A Batch Script Generator Web Service for Computational Portals

Steve Mock, Kurt Mueller
University of California at San Diego, San Diego Supercomputer Center
{mock,kurt}@sdsc.edu
Postal Address: UC San Diego, MC 0505
9500 Gilman Drive
La Jolla, CA 92093-0505
Phone for Corresponding Author : (858) 534-8398
FAX: (858) 822-5407

Marlon Pierce, Choonhan Youn, and Geoffrey Fox
Community Grid Labs, Indiana University
{marpierce,cyoun,gcf}@indiana.edu
Postal Address: 501 N. Morton Street
Bloomington, IN 47401
Phone for Corresponding Author: (812) 856-1212
FAX: (812) 856-7972

Mary Thomas
Texas Advanced Computing Center, The University of Texas at Austin
mthomas@tacc.utexas.edu
Phone: (512) 471-6363
FAX: (512) 475-9445

Submitted to Communications in Computing '02

Abstract

Web services have begun to receive a great amount of attention as the appropriate backbone for business-to-business interactions. Web services essentially do not present new concepts for distributed computing but rather implement important simplifying, standards-compliant ways for entities to find and invoke the appropriate remote service. These have important implications to the field of computational, or science, portals. We examine the basic web services architecture from the perspective of these portals, using a batch script generation service as a test case.

1. Introduction

Computational grid portals, and more generally grid computing environments, are designed to simplify access to grid technologies and to compose basic grid services into specialized application and working environments suitable for scientists and engineers. Developing these portals, including integrating and coordinating multiple, distributed resources and differing grid technologies with various APIs, requires a significant amount of effort. Portal developers often re-implement functionality found in existing portals because there are no common discovery and access mechanisms in place to enable sharing of portal functions.

The emerging web services architecture has the potential to help simplify the job of the portal developer, while at the same time increasing the flexibility and power of portals for users. "Web services" refers to an architecture in which atomic services, such as job submission or storage management, are published for use by portals, applications, or other services using XML for the service interface, data description, and invocation protocol. Dynamic discovery of published services is facilitated by registering service information with various queryable directory systems. Widespread adoption of web services protocols, systems, and interfaces will allow Grid portals, applications, and users to publish, reuse and dynamically share data, metadata, and services and will have a significant impact on how the Grid evolves.

Using the web services model (which we describe in more detail in Section 2 of this paper), portal projects can be constructed as specific implementations and composites of basic underlying web services components and can also provide services that may be shared among different projects. Legacy applications and services can be 'wrapped' to make them web services that are available to appropriate users, including other portals and applications. We have experience with the HotPage and Gateway portals [17] [1], and by following the Web services approach, we are able to share existing services that we previously developed independently.

In our approach, we will define basic categories of Web services:

1. Site-specific services: many portals have spent a great deal of effort customizing services such as authentication, job submission and monitoring for particular deployment sites. By reconstructing these services as web services, a particular function such as "Job Submission to Site A" becomes a general service provided by the portal. Other portal projects can then implement this service when they want to use Site A, rather than go through the often extensive development process for deploying a new service.
2. Software services: these services are geared to particular applications (e.g., a finite difference method or LU solver service, a GAMESS service) that handle the details of where and when jobs get run.
3. Basic anonymous services: other services are not site-specific and may be used anonymously. Publishing basic services avoids duplication of effort between portal projects.

We suggest that these types of basic building blocks will be useful in constructing more complicated services. It will be possible for portal developers as well as users to easily create novel applications dynamically by chaining simple web services into complex sequences of operations.

As part of a community effort to explore the feasibility of Web services for Grid portals, the Grid Computing Environments Research Group (GCE-RG) [9] has begun to develop the "Interoperable Web services Testbed" [16]. The goal of this project is to explore which web services are needed by portals in order to accomplish simple tasks that span project, organizational, and international boundaries. The purpose of the testbed is not to develop standards but to set up a Grid environment to explore issues associated with interoperability among developers and organizations, security and policy mechanisms, and workflow concepts. The GCE has proposed an initial *minimal* set of web services: job management, accounts and allocations, data management, events, Grid messaging, scheduling, security, applications, and collaborations. As part of a first test case for evaluating interoperability, we chose to build a simple, anonymous, batch script generation service.

In this paper, we present our experiences with developing the Batch Script Generator Web Service (BSG-WS). We first present an overview of Web services from the perspective of the portal developer, which is a distinctly different view from that of other Grid developers, followed by an example of how a portal developer might integrate a set of simple web services. We then describe the details of the Web Service, and present specific examples of how the various components of the web services architecture play a role in portal web services.

2. Web Services and Computing Portals

Web services have been extensively discussed in many articles, so we will only briefly review them here. The Web service constituent pieces are SOAP[15], WSDL[21], UDDI [19], and WSIL [21]. SOAP and WSDL are standards supported by the World Wide Web Consortium (W3C) [24]. The collection of these autonomous parts into the Web services architecture is summarized in several recent articles [2, 20, 22]. Web services will also form the basis for the next generation of the Globus toolkit for grid computing [5].

The influence of a Web services architecture on the development of a Grid portal is demonstrated in Figure 1, which shows a simple, conceptual diagram of how portals, applications and web services might utilize distributed, interoperable, Grid Web services provided by multiple organizations. Since the developers need only program to a protocol and interface, the implementation details such as programming language, database, or Grid service used becomes irrelevant. The diagram conceptualizes how a portal accesses distributed Grid Web services to submit a job based on a set of steps that flow from authentication to authorization, data management, and job submission.

This diagram conceptualizes how multiple portals or applications (represented by the top layer), hosted across multiple webservers or resources, might utilize grid web services. The steps are as follows: (1) Texas Portal (TP) authenticates user via MyProxy Web Service; (2) TP checks resource status; (3) TP check AKENTI authorization; (4) TP pulls files from collection; (5) TP submit request to JobSubmit web service (JSWS); (6) JSWS submits request to Batch Script Generator WS; (7) JSWS submits request to GRAM gatekeeper; (8) when done TP sends request to (9) SRB-WS to migrate files into collection.

This example demonstrates the natural composite nature of web services and how easily a workflow can be constructed because of the interoperable protocols. Note that details such as the order in which tasks should be done (both sequential and parallel) will be an area of research that must be addressed and will have an impact on the efficiency of the Grid.

3. Batch Script Generator Web Service

Web Services essentially define protocols and interfaces in XML. We believe that the simplicity of this approach will make it possible to develop interoperable and reusable services that can be shared between portal groups. We must now must test these beliefs by evaluating the existing specifications and tools to see if they meet the requirements of computing portals. The Batch Script service described here is the first example of such an interoperable web service for our community.

We have created two instances of the BSG-WS that generate queue scripts for the appropriate batch queuing systems, based on a user's request for resources, residing at SDSC, and Indiana University. These are anonymous services and can be used by anyone wanting to generate queues for PBS, Load Leveler, and LSF. Support for other queuing systems will be added in the future, as will other service providers.

We selected the creation of a web service that generates batch scripts because both SDSC and IU had previously implemented this service for our existing portals (HotPage and Gateway). We wanted to develop a Web service that could be used by existing projects. It thus was a natural candidate to serve as both an intrinsically useful web service and as a prototype for more complicated services. Unlike web services such as job submission, the generator does not require special client-side authentication before use, so security and logistics issues for setting up account are not important. We do note that server-side authentication and wire privacy will be important in a production version of this service.

3.1. SOAP for Service Invocation

The BSG-WS makes one method (batchGen) available for use by clients through a SOAP-encoded remote procedure call. The Java declaration of batchGen is simply

```
public String batchGen(String xmlString)
```

The input to the RPC method batchGen is a string of XML that describes the batch job for which the client wishes to create a batch script. The string output of batchGen is the batch script generated by the web service based on the XML job input. SOAP messages between client and server are generated through Apache SOAP [26].

The BSG-WS converts the input string into an XML document object and parses and validates the object using the Xerces [25] Java DOM XML parser. The validation step detects user errors in the input XML, reducing complexity of the Web service by alleviating some of the need for error checking in the web service code. The validated and parsed XML document object is passed to a broker that extracts the batch scheduler type. If the scheduler type is not supported by the Web service, then a message to that effect is returned to the client. If the scheduler is supported, then the XML document object is passed to a handler created for that specific scheduler. The scheduler handlers are custom-written methods that generate a batch script given an XML document based on a specific DTD and return the generated batch script as a string. The web service then passes the generated string back to the client as the result of the SOAP RPC query.

3.2. WSDL for Interface Definition

WSDL is used to declare method interfaces in XML. The details of WSDL are described elsewhere. It is a straight forward to convert the method definition above into WSDL. We did note some minor incompatibility between a hand-written version of the service interface and client stubs generated using IBM's Web Services toolkit. This essentially was a problem with the toolkit, not WSDL. We otherwise found no difficulties using WSDL to invoke the service with clients written in both Java and Python. We note however, that the batch script parameter request that we currently pass as a single string is a good candidate for a WSDL complexType, effectively an XML data object. This will present a more interesting case for interoperability, as it will allow us to test object passing between different languages using XML as the intermediary.

3.3. Using UDDI for Resource Discovery

UDDI (for Uniform Description, Discovery, and Integration) is usually presented as the appropriate discovery mechanism for Web Services. UDDI specifications include a container hierarchy for describing data and an API for querying and searching a repository. General UDDI properties are described elsewhere [19].

We have created entries for our prototype web service in a UDDI repository at IU [12]. UDDI business entities map directly to the portal service providers (Gateway and HotPage). For each such entity we have one or more services (the batch script generator, plus additional services such as job submission and context management). We use the service bindings to point to the SOAP web servers, the access points for our services. We also publish our WSDL interfaces in the UDDI directory and place links to these within the service repository. The WSDL information is not stored directly within the UDDI, but rather we point to it on external web servers.

UDDI queries and subsequent web service interactions involve multiple servers. A user interacts with a web server that controls the user interface (UI). The UI server in turn contains components that implement SOAP calls and responses with the UDDI server and the service provider. The user first requests a search for a service provider, a particular service by name, or a service with certain capabilities. The latter type of request is illustrated by the search for a batch script generator that supports a particular queuing system. Based on the search results, the user can then select the appropriate service. The UI server obtains from the UDDI directory the location of the service provider (in our case, the URL of the SOAP server, but potentially also ftp and mail servers and even FAX numbers). The UI also obtains the location of the WSDL file from the UDDI directory and uses this to set up the requests to the selected service provider.

The UDDI repository's main strength is its searching capabilities, and some consideration must be given to proper data representation in the repository in order to maximize these capabilities. Each portion of the UDDI is searchable: the repository can be searched (with wild cards and partial matches) for particular business entities, which can in turn be searched for matching services, and so on. There are several different queuing systems available to end users, and these are not all supported by both services. The HotPage service is based around LSF and LoadLeveler, while the Gateway web service supports PBS queue scripts. The end user will typically search the UDDI by the particular queue that he or she wants: a search for PBS will list Gateway, and so on. Overlapping support for queues will produce multiple search results. The user can then select the desired service provider and execute the service.

Some of UDDI's mechanisms for representing data about web services are clearly designed around the needs of businesses, and agreement on UDDI usage standards amongst members of the Grid community will be required in order to use the mechanisms effectively. For instance, we put information about the specific queues applicable to each batch script generation service in the service's description field as a temporary device to enable locating services. They are represented here as a single string of the form "Gateway+Queue+PBS". It would be better to represent the queues with individual key-value pairs such as "queue=Gateway." UDDI provides this form of representation through Category and Identifier structures. However, these are currently geared toward standard business classification systems external to UDDI. See the specifications in Ref [19] for more detail. Grid web service providers will need to agree on standard Category and Identifier keys and values in order for these mechanisms to be useful in advanced searches. UDDI searches without the use of categories or identifiers are extremely limited and will lead to the adoption of ad-hoc solutions by different developer groups.

It is possible that UDDI may be overkill for the computational web services community for the foreseeable future. While UDDI was designed to meet the needs of businesses looking to establish new business relationships, WSIL on the other hand is geared more towards partners that have an existing relationship and need to manage metadata about their relationship (published services, locations of WSDL files, methods that the files and services may be accessed, and so forth). For these reasons, it appears that WSIL may be more appropriate for use by the relatively small grid community. The primary drawback to WSIL is that it is currently a proprietary specification of IBM and Microsoft, although they have announced eventual plans to make this public.

4. Conclusion

In this paper we have presented the design and development of a simple Batch Script Generator Web Service, and we have explored using a directory service to publish information about the web service. We have shown that this Web service can be used by multiple portals. This work has demonstrated to us that the conversion of existing portal functions to web services mechanisms and adherence to emerging web services standards in development of future grid application can be done and will ease the task of portal developers and provide new levels of flexibility and power to portal users.

Many decisions about specific implementation details for using Web Services for computational portals remain to be made, and through participation in the GCE Research Group of the Global Grid Forum and other community efforts, we will play an active role in determining the evolution of portals and web services. Initiatives are presently underway to integrate Globus GSI security into web services environments to enable compatibility with current infrastructure. In particular, we will be carefully monitoring the efforts of the Globus group and IBM in developing their OGSA standard for open, secure web services. We are continuing to collaborate in implementing web services for existing and new functions, and we are excited about exploring web services approaches to overcome the limitations of today's computational portal environments.

5. References

[1] Akarsu, E., Fox, G., Haupt, T., Kalinichenko, K., Kim, K. , Sheethalnath, P., and Youn, C.. Using Gateway System to provide a desktop access to high performance computational resources. Proceedings of the Eight IEEE International Symposium on High Performance Distributed Computing, August, 1999.

[2] Castro-Leon, Enrique . A Perspective on Web Services. Accessed from http://ww.webservices.org/index.php/article/articleview/114/1/3.

[3] Common Object Request Broker Architecture Web Site. Accessed from http://www.corba.org.

[4] Extensible Markup Language (XML). Accessed from http://www.w3c.org/XML

[5] Foster, I., et al. The Physiology of the Grid: an Open Grid Services Architecture for Distributed Systems Integration. Accessed from http://www.globus.org/reserch/papers/ogsa.pdf.

[6] Global Grid Forum 5, Edinburgh, Scotland. GGF website located at http://www.gridforum.org.

[7] Global Grid Forum m.Website. Accessed from http://www.gridforum.org.

[8] Grid Computing Environments Research Group Special Issue on Grid Computing Portals. To appear in Concurrency and Computation: Practice and Experience, John Wiley and Sons.

[9] Grid Computing Environments. Accessed from http://www.computingportals.org.

[10] IBM Web Services Toolkit, available at http://www.alphaworks.ibm.com/tech/webservicestoolkit.

[11] Modi. Tarak. WSIL: Do we need another Web Services Specification? Accessed from http://www.webservicesarchitect.com/content/articles/modi01.asp.

[12] Pierce, Marlon (marpierc@indiana.edu), contact for access to the UDDI repository described in this paper.

[13] Seely, S. SOAP: Cross Platform Web Service Development Using XML; Prentice Hall: Upper Saddle River, 2002.

[14] Siegel, J. ed. CORBA 3: Fundamentals and Programming. John Wiley and Sons, 2000.

[15] Simple Object Access Protocol (SOAP) 1.1. Accessed from http://www.w3c.org/TR/SOAP

[16] Thomas, M. P., et. al., The GCE Interoperable Web Services Testbed. Submitted to Special Issue of the Journal of Parallel Distributed Computing: Computational Grids, R. Wolski and Jon Weissman, co-editors (2002). Available at: http://www.computingportals.org/testbed. http://www.webservicesarchitect.com/content/articles/modi01.asp.

[17] Thomas, M. P., Mock, S., Dahan, M., Mueller, K., Sutton, D., The GridPort Toolkit: a System for Building Grid Portals. Proceedings of the Tenth IEEE International Symposium on High Performance Distributed Computing, August, 2001.

[18] Uniform Resource Identifier. Accessed from http://www.ietf.org/rfc/rfc2396.txt

[19] Universal Description, Discovery, and Integration (UDDI). Accessed from http://www.uddi.org.

[20] Vaughan-Nichols, Stephen J.. Web Services: Beyond the Hype. Computer. Vol 35, No. 2, p 19.

[21] Web Services Description Language (WSDL) 1.1. Accessed from http://www.w3c.org/TR/wsdl

[22] Web Services Inspection Language (WS-Inspection) 1.0. Accessed from http://www-106.ibm.com/developerworks/webservices/library/ws-wsilspec.html. See also http://msdn.microsoft.com/library/default.asp?url=library/en-us/dnglobspec/html/ws-inspection.asp

[23] Wolter, Roger. XML Web Services Basics. Accessed from http://msdn.microsoft.com/library.

[24] World Wide Web Consortium (W3C). Accessed from http://www.w3c.org

[25] Xerces Java Parser website: http://xml.apache.org/xerces-j/

[26] Apache SOAP: http://xml.apache.org/soap/

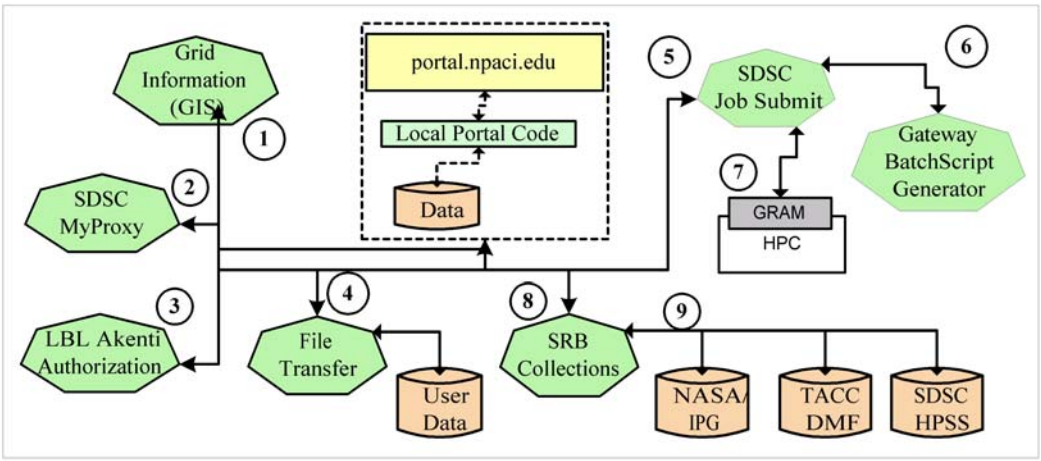

Figure 1 A schematic representation of grid web service interactions.

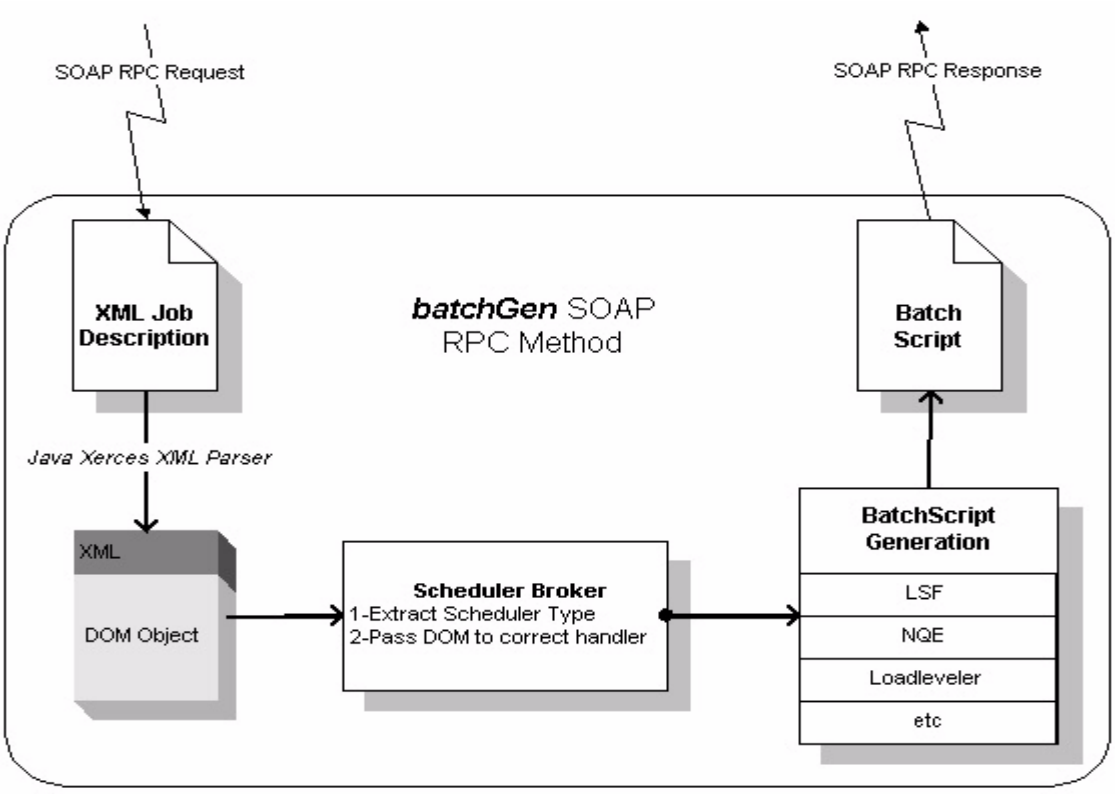

Figure 2: The steps that the ***batchGen*** remote procedure call takes to take an XML string from a SOAP RPC request and return a string containing a working batch script in a SOAP RPC response.