gridMonSteer: Generic Architecture for Monitoring and Steering Legacy Applications in Grid Environments

Ian Wang[1,2], Ian Taylor[2,3], Tom Goodale[2,3], Andrew Harrison[2] and Matthew Shields[1,2]
[1]School of Physics and Astronomy, Cardiff University
[2]School of Computer Science, Cardiff University
[3]Center for Computation and Technology, Louisiana State University

Abstract

Grid environments present a virtual black box to scientists running legacy applications, with the Grid infrastructure effectively hiding the running application on a resource over which the scientist generally has limited or no control. Existing monitoring systems that allow Grid-enabled applications to communicate their progress and receive steering information are inapplicable as they require code modification, and this not possible in true legacy scenarios. While a black box approach may be acceptable in batch execution scenarios, it means the Grid is cut off to legacy applications where interactions or intermediate results are required.

In this paper we present gridMonSteer, a simple, non-intrusive architecture that allows scientists to receive intermediate results and interact with legacy applications running in a Grid environment. This architecture is Grid middleware independent and can be employed to monitor applications submitted via any Grid resource manager (e.g. GRAM, GRMS or Condor/G) or running within a Grid service framework. We present a case study describing how gridMonSteer enables legacy applications to act as active components in Grid workflows, dynamically driving and steering the workflow execution.

1 Introduction

Within a Grid environment, the vast majority of data analysis applications executed by scientists can be considered legacy. By this we mean that they are unaware of their Grid environment and any mechanisms it provides to communicate with users or controller applications. Although systems exist that facilitate communication with Grid applications [1][2][3], there is a general reluctance to re-engineer or rewrite legacy applications to utilize any communication mechanisms available due to the cost and lack of an agreed standard. For scientists using legacy applications the Grid environment acts as virtual black box, in which jobs are submitted and executed but retrieving intermediate results or interacting with the running application is very difficult.

Opening up the Grid environment to interactive legacy applications will allow us to much better integrate these applications into distributed problem solving environments. Rather than just viewing legacy application as standalone entities, we can now employ them as active components in larger complex decision-based applications. For example, the intermediate results generated by a legacy application can be used to dynamically steer a data-driven Grid workflow.

In this paper we present gridMonSteer, a generic architecture for monitoring and steering applications in a Grid environment, together with its current implementation. This implementation provides many monitoring and steering capabilities both at the application level and in terms of a dynamic Grid workflow. The principal benefit of gridMonSteer is that simple, often generic solutions to Grid application monitoring/steering scenarios can be developed without modifying any application code. These solutions are Grid middleware independent and function equally whether the application is submitted using GRAM [4], GRMS [5] or Condor/G [6], or even run within a Grid service

framework [7].

The gridMonSteer architecture consists of two parts: an application wrapper that runs in the same execution environment as the application, and an application controller that generally runs locally to the user. As all the communication in gridMonSteer is initiated by the application wrapper, all communication is outbound from the Grid resource. This situation is generally allowed by firewalls as it implies that the application wrapper has already complied with the security requirements of that resource.

The gridMonSteer application wrapper is executed as part of the application job submission. The arguments specified in this submission tell the wrapper what information to request from/notify to the gridMonSteer controller, and these can be tailored to a specific application scenario. This information could for example be immediate notification of output files created by the application, incremental requests for input to the application (e.g. via the standard input), requests for input to be forwarded to the application via a network port, or notification of the state of the application.

A gridMonSteer application controller is a service generally run locally by the user that receives information requests and notifications from the application wrapper. A controller exposes a simple interface that can be invoked by the application wrapper; in the current implementation this is web service interface. As long as this interface is exposed, the actual behavior of controller is undefined and can be tailored to individual or generic application requirements; for example:

- a generic controller could combine incremental input to the standard input with immediate output from the standard output to conduct an interactive command-line session with a Grid application.

- a visualization controller could receive immediate notification of image files to produce a live animation of output from a Grid application.

- a workflow controller could use immediate output file notification from an application to dynamically steer the execution of a Grid workflow, e.g. to launch an immediate response to a significant event.

The rest of this paper further describes the architecture, implementation and application of gridMonSteer. In Sections 3 and 4 respectively we explain the gridMonSteer architecture and our implementation of this architecture. We present a case study in Section 5 that uses gridMonSteer in combination with Triana [8][9] and Cactus [10][11] to dynamically steer a Grid workflow. We draw our conclusions from this work in Section 6. First however, in Section 2, we look at work related to the architecture presented in this paper.

2 Related Work

The number of scientific applications developed to operate in a Grid environment is tiny when compared with the vast number that can be considered to be legacy codes when run within such environments. Grid legacy applications are not implemented with hooks into the application monitoring systems that are being developed for Grid environments, such as Mercury [2] and OMIS [3] (see [1] for a survery), and therefore have no method for communicating their progress/intermediate results or receiving steering information.

One obvious solution to this problem is to "Grid-enable" these applications by allowing them to interact with a monitoring system. However, enabling an application to interact with existing monitoring systems requires modification of the application source code. In a legacy application scenario this is often not possible because the source code is unavailable. If the source code is available, the difficulty and cost of modifying unsupported code is likely to prove prohibitive, especially when standards for monitoring architectures in Grid environments are still being developed.

Without access to monitoring systems, when a legacy application is submitted to execute on the Grid via a resource manager, such as GRAM, GRMS or Condor/G, that resource manager becomes the principal point of contact regarding the progress of that application. However, the progress information available to a user is generally extremely limited, not normally stretching much beyond the status of the job and its id. Although a user could expect to have access to the execution directory of the legacy application, this access is pull based, so retrieving intermediate results in

a timely fashion would require constant polling of the remote resource. This is very inefficient compared to the result notification approach used in gridMonSteer and unlikely to prove a usable architecture for effective application controllers.

An alternative to using a resource manager to execute a legacy application is to expose the functionality of the application as a Web or Grid service. To achieve this requires an application wrapper that mediates between the service interface and invocations of the application. A couple of approaches, including SWIG [12] and JACAW [13], have adopted a fine-grained approach where the actual source code is wrapped exposing the application interface. However, in addition to granularity and complexity considerations, this approach requires access to the application source code, a situation that is generally not applicable to legacy applications.

A coarse grained approach, where the application is treated as a black box interfaced with using stdin/stdout or file input/output, is used in other projects, such as GEMLCA [14] and SOAPLab [15]. This approach however typically provides little additional benefit over using a resource manager with regards to application interaction. This is illustrated by the fact that, rather than wrapping the application process directly, GEMLCA actually wraps a Condor job submission.

One system that does employ similar components to that of gridMonSteer is Nimrod/G [16][17], a Grid based parametric modeling system. In Nimrod/G an application wrapper is used to repeatedly to run parametric modeling experiments with varying parameters and to pre/post stage results. A central database is queried by the application wrapper to determine the next experiment to execute. Such a scenario could easily be implemented using the dynamic scripting capacity of the current gridMonSteer implementation (see Section 4.1.1), with the central database acting as the application controller. Unlike gridMonSteer, Nimrod/G does not provide a generic legacy application monitoring or steering architecture outside dynamic scripting of parameter modeling experiments.

3 gridMonSteer Architecture

The gridMonSteer architecture (see Figure 3) consists of two parts, an application wrapper that is executed on a Grid resource and an application controller that is generally run locally to the user. As gridMonSteer application wrapper is submitted to the Grid as a standard job, this architecture works with any Grid resource manager (e.g. GRAM, GRMS or Condor/G). Unlike other Grid monitoring approaches, the gridMonSteer application wrapper is non-intrusive and therefore does not require modifications to the application code.

The process of running an application under gridMonSteer is initialized when, rather than submitting an application to a Grid resource manager for execution, a gridMonSteer application wrapper job is submitted instead (Step 1 in Figure 3). The original application executable and arguments are supplied as arguments to the wrapper job. As the wrapper job is submitted to the resource manager as a standard job, all the features of the chosen resource managers job description are available, as they would be if submitting the application directly.

As well as the application executable and arguments, the address of a gridMonSteer application controller service is also supplied as an argument to the wrapper job. An application controller is a service (e.g. web service) that exposes a simple network interface (see Section 4.2). This interface is invoked by the application wrapper to request information (e.g. input files) from and notify information (e.g. output files) to the controller. Other arguments supplied to the wrapper job specify exactly which information is notified to/requested from the controller by the application wrapper, as will be discussed in Section 4.1.

The gridMonSteer application wrapper job is executed on a Grid resource as specified in the job description and determined by the resource manager (Step 2). The application wrapper is then responsible for launching the original application (Step 4), however, before this is done, any input files that are required to be pre-staged in the execution directory are requested from the controller by the application wrapper (Step 3).

After the application has been launched by the application wrapper, the application wrapper regularly monitors the execution directory for the creation/modification of output files (Step 5). If any changes occur to files required by the application controller, as specified in the wrapper job arguments (see Section 4.1), then the controller is noti-

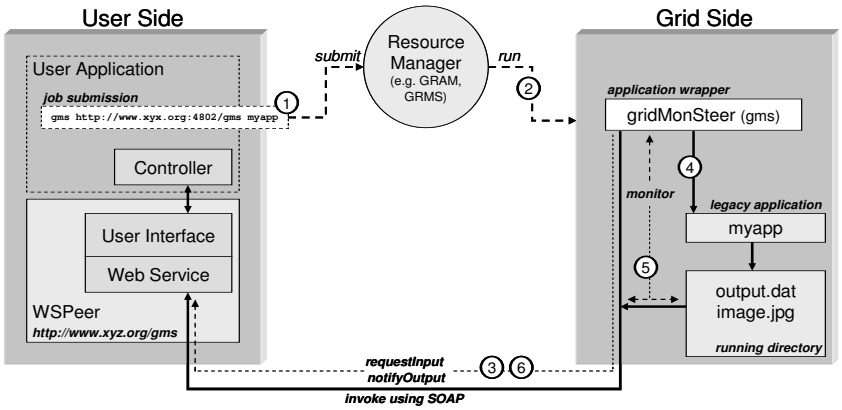

Figure 1: gridMonSteer architecture using a web service controller launched using WSPeer.

fied via its network interface (Step 6).

In addition to handling input/output files, the application wrapper can perform other notification/steering tasks such as job state notification and dynamic script execution. We detail such functionality further in Section 4.

As long the application controller exposes the predefined controller interface, the gridMonSteer architecture does not define how the controller handles the information requests/notifications that it receives from the wrapper.

3.1 Security Considerations

In a Grid environment most hosts are behind firewalls that typically create a one-way barrier for incoming traffic on specified ports, thereby disabling incoming connections from external controllers. Hosts can also employ address translation that is not Internet facing and therefore cannot be contacted directly, such as Network Address Translation (NAT) systems. Once this job is started in such an environment direct communication from an application controller becomes difficult or impossible.

In the gridMonSteer architecture all communication is initiated by the gridMonSteer application wrapper. As the application wrapper executes in the same environment as the Grid application this means that communication is always outbound from Grid resources. This is a situation that is generally allowed by firewalls as it implies the application wrapper has already complied with the security requirements of that resource. In the case of gridMonSteer, as the application wrapper is submitted as a standard Grid job, these security requirements are enforced by the resource manager used to execute the wrapper job. Outbound communication is also compatible with address translation systems such as NAT.

As long as a secure transport protocol such as HTTPS is used for communication between the application wrapper and controller, identity certificates can be used by the application wrapper to verify it is communicating with a trusted controller. Further from that, it is up to the application controller implementation not to introduce security flaws. The application controller should operate under the same trust relationship that allowed the application wrapper job to be executed on the Grid resource initially.

4 gridMonSteer Implementation

The designs of the current gridMonSteer application wrapper and application controller implementations are described in Sections 4.1 and 4.2 respectively. At present the application wrapper is implemented in Java, however a C version is under development. In the current implementation the controller exposes a web service interface, and can be implemented in any language so long as this

requirement is met.

4.1 Application Wrapper

As described in Section 3, in the gridMonSteer architecture, rather than submitting an application directly to a Grid resource manager, an application wrapper job is submitted. This application wrapper is then responsible for requesting input information from the application controller, launching the original application, monitoring the output from this application, and notifying this information to the application controller.

When the application wrapper job is submitted the following arguments must be specified:

- The address of the application controller, which in the current implementation is the address of a web service.
- The application executable and any arguments that should be passed to this application.

Additional arguments can then be used specify what information is requested from and notified to the application controller based on the individual application scenario. We outline these arguments in Sections 4.1.1 and 4.1.2.

4.1.1 Monitoring Arguments

The set of gridMonSteer arguments that we refer to as monitoring arguments are used to specify the information that is sent from the application wrapper (grid side) to the controller (user side). The following arguments are used to specify which files output by the legacy application are notified to the controller, and the notification policy used:

`-out` - Sends output files to the controller after application execution has finished.

`-monitor` - Same as `-out` except the output files are sent immediately after their creation/modification.

`-update` - Same as `-out` except incremental updates to the output files are sent periodically during application execution.

Each of the arguments above take a file list which details the names of files and/or directories to be monitored, and may include wildcards (e.g. *.jpg). The keywords `STDOUT` and `STDERR` can be used in the file list to specify monitoring the standard output and standard error streams respectively.

Although the basic file notification arguments detailed above covers most monitoring situations, in some scenarios applications generate a large number of output files of which the scientist is only interested in a subset determined during application execution. For these scenarios gridMonSteer uses a register/select system, whereby output files are registered with the application controller when they become available and the application wrapper requests the controller/user to select the files that are of interest. The `-xout`, `-xmonitor` and `-xupdate` arguments provide register/select compliments for the basic file notification arguments.

In addition to output file notification, the `-state` argument can be used to indicate that application state information, such as host name, running directory and run time, should be notified to the application controller periodically during execution. Such information is particularly useful in a grid environment where the application may reside in a job queue for some time and execute on an unfamiliar, remote resource.

4.1.2 Steering Arguments

The gridMonSteer steering arguments specify the information that is requested from the application controller (user side) by the application wrapper (grid side). The `-run` argument specifies that the wrapper asks the whether the application should be re-run after it has exited. On each run the job arguments or even the job executable can be altered. This effectively allows the controller to execute a dynamic script via the application wrapper. This offers considerable benefits over the alternative, which would be to submit a succession of independent jobs, such as not incurring multiple scheduling delays and not having to deal with the complexity of the independent jobs being run in different environments.

The application wrapper can request input files from the controller to be staged in the execution directory before the application is run/re-run. This is specified using the following arguments:

`-in` - Requests input files from the controller prior

to application execution.

`-append -` Same as `-in` except incremental updates to the input files are repeatedly requested during application execution.

As with the equivalent arguments output files (see Section 4.1.1), the above commands take a file list detailing the files to be requested. This file list can include the keyword `STDIN` to indicate requesting the standard input.

4.2 Application Controller

A gridMonSteer application controller is a process that receives input requests and output notification from the application wrapper via a network interface. An application controller is generally run locally to the user, allowing the output of an application to be visualized on the users desktop for example, however it is equally valid for a remote application controller to be used. Also, an application controller can be used to control multiple jobs. These jobs can be independent applications or separate parts of a parallel application.

In the current implementation application controllers are exposed as web services, however the gridMonSteer architecture could function equally effectively using an alternative communication framework. In this implementation, the controller is required to expose an interface containing six operations (*runJob, getInput, notifyOutput, notifyState, registerOutput, selectOutput*), and providing this interface is exposed, the actual behavior of the controller is undefined.

In the scenarios we are currently developing (see Section 5 for an example), we use WSPeer [18] to allow the application controller to dynamically expose itself as a web service for the duration of the jobs it is managing. Although this is beneficial for developing interactive application controllers, any web service implementing the gridMonSteer controller interface could be employed as an application controller.

5 Case Study: Using Cactus to Steer Triana Workflows

In this case study we illustrate how gridMonSteer can be used to steer complex Grid workflows. Here, we consider the case that the workflow itself is the Grid application rather than the current perception that the legacy code being invoked remotely is the application. Although, this example is simple, it could be extended to data driven applications that involve complex decision-based workflow interactions across the resources.

The specific scenario we illustrate in this case study is a gridMonSteer wrapped Cactus [10][19] job executed within a Triana workflow [8][9], with Triana both submitting the Cactus job to the Grid resource manager and acting as gridMonSteer controller for that job. As controller, Triana receives timely notification of intermediate results via gridMonSteer that are then used to steer the overall workflow being executed.

The original idea for gridMonSteer came from a SC04 demonstration in which Triana was used to visualize two orbiting sources simulated by a Cactus job running in a Grid environment [20]. Accessing intermediate results is a typical requirement for Cactus users who want to monitor the progress of their application or derive scientific results. For this scenario a specific Cactus thorn was coded to notify the files created by Cactus to the Triana controller after each time step, however this is a Cactus specific solution. In this example we recreate this demonstration except Cactus is treated as a true legacy application, and gridMonSteer provides non-intrusive monitoring and steering capabilities.

Cactus simulations present a number of interesting cases for gridMonSteer. The output from Cactus can be in a number of formats, including HDF5, JPEG, and ASCII formats suitable for visualizing with tools such as X-Graph and GNUPlot. In some instances, such as JPEG images, the output files are overwritten after each time step, where other output files are appended, such as X-Graph files.

Grid job submission in Triana is conducted via the GridLab GAT [5], a high level API for accessing Grid services. The GAT uses a pluggable adaptor architecture that allows different services to be used without modifying the application code. Current resource manager adaptors for the Java GAT include GRAM, GRMS and local execution. Although different adaptors can be used within the GAT, as gridMonSteer is generic to any resource manager, Triana can run gridMonSteer wrapped jobs via the GAT without having to modify the job submission to take account of different resource

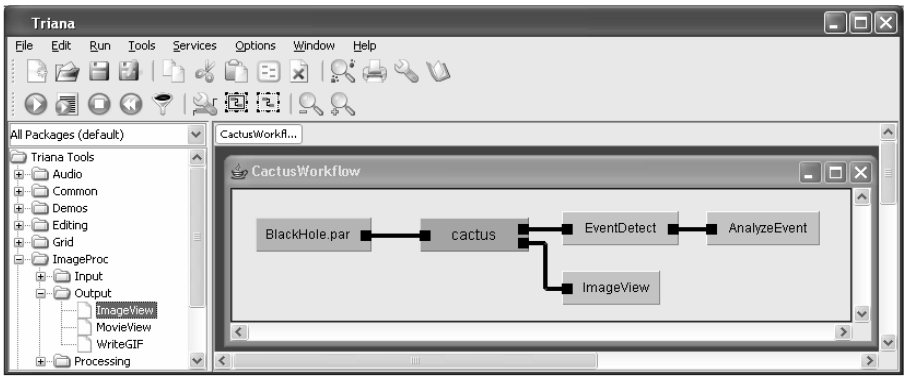

Figure 2: An example Triana workflow for executing a gridMonSteer wrapped Cactus job.

manager bindings.

When submitting a Cactus job, the arguments specified for the gridMonSteer application wrapper by Triana include:

`-xmonitor:*.jpg` - Monitor for creation and modification of JPEG files.

`-xupdate:*.xg;STDOUT` - Monitor for updates to the X-Graph ASCII files and standard output.

`-state` - Notify the state of the Cactus job (started, stopped etc.).

These arguments cover the required monitoring and steering for a Cactus job, as described above.

In Figure 5 we show a gridMonSteer wrapped Cactus job embedded in an example Triana workflow. Data flow within Triana is represented left to right, so in this example BlackHole.par is providing the input to the Cactus job. As BlackHole.par is a File object, this represents a pre-staged file for the Cactus job.

As the select/register arguments (`-xmonitor`, `-xupdate`) are specified for the gridMonSteer wrapper, the outputs from the Cactus job are registered with Triana when they are created/updated. A simple user interface allows the scientist to interactively select from the available outputs and map them to output nodes on the Cactus workflow task. This can be used for example to change the simulation elements they are studying if they observe an interesting event. The selected outputs are immediately notified to Triana by the application wrapper and used to drive the remaining workflow.

In the example shown in Figure 5, there are two outputs from the Cactus workflow task. One output is directed to ImageView, a local component that enables the visualization of the image files generated by the running Cactus job. The other output is directed to EventDetect, which represents the tools we are developing within Triana to detect interesting events in Cactus output data, such as the apparent horizon of a black hole simulation. When an interesting event is detected it can be analyzed locally (as represented by AnalyzeEvent), or alternatively further Grid jobs can be launched to process the event remotely.

6 Conclusion

In this paper we have outlined gridMonSteer, a generic architecture for monitoring and steering legacy applications in Grid environments. Unlike existing Grid monitoring solutions, the gridMonSteer approach is non-intrusive and therefore does not require any modification to the application code. Furthermore, as the gridMonSteer application wrapper is executed as a standard job, it is generic to any Grid resource manager and can be used within a Grid service framework. All communication is initiated by the application wrapper and is therefore outbound from the Grid resource, a situation that is generally allowed by firewalls as it implies the job has already complied with the security requirements of that resource. The behavior of the application controller that is employed to handle information requests/notification from the

wrapper can be tailored to individual or generic application requirements.

In this paper we presented a case study using gridMonSteer to monitor a legacy application running as a component within a Grid workflow. In this case study, the intermediate results from the legacy application where notified by gridMonSteer to the workflow controller and used to dynamically steer the workflow execution. This is an example of how gridMonSteer enables interactive legacy applications to be used within Grid environments in situations that were previously unavailable.

References

[1] S. Zanikolas and R. Sakellariou, "A taxonomy of grid monitoring systems," *Future Generation Computer Systems*, vol. 21, pp. 163–168, January 2005.

[2] Z. Balaton and G. Gombas, "Resource and job monitoring in the grid," in *Proceedings of the Ninth International Euro-Par Conference, Vol. 2790 of Lecture Notes in Computer Science*, Springer-Verlag, 2003.

[3] B. Balis, M. Bubak, T. Szepieniec, R. Wismuller, and M. Radecki, "Monitoring grid applications with grid-enabled OMIS monitor," in *Proceedings of the First European Across Grids Conference, Vol. 2970 of Lecture Notes in Computer Science*, pp. 230–239, Springer-Verlag, 1997.

[4] K. Czajkowski, I. Foster, N. Karonis, C. Kesselman, S. Martin, W. Smith, and S. Tuecke, "A Resource Management Architecture for Metacomputing Systems," in *Proc. IPPS/SPDP '98 Workshop on Job Scheduling Strategies for Parallel Processing*, pp. 62–82, IEEE Computer Society, 1998.

[5] "The GridLab Project." See web site at: `http://www.gridlab.org`.

[6] J. Frey, T. Tannenbaum, M. Livny, I. Foster, and S. Tuecke, "Condor-G: A Computation Management Agent for Multi-Institutional Grids," in *Proceedings of the 10th IEEE International Symposium on High Performance Distributed Computing (HPCD-'01)*, 2001.

[7] I. Foster, C. Kesselman, J. Nick, and S. Tuecke, "The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration," tech. rep., Open Grid Service Infrastructure WG, Global Grid Forum, 2002.

[8] I. Taylor, M. Shields, I. Wang, and O. Rana, "Triana Applications within Grid Computing and Peer to Peer Environments," *Journal of Grid Computing*, vol. 1, no. 2, pp. 199–217, 2003.

[9] "The Triana Project." See web site at: `http://www.trianacode.org`.

[10] T. Goodale, G. Allen, G. Lanfermann, J. Masso, T. Radke, E. Seidel, and J. Shalf, "The cactus framework and toolkit: Design and applications," in *Vector and Parallel Processing VECPAR 2002, 5th International Conference, Lecture Notes in Computer Science*, Springer, 2003.

[11] "The Cactus Computational Toolkit." See web site at: `http://www.cactuscode.org`.

[12] D. M. Beazley and P. S. Lomdahl, "Lightweight Computational Steering of Very Large Scale Molecular Dynamics Simulations," in *Proceedings of Super Computing '96*, 1996.

[13] Y. Huang and D. W. Walker, "JACAW - A Java-C Automatic Wrapper Tool and its Benchmark," in *Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS'02)*, 2002.

[14] T. Delaitre, A.Goyeneche, P. Kacsuk, T.Kiss, G. Terstyanszky, and S. Winter, "GEMLCA: Grid execution management for legacy code architecture design," in *Proceedings of the 30th EUROMICRO Conference, Special Session on Advances in Web Computing*, August 2004.

[15] M. Senger, P. Rice, and T. Oinn, "A resource management architecture for metacomputing systems," in *Proceedings of UK e-Science All Hands Meeting*, pp. 509–513, September 2003.

[16] R. Buyya, D. Abramson, and J. Giddy, "Nimrod/g: An architecture of a resource management and scheduling system in a global computational grid," in *HPC Asia 2000*, pp. 283–289, May 2000.

[17] "Nimrod: Tools for distributed parametric modelling." See web site at: `http://www.csse.monash.edu.au/ ?davida/nimrod/`.

[18] A. Harrison and I. Taylor, "WSPeer - An Interface to Web Service Hosting and Invocation," in *HIPS Joint Workshop on High-Performance Grid Computing and High-Level Parallel Programming Models*, 2005.

[19] "The Cactus Project." See web site at: `http://www.cactuscode.org`.

[20] T. Goodale, I. Taylor, and I. Wang, "Integrating Cactus Simulations within Triana Workflows," in *Proceedings of 13th Annual Mardi Gras Conference - Frontiers of Grid Applications and Technologies*, pp. 47–53, Louisiana State University, February 2005.