A Batch Script Generator Web Service for Computational Portals

S. Mock
San Diego Supercomputer Center
University of California at San Diego
San Diego, CA
(mock@sdsc.edu)

M. Pierce
Community Grids Lab
Indiana University
Bloomington, IN
(marpierc@indiana.edu)

C. Youn
Community Grids Lab
Indiana University
Bloomington, IN

G. Fox
Community Grids Lab
Indiana University
Bloomington, IN

M. Thomas
Texas Advanced Computing Center
University of Texas
Austin, Texas
(mthomas@tacc.utexas.edu)

Abstract

We examine the use of Web services, an XML-based distributed object system, for developing reusable, interoperable services for computational science web portals. This paper describes a batch script generation service that serves as our prototype for evaluating SOAP, WSDL, and UDDI. We describe steps we took to create this service from legacy applications, interoperability tests between multiple clients and different installations of the service, and evaluations of service discovery mechanisms.

Keywords: Web Services, Computational Portals

1 Introduction

Computational grid portals and grid computing environments are designed to simplify access to grid technologies and to compose basic grid services into application and working environments for scientists and engineers. Developing these portals is a significant effort, and portal developers often reimplement services developed by other groups because there are no common discovery and access mechanisms to enable portal service sharing.

The emerging Web services architecture has the potential to both simplify portal development and increase the flexibility and power of portals for users. In the Web services architecture, 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. Services may be discovered dynamically from published service descriptions placed in queryable directory systems.

Using the Web services model, portal projects can be constructed as specific implementations and composites of basic underlying components. They can also provide shared services distributed among different projects. We have developed the basic portal service implemetations for our separate portal projects (HotPage[1] and Gateway [2]), so our task is to convert these into interoperating services.

This work is part of the Grid Computing Environments Research Group's [3] community effort to explore the feasibility of Web services for Grid portals. For this work we are developing the Interoperable Web Services Testbed. Our goal is to determine a core set of portal Web services. The proposed initial set includes job management, accounts and allocations, data management, events, Grid messaging, scheduling, security, applications, and collaboration. The batch script generation service described below is the testbed's first test case for interoperability.

2 Web Services and Computing Portals

The Web service constituent pieces are SOAP[4], WSDL[5], UDDI[6], and WSIL[7]. SOAP and WSDL are standards supported by the World Wide Web Consortium (W3C)[8]. We use SOAP for remote method invocation between services, WSDL to describe the method interfaces, and UDDI as the service repository and discovery system. Web services will also form the basis for the next generation of the Globus toolkit for grid computing [9].

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 services provided by several organizations. Since developers need only program to a protocol and interface, implementation details of the service become 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.

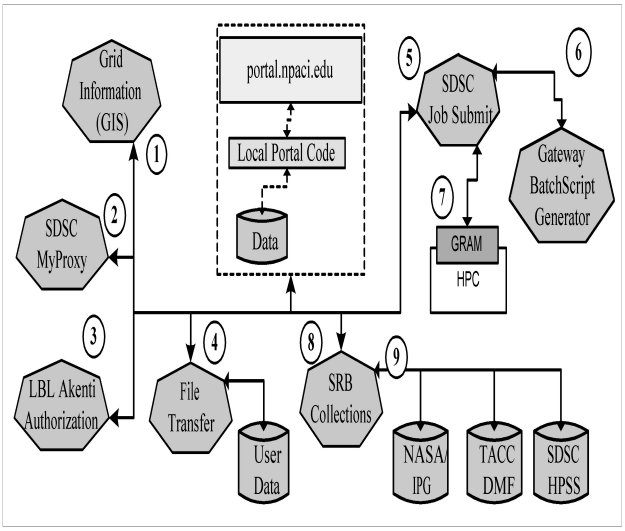

Figure 1: The next generation of computing portals will be composed of distributed, interoperable Web services

The steps are as follows: (1) the Texas Portal (TP) authenticates a user via MyProxy Web Service [10]; (2) TP checks resource status; (3) TP check AKENTI authorization [11]; (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 distributed Web services and thus the importance of interoperability. We must now test these ideas by evaluating the existing specifications and tools to see if they meet our requirements.

3 Batch Script Web Service

The Batch Script Generation Web Service (BSG-WS) described here is our community's first example of an interoperable Web service. Working script generators were previously developed independently by our groups to support PBS (Gateway), LoadLeveler (Hotpage) and LSF (HotPage). Because implementations already existed and had no special security requirements, the script generators were natural candidates to be converted into an interopable Web service.

The BSG-WS makes one method (batchGen) available to 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 parameters. The generated batch script is returned as another string.

The BSG-WS converts the input string into an XML document object, which is parsed and validated. Validation 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 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. This is summarized in Figure 2.

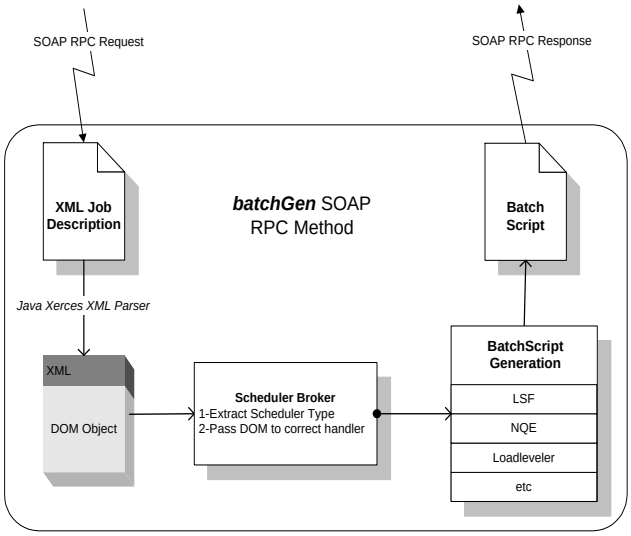

Figure 2: The batch script generation service supports multiple queuing systems through a single interface.

WSDL is used to declare method interfaces in XML. It is straightforward to convert the method definition above into WSDL. We found no important difficulties using WSDL to invoke the Java-based service with clients written in both Java and Python. We note however, that the batch script parameter request string is a good candidate for a WSDL complexType. This will present a more interesting case for interoperability, testing object passing between different programming languages using XML as the intermediary.

UDDI (for Uniform Description, Discovery, and Integration) is usually presented as the discovery mechanism for Web services. UDDI specifications include a container hierarchy for describing data and an API for querying and searching a repository.

We have created entries for our prototype Web service in a UDDI repository. UDDI business entities map directly to the portal service providers (Gateway and HotPage). Each entity has 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 that run the BSG-WS. We publish the WSDL interfaces in the UDDI directory as URLs.

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. 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. Each portion of the UDDI's data hierarchy 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.

The BSG-WS presents a twist on the UDDI search capabilties. There are several different queuing systems available to end users, and these are not all supported by both services, as described above. The end user will typically search the UDDI by the desired queue: 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.

The difficulty we encountered was that there was no straightforward way to distinguish a PBS-supporting service from a LSF- or LoadLeveler-supporting service. UDDI provides Category and Identifer structures, but these are geared toward standard, external commercial classifications and are not appropriate for our service. For a temporary workaround, we placed the distinguishing information into each service's description field, but this must use an adhoc format. In order for UDDI to be useful for interoperable computing (rather than business) services like BSG-WS, bodies such as the GCE must define a service taxonomy and ex-

tend UDDI appropriately.

Alternatives to UDDI, including WSIL and XML databases, exist and may be more appropriate for computing grids. These need more investigation. We believe that information and resource discovery remains an open area.

4 Conclusions

We have presented the design and development of a simple interoperable Batch Script Generator Web Service, which we have shared between our web portals. This has demonstrated the feasibilty of converting existing portal functions into Web services. Adherence to emerging standards in developing future grid application will ease the task of portal developers and provide new levels of flexibility and power to portal users.

The work descirbed here is the first step towards developing a suite of interoperable core services and composing them into single applications, as summarized in Figure 1. Future work will include developing common interfaces for more core services, evaluating security issues, and developing techniques to bind core services into single composite services.

References

[1] Thomas, M. P., et al. The GridPort Toolkit: a System for Building Grid Portals. Proceedings of the Tenth IEEE International Symposium on High Performance Distributed Computing, August, 2001.

[2] Akarsu, E., et al. Using the Gateway System to Provide Desktop Access to High Performance Computational Resources. Proceedings of the Eight IEEE International Symposium on High Performance Distributed Computing, August, 1999.

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

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

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

[6] Universal Description, Discovery, and Integration. Accessed from http://www.uddi.org

[7] Web Services Inspection Language 1.0 Accessed from http://www-106.ibm.com/developerworks/webservices/libray/ws-wsilspec.html.

[8] World Wide Web Consortium Web Site. Accessed from http://www.w3c.org.

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

[10] Novoty, J., Tuecke, S., Welch, V. Proceedings of the Tenth International Symposium on High Performance Distributed Computing, IEEE Press (2001).

[11] Thompson, M., et al. Certificate-based Access Control for Widely Distributed Resources. Proceedings of the Eigth Usenix Security Symposium (1999).