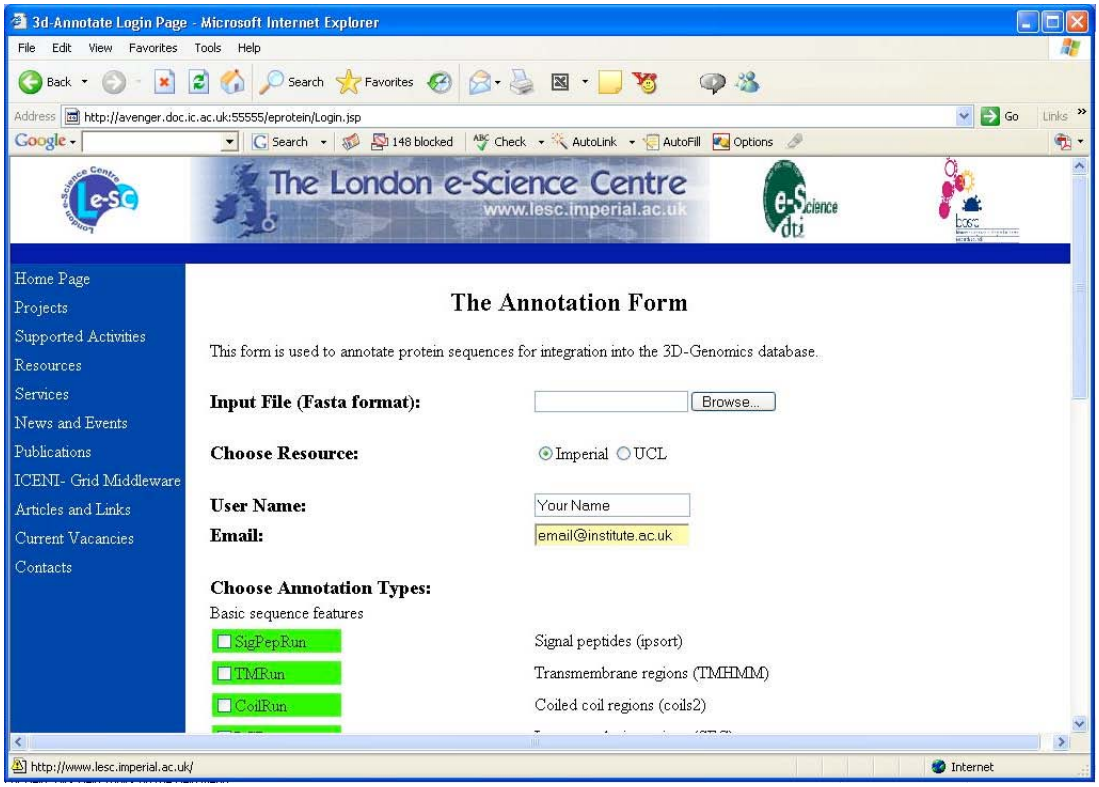

Figure 5. The e-Protein Web Portal

resources and execution of the jobs.

The process for 3D-ANNOTATE is divided into the following levels (see Figure 6):

Level 1: The first level comprises a Graphical User Interface, which offers a Web Portal to submit biological sequences. Users can indicate the location of an input sequence files in FASTA format, which may contain multiple sequences, and select the choice of applications (filters) to be run on the given sequence. Once the file is submitted, a Java Control Program records the requests and splits the sequences into single files and submits them to the 3D-GENOMICS database. The database returns a list of sequence IDs for each sequence which is utilised for processing in the next level.

Level 2: The second contains the workflow service and the brokering service. Each sequence will cause the execution of the workflow derived from the users inputs in Level 1. Each workflow component will be deployed by a call to the appropriate GridSAM instance through an auto-generated JSDL document, detailing the path for the appropriate wrapper. GridSAM returns a handle which can be used to determine when the wrapper has completed.

Level 3: The third level involves executing elements of the annotation pipeline as specified by the workflow. GridSAM submits the wrappers to the underlying DRM systems. The wrapper will execute and thus execute the component as required.

7.2. e-Protein Application Wrapping

Although the components within the workflow are reasonably portable there are still issues of portability with them which arise from the use of the existing bioinformatics tool sets and the need for access to databases. In order to ensure that all the required elements for the component are available wrapping technology is used. The wrapper contains a copy of the Perl program required for execution along with any required Perl libraries, data files and the bioinformatics tool sets that are required. In this case all of the applications were originally written to run in a batch mode, thus there is no need to explicitly handle input and output of the components. Some of the original bioinformatics tool sets did require wrappers, though these had already been wrapped as part of the 3D-GENOMICS through Perl wrappers. On execution of jobs GridSAM executes a wrapper job for each of the components in the workflow. The wrapper (implemented as a shell script) decompresses a (tar) archive containing the Perl component along with any libraries and bioinformatics tool sets; executes the Perl component; before removing any files generated on the exe-

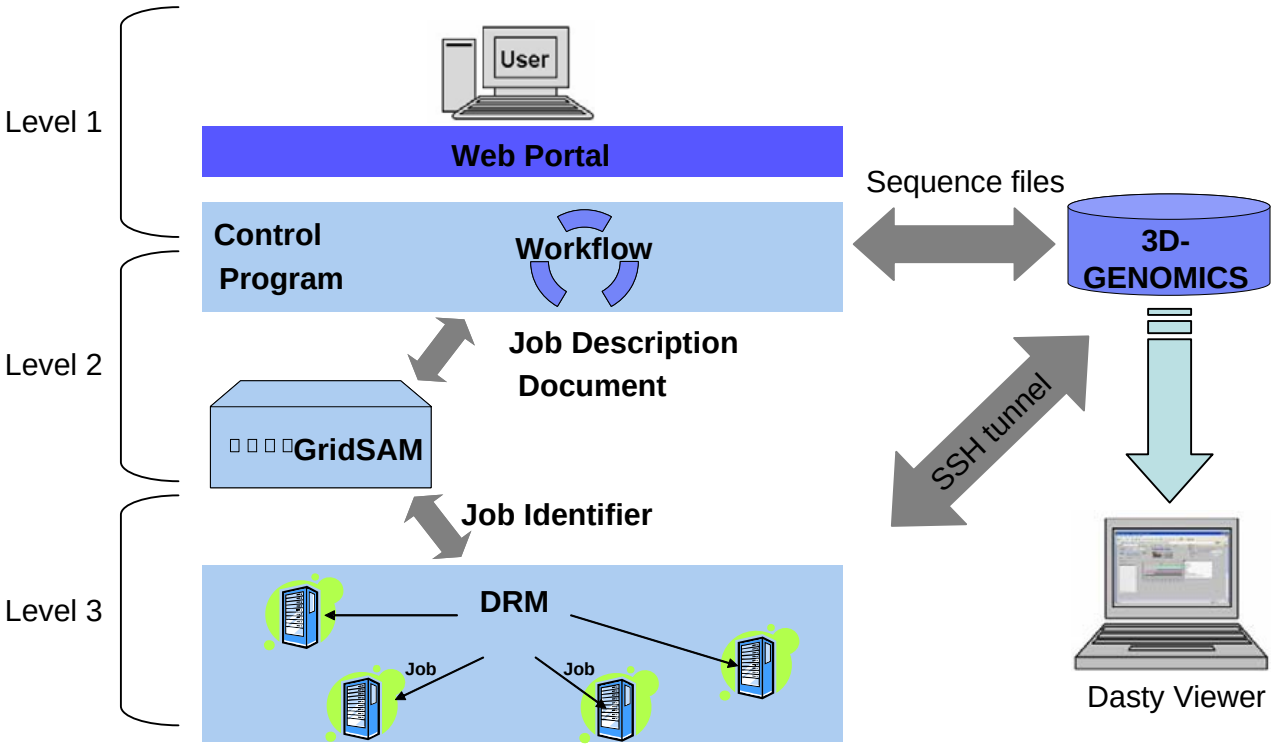

Figure 6. The 3D-ANNOTATE architecture

cution resource. As the Perl programs are reliant on having access to the 3D-GENOMICS databases they require access to the databases currently held at Imperial College London. This is currently achieved through the use of secure shell tunnelling of the database communications. However, we seek to generalise this process by adopting database access through OGSA-DAI [12].

This may make some of the components platform specific. This platform dependence can be handled through the resource requirements expressed in the JSDL document and appropriate resources selected through the brokering service.

7.3. Implementation Experiences

We have found that decomposing the application into a number of components controlled through a workflow engine shows benefits in execution of this application. Another group within the e-Protein project have kept the single Perl application and submitted this to the Grid. Their effort to use this application on the Grid was small as the whole application could be wrapped as a single Grid task with a simple control program. However, without the decomposition of the components the entire application needs to be run on a restrictive subset of the Grid. Those resources which match all the requirements of the entire workflow. In the case of the 3D-GENOMICS work only one of the components has tightly restrictive resource requirements this has lead the other group to only be able to use this small subset of the Grid. Thus highly overloading this subset of resources. However, our approach allows those components without the restriction to be submitted to a larger set of resources. This not only allows better usage of the Grid but it also reduces the load on the small subset of the Grid capable of running the restrictive component. Thus we have been able to deploy the 3D-ANNOTATE application over resources based at Imperial College London, University College London and resources on the National Grid Service (NGS) [11].

8. Conclusion

In this paper we have shown how existing legacy applications can be mapped into a Grid environment by the use of a simple architecture which exposes the application to the user through a simple web-based portal back ending to a standards based jobs submission interface (GridSAM). This allows for abstraction at different levels within the system. The end user is abstracted from the complexity of the Grid, while the portal developer is abstracted away from the heterogeneity of the underlying Grid fabric.

Further we have shown that if the application is com-

posed of a number of separate components then these components along with a workflow description document can be submitted to the Grid. This has the added benefit of allowing the Grid to best determine where each part of the workflow should be executed in order to meet the requirements of users of the Grid.

We have constructed our architecture from elements within the ICENI II system and demonstrated how they can be used in Grid-enabling the 3D-GENOMICS legacy application.

9 Acknowledgements

We would like to thank the Open Middleware Infrastructure Institute Managed Programme who have funded the "*GridSAM Simple Web Service for Job Submission and Monitoring*" project.

References

[1] activeBPEL. http://www.activebpel.org/.

[2] Biodas. http://www.biodas.org.

[3] e-Protein. http://www.e-protein.org/.

[4] Grid Engine. http://gridengine.sunsource.net/.

[5] GridSAM - Grid Job Submission and Monitoring Web Service. http://gridsam.sourceforge.net.

[6] Lsf. http://www.platform.com/.

[7] OGSA Basic Execution Services (OGSA-BES-WG) . https://forge.gridforum.org/projects/ogsa-bes-wg/.

[8] OGSA Resource Selection Services (OGSA-RSS-WG) . https://forge.gridforum.org/projects/ogsa-rss-wg/.

[9] Open Grid Services Architecture (OGSA-WG) . https://forge.gridforum.org/projects/ogsa-wg.

[10] The Global Grid Forum. http://www.ggf.org.

[11] The National Grid Service. http://www.ngs.ac.uk/.

[12] The OGSA-DAI Project. http://www.ogsadai.org.uk/.

[13] The Open Middleware Infrastructure Institute. http://www.omii.ac.uk.

[14] A. Anjomshoaa and F. Brisard and M. Drescher and D. Fellows and A. Ly and S. McGough and D. Pulsipher and A. Savva . Job Submission Description Language (JSDL) Specification v1.0 . http://www.gridforum.org/documents/GFD.56.pdf.

[15] BPEL4WS. http://www6.software.ibm.com/software/developer/library/ws-bpel.pdf.

[16] J. Cohen, W. Lee, A. Mayer, and S. Newhouse. Making the Grid Pay - Economic Web Services. In *Building Service Based Grids Workshop, GGF11*, Honolulu, Hawaii, USA, June 2004.

[17] E. Gamma and R. Helm and R. Johnson and J. Vlissides. Design patterns: Abstraction and reuse of object-oriented design. In *Lecture Notes in Computer Science*, volume 707, pages 406–431. Springer, 1993.

[18] I. Foster and C. Kesselman. The Globus Project: A Status Report. In *IPPS/SPDP '98 Heterogeneous Computing Workshop*, pages 4–18, 1998.

[19] I. Foster and C. Kesselman, editors. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, July 1998.

[20] Jeremy Cohen and John Darlington and William Lee. Payment and Negotiation for the Next Generation Grid and Web. In *UK e-Science All Hands Meeting*, Nottingham, UK, sep 2005.

[21] K. Fleming and A. Muller and R.M. MacCallum and M.J.E. Sternberg. 3D-GENOMICS: A data base to compare structural and functional annotations of proteins between sequenced genomes. *Nuc. Acid Res*, 32:D245–D250, 2004.

[22] W. Lee, A. McGough, and J. Darlington. Performance Evaluation of the GridSAM Job Submission and Monitor ing System. In *UK e-Science All Hands Meeting*, pages 915–922, Nottingham, UK, Sept. 2005. ISBN 1-904425-53-4.

[23] M. Welsh and D. Culler and E. Brewer. Seda: An architecture for well-connected scalable internet services. In *Eighteenth Symposium on Operating Systems Principles (SOSP-18)*, October 2001.

[24] A. McGough, J. Cohen, J. Darlington, E. a Katsiri, W. Lee, S. Panagiotidi, and Y. Patel. An End-to-end Workflow Pipeline for Large-scale Grid Computing. *Journal of Grid Computing*, pages 1–23, Feb. 2006.

[25] A. McGough, W. Lee, and J. Darlington. ICENI II Architecture. In *UK e-Science All Hands Meeting*, pages 441–448, Nottingham, UK, Sept. 2005. ISBN 1-904425-53-4.

[26] T. H. Project. Hibernate. http://www.hibernate.org.

[27] Q. E. Scheduler. Quartz enterprise scheduler. http://quartzscheduler.org.

[28] T. Ylonen. The Secure Shell (SSH) Protocol Architecture. http://www.ietf.org/rfc/rfc4251.txt?number=4251.

[29] Thain, D., *et al.*,. Condor and the Grid. In F. Berman, A. J. G. Hey, and G. Fox, editor, *Grid Computing: Making The Global Infrastructure a Reality*. John Wiley, 2003.

[30] Y. Patel and A.S. McGough and J. Darlington. QoS support for Workflows in a volatile Grid. In *Submitted for the Workshop on Grid Computing 2006 (GRID2006)*.

[31] L. Young, A. McGough, S. Newhouse, and J. Darlington. Scheduling Architecture and Algorithms within the ICENI Grid Middleware. In *UK e-Science All Hands Meeting*, pages 5–12, Nottingham, UK, Sept. 2003. ISBN 1-904425-11-9.