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

Marlon Pierce, Choonhan Youn, and Geoffrey Fox
Community Grid Labs, Indiana University
{marpierc,cyoun,gcf}@indiana.edu

Mary Thomas
Texas Advanced Computing Center, The University of Texas at Austin
mthomas@tacc.utexas.edu

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.

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. 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 and job submission for particular sites. By reconstructing these services as web services, a particular function such as "Job Submission to Site A" becomes a general service that can be used by multiple portals.
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. Site-independent services: other services that are function-specific rather than site-specific. The batch script generation service described in this paper is one example. Publishing basic services avoids duplication of effort between portal projects.

These basic building blocks will be used to construct 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 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. An Overview of Web Services

The GCE definition of web services is based on current commercial standards and consists of the following major components:

1. A protocol for invoking remote procedure calls;
2. An interface definition language that is used to define services; and
3. A service repository that can be used to find services.

A number of standards have been developed for each of these parts. The remote procedure call protocol is Simple Object Access Protocol (SOAP) [15] and method interfaces are defined using WSDL [21] (Web Services Definition Language). Both are standards supported by the World Wide Web Consortium (W3C) [24] and are based on XML [4]. The service repository usually is compliant with the UDDI [19] (the Universal Description Discovery and Integration) specification. UDDI is not a W3C standard but rather is an agreed-upon industry standard.

A fourth standard, the Web Services Inspection Language (WSIL) [21] is sometimes added to this list. WSIL, currently under proprietary development by IBM and Microsoft, is a simple XML-based description of web service metadata. WSIL is intended to be a light-weight registry that can be quickly developed while UDDI matures.

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].

2.1. Impact of Web Services on Portals

The influence of a web services architecture on the development of a Grid portal is demonstrated in Figure 1, which shows a simple 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, the implementation details such as programming language, database, or Grid service used becomes irrelevant, as long as the protocols can be agreed upon. The diagram depicts 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 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

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, as described above, 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 existing portals (HotPage and Gateway), and 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 accounts 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

SOAP standardizes the XML conversations used to perform remote procedure calls (RPCs) during a web service transaction. XML is used by SOAP libraries to create the messages sent both to the web service from the client, and again for the message response sent from the web service back to the client. In order for a client to make a remote procedure call, it must know:

- The URI (Uniform Resource Identifier [18]) of the target object
- A valid method name for that object

- The method parameters [13]

All of this information is contained in the WSDL description of the web service, which can be published using UDDI.

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:

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

This declaration establishes that batchGen takes one string, and returns another.

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.

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[4] 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 an XML-based specification used to describe web services. The WSDL file provides all the necessary information for a developer to make a client that uses the web service. There are toolkits like the WSTK [10] that generate code stubs for a client or for a web service. If a client developer has access to the WSDL file for a web service, then they have all the information needed to create a client program.

The Batch Script Generator web service WSDL was initially written by hand, to familiarize the developers with the process of creating WSDL descriptions of web services. This initial WSDL file was compared to one that was generated using the Java source code for the web service using the WSTK. The comparison revealed a small error in the initial hand-written WSDL file that was causing problems for some clients. The improved WSDL file generated by the WSTK was adopted and is attached as Appendix A.

3.3. Using UDDI for Resource Discovery

The UDDI repository serves as a standard way for businesses to post and search information about services. A UDDI repository contains both human-readable information and technical information needed to construct client programs and connect to the appropriate services.

A UDDI repository consists of nested containers: each repository contains zero or more business entities, each entity contains zero or more services, and so on. Each data type contains required and optional parameters, such as name, description strings, pointers to documentation and related material, and identifiers and categories for classification.

Coupled to these data types are the publishing and searching methods that constitute the programmer's interface. Publishing privileges are restricted by authentication and authorization, and allow the privileged user or application to add, modify, and delete information about business and services. Other client applications can be built from UDDI searching functions, which are keyed to each of the basic data types.

We have created entries for our prototype web service in a UDDI repository at IU [12]. The 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.

Figure 3 illustrates the full interactions of the different web services for a computational portal. A user interacts with a web server that controls the user interface (UI). The UI server in turn contains components that implement SOAP communication 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. The user can then select the appropriate service. The UI server obtains from the UDDI directory the URLs of the desired batch script service and the WSDL file and uses this to set up requests to the selected service provider.

The UDDI repository's strength is its searching capabilities, so we must consider the proper data representation in the repository. 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=PBS." 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. UDDI was designed for businesses looking to establish new partnerships. WSIL on the other hand is geared more towards partners that have an existing relationship and need to manage metadata about their published services, such as locations of WSDL files. For these reasons, it appears that WSIL may be more appropriate for use by the relatively small grid community and needs further investigation.

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 other portals. This work has convinced us that the conversion of existing portal functions to web services mechanisms and adherence to emerging web services standards in development of future grid applications will ease the task of portal developers and provide new levels of flexibility and power to portal users.

Many decisions about high-level plans and specific implementation details of 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/rearch/papers/ogsa.pdf.

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

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

[15] 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.

[16] 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.

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

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

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

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

[21] 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

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

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

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

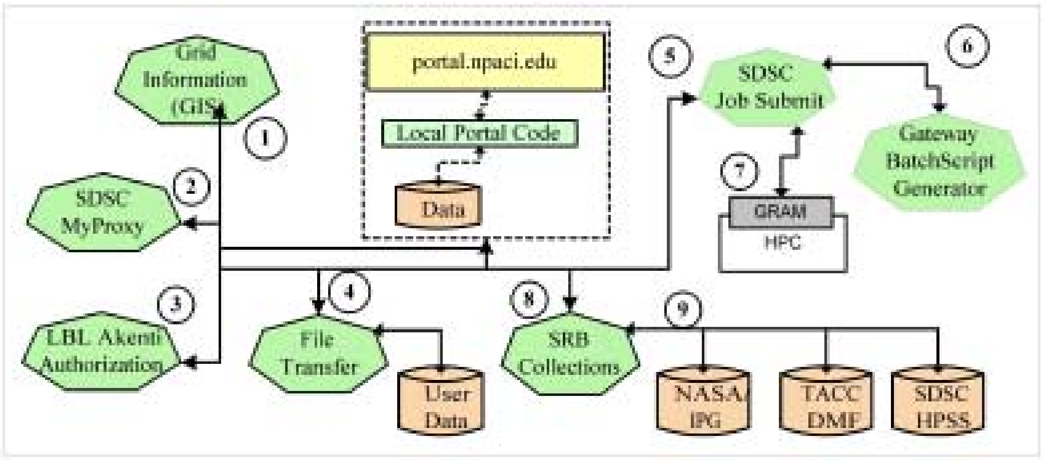

Figure 1: This diagram conceptualizes how multiple portals or applications (represented by the top layer), hosted across multiple webservers or resources, might utilize grid web services. Steps: (1) Texas Portal (TP) authenticates user via MyProxy Web Service; (2) TP checks resource status; (3) TP checks AKENTI authorization; (4) TP pulls files from collection; (5) TP submits 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.

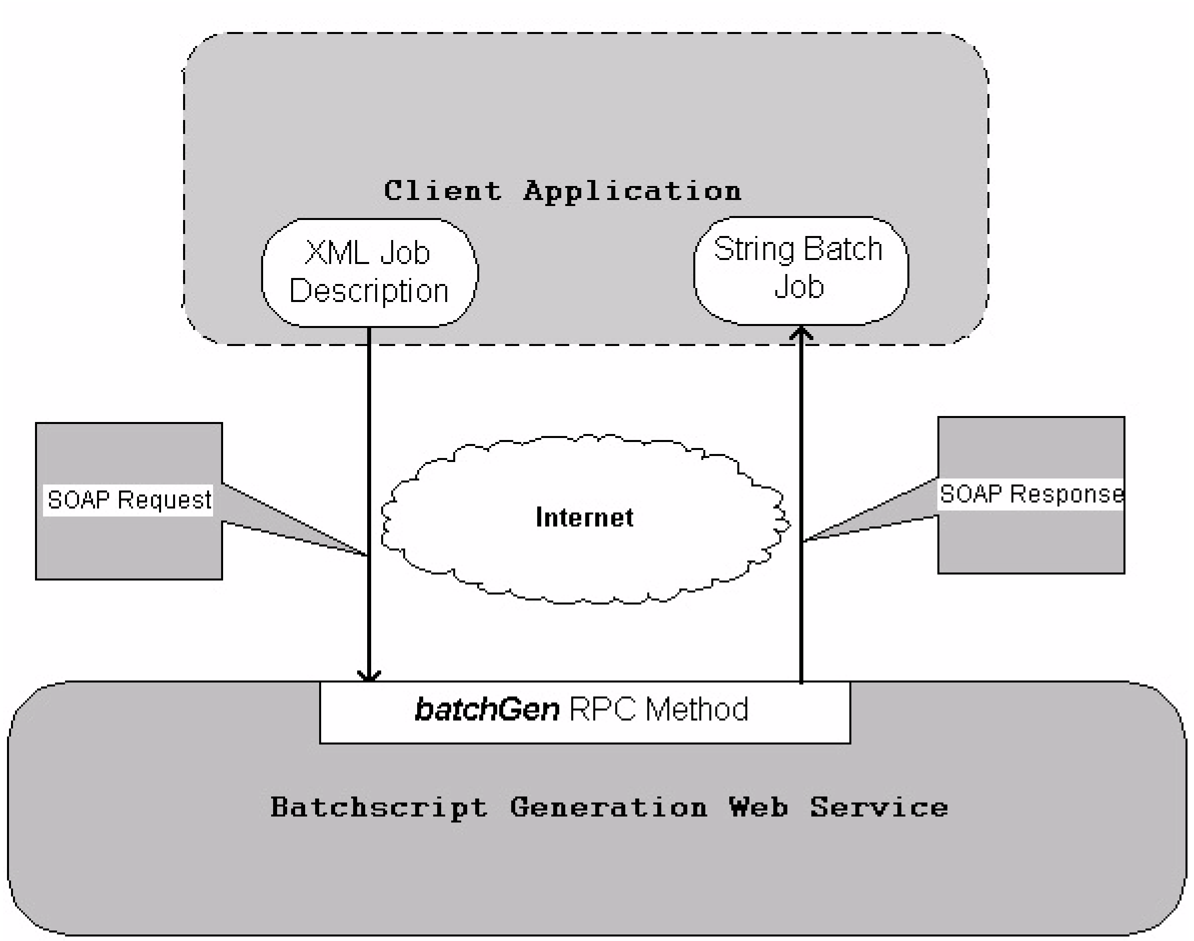

Figure 2: A general overview of how a client interacts with the Batch Script Generator Web Service using SOAP RPC.

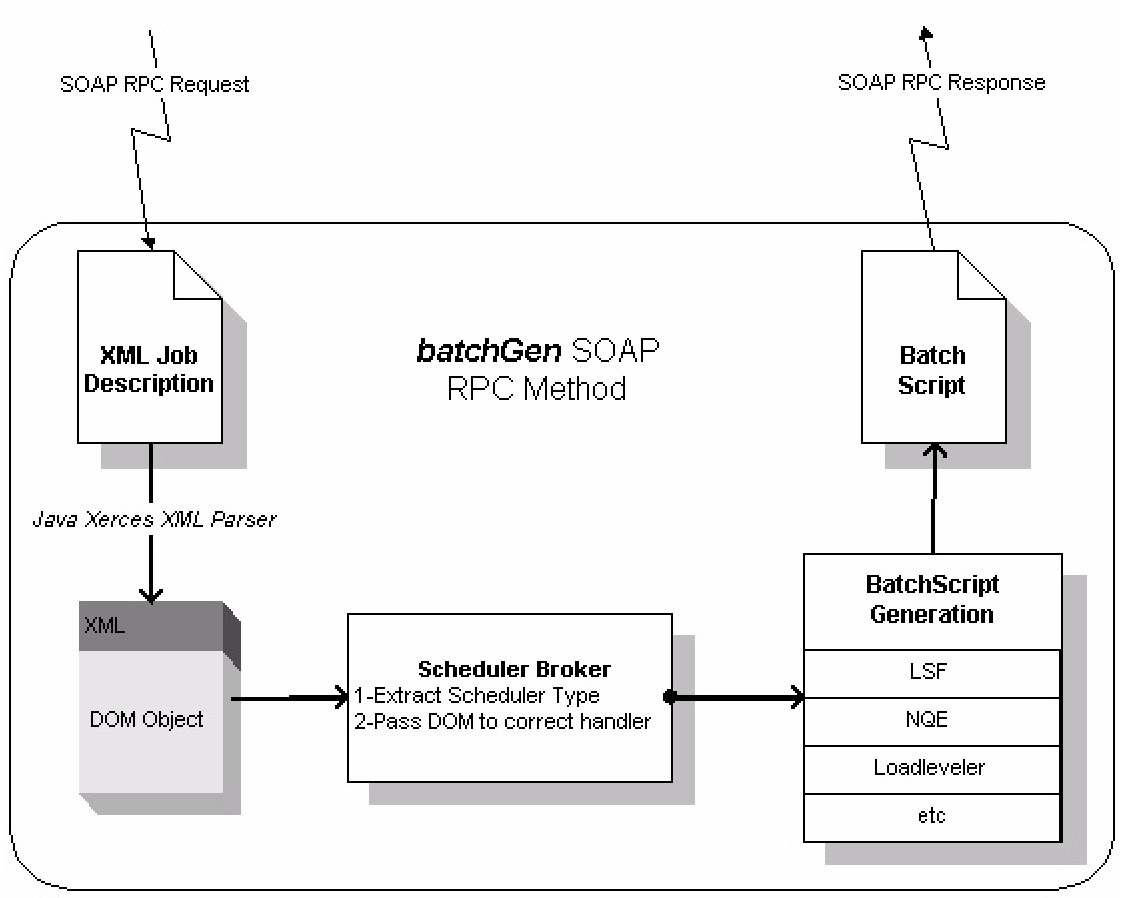

Figure 3: The steps that the ***batchGen*** remote procedure call takes to receive an XML string from a SOAP RPC request and return a string containing a working batch script in a SOAP RPC response.

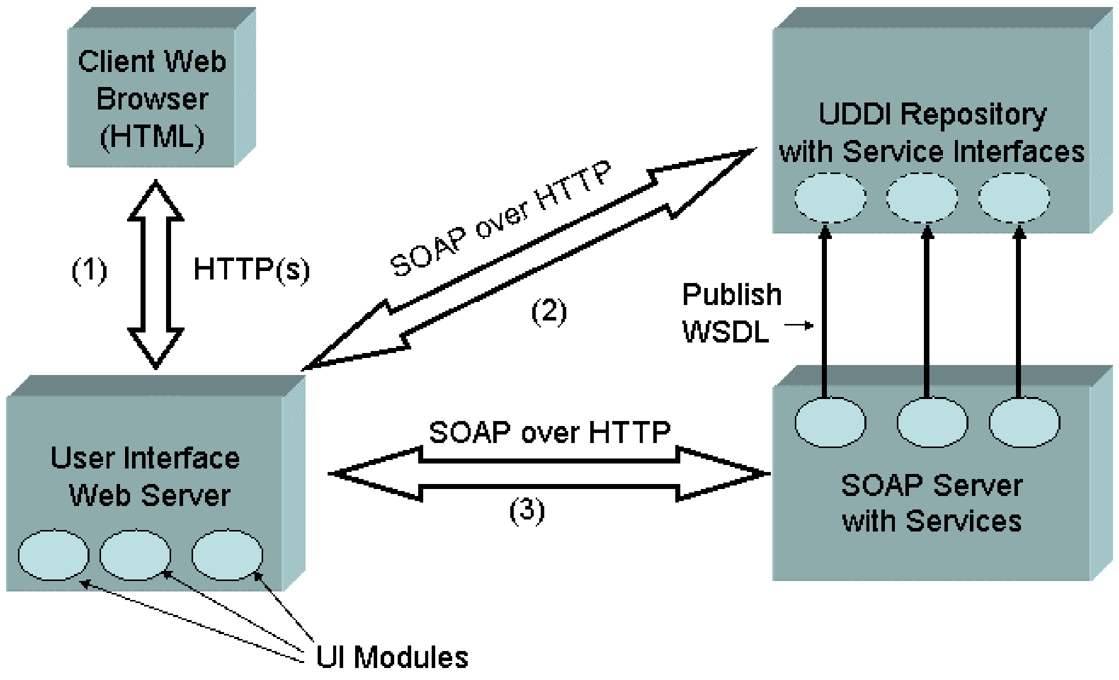
Client Web Browser (HTML)
(1)
HTTP(s)
SOAP over HTTP
(2)
UDDI Repository with Service Interfaces
Publish WSDL
User Interface Web Server
SOAP over HTTP
(3)
SOAP Server with Services
UI Modules

Figure 4: Portal-to-portal interaction is supported through web services. The client interacts with a User Interface server (1), which in turn becomes a client to both the UDDI server (2) and then the selected service provider (3).

APPENDIX A: The WSDL file from the deployed Batch Script Generator Web Service.

```
<?xml version="1.0" encoding="UTF-8" ?>
<definitions  name="BatchScriptService"
    targetNamespace="http://rewind.sdsc.edu/BatchScriptService"
    xmlns="http://schemas.xmlsoap.org/wsdl/"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:tns="http://rewind.sdsc.edu/BatchScriptService"
    xmlns:xsd="http://www.w3.org/1999/XMLSchema">
    <message name="submitRequest">
        <part name="xmljob" type="xsd:string"/>
    </message>
    <message name="submitResponse">
        <part name="response" type="xsd:string"/>
    </message>
    <portType name="BatchScriptServicePortType">
        <operation name="batchGen">
            <output message="tns:submitResponse" name="submitResponse"/>
            <input  message="tns:submitRequest"  name="submitRequest"/>
        </operation>
    </portType>
    <binding name="BatchBinding" type="tns:BatchScriptServicePortType">
        <soap:binding stype="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
        <operation name="batchGen">
            <soap:operation soapAction=""/>
            <input>
                <soap:body use="encoded" namespace="urn:BatchScriptService"
                   encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
            </input>
            <output>
                <soap:body use="encoded" namespace="urn:BatchScriptService"
                   encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
            </output>
        </operation>
    </binding>
    <service name="BatchScriptService">
        <documentation>Supports the loadleveler, lsf, and nqe/nqs schedulers as run at NPACI for parallel mpi
jobs only at this time. You fill out the XML describing the mpijob, and it returns a batch script. See
http://rewind.sdsc.edu/batchscriptservice for more info.</documentation>
        <port binding="BatchBinding" name="BatchPort">
            <soap:address location="http://localhost:8088/soap/servlet/rpcrouter/"/>
        </port>
    </service>
</definitions>
```