VLab: Collaborative Grid Services and Portals to Support Computational Material Science

Mehmet Nacar, Mehmet Aktas, and Marlon Pierce
Community Grids Lab
Indiana University
Corresponding Author: mnacar@cs.indiana.edu

Zhenyu Lu and Gordon Erlebacher
School of Computational Science and Information Technology
Florida State University
Corresponding Authors: zhenyulu@cs.fsu.edu, erlebach@csit.fsu.edu

Dan Kigelman, Evan F. Bollig, Cesar De Silva, Benny Sowell, and David A. Yuen
Minnesota Supercomputer Institute
University of Minnesota
Corresponding Author: rao@msi.umn.edu

Abstract: We present the initial architecture and implementation of VLab, a Grid and Web Service-based system for enabling distributed and collaborative computational chemistry and material science applications for the study of planetary materials. The requirements of VLab include job preparation and submission, job monitoring, data storage and analysis, and distributed collaboration. These components are divided into client entry (input file creation, visualization of data, task requests) and backend services (storage, analysis, computation). Clients and services communicate through NaradaBrokering, a publish/subscribe Grid middleware system that abstracts specific hardware information through the use of topics. We describe two aspects of VLab in this paper: 1) data entry and submission, and 2) a visualization web client/service. Grid Web Portals, Java Server Faces (JSF) and JSF Grid Beans are used to build an interface that permits input file specification, multiple code submissions, and multiple job submissions (of a given code) with backend data persistence. In addition, to investigate our collaboration and visualization infrastructure, we have developed a service that transforms a scalar data set into its wavelet representation. A client (java applet) can retrieve the coordinates of the centers of the dominant fraction of the wavelets and display the results as a collection of spheres. General adaptors are placed between the endpoints and NaradaBrokering, which serve to isolate the clients/services from the middleware. This permits client and service development independently of potential changes to the middleware.

1. Introduction

The Virtual Laboratory for Earth and Planetary Materials (VLab) is a National Science Foundation-funded interdisciplinary research project that uses computational techniques to investigate planetary materials at extreme conditions, enabling a fuller understanding of the processes that create earth-like and other planetary objects. To address challenges in collaborative and distributed computing, VLab brings together a team that includes researchers in computational material science, geophysics, scientific visualization, Grid computing, and information technology. More information on VLab is available from [1].

From the Grid computing point of view, VLab presents several interesting problems that are the subject of this paper. Driving Grid research issues include the following:

- Persistently managing user inputs as archived, hierarchical project metadata. This is described in Section 2.
- Simplifying complicated, multi-staged job submissions and monitoring using Grid portal technology. Section 3 describes this work.
- Integrating VLab applications with Grid messaging infrastructure [2] to virtualize resource usage, provide fault tolerance, and enable collaboration. This work is summarized in Section 4.

Bullets 1 and 2 are essentially client-side operations, while bullet 3 is primarily concerned with abstracting server-side capabilities. In this paper, we report our initial approaches to solving these problems.

2. A Sample Problem: Managing Plane Wave Self Consistent Field (PWSCF) Calculations

Many of VLab's workhorse simulation codes are part of the "Quantum Espresso" package developed by the Democritos group [3]. We have taken the Plane Wave Self Consistent Field [4] code in the Espresso suite as our starting point. The PWSCF code is a parallelized application that is typically submitted to supercomputing resources using a batch queuing system. Thus, we will need to use standard Grid technologies such as the WS-GRAM, Reliable File Transfer, GridFTP, and Grid security, such as provided by the Globus Toolkit [5]. The typical problem is to gather user information necessary to create a PWSCF input file, move the file to an appropriate backend resource, and run the code. Grid Web portals are used to manage this submission. This is a classic Grid portal problem [6]. As detailed in a follow-up issue to [6] (currently in preparation), most Java-based portals are developed around the "portlet" approach [7]. We follow this approach and have adopted the GridSphere [8] portal container for VLab deployment.

As a simple submission application, PWSCF does not place any unusual demands on standard Grid and portal technologies. However, as described in the introduction, we need to explore problems in user session archive management, since VLab submissions may involve dozens or hundreds of simulation runs per user. We also anticipate the addition of more codes from the Quantum Espresso suite and the need to couple these into workflows.

Java Server Faces (JSF) provides an extremely useful framework for building science portal applications. We describe JSF and Grid client integration in more detail in the next section. However, JSF alone provides an important advantage for user input form creation: all forms and actions are backed by corresponding JavaBeans. JSF eliminates the need to explicitly manage HTTP request parameter names and session variables. We thus need only follow simple JavaBean get/set methods and property name conventions and may develop our input page backing code independently of the Web server.

Using backing beans to store user input data provides another important advantage: the project input forms may be serialized as XML using Castor or XML Beans and stored using various archival technologies. For VLab, we are evaluating the use of WS-Context [9] for this purpose, since it supports the linking of lightweight XML information nuggets using simple parent-child relationships [10]. Since we are not starting from existing XML schemas, we have chosen Castor for marshalling and unmarshalling backing beans. We then store in persistent context elements organized hierarchically by user identity (parent) and user project (child). User sessions are stored as children of specific projects. More detailed information on our WS-Context implementation is available from [11]. More complicated permission systems for sharing data will be investigated in the future, but these issues are orthogonal to the storage service.

3. Reusable Portal Components: JSF Grid Beans

In the previous section, we discussed the value of using JSF "as is" to develop user input form portlets. After generating the PWSCF input files from the input forms, we must then upload this file and use it to submit the remote application. This can be done with standard Grid technologies. However, we must do this for several additional codes and will need to provide a way to simplify simple code couplings for multiply staged jobs. This corresponds to building several additional portlets. We would also like to avoid writing specialized backing code that is too closely tied to these specific portlets. Instead, we want to build each portlet interface out of as many reusable parts as possible. To accomplish this, we have developed JSF bean wrappers to general COG Kit abstractions [12], [13] to the Globus toolkit. These COG abstractions hide differences between (for example) pre-Web Service and Web Service versions of the Globus Toolkit. By wrapping COG ExecutableTask abstractions as Java Beans with appropriate get/set methods, we may couple them to JSF web applications.

JSF presents us with a small problem, however. It only manages individual session beans, but a user may need to submit many independent jobs within a single session, each needing its own bean. Also, we must link several beans into compositions of multiple grid tasks—even the simple PWSCF submission couples file transfer and job submission tasks in a single button click. The Task Manager and Task Graph Manager described in this section represent our current solution to these problems.

The *Task Manager* handles independent user requests, or tasks, from the portlet client in Grid services. The user request-generating objects are simply Java Bean class instances that wrap common Grid actions (launching remote commands, transferring data, performing remote file operations) using Java COG classes. We define a general purpose interface called GenericGridBean, which specifies the required get/set methods of the implementing class.

These are data fields such as “host name”, “toolkit provider”, and so forth. GenericGridBean implementations include JobSubmitBean, FileTransferBean and FileOperationBean.

When a client invokes a particular type of action, it does so indirectly through the Task Manager Bean. The TaskManager is responsible for passing property values and calling action methods of task beans. Once a user request is caught, the task manager creates a task bean object and its event listener. It then persistently stores them to a storage service (a HashMap in our current implementation). The storage service has methods that can access this bean instances with “taskname” unique key. A JSF validator guarantees that each “taskname” parameter is unique in the session scope.

The Task Manager is also responsible for monitoring task beans and managing their lifecycles. When a task is initially submitted, we store it as a “live” object in the Storage Service (Fig. 1). Live objects correspond to COG states such as “unsubmitted”, “submitted”, “active”, etc. When jobs enter “completed” or related states (“failed”, “canceled”), they are removed from the live storage and serialized as XML objects for persistent storage. This allows us to recover the submitted job’s properties (such as input file used and execution host) for later editing and resubmission. We currently store the serialized XML in local files for testing but will evaluate the use of WS-Context for persistent storage. One current drawback to this initial scheme is that live objects will be lost when the session expires. Globus Toolkit services provide persistence with clients through callback ID handles, but we must add this capability to the Java COG.

We are also investigating another solution to the persistence problem. One of the side effects of the JSF approach is that the backing Java Bean classes may be developed independently of the JSF framework. This will potentially allow us to extract our Task Beans from the JSF container and run them as standalone Web Services. This approach will simplify the management of user objects that are not directly tied to Tomcat session objects. To implement this, we are investigating the use of Inversion of Control pattern implementations like Spring [Johnson2003]. Thus, Spring framework will take care of lifecycles of beans and task properties will be injected to Spring with web services.

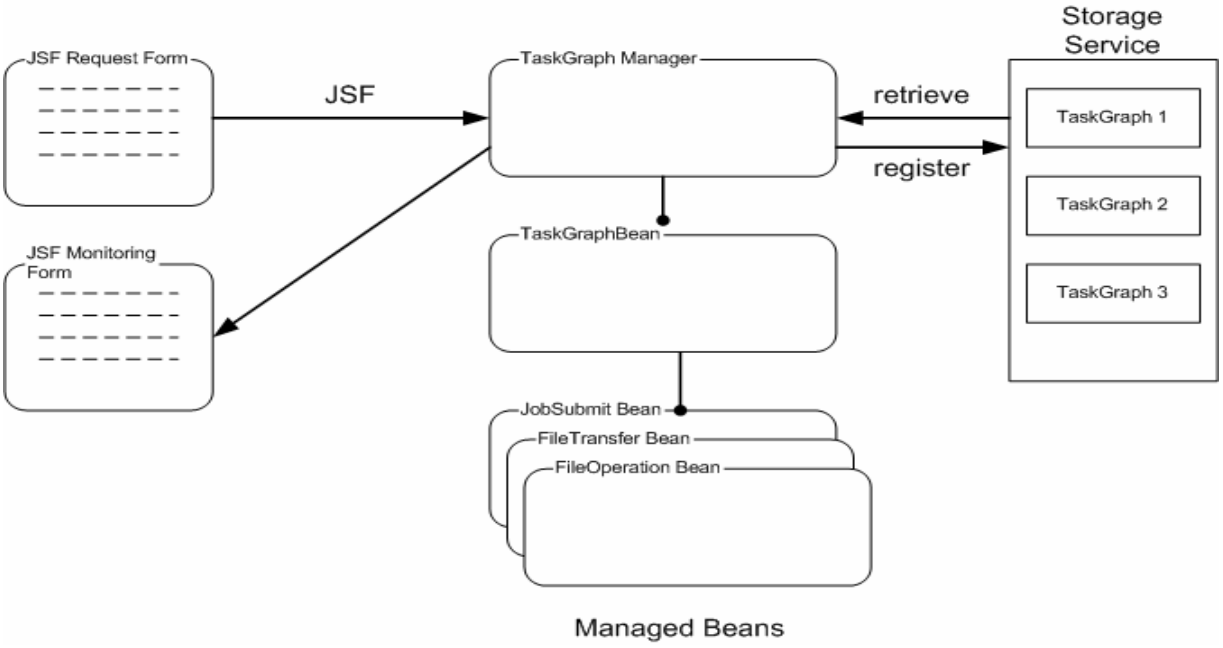

Figure 1 The Task Graph Manager can be used to manage Grid submissions with multiple steps.

In addition to managing multiple independent tasks, we must also often manage coupled tasks: a single button click may require two or more dependent actions. The Java COG Kit provides the TaskGraph class to handle directed acyclic graph-like workflows composed of atomic tasks. As with the Task Manager previously, we have defined a *Task Graph Manager* class, which is a Java Bean wrapper around the COG TaskGraph class.

The TaskGraph Manager handles multiple-step task submission, as depicted in Fig. 1. The TaskGraph Manager coordinates user requests with TaskGraph backing beans. Each TaskGraph bean is itself composed of GenericGridBean implementation instances (for file transfer, job submission, etc.). We express the dependencies

using JSF tag library extensions, so that the JSF application developer can encode the composite task graph workflow out of reusable tags. The TaskGraph Manager submits and monitors TaskGraphs through an action method when the user launches the job. The TaskGraph Manager has similar monitoring and event handling mechanisms as the Task Manager. The TaskGraph Manager redirects user inputs into the proper TaskGraph bean instance and manages its lifecycle. We may also use the Task Graph Manager to retrieve specific TaskGraph instances so that we may, for example, check the current status of a running job.

4. Collaborative Web Services

The PWSCF application represents a straightforward Grid application. VLab will also need to address more challenging problems: determining best available backend resources for scheduling applications and collaborative web services that allow multiple clients to interact with the same service. For example, the Task Manager and TaskGraph manager described in Section 3 are client-side code that still must specify specific host computers in the current implementation. Virtualizing the backend connection through proxy services is highly desirable for both load balancing and fault tolerance. As we describe in this section, we are using message oriented middleware (NaradaBrokering) to investigate solutions to these problems.

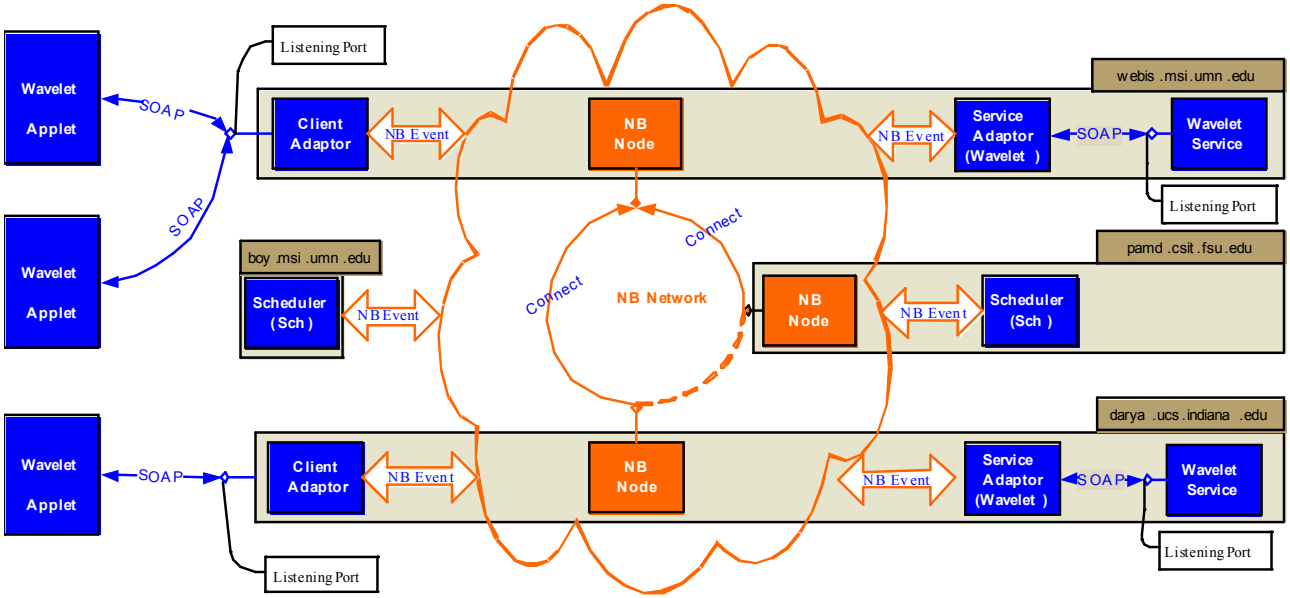

Figure 2. A network of three NaradaBrokering brokers is deployed across Indiana University, Florida State University and University of Minnesota. Attached to the network are two wavelet services, with their service adaptors, two schedulers, and several client adaptors. Clients can access the wavelet service via a wavelet applet using standard browsers.

The desired functionality ascribed to VLab demands a flexible environment that supports a variety of services related to computation, storage, visualization, database transactions, and processing. In addition, system scalability, expandability, and fault tolerance are essential components. We have chosen to use NaradaBrokering as a middleware system that already incorporates many of these elements.

NaradaBrokering [2] is a Grid and Web Service-compatible middleware system formed by a cooperating set of brokers whose role is to process messages sent to it, each with a topic tag, and route them to any subscribers to that tag. The use of a topic tag is the hallmark of publish/subscribe systems, which make it possible to make requests and receive replies without regard to the specific system processing the requests. Using NaradaBrokering, we have developed a prototypical network, geographically distributed to demonstrate the features of a future VLab system. The components of our architecture are illustrated in Fig. 2.

We have developed a wavelet transform service for demonstration purposes. Large-scale datasets are stored on two servers in Minnesota (implemented) and Indiana (not yet implemented). A client can request the wavelet transform of a specific dataset, and have a specified number of wavelet coefficients transmitted to the client. We implement the client as an applet (which can itself be embedded in a portlet) for acceptable interactivity and image rendering. However, the approach is general and can be applied also to other Web applications (i.e. JSF portlets described in

Section 3). In our wavelet application, the applet displays the coefficients as spheres centered at the location occupied by the center of the 3D wavelet. High performance graphics are obtained through the use of JOGL, an OpenGL API for Java.

Although we demonstrate our framework with a wavelet transformation service for visual impact, this service can be replaced with any other service that conforms to the WSDL standard. The system has several components that work together to isolate the user from the task of finding available services, finding the servers that support them, and distributing the workload. As shown in Fig. 2, these components are the Broker Network, the schedulers, the services (with associated adaptors), the clients (with their adaptors), and the service registries (not shown). The *Broker Network* encompasses a collection (in this case three) of NaradaBrokering nodes connected to one another. Any entity (client or service) publishing to any of these nodes will have its request propagated through the entire Broker Network towards one or several destinations. NaradaBrokering is responsible for efficient routing, guaranteed delivery, or in the event of problems, appropriate error feedback to the sender. The *Schedulers* manage the task execution process. There are several schedulers to provide fault tolerance into the system, in the even that one or more servers that host the scheduler become unavailable. Different schedulers might also handle different types of requests. When a client makes a request, a scheduler is identified to handle the request and establish a connection between service and client. The scheduler is also responsible for splitting larger tasks into sub-tasks (not implemented), finding the appropriate services and submitting these tasks with regard for task order. The *Services* are responsible for completion of requests initiated either by the clients or by the schedulers on their behalf. Data generated from these services are either returned to the clients or to the schedulers who then forward the data to the clients. *Service adaptors* are interfaces between the service (which has no knowledge of the NaradaBrokering middleware and communicates via the SOAP protocol), and NaradaBrokering, which understands messages with an associated topic. Thus, the adaptors wrap and unwrap messages received by and sent to the services to make them conform to the NaradaBrokering protocol. To this end, the adaptors simply add a publishing topic, state whether message refers to a subscription or a publishing message, and optionally adds some headers to the message, such as IP addresses. *Services* are responsible for executing the tasks generated by the clients. *Clients* may be applets, either standalone or within a portal environment, or portlets. They communicate with the middleware through the use of client adaptors, which play the same role with respect to clients as do the service adaptors with respect to services.

The key concept within our system is that each entity (any one of the system's components with the exception of the brokers), has a unique ID, and the entity subscribes to its own ID. Furthermore, each entity subscribes to a category ID related to the services it can provide. For example, a scheduler subscribes to the ID "sch", while the wavelet server might subscribe to the ID "wav". A client first sends a message with the topic "sch". Through the publish/subscribe mechanisms, all entities that subscribed to "sch" will respond, namely all the schedulers. Currently, the first scheduler to respond will be chosen. Note that if one scheduler is unavailable, the second scheduler will be chosen (both messages reach the requesting client). Along with responding message from the scheduler, its unique ID is included, so that the client may then connect directly to it. The client then sends its specific task request (in this case, the request to perform the wavelet transform on a specific file) to the scheduler. The message headers include the name of the desired service (in this case the service has topic "wav"). The scheduler sends a message out requesting a wavelet service that has subscribed to the topic "wav". Once found, the wavelet service returns a message with its own unique ID. The scheduler can now act as a proxy for the client, or link the client to the service directly. We note that the original client task has its own ID associated with it, and any other client that uses that same ID will share its display with the original client. The *registry* entities provide the mechanism through which additional clients can enquire as to existing client interactions (implemented). Whether or not this client can connect to the ongoing sessions is a function of the authentication model (not yet implemented).

5. Conclusions and Future Work

The work described above represents the first steps in the development of a robust, collaborative system to provide simple, yet flexible workflows for research scientists interested in conducting complex scientific computational tasks rather than worry about the intricacies of the underlying computational frameworks. To this end, we have addressed three important components of this framework: data entry, job submission, and backend services. Our work is guided by the principle of ease of use, fault tolerance, collaboration, and persistent records. The use of the PWSCF code serves as a prototypical code, which forms a single cog in a more complex data entry, job submission, and data analysis cycle. More generally, there are several codes that must be submitted, often in large numbers, and there is often causality between results already generated and the codes to be submitted as a result of particular

analyses. Implementation of a system that takes care of these dependencies (quasi) automatically is an end objective of this project. To this end, we will focus our attention on several important components of the system:

- Although we include at two entities of each type in our distributed system (2+ schedulers, 2+ wavelet services, etc.), there is as yet no attempt to take network and server load into account when choosing which units will perform the actual work. It is currently a first come first chosen approach. We will evaluate existing work and enhance our schedulers and services to activate themselves based on a more realistic measure of instantaneous or extended load.
- Collaboration is a natural attribute of our system. Two user tasks that subscribe to identical topics automatically receive the same information. We will investigate approaches to achieve this collaboration both at the visual level (shared user interfaces and displays), with the possibility of multiple users controlling the input. Much work has been done in this area, albeit (to the authors' knowledge) not within the context of publish/subscribe middleware.
- Complex workflows are important within VLab. Recent research has shown how to implement strategies for specifying workflows across multiple services. This work will be integrated within our system to properly link input, job submission, analysis, feedback to the user, and finally, automatic (or semi-automatic) decisions regarding the next set of simulations to submit.

Our future work will focus on improving the flexibility of collaborative environment, utilizing data flow model to formulate the task execution and providing the directory service to search the available services.

The Web Service Resource Framework [5] and particularly the WS-Notification specification family may be used to replace our pre-Web Service topic system. Support for WS-Notification is currently being developed in NaradaBrokering [15]. When this is available we will evaluate its use in our system.

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.

6. References

[1] Virtual Laboratory for Earth and Planetary Materials Project Web Site: http://www.VLab.msi.umn.edu/.
[2] Shrideep Pallickara and Geoffrey Fox NaradaBrokering: A Distributed Middleware Framework and Architecture for Enabling Durable Peer-to-Peer Grids in Proceedings of ACM/IFIP/USENIX International Middleware Conference Middleware-2003, Rio Janeiro, Brazil June 2003.
[3] Democritos Scientific Software Web Site: http://www.democritos.it/scientific.php.
[4] Plane-Wave Self Consistent Field Web Site: http://www.pwscf.org/.
[5] Globus Toolkit Version 4: Software for Service-Oriented Systems. I. Foster. IFIP International Conference on Network and Parallel Computing, Springer-Verlag LNCS 3779, pp 2-13, 2005.
[6] Hey, A. and Fox, G., eds. Concurrency and Computation: Practice and Experience, Vol. 14, No. 13-15 (2002). Special issue on Grid Computing Environments.
[7] Abdelnur, A., Chien, E., and Hepper, S., (eds.) (2003), Portlet Specification 1.0. Available from http://www.jcp.org/en/jsr/detail?id=168.
[8] The GridSphere Portal web site: http://www.gridsphere.org.
[9] Bunting, B., Chapman, M., Hurlery, O., Little M., Mischinkinky, J., Newcomer, E., Webber J, and Swenson, K., Web Services Context (WS-Context), available from http://www.arjuna.com/library/specs/ws_caf_1-0/WS-CTX.pdf.
[10] Mehmet S. Aktas, Geoffrey Fox, and Marlon Pierce. Managing Dynamic Metadata as Context. June, 2005. Available from http://grids.ucs.indiana.edu/ptliupages/publications/maktas_iccse05.pdf.
[11] Fault Tolerant High Performance Information Services Web Site: http://www.opengrids.org
[12] Gregor von Laszewski, Ian Foster, Jarek Gawor, and Peter Lane, "A Java Commodity Grid Kit," Concurrency and Computation: Practice and Experience, vol. 13, no. 8-9, pp. 643-662, 2001, http:/www.cogkit.org/.
[13] Kaizar Amin and Gregor von Laszewski. Java Cog Kit Abstractions. 2005. Available from http://www.cogkit.org/release/4_1_2/manual/abstractions.pdf.
[14] Rod Johnson, Expert One-On-One J2EE Design and Development. Wrox, 2003.
[15] Shrideep Pallickara, Harshawardhan Gadgil and Geoffrey Fox On the Discovery of Topics in Distributed Publish/Subscribe systems. Accepted for publication in proceedings of Grid 2005.