The Gateway Computational Web Portal: Developing Web Services for High Performance Computing

Marlon Pierce[1], Choonhan Youn[2], and Geoffrey Fox[3]

[1] Community Grids Laboratory, Indiana University, Bloomington IN, 47405-7000, USA,
pierceme@asc.hpc.mil,
WWW home page: http://www.gatewayportal.org
[2] Department of Electrical Engineering and Computer Science, Syracuse University, Syracuse NY 13244, USA
cyoun@indiana.edu,
[3] Departments of Computer Science and Physics, School of Informatics, Community Grids Laboratory, Indiana University, Bloomington, IN 47405-7000, USA
gcf@indiana.edu

Abstract. We describe the Gateway computational web portal, which follows a traditional three-tiered approach to portal design. Gateway provides a simplified, ubiquitously available user interface to high performance computing and related resources. This approach, while successful for straightforward applications, has limitations that make it difficult to support loosely federated, interoperable web portal systems. We examine the emerging standards in the so-called web services approach to business-to-business electronic commerce for possible solutions to these shortcomings and outline topics of research in the emerging area of computational grid web services.

1 Overview: Computational Grids and Web Services

As computational grid technologies such as Globus[1] make the transition from research projects into mainstream, production-quality software, one of the key challenges that will be faced by computing centers is simplifying the use of the Grid for users, making contact with the large pool of scientists and engineers who are using computational tools on PCs and who are unfamiliar with high performance computing.

Grid computing environments[2, 3] seek to address this issue, typically through the use of browser-based web portals. Most computational portals have adopted a multi-tiered architecture, adapted from commercial web portals. The tier definitions vary from portal to portal, but in general fall into three main categories, analogous to the three components of the model-view-control design pattern: a front-end that is responsible for the generation, delivery, and display of the user interface, a back end consisting of data, resources, and specific implementations of services, and a middle tier that acts as a control layer, handling user requests

and brokering the interactions through a general layer to specific back end implementations of services. Each of these layers can consist of several sub-layers.

Most computational portals implement a common set of services, including job submission, file transfer, and job monitoring. Given the amount of overlap in these projects, it is quite natural to propose that these portals should work together so that we can avoid duplicating the effort it takes to develop complicated services and deploy them at specific sites. The challenge to interoperability is that portals are built using different programming languages and technologies. We believe the key to portal interoperability is the adoption of lightweight, XML-based standards for remote resources, web service descriptions, and communication protocols. These are by their nature independent of the programming languages and software tools used by different groups to implement their clients and services. We thus can build distributed object systems without relying on the universal use of heavyweight solutions such as CORBA[4], although these will certainly be included in the larger web services framework.

In this paper we describe our work in developing and deploying the Gateway Computational Web Portal. We consider lessons learned about the limitations of our current approach and examine how these limitations can be overcome by adopting a web services approach. We describe specific requirements for high performance computing web services and outline our future research work in these areas.

2 The Design of the Gateway Computational Web Portal

Computational web portals are designed to provide high-level user interfaces to high performance computing resources. These resources may be part of a computational grid, or they may be stand-alone resources. As argued in [5], the real value of computational portals is not just that they simplify access to grid services for users but also that they provide a coarse-grained means of federating grid and non-grid resources. Also, because they are web-based, portals provide arguably the correct architecture for integrating the highly specialized grid technologies (suitable for development at national laboratories and universities) with commercial software applications such as databases and collaborative technologies.

We have developed and are deploying a computational portal, Gateway [6, 7, 5], for the Aeronautical Systems Center and Army Research Laboratory Major Shared Resource Centers, two high performance computing centers sponsored by the US Department of Defense's High Performance Computing Modernization Program. The primary focus of this project is to assist PC-based researchers in the use of the HPC resources available from the centers. In addition, as we have described elsewhere [8], this approach also has potential applications in education.

2.1 Gateway XML Descriptors

XML metadata descriptions serve as the interface to host machines, applications, and available services, describing what codes and machines are available to the user, how they are accessed and used, and what services can be used to interact with them. We store this information in a static data record on the web server and use it to both generate dynamic content for the user and to generate back end requests. For example, a description of an application includes the number of input files required to run it, so the same display code can be used to generate input forms for different codes.

Application Description By application, we refer specifically to third party scientific and engineering codes. All of these have common characteristics for running on a command line, so in our application description we seek to capture this information in an XML data record. For a particular application, we need to capture at least the following to run it:

1. The number of input files the code takes.
2. The number of input parameters the code takes.
3. The number of output files the code generates.
4. The input/output style the code uses.
5. Available host machines (see below).

By input and output files, we refer specifically to data files. Parameters are anything else that needs to be passed to the code on the command line, such as the number of nodes to use in a parallel application. This is highly code specific. I/O style includes standard Unix redirects or C-style command line arguments.

An example of an XML description is given in [8]. We note here that we have found it useful to adopt a shallow tree structure with general tag names. The information in the XML tree consists of name/value pairs, and we use general code for extracting this information from the data record. Thus, we do not make the pretense that our description above is complete. We can extend the XML description later with additional name/value pairs without having to rewrite any code that handles the XML.

HPC Description We have developed a description mechanism for HPC systems using the same viewpoint as our Application Description: we want a data record that contains a minimal amount of information, but which can be extended as needed, since we cannot anticipate all requirements of all codes and hosts from the beginning. The software that handles this code must of course be general in order to handle future description extensions.

Let us now examine the minimal contents of a Host Descriptor. This would include the following:

1. The DNS name of the host.
2. The type of queuing system it uses.

3. The full path of the executable on the host.
4. The working or scratch directory on the host.
5. The full path to the queue submission executable.

Again, an example of this can be seen in [8]. Note we have adopted as a convention that the Application Descriptor is the root of the tree, and the host machines are sub-nodes.

Service Description As described in 2.2, Gateway provides a number of generic services. These are implemented using WebFlow (Java- and CORBA-based middleware). However, the services themselves should be independent of the implementation. Thus all computational portals could potentially use the same interface description for a particular set of services, and any particular portal could radically redesign its middleware without changing the service interface.

An example of one of Gateway's Service Descriptors is given in [8]. For a single implementation, this is somewhat redundant, since for WebFlow we must translate this into IDL suitable for dynamic CORBA. Now that a standard for service descriptions (the Web Services Description Language, or WSDL[10]) is available from the larger web community, we will adopt this in future development work.

2.2 Gateway Architecture

For Gateway, we identified the following as the core services that we must implement in the portal:

1. Secure identification and authorization over an extended session;
2. Information services for accessing descriptions of available host computers, applications, and users;
3. Job submission and monitoring;
4. File transfer;
5. Remote file access and manipulation;
6. Session archiving and editing.

These services constitute what we term a system portal: it is not tied to specific back end applications, instead implementing these services in a generic fashion that is application neutral. Applications from specific computational areas such as chemistry or structural mechanics can be added to the portal in a well-defined fashion through the XML descriptors.

Gateway's implementation of these services is schematically illustrated in Figure 1. Clients, typically browsers but also custom applications for file browsing, contact a web server over a secure HTTP connection. Security details are highly site-specific. The user interface is implemented using JavaServer Pages (for display) combined with specially written Java components maintained by the web server's servlet engine. These latter represent a set of objects local to

the server. These components can handle specific server-side tasks but can also act as proxies for our distributed object software, WebFlow. WebFlow components can be distributed amongst different, geographically distributed computers. WebFlow components in turn act as proxies to back end resources, including stand alone HPC resources plus also computing grids and batch visualization resources. These back end resources are reached via remote shell operations. We are in the process of developing specific components for Globus using the Java CoG Kit[9]. Thus WebFlow provides a single entry point to distributed resources. For a more comprehensive description Gateway, please see [7, 5].

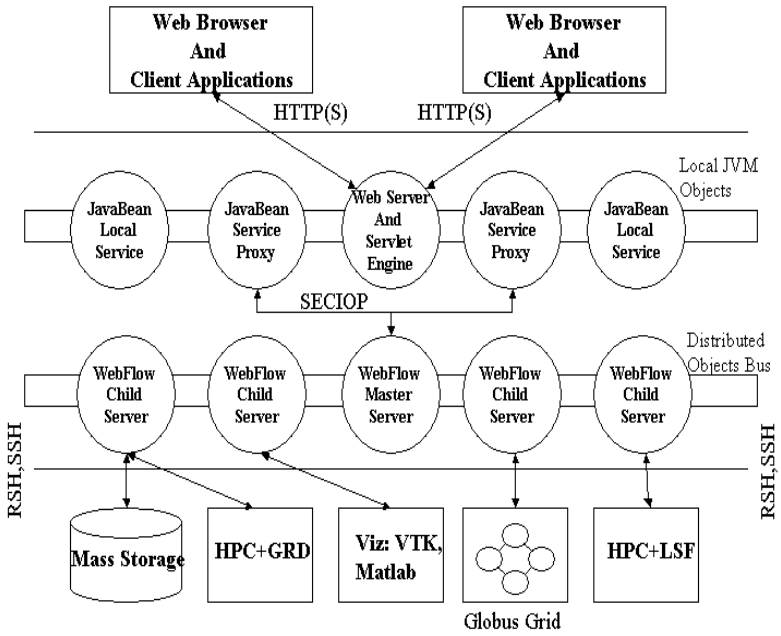

Fig. 1. The Gateway computational portal is implemented in a multi-tiered architecture. SECIOP is a wire protocol for secure CORBA. RSH and SSH are secure remote shell commands.

The services and general architectural approach described above are not unique to Gateway. Most other portal projects implement their versions of these services as well [2, 3]. As we will describe below, these services thus should be viewed as the atomic services of a computing grid web services system in the next generation of portal systems

2.3 The Need for Interoperability

There are some limitations to the traditional portal approach. Primarily, it is concerned with accessing heterogeneous back end resources through a particular

middle tier software implementation. Thus as designed now, most portal projects are not interoperable as they are tied to their particular implementation of the control layer. Yet interoperability is desirable both from the point of view of good software design and the realities of deployment. Many different groups have undertaken portal projects. Even within the DOD's Modernization Program there have been several such projects. Each portal tends to carve out a fiefdom for itself and none of these portals should expect to become the single solution for the entire program. Instead, it is in their interest to work together in order to take advantage of each project's particular strengths and avoid duplication of effort.

To see how this is an advantage, consider the difficulties (technical and political) of getting a portal installed at a particular site. Software must be installed and tested, and particular features such as security must sometimes be developed from scratch and approved by the appropriate management groups. This is a time consuming process that can become a significant portion of the total time spent on the project. But recall that the general portal services such as secure authentication and job submission are common to most portals. Thus, once one portal has successfully been deployed at a given site, it is redundant for other portals to do the same if they can instead make use of the security and job submission capabilities of the portal already in place. This can be thought of as a site-specific grid computing web service.

As a second example, consider the problem faced by portals of supporting multiple queuing systems on HPCs. Gateway and other portals have developed solutions for this. It is wasted effort for other projects to repeat this work. Unfortunately, Gateway's solution is pluggable only into portals with a Java-based infrastructure. The solution is to make queue script generation into a service that can be accessed through a standard protocol. A portal can then access the web service, provide it with the necessary information, and retrieve the generated script for its own use. This is an example of a function-specific grid computing web service.

3 Towards a Grid Web Service Portal Architecture

Given the potential advantages of interoperability, the proper architecture must now be determined. For this we can take our cue from the design goals of the underlying Grid software[11]: portal groups need to agree to communication protocols. Once this is done, they don't need to make further agreements about control code APIs and service implementations so long as the protocols are supported. The web services model [12, 13] has the backing of companies such as IBM[14] and Microsoft[15] as the standard model for business-to-business commerce. Let us now consider the aspects of this model for high performance computing.

Distributed Object Model The point of view that we take is that all back end resources should be considered as objects. These may be well-defined software

objects (such as CORBA objects) but may also be hardware, applications, user descriptions, online documents, simulation results, and so forth. As described below, these objects will be loosely coupled through protocols rather than particular APIs.

Resource and Service Descriptions The generalized view of resources as objects requires that we describe the meta-data associated with the object and provide a means of locating and using it. XML is appropriate for the first and we can assign a Uniform Resource Identifier (URI) for the latter. We must likewise provide an XML description of middle-tier services (see Section 2.2). WSDL is an appropriate descriptor language. Note that this defines objects in a language independent way, allowing them to be cast into the appropriate language later, using for example Castor[17] to create Java objects. Likewise, this distributed object system is protocol-independent. The object or service must specify how it is accessed, and can provide multiple protocols.

Resource and Service Discovery Once we have described our object, it must be placed in an XML repository that can be searched by clients. Clients can thus find the resource they are looking for and how to access it. Commercial web services tend to use UDDI[16], but other technologies such as LDAP servers may also be used. Castor provides a particularly powerful way of converting between Java components and XML documents and when combined with an XML database, can serve as a powerful object repository specialized for Java applications [8].

Service Binding Following the discovery phase, the client must bind to the remote service. WSDL supports bindings to services using different mechanisms (including SOAP and CORBA). Thereafter, interactions can be viewed as traditional client-server interactions.

There is nothing particularly new about this architecture. Client-server remote procedure calls have been implemented in numerous ways, as has the service repository. WebFlow, as just one example, implements a specialized naming service for looking up servers and their service modules. The difference here is that the repository is designed to offer a standard XML description of the services using WSDL. The service description itself is independent of wire protocols and Remote Procedure Call (RPC) mechanisms.

There still is the problem that the client must implement the appropriate RPC stubs to access the server. Given the number of protocols and RPC mechanisms available, there is an advantage in the various grid computing environment projects coming to an agreement on a specific RPC protocol. Guided by the surge of development in support of SOAP[18] for web services and its compatibility with HTTP, we suggest this should be evaluated as the appropriate *lingua franca* for interoperable portals. Legacy and alternative RPC mechanisms (such as used by WebFlow) can be reached through bridges.

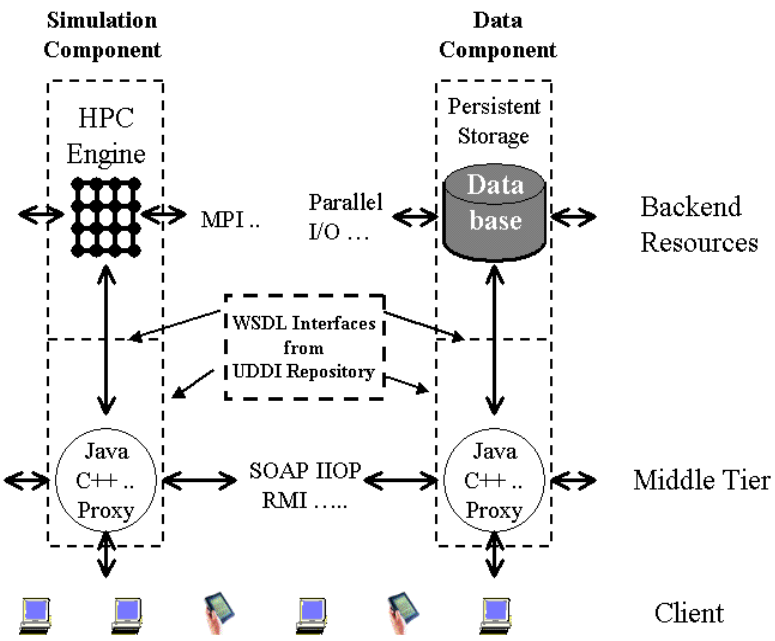

Fig. 2. The extended multi-tiered architecture for web services will be used to support interoperability between portals.

The architecture for this kind of web services system is illustrated in Figure 2. The basic point is that common interfaces can be used to link different multi-tiered components together. In the shown instance we have two distinct services (and HPC service and a database service) perhaps implemented by different groups. Clients can access these services by first contacting the service description repository. Essentially, the local object sub-tier and the distributed object sub-tier of the control layer in Figure 1 become decoupled. The local object service proxy must consult the service repository and find a service to use. It then binds to that service with the protocol and mechanism prescribed in the service description.

The above constitutes the basis for web services. We now identify some important extensions that will need to be made.

Security for high performance computing services needs consideration. First, the very fact that a particular web service is available from a certain site may be sensitive information. Thus the service repository must have access and authorization controls placed upon it in order to allow clients to only see services that they are authorized to use. Furthermore, the services themselves must be secure, as they typically cannot be used anonymously. Specific security extensions required by the service description include

1. Authentication mechanisms for client and/or server.
2. Means of identification (certificate, public or private key, session ticket).

3. Requirements for privacy and data integrity.

Within a limited realm (a Globus Grid or interoperating Kerberos realms) this can be greatly simplified through existing single sign-ons capabilities.

A second important consideration for HPC services is the relationship between job composition, workflow, and events. The simpler web services can be thought of as atomic components that can be used to compose complicated services. For example, a job can be scheduled and its output filtered and transfered to a visualization service for analysis. This entire task can be considered to be a single, composite job. Thus we need to define an XML dialect for handling the above type of workflow, together with appropriate software tools for manipulating these workflow documents and executing their content. We specifically see this as an agent that executes the workflow commands, contacting the service repository at each stage and reacting appropriately to event messages from the back end. Job composition is not new: WebFlow has this capability, for example. The difference now is that we must define workflow in an implementation-independent way (again in XML) so that different groups can provide different pieces of the combined service.

Workflow for grids is a complicated undertaking because of the numerous types of error conditions that can occur. These include failures to reach the desired service provider at any particular stage of the workflow either because of network problems or machine failure. Likewise, even if the resource is reached and the service is executed, failure conditions exist within the application. For example, the file may be corrupt and generate incorrect output, so there is no need to attempt the next stage of the workflow procedure. We can thus see that any workflow description language must be coupled with reasonably rich event and error description language.

4 Acknowledgments

Development of the Gateway Computational Web Portal was funded by the Department of Defense High Performance Computing Modernization Program through Programming Environment and Training.

References

1. The Globus Project. http://www.globus.org
2. Grid Computing Environments. http://www.computingportals.org.
3. Grid Computing Environments Special Issue.
   Concurrency and Computation: Practice and Experience.
   Preprints available from http://aspen.ucs.indiana.edu/gce/index.html.
4. Common Object Request Broker Architecture. http://www.corba.org
5. Pierce, M., Fox, G., and Youn, C.: The Gateway computational web portal. Accepted for publication in Concurrency and Computation: Practice and Experience.
6. Gateway Home Page. http://www.gatewayportal.org.

7. Fox, G., Haupt, T., Akarsu, E., Kalinichenko, A., Kim, K., Sheethalnath, P., Youn, C.: The Gateway system: uniform web based access to remote resources. ACM Java Grande Conference (1999).
8. Fox, G., Ko, S.-H., Pierce, M., Balsoy, O., Kim, J., Lee, S., Kim, K., Oh, S., Rao, X., Varank, M., Bulut, H., Gunduz, G., Qiu, X., Pallickara, S., Uyar, A., Youn, C.: Grid Services for Earthquake Science. Accepted for publication in ACES Special Issue of Concurrency and Computation: Practice and Experience.
9. von Laszewski, G., Foster, I., Gawor, J., Lane, P.: A java comodity grid kit. Concurrency and Computation: Practice and Experience, **13** (2001) 645-662.
10. WSDL: Web Services Framework. http://www.w3c.org/TR/wsdl.
11. Foster, I., Kesselman, C., Tuecke, S.: The Anatomy of the Grid: Enabling Scalable Virtual Organizations. Intl. J. Supercomputer Applications, **15** 2001.
12. Saganich, A.: Java and Web Services. http://www.onjava.com/pub/a/onjava/2001/08/07/webservices.html
13. Curbera, F.: Web Services Overview. http://www.computingportals.org/GGF2/WebServicesOverview.ppt
14. IBM Web Services Zone. http://www.ibm.com/developerworks/webservices
15. Microsoft Developer Network. http://msdn.microsoft.com
16. UDDI: Universal Descriptors and Discovery Framework. http://www.uddi.org.
17. The Castor Project. http://castor.exolab.org
18. Simple Object Access Protocol (SOAP) 1.1. http://www.w3c.org/TR/SOAP.