A Portal for Grid-enabled Physics

Brett Beeson[1] **Steve Melnikoff**[1] **Srikumar Venugopal**[2] **David G. Barnes**[1]

[1]School of Physics
The University of Melbourne
Parkville, VIC 3010, Australia
Email: {bbeeson, stevexm, barnesd}@physics.unimelb.edu.au
[2] GRIDS Laboratory and NICTA Victoria Laboratory
Department of Computer Science and Software Engineering
The University of Melbourne
Parkville, VIC 3010, Australia
Email: srikumar@cs.mu.oz.au

Abstract

This paper presents the motivation for development and implementation of a computational portal for the processing of astrophysical and high energy physics data on global Grids. The requirements for the portal, its design, and choices leading to the utilisation of Globus, Gridbus Broker and GridSphere as the middleware, resource broker and portal framework respectively are discussed, along with implementation details and potential future directions for the project.

1 Introduction

Theoretical astrophysics and experimental particle physics are major clients of high performance computing (HPC) facilities worldwide. They are also key application drivers and early adopters for grid technology and distributed computing paradigms. The codes for both are typically numerically intensive and must process, access, or generate massive data sets.

Virtual Observatories (VOs) are one of the leading applications of compute and data grid technology. They deliver a new paradigm for undertaking experimental astronomy, and in particular offer exciting possibilities for completely integrating observational astronomy and theoretical astrophysics. Experimental particle physics is also placed to be an important application of grids. The Belle Analysis Data Grid (Winton et al. 2003) project is developing a global data grid to access and process terabytes of data from the KEK B-meson factory in Japan, searching for the violation of the fundamental Charge-Parity, or CP, symmetry.

Making globally distributed computational resources — from VO data centers to remote instrumentation and experimental facilities — readily available to individual scientists and large-scale collaborations, is part of the vision brought forward by the Grid computing community over the last few years. Communities with common objectives can come together as virtual organizations (Foster, Kesselman & Tuecke 2001) in order to pool their resources and coordinate research among their members. However, this requires establishment of standards for common access to resources and tools to harness the power of the distributed infrastructure. There has been success in specific areas, such as the development of low-level Grid *middleware*. However, the complexity of composing and deploying applications on a Grid is still daunting and a familiar, easy to use environment to access these resources is required for scientists to make use of Grids for their day-to-day research.

Portal-based computing is a strategic enabler for the new and rapidly developing e-Science (or e-Research) methodology. Grid portals address the gap between current Grid computing technology and reliable production systems which scientists can make part of their everyday research toolkit. They have been described as a "problem solving environment that allows scientists to program, access and execute distributed Grid applications from a conventional web browser and other desktop tools" (Gannon, Fox, Pierce et al. 2003). In addition, science application-oriented Grid portals can be an interface between legacy, single CPU programs and their redeployment in a Grid computing environment. By presenting an integrated web browser based interface, sophisticated and powerful Grid tools which have been daunting to employ are made easily available. Portals provide a standard architecture that the user or developer can extend to suit their particular situation. The end result is scientists focusing on doing science, and on the products of their research activities.

A portal for Grid-enabled physics, in particular astrophysics and high energy particle physics, would provide an ideal environment to host user-level services, such as simulation, or handling analysis job configuration (including parameter sweep support), submission to Grid resources, task progress monitoring, and input and output data management and archiving. Built as a general tool, a physics portal needs to support a wide range of existing codes (parallel and serial), and ideally be extensible in the sense that users could upload their own code libraries or configuration routines. This environment would also offer a path to integration with observational astronomy data by using International Virtual Observatory Alliance standards to describe input and output formats, enabling generic VO tools to parse and manipulate the results of simulations as easily as they can direct observational data.

In Section 2, we expand on the above requirements that motivated our development of a Grid-enabled portal for physics. In following sections, we describe our technology choices (§ 3) and design (§ 4) to meet these requirements, and comment on its implementation (§ 5) as well as future directions (§ 6) for the portal.

2 Requirements

Our Grid-enabled portal for physics experiments had several basic requirements. Primarily its function is to allow users to easily run applications over grid resources. A second, key, goal of this project is to not only create a functional portal, but also one that advances the Grid computing effort by creating reusable components that integrate existing technologies. To elucidate these points, we describe a typical *use-case* in the form of steps that a user would take when connected to the portal, its associated requirements, and their generalization.

1. **Authentication.** Using a standard web-browser, the user logs in to the portal with a username and password. New users can request an account by contacting the site's administrator and presenting valid authentication credential(s). Once the user's authenticity is established, access to Grid resources is delegated on their behalf, by uploading a Grid credential, or by configuring the portal to access a credential repository.

2. **Job selection and configuration.** The portal then presents options for a particular application to be run. For instance our first-deployed code is the magneto-hydrodynamic (MHD) astrophysics code ZeusMP, a multi-processor version of Zeus2D (Stone & Norman 1992). Choosing ZeusMP, specific parameters are then displayed for the user to set. They may opt for a collapsing star simulation, to explore the effects of differing values of a star's initial mass. The portal creates a number of seperate jobs, each with a different set of initial conditions, i.e. initial mass. Additional parameters can be set to a range, such that the entire parameter space is available for study.

3. **Resources Selection and Job Execution**. Ideally, the user should not be required to specify any resources leaving the portal backend to obtain these automatically from a registry such as a possible list of common computational and storage resources available to the members of a single virtual organisation. However, the user may want to specify a set of resources he/she is interested in utilising, such as a particular data resource to access. Additionally he/she may specify a deadline by which the results of the execution are required. The portal backend then creates the individual jobs from the user description and maps them to resources in such a way that the user's deadline is met. It submits the job(s) for processing and and monitors system loads in order to efficiently utilise resources and to ensure that the job execution happens within the allocated time.

4. **Monitoring.** Often these jobs will take hours to run. Users may want to monitor progress immediately, via the portal,or alternatively, disconnect and return later if the jobs are likely to be long-lived. For which, the portal (and web) server must be able to be restarted without dropping queued jobs. Users may have several experiments running simultaneously and should be able to choose which to monitor. The end result is that at some point a user can return and verify whether one or more of their experiments have finished.

5. **Output.** An error log displays exceptions for the user to examine. Successful jobs will store their results in a grid storage location such as MySpace (Qin, Davenhall, Noddle & Walton 2003) or on the web server machine. The user will be presented with an interface to browse files, and, for common file formats, such as FITS and HDF, server-side visualisation tools could be made available within the portal too. Depending on these results, the user may alter a number of configuration parameters and re-run the simulation. But once satisfied with the results, pertinent data can be downloaded to a local machine, or stored in a more permanent location on the Grid, if required. Because the resulting output files may be very large, it is important to provide some rudimentary information — through basic visualisation or statistical information — to check data quality or validity before initiating large data transfers.

Thus,from the above requirements it can be seen that we needed to pay special attention to the following aspects of portal development:

Security. Since the user delegates authority to the portal and the portal potentially has many users, security is an important concern. Ideally users' authentication credentials will not be stored on disk of the web server — a very public machine — but rather on a different machine with extra security. All connections to the web server should be made via Secure Sockets Layer (SSL) to protect usernames and passwords. The portal itself should run with special permissions to reduce damage if it is compromised. Portal user accounts need only relate to the portal and not to system accounts. The usual drawback of enhanced security is in reduced ease-of-usage. Our portal tries to strike a balance by having the users delegate their credentials a single time. Subsequent uses of the portal simply require a password to activate the credentials. Such a system allows users to connect to the portal from any machine with a web browser without having their credentials stored on their local workstation. No other software should be required. For example, visualisation services should not require downloading software, other than, perhaps, an embedded Java applet.

Resource Brokering Grid resource brokers (GRBs) undertake the tasks for resource discovery, job scheduling, execution and monitoring and job output retrieval among others. Linking the portal with a GRB backend has the advantage of keeping the portal development free to focus on presentation of content to the users and leaving the complexity of resource allocation and scheduling to the broker. Grid middleware and standards are still in the process of evolution and by keeping the portal free from the vagaries of the underlying infrastructure, the cost of code maintenance is reduced. Also, portal users are able to take advantage of any new scheduling algorithms that may have been introduced by the GRB developers.

Output Visualization Visualization of simulation and analysis data is of most interest to the end-users who, in this case, are physicists. The visualization requirements will vary with the stream of research that is being conducted. For example, astronomy researchers will want to look at the 3D visualizations of the collapsing star simulation described above while high-energy physicists will want to look at histograms generated by analysis of data obtained from particle accelerators. The portal should be customizable to accomodate these varying needs.

There are a number of extra requirements needed to make our portal a valuable addition to the Grid community and not an *ad hoc* and limited solution specific to a domain. For example, we need to reuse existing components where possible, and certainly any portal framework should make use of existing and relevant standards. Much of the current Grid software is in the beta stage, so any software must also be carefully selected to ensure it is well supported and likely to remain available and supported in the long term. A benefit of using beta software is, of course, the ability to give feedback during its development. This not only helps tailor it to any specific needs, but also improves Grid software as a whole.

3 Technology Choices

One of our primary aims is to reuse existng software components and to develop our own reusable components too. In this section we outline which technologies we choose to achieve this goal.

3.1 Grid Middleware

The Globus Toolkit (GT) (Foster & Kesselman 1997) provides functionality to glue together distributed resources to form computational grids and to create Grid-based applications. It provides a secure, uniform interface for resource access and provides file transfer,remote job submission and meta-data query services and is one of the most widely-used middleware packages for the Grid. Typically a choice of Globus implies the need to interoperate with version 2.4 installations of the toolkit; we have found that GT version 3.0.2 clients work well with version 2.4 services. The GT is also the chosen Grid middleware for many international Grid and eScience projects such as the UK eScience Project, LHC Grid, International Virtual Observatory and the Griphyn project. As astronomy and high energy physics communities in Australia are already part of some of these collaborations, it makes sense for us to adopt Globus as the standard for our implementation. GT is increasingly being developed in Java, which is also well suited for building complex web applications. Java provides for stable and rapid development and includes a good Application Programming Interface (API) for security. Java Remote Method Invocation (RMI) is an easy-to-use method for communicating between seperate processes. We employ it to seperate the web server process from the job submission process, allowing us to restart the web server without interrupting users' jobs. We also prefer Java-based frameworks for structured development and easy use of the Globus Toolkit.

3.2 Resource Brokers

Most of the tasks that we have envisaged for Grid execution within astro and particle physics consist of running the application over combinations of various parameters as was shown in the use-case. This is known as the parameter-sweep model (Abramson, Sosic, Giddy & Hall 1995) of computation and the independent nature of the jobs produced within this model makes them suitable for execution on grids which may be prone to fluctuating loads and dynamically appearing and disappearing resources. Nimrod/G (Buyya, Abramson & Giddy 2000) is a resource broker for the parameter-sweep model of computation wherein jobs are created by combining values within the parameter space of an application. The Gridbus Broker (Venugopal, Buyya & Winton 2004) extends the computation-centric Nimrod/G model to distributed data-intensive applications by introducing data-oriented scheduling and the concept of dynamic parameters. The Gridbus Broker has a service-oriented architecture that discovers services at run-time and uses them to shortlist suitable resources for execution and to refine the parameter search space. The Broker was written in Java for easy integration with web application containers such as Tomcat, and can be used standalone or as a library through a well-defined Java API.

The Gridbus Broker has been utilised in the analysis of data generated by the Belle High Energy Physics experiment. It was able to schedule jobs to resources based on their availability and their proximity to the sources of data. The scheduler minimizes the cost and time involved in transmission of the data while guaranteeing shortest job turnaround time. Large repositories of data that have been replicated selectively through a distributed infrastructure are the norm in many scientific grid collaborations. The data requirements within high energy physics and astrophysics communities are well-known (Winton 2003). We found the ability of the Gridbus Broker to access remote data catalogs for scheduling and executing jobs, essential for the usability of this portal.

3.3 Web Technologies

Servlets are a viable choice for reusable web applications, and, therefore, we looked at a number of different frameworks built on top of servlets. This choice of framework was a time consuming effort, as one must find, install and test each. Time well spent though, since the framework choice has an extensive impact on subsequent development.

Struts[1] was used to construct a prototype of the portal interface. Struts has a wide user-base and is relatively easy to use. Although it strictly enforces seperation of content and code, we still found it difficult to create reusable components. Moreover, generic functionality such as file uploading, are not part of the basic Struts framework and had to be written by us. We concluded from the results of the prototype development that decoupled, individual components were needed to create a flexible, reusable portal.

Portlets provide a standard technique for creating web applications for deployment in a *container portal*. Each application is self-contained and is displayed in a sub-window of the portal. The portlet framework allows for communication between portlets without the need for tight inter-portlet coupling. Java Specification Request (JSR) 168 (Abdelnur, Chien, Hepper et al. 2003) defines the standard for portlets, and several systems implement it, including Apache Jetspeed[2] and GridSphere (Novotny, Russell & Wehrens 2003). Since portlets are in their infancy, many of the portlet containers are incomplete and function more as proof of concept vehicles. As well as meeting our technical requirements, we needed a system with good user support and longevity. In summary, the portlet container requirements were that it:

- implement JSR 168 and not extend functionality in a non-portable way,
- be well designed, following software engineering principles,
- be relatively easy to install and get started,
- be robust in a production environment,

[1]`http://jakarta.apache.org/struts`
[2]`http://portals.apache.org`

- come with source code, to allow for modification and bug fixes,
- be well supported, preferably providing direct contact details for development programmers,
- be long-lived, with a clear plan for future development,
- come with generic portlets for functions such as file uploading, and
- enable rapid development.

GridSphere was our choosen, preferred portlet container. It is based on the popular Apache Tomcat[3] web-application container. Jetspeed was another possibility which we explored, but anecdotal evidence suggests it is too immature at this point for deployment in a working environment. Several other portlet containers were examined but either they, too, were immature or didn't have sound user support.

Unfortunately JSR 168 does not provide all the functionality we needed, particularly with respect to inter-portlet communication. GridSphere has extended JSR 168, but this means some parts of our code will be non-portable. Another drawback of GridSphere was the slow development cycle, even on fast machines. This was accentuated because we were learning a system which involved much trial and error. New code had to be compiled, deployed and then the Tomcat server needed to be restarted. The ApacheAnt[4] build system, which is used by GridSphere, allowed us to semi-automate this process.

4 Design

Having selected GridSphere and Gridbus Broker we needed to design a functional system which is reusable, secure and extensible. We outline the major design choices here, beginning with an overview of the whole system and then examining the various components.

A schematic of the portal system is shown in Figure 1, and a component-oriented diagram is shown in Figure 2. At the core of the system is the web server machine, and at a simple level, the user is aware only of this machine. It runs a web server with Tomcat and GridSphere together providing the portlet container. GridSphere handles user sessions, but between sessions it cannot run background jobs. Instead, it uses a background process, Gridbus Server — based on Gridbus Broker — to handle submission and control of jobs. This allows the user to exit the portal and return later whilst her jobs continue to execute. In addition, GridSphere can be restarted without affecting users' jobs. The Gridbus Server uses Globus to submit jobs to various resources. Credential management is handled by generic GridSphere portlets which pass the necessary information to the Gridbus Server.

GridSphere and Java allow us to use services and interfaces to decouple individual components. Portal services can be used by any portlet, and they provide a simple interface to subsystems which provide security, job submission and monitoring. Each service is defined by a Java interface which can be implemented in several different ways without affecting the portlets. For instance, the job submission services can use Nimrod/G or Gridbus Broker without the portlet being aware of the difference.

Plan files encode application requirements for an experiment using Nimrod/G's simple declarative programming language which is also supported by the Gridbus Broker. A plan file specifies the parameter values, the input files if present, the program(s) to be executed and the output files to be copied back if necesssary. They are created by application-specific portlets after which generic portal components treat all plan files alike. Therefore we can can add new applications just by creating a new portlet which writes a plan file based on the application's parameters. Thus, by hiding the plan file format from the user, a further layer of abstraction is provided and increases the ease-of-use of the portal.

An experiment is a set of related jobs created using a single work-flow plan. A single instance of the Broker is created for each experiment. The portal needs to support multiple users and multiple user experiments, as well as persistence across user logins. Using Java RMI we created a Gridbus Broker server which runs as a standalone process (see figure). The portal communicates with the server via a set of RMI remote objects which are delegates for the real object in the server. Using this system we can restart the web server and monitor the state of jobs using other applications. Without it, a shutdown of the web server would terminate all the on-going users' jobs.

By choosing Globus, Java and GridSphere we can utilise existing tools of credential management. We employed MyProxy[5] to store user credentials and to delegate proxies for user resources. The server runs in a seperate process, possibly on a seperate machine. Users can specify a MyProxy server to use, or use one associated with the portal. The portal uses GridSphere credential management portlets to interact with MyProxy via a network socket. There are portlets to upload credentials, map credentials to resources and to obtain a short-term delegation proxy to use on those resources. Once the delegated proxy is obtained, the portlet passes it to Gridbus Server to use.

Users need to see output (that is, `stdout` and `stderr`) as well as the data files from the jobs. The Broker uses Globus Resource Allocation and Management (GRAM) batch submission to reduce the gatekeeper's load. This increases reliability as the GRAM job handle can be used to retrieve job status and output even in the case of Broker failure. If standard out is required (for example, if a job fails) we ask GridbusBroker to fetch it in order to display it in HTML to the user. Data files are copied back to the web-server using Global Access to Secondary Storage (GASS). Other protocols can be used, as specified in the plan file so we can easily add other storage locations such as MySpace. Files must be stored in a user-specific, experiment-specific directory for the user to browse. We provided a generic file browsing portlet which uses existing tools to provide data visualisation to users. We plan to also provide the distributed volume renderer (Beeson, Barnes & Bourke 2003) and the Remote Visualisation Service[6] which use remote machines to perform visualisation, reducing the load on the web-server and allowing us to visualise large, remote datasets.

5 Implementation and Experiences

GridSphere provided a solid foundation for our portal implementation but had a steep learning curve. Implementing the infamous "Hello World" in a portlet is not a trivial exercise! To a large extent, this is the nature of such containers. The software engineering

[3]`http://jakarta.apache.org/tomcat`
[4]`http://ant.apache.org`

[5]`http://grid.ncsa.uiuc.edu/myproxy`
[6]RVS, under development by the Australian Virtual Observatory

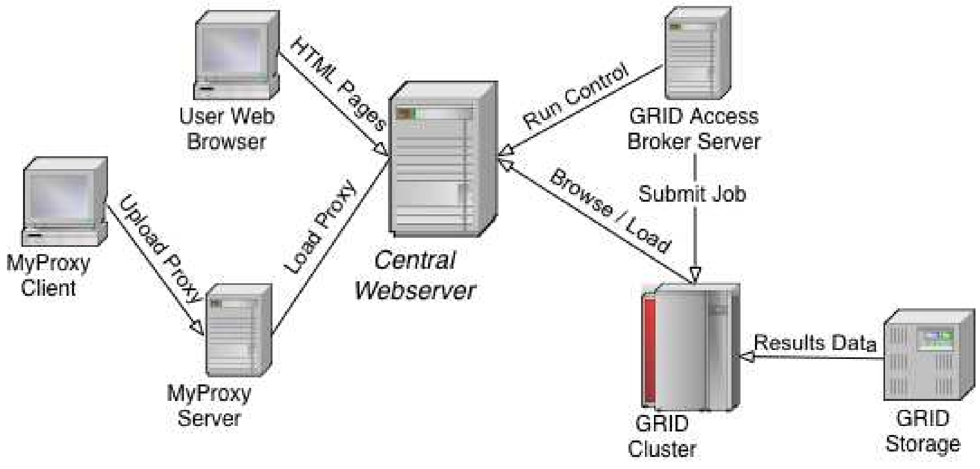


Figure 1: Schematic of the Grid-enabled physics portal system.

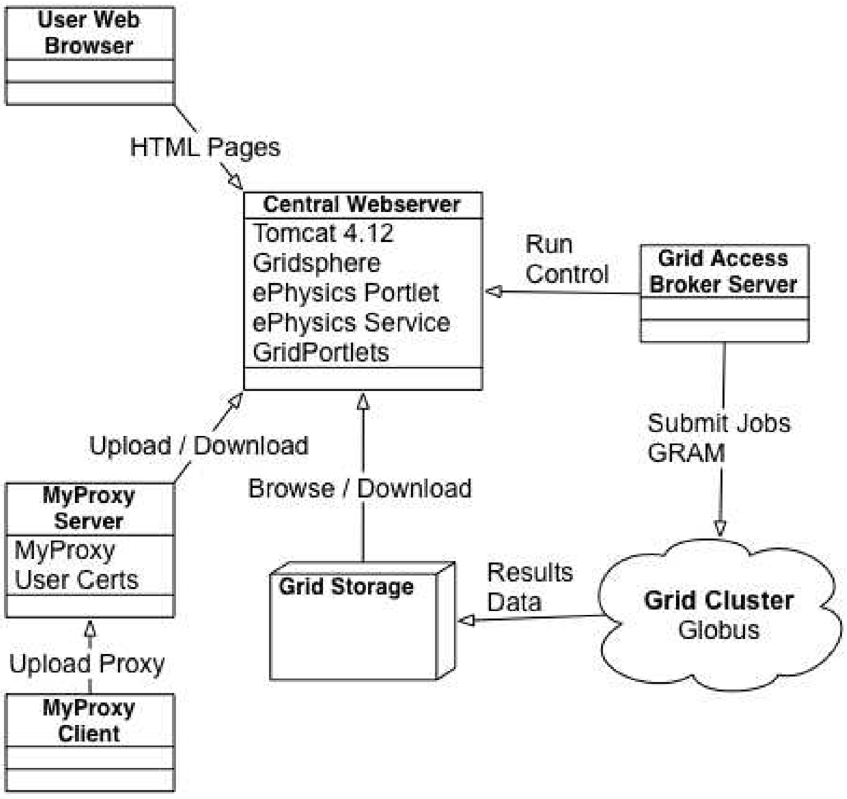


Figure 2: Components of the Grid-enabled physics portal system.

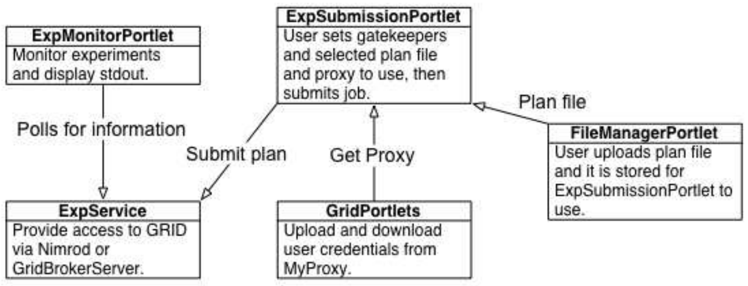

ExpMonitorPortlet
Monitor experiments and display stdout.
ExpSubmissionPortlet
User sets gatekeepers and selected plan file and proxy to use, then submits job.
Plan file
FileManagerPortlet
User uploads plan file and it is stored for ExpSubmissionPortlet to use.
Polls for information
Submit plan
Get Proxy
ExpService
Provide access to GRID via Nimrod or GridBrokerServer.
GridPortlets
Upload and download user credentials from MyProxy.

Figure 3: Portlets developed and used in the Grid-enabled physics portal system.

focus, good documentation and user forums of GridSphere are indispensible to its users. But, once familiar with GridSphere, there are many useful elements available to the programmer. These include message passing between portlets, persistence support via databases and user session support. GridSphere uses the Model-View-Controller (MVC) paradigm which is already popular and widespread. A key element of our portal was reusability: MVC enforces the separation of form and content which greatly facilitates this reusability. The portlets we developed (or used, in the case of GridPortlets) for deployment in GridSphere are shown in Figure 3. We found Tomcat to be a good, stable server platform with a wide user base. GridSphere is specifically designed for Tomcat and the two work together well. When using Tomcat there is a known problem which affects the restart of modified and redeployed web applications. Often the whole container must be restarted, which is time-consuming and error-prone. This problem is exacabated by the extra layer of GridSphere. The resulting slow development cycle significantly impacted our development speed.

An advantage of developing complex web applications in Java is the good support via build and test tools. We used Junit, Cactus and Ant to provide testing, container-based testing and building support. Such tools are really essential due to the complex build and deploy system. They were all easy to use and alleviated some of the problem concerning slow development mentioned above. Figure 4 shows the completed portal, with job submission, monitoring and file management portlets active. Behind the user interface lies the system which performs the work — Gridbus Server, Gridbus Broker, Globus and MPI. Gridbus Broker uses Globus middleware to submit jobs to remote machines, although it allows for other grid middleware. Globus calls are complex and not always robust. Gridbus Broker provided an invaluable abstraction from Globus: its simple interface meant that Globus-specific problems could be handled at a low level. We found it essential to allow Globus calls to fail and then retry, rather than failing on the first attempt.

We feel we made a good decision to use RMI in Gridbus Server to communicate between processes. The alternative was a web service, but in our experience development of web services is extremely slow and error prone, particularly for stateful services. RMI was a quick and elegant solution and most of the code can be reused when we create a web service.

The atomic unit of computation within the Gridbus Broker is called a job. Monitoring of executing code is limited to whether a job is waiting, running or finished. Depending on the code being run, jobs can be very long, and so the user may often want to know how far through an individual job is, not just that it is executing. To an extent, we can create more, smaller jobs, but really there needs to be some feedback system. The core problem is that each code is different but we handle them generically. We must accept this limitation since by design we choose not to modify existing codes.

6 Future Directions and Conclusion

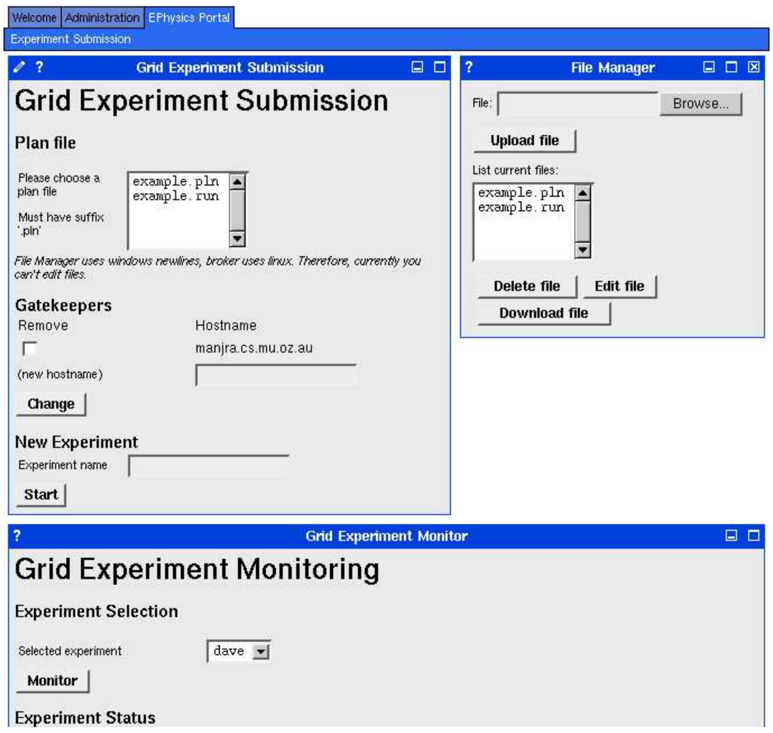
As well as adding more applications (beyond ZeusMP) there are a number of enhancements which would provide a more flexible and user-friendly system. We describe these possiblities here.

When a new application is deployed using the portal, it requires the user to set a number of parameters. We use the plan file to record these parameters. Currently users must write the plan file which is easy but time-consuming. A better alternative would be an HTML form for the user to fill out. The form would then be processed to create a suitable plan file. For simple applications, this would suffice, but many applications have numerous related parameters. A better user-interface could be provided, using a Java applet for example.

The portal interface to the broker has been made as independent of the actual broker as possible. We would like to add support for other resource brokers such as Nimrod/G in order to take advantage of their capabilities. A significant portion of scientific workloads is in the form of tasks linked together to form workflows. We would like to add support for Grid workflow systems (Yu & Buyya 2004) so that such workloads can be realised through the portal. We envisage presenting the user with a choice of resource brokers and workflow management systems so that he/she can select one depending on his requirements.

It is important to store the data generated by the resource broker (or a workflow management system) during job execution so that in case of failure of the broker, the user can study this data and restart the execution from the point of failure. This data would also be valuable as a record of Grid performance and also for accounting purposes. Currently, the Gridbus broker lacks this capability as the data lasts only for an instance of the broker. Nimrod/G is based on a SQL database and provides better job persistence. We plan to actively engage developers wherever possible so that this requirement is fulfilled for all resource brokers and workflow systems that interface to the portal.

To conclude, the portlet approach to Grid computing provides scientists with a productive environment and a rich tool set for large-scale, distributed data processing. As has been presented here, for astrophysics and high energy physics data, legacy analysis software can be redeployed within this framework. And, as these frameworks mature, web-based application tools, like GridSphere, will occupy a central place in the development of global research networks.



Grid Experiment Submission

Plan file

Please choose a plan file

Must have suffix '.pln'

example.pln
example.run

File Manager uses windows newlines, broker uses linux. Therefore, currently you can't edit files.

Gatekeepers

Remove	Hostname
☐	manjra.cs.mu.oz.au
(new hostname)	

New Experiment

Experiment name

File:

List current files:

example.pln
example.run

Grid Experiment Monitoring

Experiment Selection

Selected experiment: dave

Experiment Status

Figure 4: The portal user interface, showing job submission, monitoring and file management portlets.

Following which, scientists may well have most of the tools for real e-Science work, bringing worldwide computing resources to their local laboratory.

7 Acknowledgements

We thank a number of people for their ideas, contributions and effort in developing the portal: Slavisa Garica, Colin Enticott, Lyle Winton, Andrew Melatos and Rajkumar Buyya.

References

Abdelnur, A., Chien, E., Hepper, S. et al. (2003), *JSR-000168 Portlet Specification.* *`http://jcp.org/aboutJava/communityprocess/final/jsr168/index.html`

Abramson, D., Sosic, R., Giddy, J. & Hall, B. (1995), Nimrod: A Tool for Performing Parametised Simulations using Distributed Workstations, *in* 'Proc. of 4th IEEE Symposium on High Performance Distributed Computing', IEEE CS Press, Los Alamitos, Calif., USA, Virginia.

Beeson, B., Barnes, D. G. & Bourke, P. D. (2003), 'A distributed-data implementation of the perspective shear-warp volume rendering algorithm for visualisation of large astronomical cubes', *Publ. Astron. Soc. Aust.* **20**, 300–313.

Buyya, R., Abramson, D. & Giddy, J. (2000), Nimrod/G: An Architecture for a Resource Management and Scheduling System in a Global Computational Grid, *in* 'Proc. 4th Int. Conf. on High Performance Computing in Asia-Pacific Region (HPC Asia 2000)', IEEE, IEEE CS Press, Los Alamitos, Calif., USA.

Foster, I. & Kesselman, C. (1997), 'Globus: A Metacomputing Infrastructure Toolkit', *The International Journal of Supercomputer Applications and High Performance Computing* **11**(2), 115–128.

Foster, I., Kesselman, C. & Tuecke, S. (2001), 'The Anatomy of the Grid: Enabling Scalable Virtual Organizations', *Lecture Notes in Computer Science* **2150**.

Gannon, D., Fox, G., Pierce, M. et al. (2003), *Grid Portals: A Scientist's Access Point for Grid Services.* *`http://www.extreme.indiana.edu/\~gannon/ggf-portals-draft.pdf`

Novotny, J., Russell, M. & Wehrens, O. (2003), GridSphere: A Portal Framework for Building Collaborations, *in* 'Proceedings of the 1st International Workshop on Middleware in Grid Computing', Rio de Janeiro, Brazil.

Qin, C. L., Davenhall, A. C., Noddle, K. T. & Walton, N. A. (2003), *MySpace: Personalized Work Space in AstroGrid.* Proceedings of UK e-Science All Hands Meeting 2003, 2-4th September, Nottingham, UK.

Stone, J. M. & Norman, M. L. (1992), 'ZEUS-2D: A radiation magnetohydrodynamics code for astrophysical flows in two space dimensions. I - The hydrodynamic algorithms and tests', *Astroph. J. Supp.* **80**, 753–790.

Venugopal, S., Buyya, R. & Winton, L. (2004), A Grid Service Broker for Scheduling Distributed Data-Oriented Applications on Global Grids, *in* 'Proc. of the 2nd International Workshop on Middleware in Grid Computing(MGC 2004)', ACM Digital Library, Toronto, Canada. to be published.

Winton, L. (2003), 'Data Grids and High Energy Physics: A Melbourne Perspective', *Space Science Reviews* **107**, 523–540.

Winton, L. et al. (2003), 'Australian Belle Analysis Data Grid'.
*http://roberts.ph.unimelb.edu.au/epp/grid/badg/

Yu, J. & Buyya, R. (2004), A Novel Architecture for Realizing Grid Workflow using Tuple Spaces, *in* 'Proc. of the 5th International Workshop on Grid Computing(GRID 2004)', IEEE CS Press, Los Alamitos, Calif., USA, Pittsburgh, USA.