Extending GTLAB Tag Libraries for Grid Workflows

Mehmet A. Nacar[1, 2], Marlon E. Pierce[1], Geoffrey C. Fox[1, 2]

[1]*Community Grids Lab, Indiana University*
501 N. Morton St. Suite 224 Bloomington, IN 47404 USA

[2]*The Department of Computer Science, Indiana University*
Lindley Hall Bloomington, IN 47404 USA

{mnacar, marpierc, gcf}@indiana.edu

***Abstract*— Portlet-based Grid portals have become a crucial part of the cyberinfrastructure by providing component-based problem solving environments for scientists. Although portals aim to provide user-friendly environments with easy-to-use interfaces, the development of portals and their portlet components are time consuming. We aim to provide reusable components for rapid portlet development. Our approach, Grid Tag Libraries and Beans (GTLAB), encapsulates common Grid operations with reusable XML tags. GTLAB also provides a way for creating composite tasks that models the requirements of computational science portals. In previous work, we have introduced Grid tags libraries for the Globus toolkit. In this study, we extend GTLAB to support widely used Condor DAGMan and Taverna workflows for the Grid community. These extended tags demonstrate that large workflows can be integrated within Grid portlets without burdening of developers.**

I. INTRODUCTION

Science gateways have gained importance by providing scientific communities with web-based access to computing and data intensive applications. Examples of large virtual organizations provide data storage, computing power, legacy applications and Grid services include TeraGrid [1] and Open Science Grid (OSG) [2]. There are a variety of application portals available to solve numerous problems ranging from atmospheric discoveries in LEAD [3] to virtual observatories described in NVO [4]. Our Virtual Laboratory for Material Sciences (VLab) project [5] is an example of a science gateway. VLab has provided the specific motivating cases for our work.

The VLab portal [6] is based on Java-centric web technologies. The VLab portal facilitates data transfers, simulation processing, and scientific visualization. VLab portal paper describes our work to develop portlet tag libraries that encapsulate common workflows we have encountered in portlet development. Our work is intended to extend the Open Grid Computing Environments (OGCE) Grid portlets and to simplify the process of scientific application portlet development for Grids.

We will first summarize our work and motivate our research in the perspective of VLab portal [6] and Big Red portal [7] experiences.

A. Lessons Learned from Initial VLab Portal

The VLab science gateway is based around the JSR 168 [8] portlet model, and the initial set of VLab portlets are described in detail in [5]. We began by developing Grid portlets using the OGCE [9] software. In this model, each portlet application was responsible for an individual task. For example, one portlet is getting Grid credentials from MyProxy [10] repository, another one is for GridFTP file operation. A third portlet is used to execute Quantum Espresso [11] package, which is major high performance computing application for VLab material science research. This approach is useful for general user portals but needs to be modified for application-specific portals like VLab. That is, we need to collect multiple capabilities within science application-specific portlets and to handle complicated Quantum Espresso job executions and file transfers in a sequence; that is, we must define dependencies between atomic job tasks. Consequently, we have determined that we can represent job dependencies using Directed Acyclic Graphs (DAG) [12].

In order to implement these graphs, we chose the Java CoG abstraction [13, 14] interfaces for DAG executions in Grid. These provide a convenient programming interface that can be easily integrated into portlets. However, we identified the need to provide a higher-level development environment that encapsulates common tasks needed to assemble a DAG in a portlet. We have described our tag libraries so called Grid Tags Libraries and Beans (GTLAB) in [15]. Our approach is to design XML-based tag libraries for expressing DAGs and to embed them in the web pages. For that reason, we have found the Java Server Faces (JSF) [16] application framework to be appropriate. JSF is a component-based web framework that can be extended to add new components, such as our Grid tags. As described in this paper, we extend our earlier approach to provide Grid tags for Condor DAGMan [17]. We also integrate Taverna [18] workflow execution into GTLAB framework.

Using the VLab portal as a case study, we have derived requirements for tag libraries that support more comprehensive workflows. In order to support loops, parallel

```
<o:submit id="DAG" action="submit" />
<o:multitask id="multi" taskname="myDAG" persistent="true" >
   <o:myproxy id="proxy" hostname="gf1.ucs.indiana.edu" port="7512"
        lifetime="2" username="anonym" password="#{resource.password}" />

   <o:filetransfer id="jobA" from="gridftp://gf1.ucs.indiana.edu/home/anonym/input_file"
        to="gridftp://cobalt.ncsa.teragrid.org/tmp/input" />

      <o:jobsubmit id="jobB" hostname="cobalt.ncsa.teragrid.org"
        provider="GT4" executable="/tmp/run"
        stdin="input" stdout="result" stderr="error" />
    <o:dependency id="depend" task="jobB" dependsOn="jobA" />
</o:multitask>
</o:submit>
```

Figure 1 GTLAB example for creating Grid portlets. These are embedded in JSF pages that generate HTML for portlets. The example shows how a portlet developer can create a simple DAG that fetches a user proxy, transfers an input file, and submits a job. Inter-tag dependencies can be expressed. User supplied parameters are managed by a Resource Bean ("resource" in the listing).

processing of the jobs and conditional branches, GTLAB needs to integrate sophisticated workflow engines.

In our work, we extensively use and extend the JSF tag library approach. Summarizing JSF tag development is out of scope for this paper, but we summarize the primary concepts later.

The rest of the paper is organized as follows. In the next section we present GTLAB framework. Section 3 motivates and describes DAG extensions followed by Section 4 describing workflow extensions. Next section discusses Taverna use case. Section 6 summarizes related work followed by conclusion.

II. GTLAB

Grid Tag Libraries and Beans (GTLAB) provide a set of JSF tag libraries for Grid portal development. This library encapsulates atomic Grid operations as well as multi-staged operations. We explain GTLAB component model and its job management capabilities in detail as follows.

GTLAB provides several important features for application developers. First, it provides modular components (tags and beans) to construct science gateways in turn portlet pages. Second, it represents Grid service clients in an abstract fashion which is tags within XML. Therefore, portal developers do not need to understand underlying details of Grid services. Finally, it provides a component model for developing Grid portlets out of reusable parts. This model suggests the ways to use tags and beans.

Grid users typically must submit jobs to batch queues where the jobs are waited for days or longer before running, and even interactive jobs possibly take a several minutes to finish. Thus we must provide a call-back system that let jobs run while allowing the portal to return control to the user. Thus the GTLAB tags need to track the jobs' lifecycle and monitor their status, displaying this information back to the user.

GTLAB creates a handler for every submitted job by the users and displays status information using JSF data tables. These data tables are fed by job handlers that are saved in hash tables within the user session. The visual design of the job monitoring pages is left to application developers so that the developers are able to modify tables and to filter the table values.

The users can manage, stop, or cancel running jobs, after they submit them. The job archiving is also tied to job handlers. For example, users can keep good samples, remove old jobs or failed jobs, and otherwise organize their repository. The job's metadata features (submit time, status, finish time, output location and input parameters) are stored and can also be listed.

Application users compose their DAG scenario by simply using Grid tags and beans within GTLAB framework. An example DAG is illustrated in Figure 1. In this case, DAG attributes are filled by the developers. Some of the attribute values are application dependant and so they are static such as Globus Toolkit provider (version) can be set for entire portal. On the other hand, some parameters are set by the end users submitting web forms. These user-supplied parameters are managed by backing JavaBean class, ResourceBean, which we provide. Grid beans are essential to fire off the actions of Grid operations. Grid beans collect property values of operations as binding to Grid tag attributes.

III. DAG SUPPORT IN GTLAB

GTLAB is designed to utilize several DAG frameworks in Grid computing including Globus [19] toolkit (by using Java CoG interface "taskgraph") and Condor DAGMan (by using Birdbath [20] Web services interface). DAGs are built by application programmers and are embedded into JSF portal pages. Grid tags help to compose DAGs with dynamic parameters entered by end users within portlet pages. Grid tags are also responsible for executing workflow by initiating 'submit' tags. In Figure 1 the first job moves the input file from a remote host to the execution host. The second job runs a script on the execution host depending on completion of the first job. In other words, the script cannot run unless input file is ready on the execution host. Finally, Grid tags allow users to keep track of the execution of the DAG by facilitating handler tags. The listing in Figure 1 is part of a larger, JSF-based portlet that would also include input forms for collecting information from the users.

CoG Taskgraph: Java CoG encapsulates clients to Globus service separate Grid services clients in abstract interfaces. CoG interfaces also introduce an additional DAG capability which is called *taskgraph*: pipelining Globus services in a context of DAG. A CoG taskgraph is a DAG interface that is built using the Java CoG API. The CoG DAG builds workflows on top of Globus toolkit services such as GRAM and GridFTP. All tasks should be defined as abstract *Task* object and their relation to other objects are defined as dependency.

GTLAB implements a layer on CoG API that is encapsulated by XML tags. For instance, the *taskgraph* interface is used by 'o:multitask' tag. These XML Grid tags are supported by Grid beans. Grid tags are injectors for Grid beans (Inversion of Control [21]). They initialize beans and manage their lifecycles. In Figure 2, we show the XML schema that summarizes GTLAB libraries. As illustrated in the figure, *multitask* can define attributes for taskgraph including *id, taskname, handler and persistent*. *multitask* also can contain dependent task objects are represented as sub tags including o:myproxy, o:fileoperation, o:jobsubmit, o:filetransfer and o:dependency.

Application users compose their DAG scenario by using Grid tags and beans together within GTLAB framework. In this case, DAG attributes are filled by the developers. Some of the attribute values are application dependent and so they are static. For example, the Globus toolkit provider attribute can be set as *GT4* for entire portal. On the other hand, some parameters are set by the end user. These attributes must bind HTML input text by using expression language semantics within JSF.

Condor DAGMan: Condor [22] is an environment for scheduling and executing applications on distributed networks of computers. DAGMan is a tool for describing complex application workflows to be executed on Condor in terms of directed acyclic graphs. In this case, GTLAB allows the user to prepare or transfer descriptions of Condor jobs or workflow scripts described with the DAGMan. End users can return later and monitor the progress of the jobs.

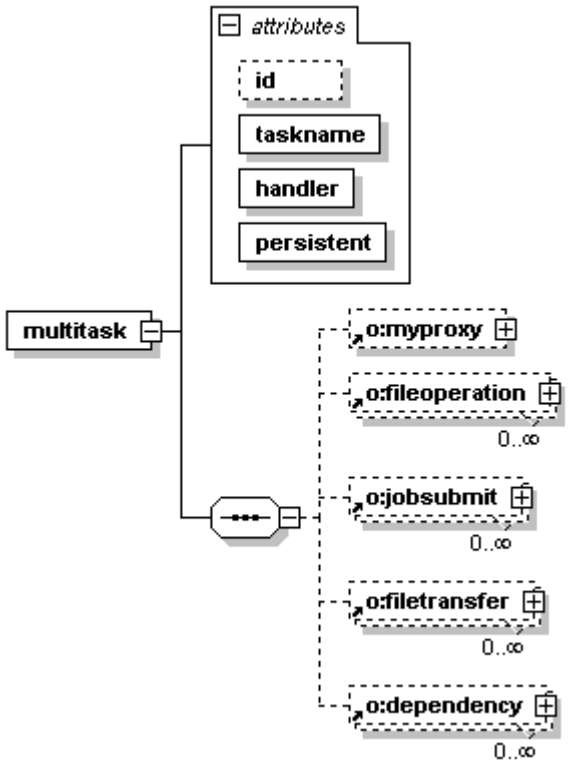

Figure 1 XML schema of multitask represents a DAG. It shows the relationship of Grid tags by defining dependency tag in GTLAB.

Condor manages job submissions to Globus-based Grids through Condor-G [23]. The GTLAB client libraries for Condor-G jobs are similar to our earlier work with CoG task graphs. GTLAB provides a portlet client environment for Condor DAGMan by introducing two additional JSF Grid tags: <o:condorDagman/> and <o:condorSubmit/>, which we describe below.

<o:condorSubmit/> is for single job submission to Condor-G resources. <o:condorDagman/> is used to describe composite DAGMan jobs and their dependencies along with a scripting file. Similar to our standard *multitask* tag, these tags provide access to Condor services in terms of using Condor beans. Our Condor beans have capabilities to prepare Condor jobs, submit jobs to Condor resources and manage the lifecycle of submitted jobs.

Condor has no equivalent Java client libraries that correspond to the Java CoG for the Globus toolkit. However, Condor provides Web services interface called Birdbath [20]. This provides an XML abstraction of the programming interfaces that can be bound to different languages such as (in our case) Java. Our Condor beans are built on top of Birdbath Web services clients. The Birdbath layer allows us to program Condor capabilities within Java Beans and associated Grid tags allow us to describe job parameters by using dynamic web interfaces. In this case, we use Web services clients to program Condor, instead of using the more well-known

command-line interface. We provide an overview of Birdbath in the following section.

Birdbath: Birdbath is Condor's Web services interface to that enables Condor to be accessible programmatically and remotely rather than by command line scripts. Birdbath has two main interfaces. First, the Scheduler interface is for submitting and querying jobs. Second, the Collector interface is for retrieving information about Condor resources.

Birdbath client stubs can be developed from the BirdBath WSDL or generated using tools such as Apache Axis's WSDL2Java tool. To enable GTLAB using client API, we have packaged Java clients as jar and add to web application library.

IV. Workflow Support in GTLAB

We consider in this section strategies for supporting more complicated workflows than can be represented by DAGs. Our goal in GTLAB is not to reproduce extensive pre-existing work in this field but to instead take advantage of it.

DAGs are very useful in case of simple workflows such as submitting a few tasks in a group. We have added new features to GTLAB such as the ability to build sub-graphs to allow partially ordered tasks. Partially ordered tasks can group the sequence of the tasks based on their dependency. But in case of enhanced workflows, DAGs are not sufficient. For example, if a user needs to try and run the simulation many times with a DAG, the DAG has to maintain loops. If an application portlet needs to provide dynamic flow control based on constraints, the DAG has to support conditional branches. Those features do not exist within DAGs. Thus, a scientific community has to facilitate these capabilities; they need to use workflows that cannot be expressed as simple DAGs. Directed graphs naturally do not handle this type of data structures. Our solution for supporting these more complicated workflows is described in the next section.

Workflows are sophisticated flow control mechanisms of group of tasks. The foundations of Grid workflows are described in a special issue of *Concurrency and Computation* [24]. The tasks could be parallel, sequential, or concurrent. Workflows can handle loops, branches and conditional branches. Workflows can be divided into three main parts: 1) Composer, 2) Enactor, and 3) Monitor.

Composer: The composer is an essential part of the workflow representations. Workflows represent services as nodes and constraints as edges to the nodes. In this case, the top node is the starting point and intermediate nodes denote tasks and local filters, however, edges denote dependencies. This structure could be a graph where nodes correspond to tasks and edges corresponds to relations. Also direction of the edges can limit the flow similar to flow charts.

Enactor: An enactor is a workflow engine that process nodes in the order determined by the composed graph. End users provide values for workflow inputs. Workflow processing results in with workflow outputs. An enactor can pipe inputs to one action that is output of the previous one. An enactor also maintains constraints, branches, loops and parallelism.

Monitoring: Monitoring follows up the processing steps. It also manages lifecycle of the workflow. End users are able to interrupt the workflow to pause or cancel the execution of the workflow at each step.

Unlike DAG composition existing within GTLAB, workflow composition is out of scope of this work. In case of DAGs there is only a few tags definition exist such as *multitask, jobsubmit, myproxy, fileoperation* etc. Because GTLAB is supposed to provide one-one mapping for each entity in DAG definition. But workflows are more comprehensive than DAGs and there are many entities to define a complete workflow. Workflow policies are described by their own composition language. We either need to provide one by one mapping of each entity exist in the workflow language or Grid tags could import workflow policy as whole within GTLAB. We rather prefer the latter to embed built-in workflow files into the enactor. Enactor takes that file as an input to start workflow engine for execution.

Our strategy for supporting workflows is as follows: GTLAB framework binds enactor engine to a 'submit' button within a web form on the portal page. Once the button is clicked by an end user, the enactor engine takes control of workflow along with the composition document. These workflow documents are already checked against validity. Workflow frameworks define their composition rules as explained in great detail in the next section. Finally, the engine starts running at the backend to process action steps.

GTLAB monitoring features are listed as status updating, cancelling, pausing, and resuming the jobs. GTLAB assigns unique handlers for all submitted workflows within the user session. These handlers are associated with 'handler' tags. The handler tag utilizes the capabilities of monitoring bean by using attributes and sub-tags.

V. Taverna use case

Taverna is workflow tool for composing and executing Web Services. Its main target is bioinformatics applications but it can in fact be applied to general workflow composition problems. Taverna includes a graphical user interface workbench that is used to formulate workflows. The Taverna workbench solves issues of complexity of the workflows by providing user friendly interface. The workbench facilitates diagrammatic and explorer representation of workflows. It allows users to compose their own workflows or to load previously designed workflows, such as may be obtained from a community repository with expert contributors. The workbench also lists the available resources (e.g. web services) where the workflows can run. After the resources and enactor engine types are selected by a user, he or she can start the workflow and can monitor progression. The user can interrupt the workflow for cancelling a step or stopping the workflow.

The Taverna workbench relies on XML-based *Simple conceptual unified flow language* (Scufl) [25]. Scufl consists of a network of processors and links. In addition to basic entities, Scufl also can have input and output nodes and constraints for processors. The Scufl language primarily is designed for users who are familiar with web forms and scripting languages to use web resources. Scufl is practical and is designed with extensibility features.

Workflow portlet: a workflow portlet should support workflow features. Generally a workflow portlet should contain these three major parts: 1) Defining workflow components and their relationships. 2) Executing the workflow: in case of Scufl they use Freefluo [26] enactor engine. 3) Monitoring execution flow and apply capabilities like resume, checkpoint, cancel, remove, etc. Typically the first and third steps supposed to be tied with strong graphical user interface such as the Taverna workbench.

Building a workflow composition environment with the graphical user interface features require many visual designs to accomplish with a success. The Taverna workbench is already available for composing workflows. Building a workflow composer out of Taverna is out of GTLAB's scope. But we can alternatively provide a text field to compose workflows in XML (e.g., Scufl) on the portlet page. However there are two drawbacks of this approach: 1) it is hard to catch syntax errors when composing a workflow, and 2) the Scufl document should be validated against Scufl schema. This process is offline and requires additional efforts.

The practices of Taverna demonstrate that the bioinformatics community is relying on pre-built workflows. Their major concern is not how to build their own workflow each time; rather they want to use well-prepared and proven workflows for their own applications. Consequently, GTLAB can provide significant support for Taverna by supporting its execution engine. We will assume that workflow composition is done by another tool (that is, the Taverna workbench user interface).

Workflow portlet application that utilizes extended GTLAB features to submit Scufl workflows. This portlet loads a Scufl workflow file, collects input values from end users, submits the workflow on Taverna, and monitors the results inside the GTLAB session framework.

Figure 3 illustrates the handling of Taverna tags within GTLAB. In this case, Taverna tags are embedded into JSF portlet page integrated with a Web form. End users only see the Web form with a few text fields and submit button. They never see the Grid tags and JSF tags that build the portlet page. This is common for all web applications. When the end user submits a web form through the portlet page, JSF intercepts this request and calls the associated action methods of Grid beans. Next, Grid beans load the appropriate Scufl document and input parameters to Taverna bean. Finally, the bean method starts execution of the workflow on Taverna enactor.

GTLAB assigns job handlers to each submitted workflow within the user session so that keeping track of the progression. In case of Taverna, the handlers synchronize with Taverna monitoring services to follow the workflow states.

Taverna Security: Taverna generally works in non-secure environments with Web services that can be used anonymously. The Taverna workbench uses local filters and scripts. The main concern of bioinformatics community is to process massive data by using complicated workflows. However, security is a critical issue in Grid services that rely on secure connections. Traditional Grid services apply GSI security [27] by using X.509 certificates. Since Taverna can facilitate emerging Web services technologies, we need to employ WSRF [28] and GT4 services within Taverna. Therefore, it is essential to address whether WS-Security [29] is applicable to Taverna processors. Similar problem has been addressed by [30] extending Taverna workbench adding new processors that support WS-Security.

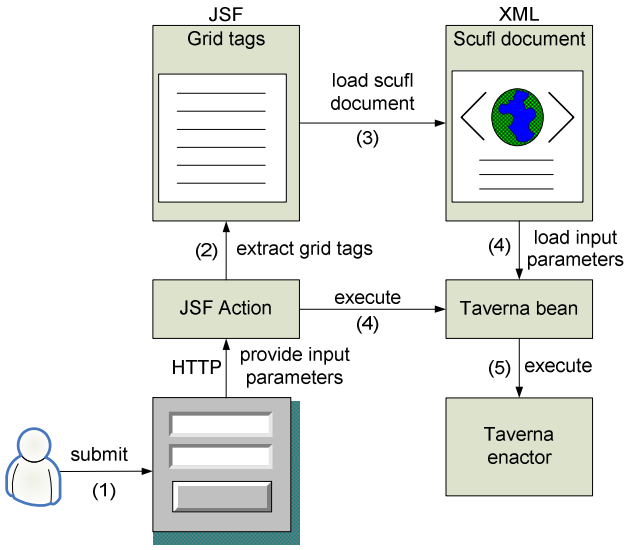

Figure 3 A user interacts with a workflow portlet to utilize Taverna enactor. User provides parameters by submitting a web form that start the chain of events in order.

Supporting Grid services within Taverna is an interesting approach. Taverna can utilize some services with local clients such as MyProxy. Although other Grid services already have Web services interfaces in GT4. These services can be scavenged to Taverna. Therefore, we are able to manage Grid services workflows through Taverna.

VI. Related Work

In this section we overview three client applications for workflow frameworks. These are Karajan [31], My Grid Portal Interface (MPI) portlets [32], and Condor portlets [33]. Each of these client packages can support workflow mechanisms through either stand-alone applications or web applications. These clients are related to our work because they are supporting similar workflow systems that we support.

In other words, GTLAB also supports the same workflow systems as client application in tag library level. GTLAB provides tags and bean modules to workflow services. Thus, application developers can program these workflows in an abstract fashion.

Karajan workflow framework provides access to Grid services by using XML-based definition language. Karajan can be utilized in various platforms. Karajan has its own parallel and structural language that is adopted for Grid services needs. Users can define jobs and their lifecycle management using the Karajan language. Karajan scripts runs on Karajan engine so called Karajan service.

Karajan service is a workflow engine that can be accessible by several ways such as polling, call-backs, and persistent data retrieving. When users submit their workflows, Karajan service interprets these inputs through Karajan libraries. Then the engine creates client stubs for the tasks defined in Karajan script. These tasks are submitted to the Grid services using Java CoG abstractions.

Taverna is a workflow tool for composition and execution of Web Services. Its main target is bioinformatics applications, but it can in fact be applied to general workflow composition problems. Taverna typically runs within Taverna workbench that is a desktop application for application scientists. It has many graphical interfaces to compose, enact and monitor workflows. At the same time it is not portable as web application.

One attempt to create web application with Taverna is to create MPI portlets. In this case, the major concern of scientists is not how to compose a workflow each time; rather they want to use well-prepared community-supplied workflows for their own applications. To this end, MPI is only responsible for executing (rather than composing) workflows. MPI has a portlet web application to execute and enact Scufl workflows. MPI loads selected workflow and dynamic input parameters from the user and execute it. MPI registers input parameters and results through My Grid Information Repository (MIR). It also allows the users to navigate on the execution flow and display results in different formats like image, xml or text.

Condor DAGMan is a DAG supported workflow mechanism. OGCE Condor Job Submission Portlet enables users to submit batch jobs to remote resources via Condor using the BirdBath project's SOAP and WSDL enabled Collector and Scheduler daemons. Condor portlets allow a user to specify job parameters, submit the job, view job status information, download output files, and delete old jobs. These portlets allow end users to connect to any Birdbath-enabled Condor flock.

VII. CONCLUSIONS

In this paper, we have evaluated our initial VLab portal development work, which constructed workflows for Material Sciences that are based on DAGs. We provided support Globus toolkit by using Java CoG. Since this initial work, we have added support for Condor DAGMan by using Birdbath services, as described in this paper. We have extended our architecture based on emerging Web Services technologies to construct a Web Services workflow. Taverna workflow allows us to deploy and manage more comprehensive workflows using Web services. We have designed additional Grid tags for Condor DAGMan and Taverna workflow. In conclusion, we showed that our GTLAB framework is extensible and applicable to different types of workflow frameworks.

In future work, we will adapt GTLAB framework to work with BPEL workflows. To this end, we will investigate how to extend GTLAB with well known workflow frameworks in the Grid community.

ACKNOWLEDGMENT

This work is supported by the National Science Foundation's Information Technology Research (NSF grant ITR-0428774, 0427264, 0426867 VLab) and Middleware Initiative (NSF Grant 0330613) programs.

REFERENCES

[1] (2007) TeraGrid website. [Online]. Available: http://www.teragrid.org/

[2] (2007) Open Science Grid. [Online]. Available: http://www.opensciencegrid.org/

[3] K. K. Droegemeier, V. Chandrasekar, R. Clark, D. Gannon, S. Graves, E. Joseph, M. Ramamurthy, R. Wilhelmson, K. Brewster, B. Domenico, "Linked environments for atmospheric discovery (LEAD): A cyberinfrastructure for mesoscale meteorology research and education", 20th Conf. on Interactive Information Processing Systems for Meteorology, Oceanography, and Hydrology, 2004.

[4] D. Ewa, et al., "Grid-Based Galaxy Morphology Analysis for the National Virtual Observatory", in Proceedings of the 2003 ACM/IEEE conference on Supercomputing. 2003, IEEE Computer Society.

[5] M. A. Nacar, M.S. Aktas, M. Pierce, Z. Lu and G. Erlebacher, D. Kigelman, E. F. Bollig, C. De Silva, B. Sowell, and D. A. Yuen, "VLab: Collaborative Grid Services and Portals to Support Computational Material Science", Concurrency and Computation: Practice and Experience, 2007(Special issue on Grid portals).

[6] E.F. Bollig, P. A. Jensen, M. D. Lyness, M. A. Nacar, P. R. da Silveira, G. Erlebacher, M. Pierce, D. A. Yuen, "VLAB: Web Services, Portlets, and Workflows for Enabling", 2007, submitted to the Journal of Physics of the Earth and Planetary Interiors (PEPI).

[7] M. A. Nacar, J. Y. Choi, M. E. Pierce, and G. C. Fox. "Building a Grid Portal for Teragrid's Big Red", in Proceedings of TeraGrid 2007, 2007. Madison, WI.

[8] A. Abdelnur, E. Chien, and S. Hepper, (eds.), Portlet Specification 1.0., 2003, Available: http://www.jcp.org/en/jsr/detail?id=168.

[9] J. Alameda, M. Christie, G. Fox, J. Futrelle, G. Gannon, M. Hategan, G. von Laszewski, M. A. Nacar, M. Pierce, E. Roberts, C. Severance, M. Thomas, G. Kandaswamy, "Open Grid Computing Environments collaboration: portlets and services for science gateways", Concurrency and Computation: Practice and Experience, 2007. 19(6):921-942.

[10] J. Novotny, T. Steven, and V. Welch, "An Online Credential Repository for the Grid: MyProxy", in Proceedings of the Tenth International Symposium on High Performance Distributed Computing *(HPDC-10)*, 2001.

[11] S. Scandolo, P. Giannozzi, C. Cavazzoni, S. de Gironcoli, A. Pasquarello, S. Baroni, "First-principles codes for computational

crystallography in the Quantum-ESPRESSO package", Zeitschrift für Kristallographie, 2005. vol:220: p. 574-579.

[12] F. Harary, *Graph Theory*. 1994, Reading, MA: Addison-Wesley.

[13] K. Amin, M Hategan, G. von Laszewski, N. J. Zaluzec, "Abstracting the Grid", Proceedings of the 12th Euromicro Conference on Parallel, Distributed and Network-Based Processing, 2004.

[14] G. von Laszewski, I. Foster, J. Gawor, P. Lane, "A Java commodity grid kit",. Concurrency and Computation: Practice and Experience, 2001. 13(8-9): p. 645-662.

[15] M. A. Nacar, M. Pierce, G. Erlebacher, G. Fox, "Designing Grid Tag Libraries and Grid Beans", in Second International Workshop on Grid Computing Environments GCE06 at SC06, 2006, Tampa, FL.

[16] C. McClanahan, E. Burns, and R. Katain (eds), "Java Server Faces Specification 1.1", Sun Microsystems, Inc., 2004, Santa Clara, USA. Available: http://java.sun.com/javaee/javaserverfaces/download.html

[17] D. Thain, T. Tannenbaum, and M. Livny, *Condor and the Grid*, Grid Computing: Making The Global Infrastructure a Reality, A. Hey. F. Berman, G. Fox, (eds), 2003, John Wiley.

[18] T. Oinn, et al., "Taverna: a tool for the composition and enactment of bioinformatics workflows", Bioinformatics, 2004. 20(17): p. 3045-3054.

[19] I. Foster, C. Kesselman, and S. Tuecke, "The Anatomy of the Grid: Enabling Scalable Virtual Organizations", International Journal of High Performance Computing Applications, 2001. 15(3): p. 200-222.

[20] C. Chapman, C. Goonatilake, W. Emmerich, M. Farrellee, T. Tannenbaum, M. Livny, M.Calleja, M. Dove, "Condor Birdbath: Web Service interfaces to condor", in Proc. UK e-Science All Hands Meeting, 2005, Nottingham UK.

[21] R. Johnson, *Expert One-On-One J2EE Design and Development*. Wrox. 2003, Indianapolis, IN, USA: Wiley Publishing, Inc.

[22] (2007) Condor DAGMan, Available: http://www.cs.wisc.edu/condor/dagman/

[23] J. Frey, T. Tannenbaum, M. Livny, I. Foster, S. Tuecke, "Condor-G: A Computation Management Agent for Multi-Institutional Grids", Cluster Computing, 2002. 5(3): p. 237-246.

[24] G. C. Fox, D. Gannon, "Special Issue: Workflow in Grid Systems", Concurrency and Computation: Practice and Experience, 2006. 18(10): p. 1009-1019.

[25] T. Oinn, et al., "Delivering web service coordination capability to users", in Proceedings of the 13th international World Wide Web conference on Alternate track papers and posters, 2004, ACM Press: New York, NY, USA.

[26] (2007) Freefluo, Available: http://freefluo.sourceforge.net

[27] I. Foster, C. Kesselman, G. Tsudik, S. Tuecke, "A security architecture for computational grids", in Proceedings of the 5th ACM conference on Computer and communications security, 1998, ACM Press: San Francisco, CA.

[28] K. Czajkowski, D. Ferguson, I. Foster, J. Frey, S. Graham, I. Sedukhin, D. Snelling, S. Tuecke, W. Vambenepe, "The WS-Resource Framework", 2004, Available: http://www.globus.org/wsrf/specs/ws-wsrf.pdf

[29] B. Atkinson, G. Della-Libera, S. Hada, M. Hondo, P. Hallam-Baker, C. Kaler, J. Klein, B. LaMacchia, P. Leach, J. Manferdelli, H. Maruyama, A. Nadalin, N. Nagaratnam, H. Prafullchandra, J. Shewchuck, D. Simon, (eds.), "Web services security (ws-security)", 2004, Available: http://www-106.ibm.com/developerworks/webservices/library/ws-secure

[30] S. Perera, D. Gannon, "Enabling Web Service Extensions for Scientific Workflows", in HPDC2006 Workshop on Workflows in Support of Large-Scale Science (WORKS06), 2006, Paris, France.

[31] G. von Laszewski, M.Hategan, D. Kodeboyina, "Work Coordination for Grid Computing", 2007, Available: http://www.mcs.anl.gov/~gregor/papers/vonLaszewski-work-coordination.pdf

[32] S. R. Egglestone, M.N. Alpdemir, C. Greenhalgh , A. Mukherjee, I. Roberts, "A portal interface to myGrid workflow technology", 2005, UK e-Science All Hands Meeting.

[33] D. Gannon, L. Fang, G. Kandaswamy, D. Kodeboyina, S. Krishnan, B. Plale, A. Slominski, "Building grid applications and portals: an approach based on components, Web services and workflow tools", 2004, in Proceedings of 10th International Euro-Par Conference (Lecture Notes in Comput. Sci. Vol.3149).