Available online at www.sciencedirect.com
ScienceDirect
Future Generation Computer Systems 24 (2008) 731–743
FGCS FUTURE GENERATION COMPUTER SYSTEMS
www.elsevier.com/locate/fgcs

A standards based approach to enabling legacy applications on the Grid

A. Stephen McGough[a,*], William Lee[b], Shikta Das[c]

[a] *Imperial College London, London, UK*
[b] *InforSense, London, UK*
[c] *Voluntary Work, London, UK*

Received 22 February 2007; received in revised form 21 January 2008; accepted 7 February 2008
Available online 25 February 2008

Abstract

The Grid holds great potential for users, software developers and resource owners. Users of the Grid are abstracted away from its complexity, while software developers are being provided with a rich middleware in which to develop their applications without the need for a large investment in resources. Resource owners are able to expose their resources, potentially for financial reward, without the need to invest in software. However, as most of the applications which currently exist pre-date the concept of the Grid, they lack the appropriate functionality and/or modularization to exploit it. In this paper we propose the use of a standards based job submission and monitoring system which is capable of deploying legacy applications onto existing resources within the Grid, environment containers for mapping applications into Grid applications, along with the use of web-based portals to expose these applications to the end user. 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. We show how the use of this technique over the Grid can significantly reduce the makespan required for the application.
© 2008 Elsevier B.V. All rights reserved.

Keywords: Grid; JSDL; GridSAM; Workflow; Standards; OGSA-BES

1. Introduction

The Grid [1] has great potential for users, developers and resource owners. For the user there are vast numbers of resources available, presented through an abstraction which hides the heterogenous nature of the Grid. For the user this means that the software 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. For the resource owner it provides an environment in which they can make their resources available, potentially for financial reward [2], 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 [3]. 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 a user could specify the requirements of the application they wish to run. These specifications can then be processed through a "brokering" service which selects which software implementation and resource selection pairings are most appropriate for the users requirements.

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 specific to a particular operating system and/or processor type. Legacy applications are often designed to be executed from the command line on an interactive resource which is easily accessible by the user. Interaction is either through text

* Corresponding author. Tel.: +44 2075948409; fax: +44 2075818024.
E-mail addresses: asm@doc.ic.ac.uk (A. Stephen McGough), william@imageunion.com (W. Lee), shikta_das@yahoo.co.uk (S. Das).
URL: http://www.lesc.doc.ic.ac.uk (A. Stephen McGough).

0167-739X/$ - see front matter © 2008 Elsevier B.V. All rights reserved.
doi:10.1016/j.future.2008.02.004

based communications, which may happen in either direction, or through the use of some Graphical User Interface (GUI). Without significant effort these executables cannot be converted into web based services.

In many cases interaction with a running application is inhibited by the nature of the Grid. Text flowing to and from an application may not be possible due to the batch nature of job submission. The configuration of a remote resource is also often unknown – this makes the process of communication more complex. The use of interactive GUI interfaces may not be possible if the application is not running on the users own box or security may prevent GUI interfaces communicating from remote resources. The availability of libraries and other elements that are required for execution are often difficult to ascertain. Other restrictions may exist which makes the running of applications on the Grid more difficult.

Often executables can work in batch mode where the inputs to the executable can be pre-determined 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.

There are two main alternatives when exposing legacy applications into the Grid. In the first executables are wrapped up as services which are exposed to users. This allows for a bespoke environment to be matched to the particular executable and a service interface to be tailored to the specific executable. In general the resource used for executing the legacy code will be fixed with the service — thus breaking our view that the software developer and resource provided need not be the same. Here we propose an alternative approach in which resources are exposed as generic job execution services consuming jobs described in a standardized language exposed through a standardized interface. Thus the tie between the resource owner and software provider is broken. This, however, does not preclude the advantages of the previous approach. Software providers may still expose bespoke service interfaces to their code. Though rather than executing the code local to the service they can submit jobs to the underlying Grid on behalf of the user. This approach would appear to fit in well with the emerging Grid infrastructure where large amounts of computing power are being exposed as generic computational resources such as the European EGEE Grid [4], the American TeraGrid [5] and the UK based National Grid Service [6].

1.1. Environment containers

In many cases these executables can be wrapped inside of an "environment container", Fig. 1, 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. Libraries and files can be made part of the batch job with input and output services dealing with interactive requirements. In general this works best when the input and output are simple text based interactions, though

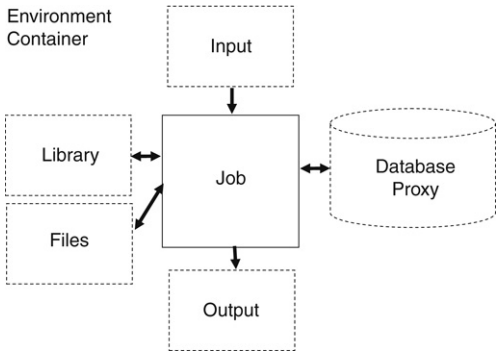


Fig. 1. An environment container.

this could be done for more complex interaction methods. Communication with external services such as databases can be handled through database proxies which are then capable of communicating with the actual database service. Thus in general we are able to use a significant amount of existing software without the need to alter it in any manner.

It is worth noting that in general environment containers tend to be bespoke to the the particular legacy code being wrapped. However, tooling is now becoming available to simplify this process such as the Application Hosting Environment [7].

1.2. Standards based job submission

Execution of a legacy executable anchored at a specific location within the Grid can be achieved quite simply within Grid software. However, there is great complexity involved if there is a desire to run the legacy executable on an arbitrary resource in the Grid — thus allowing for greater use of Grid resources. The Grid software needs to be aware of not only where the legacy executable could be deployed but also how to deploy it onto these resources. 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 [8], Globus [9], Grid Engine [10] and LSF [11]. 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. This heterogeneity can be hidden from the end user by writing a user interface capable of understanding each of the different underlying DRM submission interfaces and their submission languages. However, this would lead to highly complicated user interface services requiring significant work each time a DRM system updates, changes or adds a submission interface. In many cases the user interface, or a proxy for it, would need to be deployed onto a specific resource which is capable (and allowed) to submit jobs to the DRM system.

In this paper we propose the use of a common (standardized) job submission interface for deploying jobs onto the Grid. This allows the Grid application developer, such as a Web portal implementer, to concentrate on the usability and development of their code independently of the complexity of the job

submission process. They need only be aware of one (standard) job submission interface thus allowing simpler user interfaces which are independent of changes in the underlying fabric of the Grid. Thus providing a clear abstraction between the user interface and the underlying DRM system. Jobs are described using a standardized Job Submission Description Language (JSDL) [12] and is submitted through a standardized job submission interface (OGSA-BES) [13]. This greatly simplifies the user interface developers effort as they no longer need to be aware of the heterogeneity of the different DRM systems.

1.3. Application workflows

Many applications comprise of a number of sub-jobs, often referred to as components, which are performed to achieve the overall goal of the application. These components are often composed together within some form of control structure (a control script), this control structure along with the individual components can be submitted to the underlying Grid Fabric as a single job allowing the whole application to be executed. However, this 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 which best matches the requirements of all of the components. Alternatively, if we allow the set of components to be exposed to the Grid independently along with a workflow document describing how these components interact – replacing the original control structure – 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 application. Parallelism within the workflow may be exploited and resilience added in the cases where resources fail during execution.

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. We discuss related work in Section 8 before concluding within Section 9.

2. Architecture

In previous work [14,15] 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. Fig. 2 illustrates how these elements can be combined together. We briefly describe these services below before expanding on them more fully later in the paper.

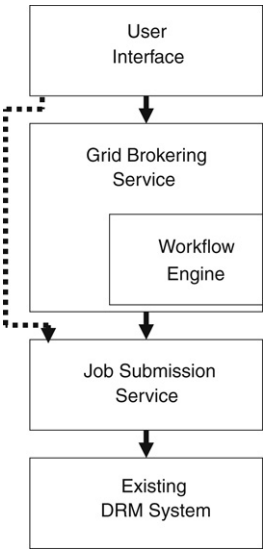


Fig. 2. Architecture.

- **User Interface**. The user interface 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 only to produce a service which uses a standard job submission description language (JSDL) 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 service. 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 user interface developed as a 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 service 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 Service**. In order to abstract away from the heterogeneity of the underlying Grid fabric we are developing a standards based Grid Submission and Monitoring (GridSAM) [16,17] service. GridSAM is

designed to be situated between existing DRM systems and the Grid providing a standard job submission interface into these DRM systems. The power of GridSAM comes from its use of the 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, or developer, 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 recognized 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 inter-operability across organizational and technical boundaries in order to attain the vision of a global computational network. In particular, the Job Submission Description Language (JSDL) [12] standardized through the Global Grid Forum (GGF) [18] – now part of the Open Grid Forum (OGF) [19] – and the Open Grid Services Architecture Basic Execution Service (OGSA-BES) [13] interface from the OGF are essential standards to promote inter-operability among DRMs.

JSDL is a language for describing a job that you wish to have executed. The document is broken down into four main sections of job identification, application description, resource requirements and data staging requirements. The job identification section contains human readable information describing the job. The application description section describes the executable. In the case of a legacy piece of code this could contain the name of the executable, the arguments required by the executable and the environment variables required. The resource description section describes the resource requirements, again this could be such information as the type of processor required, the amount of memory required and/or the operating system. The data staging section contains a description of those files which are to be staged in before execution and those to be staged out after execution, the protocols to be used and the locations on which these files may be found or should be staged back to. The JSDL document is rendered in XML.

OGSA-BES provides an interface definition for how to communicate JSDL documents between clients and servers. It defines the Web Service interface to use and the Web Service Description Language (WSDL) for the service. This document also defines the interaction process between client and server and defines the expected responses from the server.

Here we present a standards-based job submission system, GridSAM,[1] undertaken as part of the Open Middleware Infrastructure Institute (OMII) Managed Programme [20], as one of the first systems adopting JSDL, OGSA-BES and Web Services for job description and interaction. We adopt the OGSA-BES standard by providing a service which provides the functionality described within the OGSA-BES specification and WSDL documents [13]. This allows users to specify their job requirements using the JSDL [12] language which GridSAM consumes, parses into an internal (object representation of the JSDL document) format before converting this into a format required for the underlying DRM system. The specifics of how this conversion is performed is dependent on the DRM system to be used. GridSAM provides a transparent and efficient bridge between users and existing DRM systems, such as *Condor* [8,21], *Globus* [9] and *Grid Engine* [10]. The encapsulation allows the functions to be used through the GridSAM Web Service as well as utilizing it as an independent library.

3.1. GridSAM Architecture

In this section we present the architecture for GridSAM as illustrated in Fig. 3. Requests for execution of jobs arrive through the Web Service interface as JSDL documents while the underlying DRM system is interacted with through a collection of DRMConnectors representing different functionalities of the DRM. These DRMConnectors map the desires captured within the JSDL document into DRM specific format and invoke these on the underlying DRM system.

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 Web Service interface, 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)*.

[1] Downloadable from http://gridsam.sourceforge.net.

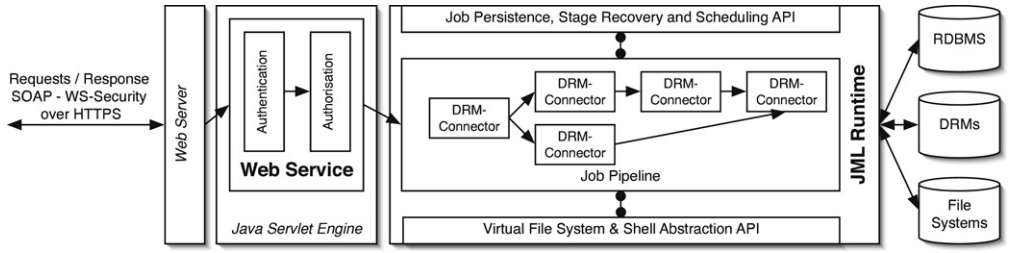


Fig. 3. GridSAM system architecture.

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) [22]. 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.

Fig. 4 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. Alternatively these services can be replicated thus allowing simultaneous execution.

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 [23] that encourages component reuse and action encapsulation. For example, the *Forking* and *Secure 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* [24] 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 [25] to schedule stages and allocate threads. Welsh et al. [22] 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.

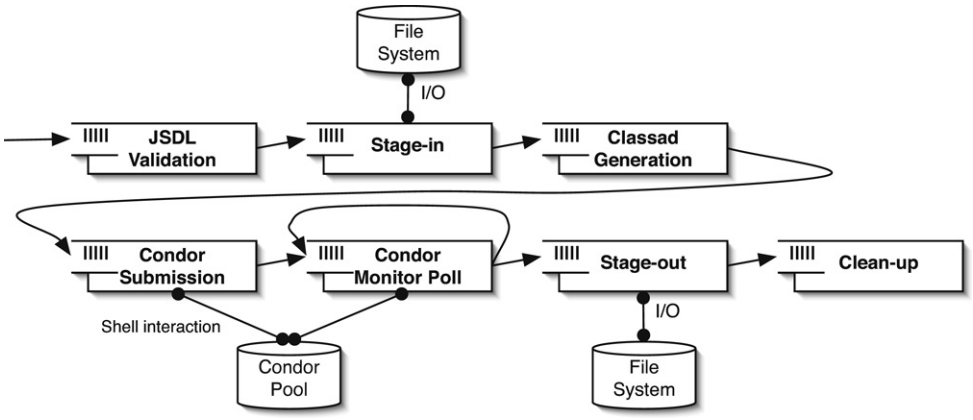


Fig. 4. Submission pipeline for Condor job in GridSAM.

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 may sit on top of are capable of performing 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. This requires the use of a resource discovery service, which may be provided by a look-up service such as GRIMOIRES [26] along with a sibling service to GridSAM which publishes information on resources to GRIMOIRES.

In previous work [14,27–29] 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 has been determined. This process may be carried out in a one pass process [27], through a constantly updating process [28] or by balancing the selection of services over many applications [29]. In either case the elements of the workflow 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 [14]. Many of the workflow optimisation stages presented in these papers 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) [14] 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 OGF in the areas of brokering through both the Open Grid Services Architecture (OGSA) [30] and OGSA-Resource Selection Service (OGSA-RSS) [31] working groups.

5. Workflow service

Often a user will make use of a number of separate components to make their application. The user may choose to execute each of these components in turn by making separate job submissions to the Grid. This places a high burden on the user who needs to be aware of how the overall application is progressing and to be able to submit the next stage in a timely manner. 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, standardized, solutions wherever possible. As such we are adopting the Business Process Execution Language (BPEL) [32] workflow description language and are working with the ActiveBPEL [33] workflow engine. Bespoke BPEL documents have been developed for each of the workflows required with each of the workflows calling GridSAM instances through a brokering service for the execution of the tasks.

The use of a standard job description language has greatly simplified the process of developing these workflows as each of the invocations of GridSAM instances requires only one job description document, which can be pre-prepared and can be passed straight from the BPEL engine, via brokering services if desired, to the GridSAM instance. Thus no extra functionality need be added to the BPEL engine nor the brokering service. The use of a standardized job submission interface allows

for one single invoke element to be used within the BPEL document thus simplifying the structure of the BPEL and allowing quicker development of documents.

At present our brokering works in a in-line manner. When the BPEL engine calls out to the broker a decision is made as to the best location for execution. We seek in the future to use the more powerful features of the BPEL specification to allow for the use of just-in-time GridSAM instance selection. In this approach the broker can select a GridSAM instance before the BPEL invoke is reached and modify the workflow such that this service is called directly.

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 user interface 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 interactions 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 pre-determined 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 [34] or through an SSH tunnel [35].
- **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.

One issue which wrappering an executable can lead to is the increase in execution time for calling the legacy code over the Grid. Time is required for staging the executable along with the wrapper to the remote service. If the execution time for the service is short, relative to this staging time, it can defeat the benefit of using the Grid. One solution to this is to batch together several executions of the legacy code for each deployment onto the Grid.

7. Proteome annotation in e-Protein

The protein structure prediction problem is one of the most challenging in biological sciences. Knowledge of the structure is vital in understanding the function of the protein. A possible solution that enables protein structure predictions is the notion of homology. The e-Protein project [36], a pilot initiative funded by the Biotechnology and Bioinformatics Sciences Research Council (BBSRC), for academics to functionally annotate protein sequences using homology and fold recognition methods. The work is to combine structural and functional genome annotation databases from Imperial College London, University College London and the European Bioinformatics Institute, through a single interface using the Distributed Annotation System (DAS) [37] and utilising Grid middleware technology. 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 the proteome annotation pipeline.

We have taken a bioinformatics workflow developed by the Structural Bioinformatics Group at Imperial College London. This provides a structural and functional database, called 3D-GENOMICS [38] 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 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 Fig. 5.

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

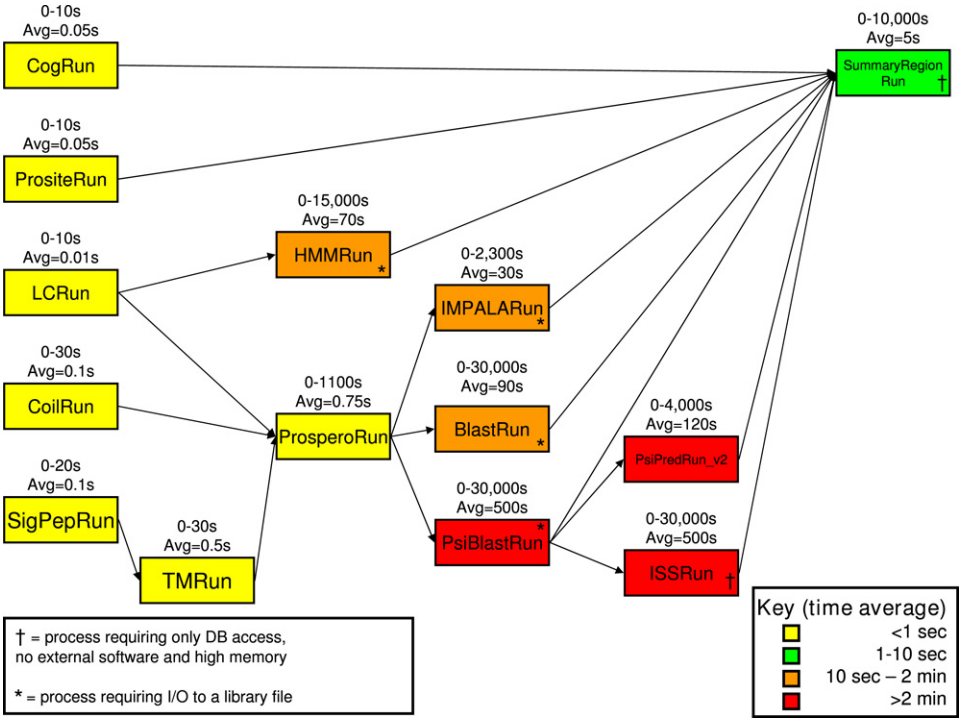


Fig. 5. The components within the e-Protein workflow.

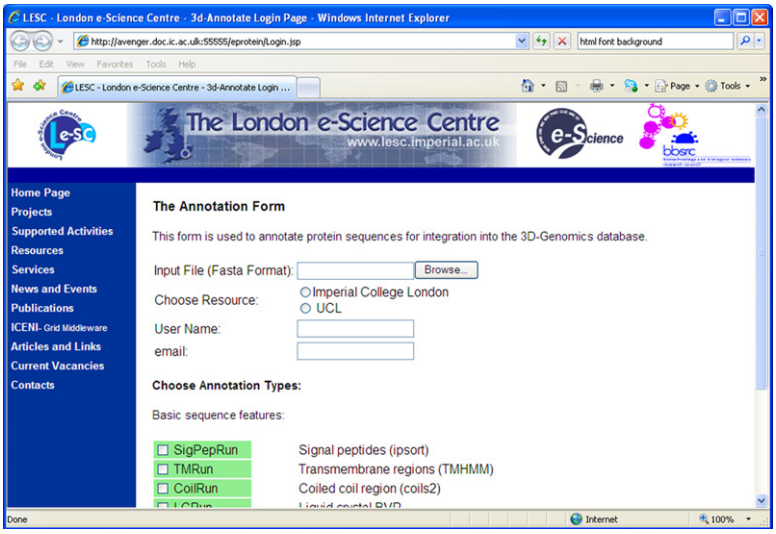


Fig. 6. The e-Protein web portal.

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 Fig. 6) 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 [14]. Here the goal description stage is our Web Portal, workflow is handled by the Control Program and execution is enabled through the use of GridSAM.

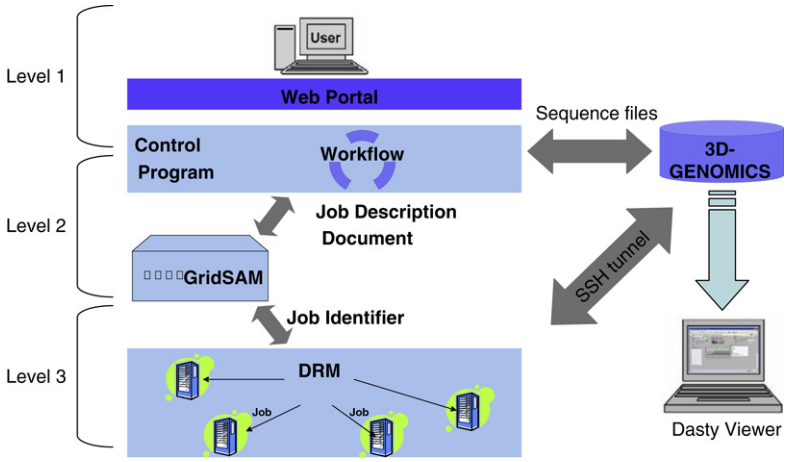


Fig. 7. The 3D-ANNOTATE architecture.

The process for 3D-ANNOTATE is divided into the following levels (see Fig. 7):

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 are utilized for processing in the next level.

Level 2: The second level 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 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. 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 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 execution resource. As the Perl programs are reliant on having access to the 3D-GENOMICS databases they require access to the database server currently hosted at Imperial College London. This is 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 [34].

The Perl code has the advantage that it is highly portable. However, many of the bioinformatic tools are 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. By using the resource requirements within the JSDL document appropriately, not only can we target executables to resources which have strict requirements for those resources, but we can also reduce the chance of other jobs being deployed onto those resources – through use of the resource broker. This will tend to a quicker overall execution time as those jobs which can only be run on a small subset of the Grid resources can have those resources kept free for just those jobs.

7.3. Experimental setup

Annotation of the entire Human genome is a significant task which, using the original Perl application, was estimated to take in excess of two years to complete using currently available resources. The aim of this experiment was to determine if the use of Grid technology could be practically used for bioinformatic experiments. The Human genome contains 35 701 sequences for which each needs to be passed through the annotation pipeline. The stages for the annotation pipeline which were used for this experiment are listed in Table 2.

Although the Web portal interface was capable of accepting multiple sequences at a time it was decided for the purpose of

Table 1
Cluster specification

Cluster	Specification
Mars	160 dual 1.8 GHz Opteron processors with 2 Gb memory
	40 dual 1.8 GHz Opteron processors with 4 Gb memory
	4 dual 2.2 GHz Opteron processors with 4 Gb memory
Viking	96 dual 2 GHz Pentium4 (XEON) processors with 2 Gb memory
	32 dual 2 GHz Pentium4 (XEON) processors with 1 Gb memory
	128 dual 2.8 GHz Pentium4 processors with 2 Gb memory
	4 dual 2 GHz Pentium4 processors with 2 Gb memory

this experiment to use an alternative submission system which submitted each sequence separately to the control program. This prevented the web portal from becoming a bottleneck to the application and allowed for the timing of each individual submission.

Two Grid enabled clusters were used for the experiment: a 408 processor Opteron based beowulf cluster running RedHat Enterprise Linux (Mars cluster); along with a 260 dual processor Intel/Linux cluster (Viking). The details of the clusters are given in Table 1. Both clusters run the Grid Engine [10] DRM system, version 6 of the software was used as this allowed GridSAM to submit jobs through the DRMAA [39] interface. To reflect the real nature of the Grid we did not request exclusive use of these clusters during the experiment. Other users were able to submit jobs with the clusters balancing requests based on the Grid Engine fair-share policies.

Instances of GridSAM were run on the interactive nodes on both the Mars and Viking clusters. In both cases to aid with load-balancing each cluster had two GridSAM instances running on separate interactive nodes. The clusters were selected in a round-robin manner unless the job could only be run on one of the clusters.

7.4. Experimental results

The Human genome annotation pipeline was executed on the Mars and Viking clusters under normal load conditions. The entire experiment took just over 11 days to perform (270 h, 39 min). The vast reduction in execution time here is a consequence of the ability to run multiple sequences in parallel. A full breakdown of the total time (in seconds) spent in executing each of the components is given in Table 2. As can be seen from this table most of the initial components have small average execution times while the latter four have much longer execution times. This is partly due to the fact that the last four components were only run on the Viking cluster due to some software incompatibility when run on the Mars cluster. As the execution times of the initial components are short it would seem that some form of batching jobs together would be sensible. This could be either to run the same component on multiple sequences for each job submitted or to perform multiple component stages on one sequence for each job submitted.

Fig. 8 shows the cumulative completion of sequences through the entire annotation pipline in relation to the experiment execution time. The rate of completion throughout

Table 2
Breakdown of time for each component

Component	Cumulative time (s)	Average time (s)
LC	84 300	2
PROSPERO	146 344	3
TM	87 603	2
SIGPEP	90 407	2
COIL	86 225	2
COG	88 274	2
HMM	3 509 592	95
PROSITE	110 795	2
IMPALA	6 527 053	175
BLAST	5 197 655	139
PSI-BLAST	25 148 730	683
PSIPRED	26 164 327	712
Total	67 241 305	

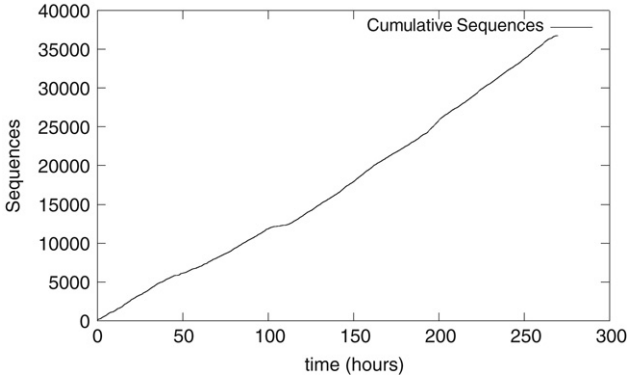


Fig. 8. Cumulative sequence completion.

the 270 hours of the experiment is reasonably constant. This is partly due to the load-balancing functionality built into Grid Engine. It should be noted that although there are 35 701 sequences within the Human genome most of the components were executed more times than this due to execution failures. Up to 5% of the submissions failed and had to be re-submitted. The reason for these failures was not always clear, sometimes the bioinformatic tools would fail to execute, or the underlying DRM system would fail to run the job. However, we were able to use GridSAM to re-submit the jobs. If however the job failed due to execution errors the job was submitted to the other cluster to be executed.

7.5. 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. They also removed the database requirements by using flat files. 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) [6].

8. Related work

Many job submissions systems exist such as Condor [21], Grid Engine [10], PBS [40] and LSF [11]. GridSAM is different to these systems as it does not in itself provide the functionality of a Distributed Resource Manager. Instead it acts as a standardized interface on-top of these pre-existing services. To this effect we see that GridSAM acts in a complementary manner to these systems.

Triana from Gidlab [41] provides a generic workflow-based graphical problem-solving environment that handles a range of distributed elements such as Grid jobs, Web Services, and P2P communications. The distributed components considered by Triana fall into two categories: Grid oriented and service-oriented. Thus legacy code is wrapped up with a resource as a service within the Grid — a model we have sought to break allowing for a much more open model of Grid use. As the bioinformatics activity which was Grid enabled as part of this work was already wrapped and developed for batch execution, the use of standard job submission services as opposed to wrappering the executables up as services was much simpler. Furthermore, Triana and the Visual GAT represent explicitly Grid operations such as job submission and file transfer (by the introduction and use of job components and file components). Although Triana can be integrated with and operate over a range of Grid technologies, such as resource managers and data management systems, it only focuses on the implementation of an environment for the specification and execution of a component workflow. This means that other aspects such as optimization, interaction with resource managers, and data management systems are not addressed by the framework. JACAW [42] is an automatic process for generating Java wrappers for C code. As we do not limit ourselves only to the use of C code nor do we assume that the code is available the JACAW library is not applicable to the execution of our legacy code.

Taverna [43] is the workflow editor environment for the MyGrid [44] project. Taverna and the MyGrid middleware are designed for performing in silico experiments in biology. Again the Taverna approach is to wrapper legacy code and resources together as a service. The MyGrid middleware lacks the facility to perform scheduling; instead, it looks up anchored services that the user can compose.

GEMLCA [45] adopts the alternative approach to execution of legacy code on the Grid by developing a separate enactor into their system for each underlying DRM system they wish to utilize. This we feel, although an equivalent exercise to the development of GridSAM in isolation, does remove the ability to exploit other JSDL/OGSA-BES implementations which are emerging in the way that we can with our work.

Kepler [46] is similar to Taverna, providing a workflow editing environment with the ability to invoke wrappered legacy and resource Web services. Although Kepler provides a useful way to model workflows, it lacks the ability to adapt these workflows in the presence of changes in the Grid. All of these workflow systems use their own bespoke workflow language. Again this is an area we seek to use standard solutions thus leveraging commodity quality workflow engines.

9. 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 user interface 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 user interface developer is abstracted away from the heterogeneity of the underlying Grid fabric.

Further we have shown that if the application is composed 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. We have demonstrated how we can use this technology to perform an annotation of the Human genome. Thus demonstrating the useability of this approach, and also showing that this approach can allow a user community to exploit the power of the Grid to reduce the execution time of their application from the order of years to less than a fortnight.

Acknowledgements

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

We would like to thank the BBSRC who have funded the "e-Protein" project as part of their e-Science funding.

The second and third author's work was carried out whilst at Imperial College London.

References

[1] I. Foster, C. Kesselman (Eds.), The Grid: Blueprint for a New Computing Infrastructure, Morgan Kaufmann, 1998.

[2] J. Cohen, W. Lee, A. Mayer, S. Newhouse, Making the grid pay — economic web services, in: Building Service Based Grids Workshop, GGF11, Honolulu, Hawaii, USA, 2004, pp. 1–7.

[3] J. Cohen, J. Darlington, W. Lee, Payment and negotiation for the next generation Grid and Web, Concurrency and Computation: Practice and Experience. http://pubs.doc.ic.ac.uk/payment-concurrency-computation/.

[4] Enabling Grids for E-sciencE (EGEE). http://public.eu-egee.org/.

[5] Teragrid. http://www.teragrid.org/.

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

[7] P. Coveney, R. Saksena, S. Zasada, M. McKeown, S. Pickles, The application hosting environment: Lightweight middleware for grid-based computational science, Computer Physics Communications 176 (2007) 406–418.

[8] D. Thain, T. Tannenbaum, M. Livny, Condor and the Grid, in: F. Berman, A.J.G. Hey, G. Fox (Eds.), Grid Computing: Making The Global Infrastructure a Reality, John Wiley, 2003, pp. 299–335.

[9] I. Foster, C. Kesselman, The globus project: A status report, in: IPPS/SPDP '98 Heterogeneous Computing Workshop, 1998, pp. 4–18.

[10] Sun Source, Grid Engine. http://gridengine.sunsource.net/.

[11] Platform, LSF. http://www.platform.com/.

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

[13] I. Foster, A. Grimshaw, P. Lane, W. Lee, M. Morgan, S. Newhouse, S. Pickles, D. Pulsipher, C. Smith, M. Theimer, OGSA Basic Execution Service Version 1.0. http://www.ogf.org/documents/GFD.108.pdf.

[14] A. McGough, J. Cohen, J. Darlington, E.a Katsiri, W. Lee, S. Panagiotidi, Y. Patel, An end-to-end workflow pipeline for large-scale grid computing, Journal of Grid Computing (2006) 1–23.

[15] A. McGough, W. Lee, J. Darlington, ICENI II architecture, in: UK e-Science All Hands Meeting, Nottingham, UK. ISBN 1-904425-53-4, 2005, pp. 441–448.

[16] W. Lee, A. McGough, J. Darlington, Performance evaluation of the GridSAM job submission and monitor ing system, in: UK e-Science All Hands Meeting, Nottingham, UK. ISBN 1-904425-53-4, 2005, pp. 915–922.

[17] W. Lee, A. McGough, V. Novov, GridSAM — Grid Job Submission and Monitoring Web Service. http://gridsam.sourceforge.net.

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

[19] The Open Grid Forum. http://www.ogf.org.

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

[21] D.H.J. Epema, M. Livny, R. van Dantzig, X. Evers, J. Pruyne, A worldwide flock of condors: Load sharing among workstation clusters, Future Generation Computer Systems (1996) 53–65.

[22] M. Welsh, D. Culler, E. Brewer, Seda: An architecture for well-connected scalable internet services, in: Eighteenth Symposium on Operating Systems Principles (SOSP-18), 2001, pp. 230–243.

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

[24] The Hibernate Project, Hibernate. http://www.hibernate.org.

[25] Quartz Enterprise Scheduler, Quartz enterprise scheduler. http://quartzscheduler.org.

[26] Simon Miles, Juri Papay, Terry Payne, Keith Decker, Luc Moreau, Towards a protocol for the attachment of semantic descriptions to grid services, in: The Second European across Grids Conference, Nicosia, Cyprus, 2004, p. 10.

[27] L. Young, A. McGough, S. Newhouse, J. Darlington, Scheduling architecture and algorithms within the ICENI grid middleware, in: UK e-Science All Hands Meeting, Nottingham, UK. ISBN 1-904425-11-9, 2003, pp. 5–12.

[28] Y. Patel, S. McGough, J. Darlington, QoS support for workflows in a volatile grid, in: 7th IEEE/ACM International Conference on Grid Computing, 2006, pp. 64–71. http://pubs.doc.ic.ac.uk/grid-2006/.

[29] A. Afzal, A.S. McGough, J. Darlington, Capacity planning and scheduling in Grid computing environments, Future Generation Computer Systems 24 (5) (2008) 404–414.

[30] I. Foster, K. H, A. Savva, D. Berry, A. Djaoui, A. Grimshaw, B. Horn, F. Maciel, F. Siebenlist, R. Subramaniam, J. Tredwell, J.V. Reich, The Open Grid Services Architecture, Version 1.5. http://www.ogf.org/documents/GFD.80.pdf.

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

[32] T. Andrews, F. Curbera, H. Dholakia, Y. Goland, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, S. Thatte, I. Trickovic, S. Weerawarana, Business Process Execution Language for Web services version 1.1, (BPEL4WS). http://www6.software.ibm.com/software/developer/library/ws-bpel.pdf (May 2003).

[33] Active Endpoionts, activeBPEL. http://www.activebpel.org/.

[34] M. Antonioletti, M.P. Atkinson, R. Baxter, A. Borley, N.P. Chue Hong, B. Collins, N. Hardman, A. Hume, A. Knox, M. Jackson, A. Krause, S. Laws, J. Magowan, N.W. Paton, D. Pearson, T. Sugden, P. Watson, M. Westhead, The design and implementation of grid database services in ogsa-dai, Concurrency and Computation: Practice and Experience 17 (2005) 357–376.

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

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

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

[38] K. Fleming, A. Muller, R.M. MacCallum, M.J.E. Sternberg, 3D-GENOMICS: A data base to compare structural and functional annotations of proteins between sequenced genomes, Nucleic Acids Research 32 (2004) D245–D250.

[39] Hrabri Rajic, Roger Brobst, Waiman Chan, Fritz Ferstl, Jeff Gardiner, amd Andreas Haas, Bill Nitzberg, Hrabri Rajic, John Tollefsrud, Distributed Resource Management Application API Specification 1.0. http://www.ggf.org/documents/GFD.22.pdf.

[40] Portable Batch System. http://www.pbspro.com.

[41] I. Taylor, M. Shields, I. Wang, O. Rana, Triana applications within grid computing and peer to peer environments, Journal of Grid Computing 1 (2003) 199–217.

[42] Y. Huang, I. Taylor, D. Walker, R. Davies, Wrapping legacy codes for grid-based applications, in: Parallel and Distributed Processing Symposium, 2003, p. 139b.

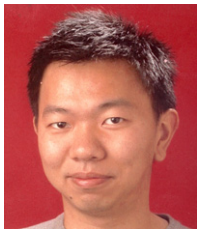
[43] T. Oinn, M. Addis, J. Ferris, D. Marvin, M. Senger, M. Greenwood, T. Carver, K. Glover, M.R. Pocock, A. Wipat, P. Li, Taverna: A tool for the composition and enactment of bioinformatics workflows, Bioinformatics 20 (2004) 3045–3054.

[44] R. Stevens, A. Robinson, C. Goble, myGrid: Personalised bioinformatics on the information grid, in: In 11th International Conference on Intelligent Systems in Molecular Biology, Bioinformatics 19 (2003) i302–i304.

[45] T. Delaitre, T. Kiss, A. Goyeneche, G. Terstyanszk, S. Winter, P. Kacsuk, GEMLCA: Running legacy code applications as grid services, Journal of Grid Computing 3 (2005) 75–90.

[46] I. Altintas, C. Berkley, E. Jaeger, M. Jones, B. Ludascher, S. Mock, Kepler: An extensible system for design and execution of scientificworkflows, in: In 16th Intl. Conference on Scientific and Statistical Database Management (SSDBM), 2004, pp. 423–424.

Dr. A. Stephen McGough has worked for the London e-Science Centre since its foundation. He has worked in the areas of job submission and resource selection and deployment over a range of different DRM systems (e.g. Condor, Globus and Sun Grid Engine). Stephen worked in the European DataGrid project and was the primary author of the JDML specification. From this work Stephen became one of the founding members and co-chair of the Global Grid Forum Job Submission Description Language Working Group (JSDL).



William Lee holds an M.Sc. in E-commerce from Birkbeck College, University of London, U.K. and a B.Sc. in Computing from Imperial College London, U.K. He has worked as a Java developer for many companies and worked as a core developer for ICENI and GridSAM before leaving to work for Inforsense. William was active in the development of the OGSA-BES specification.

Shikta Das holds an M.Sc. in Bioinformatics from Birkbeck College, University of London, U.K. and a B.Sc. in Zoology from the University of Patna, India. While at the London e-Science Centre at Imperial College in 2004, she has been working on developing the use of ICENI and GridSAM for the e-Protein Project with John Darlington and A. Stephen. McGough.