An approach to encapsulation of Grid processing within an OGC Web Processing Service

Andrew Woolf (andrew.woolf@stfc.ac.uk), Arif Shaon (arif.shaon@stfc.ac.uk),
STFC e-Science Centre, Rutherford Appleton Laboratory, Oxon OX11 0QX, UK

1 Introduction

The Geospatial community requires an ability to analyse and process geographic datasets using a combination of various spatial software and analytical methods. Increasingly complex geo-processing operations drive a concomitant increase in required computing capability – it is no longer always possible to rely on a desktop GIS to perform spatial analysis of interest, especially where very large datasets are concerned, or the results of complex simulation models need to be integrated.

There is also a need to be able to share processing and analytical capability across the community in an interoperable fashion. A research group having developed a new analysis technique may wish to make it available to the broader research community, or the owner of a high-performance computing resource may wish to allow use by collaborators for a defined purpose.

2 The problem

The recent development of web services has provided a powerful technical solution to simplify the sharing of computational resources and algorithms. Software may be exposed for use through simple web-based protocols, and chains of such services may be orchestrated in value-adding workflows. These 'service-oriented architectures' provide a new paradigm for enabling collaboration.

To date, most web-based geo-processing services take a traditional 'RESTful' 'stateless' view of resources: data are not distinguished from their access service instances, asynchronous interaction sequences are poorly supported, there is no notion of other more generic resource types (e.g. computational) separate from data and service instances. There are no efficient means for handling resource-intensive processes. For example, an application for predicting future global climate change may involve analysing significantly large volumes of past weather data collected over a number of years, and running compute-intensive forecast models. Such a complex application will require a large amount of computational resources, such as disk space, memory and CPU power. Executing this application through a standard web service allows no scheduling capability, and could utilise all available computational resources, resulting in significant delays for other processes.

In addition, typical geo-processing web services do not take a sophisticated approach to security-related issues associated with the underlying processes and datasets. In fact, service providers often rely on ad-hoc access control at the client level to ensure security of the

resources exposed. This leads to non-interoperability in workflows that require interaction between multiple services, each with different security protocols.

The aforementioned limitations are precisely the niche of Grid computing[1]. While Grid architectures and technologies vary, they share in common an attempt to abstract models of stateful resource (data, storage, compute, etc.) within a standardised framework in order to simplify the construction on demand of complex workflows. For instance, using Grid technology, execution of a large-scale parallel process may be accelerated by load-balancing across different Grid nodes. In addition, Grid architectures enable allocation of specific amounts of computational resource, such as disk space, to a particular process. In terms of access control, many Grid architectures employ a common security framework (e.g. the Grid Security Infrastructure (GSI)[2]) to provide secure, robust and interoperable communications. Authentication in the GSI architecture is based on a public-key infrastructure (PKI) with x.509 certificates containing information for identifying users or services.

From this perspective, geoprocessing applications and services should benefit from integration with Grid computing resources and technologies to enable applications to scale out. This goal is reflected in a Memorandum of Understanding (MoU) on collaboration signed in late 2007[3] by the Open Geospatial Consortium (OGC)[4] and the Open Grid Forum (OGF, the principal standards body for Grid computing)[5]. The initial targets of the collaboration are: integration of the OGC Web Processing Service (WPS) with Grid processing and workflow tools, and integration of geospatial catalogues with Grid data movement tools. From a wider perspective, it is intended to promote the use of Grid-based resources within the Geospatial community.

3 General approach

The OGC standard Web Processing Service (WPS)[6] interface is specifically designed to facilitate publishing, discovery and use of geospatial processes in a standardised and thus interoperable manner. In general, the WPS interface may be implemented as a Web Service, with operations for: describing functionality in terms of inputs and outputs, triggering its execution, monitoring its status and finally retrieving its output.

We present here an approach to developing a Grid-enabled WPS by exploiting the similarities and differences between WPS and Grid processing paradigms. Both provide a remote interface for invoking computational processes. WPS functionality includes remote data inputs/outputs, progress monitoring, and asynchronous delivery. Beyond this, Grid job

[1] http://en.wikipedia.org/wiki/Grid_computing

[2] http://www.globus.org/security/overview.html

[3] http://www.opengeospatial.org/pressroom/newsletters/200801#C5

[4] http://www.opengeospatial.org/

[5] http://www.ogf.org/

[6] http://www.opengeospatial.org/standards/wps

submission mechanisms typically add the ability to stage data, and specify required computational resource requirements[7].

Our approach utilises the conceptual overlaps and differences between WPS and the Job Submission Description Language (JSDL)[8] (Figure 1). JSDL is an OGF specification that provides a standardised description of a computational job and the resources (data and computational) it requires. It provides a normative XML schema for describing computational job requirements, including job description and identification, software application to be run, resource requirements (filesystem parameters, disk space, operating system, CPU, etc.), and data staging instructions for input and output. It also includes standardised POSIX extensions for executable filename, command-line arguments, etc. The purpose of JSDL is to enable a standardised specification of job submission to Grid infrastructures irrespective of the back-end schedulers or resource managers used. JSDL job descriptions may be submitted to consuming services like GridSAM[9].

At the specification level, both the WPS and JSDL parameters include process description information (Identifier – WPS, JobIdentification/JobName – JSDL), and process outputs and inputs (DataInputs – WPS, POSIXApplication/Input,Argument,Output, DataStaging/Source – JSDL). These overlapping parameters are sufficient to produce a valid JSDL document. This is fundamental to the approach presented in this paper.

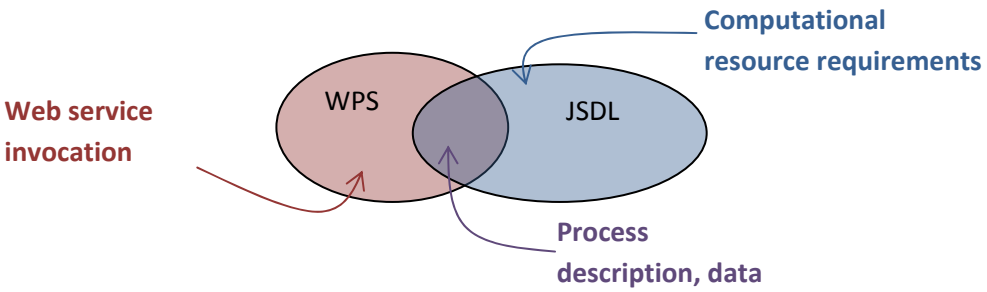

Figure 1: Conceptual overlaps and differences between WPS and JSDL

4 Solution

We have developed a solution that combines the simple web service interface of WPS with the ability of JSDL to specify resource requirements for a large computational process. Specifically, we have developed a WPS-Grid profile to provide a WPS interface to a Grid computing backend. The infrastructure of the Grid enabled WPS developed may be viewed as Grid computing resources encapsulated behind the WPS; the WPS architecture is designed to act as an interface to the resources on the Grid.

[7] Woolf (2006), "Wrappers, portlets, resource-orientation and OGC in Earth-System Science Grids", OGC TC Edinburgh, [Online: http://portal.opengeospatial.org/files/?artifact_id=15966]

[8] http://www.gridforum.org/documents/GFD.56.pdf

[9] A job submission and monitoring web service accepting JSDL input, http://gridsam.sourceforge.net.

4.1 WPS Execute request

A WPS-Grid profile is achieved by enabling encoding of JSDL related information as part of the WPS Execute request defined in the WPS 1.0 specification. The JSDL-related information is then used by the WPS instance for constructing a JSDL document, and submitting the process for execution to a Grid backend through a JSDL-enabled Grid client, such as GridSAM (Figure 2).

The rationale of this approach is to incorporate JSDL handling ability into a WPS while ensuring its conformance to the WPS 1.0 specification. Moreover, this allows the capability of Grid computing to be combined with the simplicity of the WPS interface.

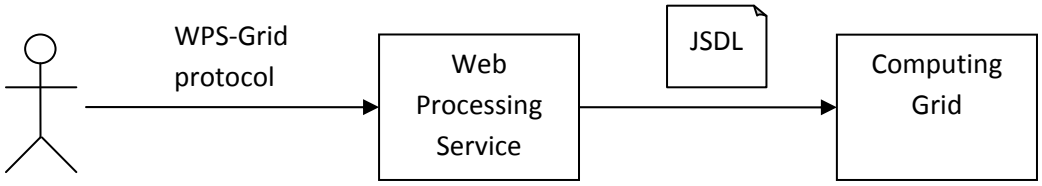

Figure 2: WPS-Grid generates JSDL document for submission to Grid

Two broad options are available for including JSDL-related information within the WPS Execute request:

- *where JSDL-related parameters are regarded as conceptually distinct from other WPS input parameters*: through a specific 'JSDL' parameter as part of the WPS DataInputs
- *where JSDL-related parameters are regarded simply as additional WPS input parameters*: through individual WPS DataInput parameters

We outline in the next two sections these different approaches to specifying JSDL input parameters in a WPS Execute request.

4.1.1 Specific 'JSDL' input parameter

Typically, JSDL-related WPS input parameters will be concerned with specifying resource requirements for executing the process on a computing Grid backend, e.g. required disk space, CPU time, etc. Conceptually, such parameters are not process-related; they only determine whether, and perhaps how quickly, a result is computed. The output results themselves are independent of them. There is a strong case, therefore, to regard the JSDL-related input parameters as different in nature to other process-related WPS input parameters. In this case, a specific 'JSDL' WPS input parameter may be used to specify resource requirements. The use of this special parameter 'JSDL' enables a WPS to distinguish between JSDL parameters and the other (process-related) parameters.

JSDL input parameters may be supplied in a conformant XML format, or as individual elements, as described in the next two sections respectively.

4.1.1.1 Full JSDL document or valid snippet

In this case, a URL may be provided to a complete pre-created JSDL document, or valid snippet (e.g. specifying just the JSDL 'Resource' XML elements), Listing 1.

Advantages of this approach are that JSDL input may be validated against the JSDL schema.

```
http://foo.bar.1/wps?version=1.0.0&request=Execute&service=WPS&Identifier=WeatherGenerator
&DataInput=other_inputs=xxx;JSDL=http://www.foo.com/myfoo.jsdl@Format=text/xml&storeExecu
teResponse=true&status=true
```

Listing 1: WPS Execution request with URL to JSDL document

Alternatively, the JSDL document or snippet may be URL-encoded and included within the WPS request, Listing 2.

```
http://foo.bar.1/wps?version=1.0.0&request=Execute&service=WPS&Identifier=WeatherGenerator
&DataInput=other_inputs=xxx;JSDL=%3CJobDefinition%3E%3CJobDescription%3E%3CResources%3E
%3CTotalDiskSpace%3E%3CExact%3E5%3C%2FExact%3E%3C%2FTotalDiskSpace%3E%3CTotalCPUCo
unt%3E%3CExact%3E1%3C%2FExact%3E%3C%2FTotalCPUCount%3E%3C%2FResources%3E%3C%2FJ
obDescription%3E%3C%2FJobDefinition%3E@Format=text/xml@Schema=http://server/jsdl.xsd&sto
reExecuteResponse=true&status=true
```

Listing 2: WPS Execution request with URL-encoded JSDL snippet

Note that such a JSDL document or document fragment may also be supplied very naturally through a HTTP POST request (Listing 3) facilitating the service invocation in XML-based workflows.

```
<wps:Execute service="WPS" version="1.0.0" xmlns:wps="http://www.opengis.net/wps/1.0.0"
xmlns:ows="http://www.opengis.net/ows/1.1" xmlns:xlink="http://www.w3.org/1999/xlink"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:jsdl="http://schemas.ggf.org/jsdl/2005/11/jsdl"
xsi:schemaLocation="http://www.opengis.net/wps/1.0.0
..\wpsExecute_request.xsd">
        <ows:Identifier>WeatherGenerator</ows:Identifier>
        <wps:DataInputs>
                <wps:Input>
                        ...
                </wps:Input>
                <wps:Input>
                        <ows:Identifier>JSDL</ows:Identifier>
                        <wps:Data>
                                <wps:ComplexData encoding="" schema="http://server/jsdl.xsd">
                                        <jsdl:JobDefinition
                                        xmlns:jsdl="http://schemas.ggf.org/jsdl/2005/11/jsdl"
                                        xmlns:xsi="http://www.w3.org/2001/XMLSchema-
                                        instance">
                                                <jsdl:JobDescription>
                                                        <jsdl:Resources>
                                                                <jsdl:TotalDiskSpace>
                                                                        <jsdl:Exact>5</jsdl:Exact>
                                                                </jsdl:TotalDiskSpace>
                                                                < jsdl:TotalCPUCount>
                                                                        < jsdl:Exact>1</ jsdl:Exact>
                                                                </ jsdl:TotalCPUCount>
                                                        </ jsdl:Resources>
                                                <jsdl:JobDescription/>
                                        </jsdl:JobDefinition>

                                </wps:ComplexData>
                        </wps:Data>
                </wps:Input>
        </wps:DataInputs>
        <wps:ResponseForm>
                <wps:ResponseDocument storeExecuteResponse="true" status="true"/>
        </wps:ResponseForm>
</wps:Execute>
```

Listing 3: WPS Execution request with HTTP-POST JSDL snippet

4.1.1.2 JSDL elements microformat

In this approach, JSDL-related parameters are represented as key-value pairs, with the keyword deriving directly from the relevant JSDL element name. As well, an Xpath-like syntax can be used (replacing '/' with '#') to specify the JSDL element name.

JSDL parameters to be included in the WPS request URL are specified in a microformat as the value of the complex DataInput parameter, 'JSDL' (Listing 4). JSDL key-value pairs are

enclosed within square brackets [], with the WPS data input attribute ('Format') set to the special value 'text/kvp'. (Note that the entire URL must be URL-encoded as per IETFC RFC 1738, although for clarity the examples in the listings below are left unencoded.)

```
http://foo.bar.1/wps?version=1.0.0&request=Execute&service=WPS&Identifier=WeatherGenerator
&DataInput=other_inputs=xxx;JSDL=[TotalDiskSpace=5;TotalCPUCount=1]@Format=text/kvp&store
ExecuteResponse=true&status=true
```

Listing 4: WPS Execution request with microformat for JSDL input parameters

As an alternative to specifying full Xpath JSDL element names, simplified keywords may be specified for predefined common JSDL elements (e.g. 'Source_URI' for the JSDL 'DataStaging/Source/URI' XML element).

Multiple values for a JSDL parameter are separated using "," (Listing 5).

```
http://foo.bar.1/wps?version=1.0.0&request=Execute&service=WPS&Identifier=WeatherGenerator
&DataInput=other_inputs=xxx;JSDL=[DataStaging#Target#URI=http://www.foo.com/myfoo.xml,http
://www.foo.com/myfoo2.xml]@Format=text/kvp&storeExecuteResponse=true&status=true
```

Listing 5: WPS Execution request with multiple JSDL element values

4.1.2 Individual JSDL input parameters

Rather than regarding JSDL parameters as special, they may be regarded as simply additional WPS literal input parameters individually. In this case, they may be included as key-value pairs in a manner similar to that described in section 4.1.1.2 above, except as individual parameters instead of within an aggregate microformat value of the parameter 'JSDL' (Listing 6).

```
http://foo.bar.1/wps?version=1.0.0&request=Execute&service=WPS&Identifier=WeatherGenerator
&DataInput=other_inputs=xxx;TotalDiskSpace=5;TotalCPUCount=1&storeExecuteResponse=true&st
atus=true
```

Listing 6: WPS Execution request with individual JSDL input parameters

4.2 Progress monitoring

This JSDL-enabled WPS is also able to monitor the status of the process on the Grid using the same Grid client as was used for the job submission (e.g. GridSAM).

4.3 Generation of JSDL document

The generation by the WPS server of a conformant and appropriate JSDL document for submission to a Grid backend is implementation-defined. Typically, an internal configuration would assign a default JSDL to a specific WPS process, with user-supplied JSDL parameters over-riding the defaults. As mentioned earlier, in fact a conventional WPS request contains sufficient information to generate a minimal compliant JSDL document.

4.4 Security

Secure access to OGC web services is the subject of considerable ongoing work. Rather than develop a divergent solution, a very lightweight 'placeholder' approach has been taken to security within the WPS-Grid profile. We allow a user to embed MyProxy[10] parameters (host, port, username, password) within the WPS request using the microformat encoding mechanism mentioned earlier ($4.1.1.2), as the value of a special DataInput parameter, "**MyProxy**" (Listing 7). These are processed by the WPS server and used at job submission for authentication.

```
http://foo.bar.1/wps?version=1.0.0&request=Execute&service=WPS&Identifier=WeatherGenerator
&DataInput=otherinput@datatype=string;JSDL=[TotalDiskSpace=5]@Format=text/kvp;MyProxy=[us
ername=xxxx;password=xxxx;host=xxx;port=xxxx]@Format=text/kvp&storeExecuteRepose=true&sta
tus=true
```

Listing 7: WPS Execution request with MyProxy input parameters

Including sensitive information, such as a user's MyProxy credential in a WPS Execute request does pose security risks. Therefore, this approach assumes that implementation of this WPS-JSDL profile will incorporate necessary security protocol (e.g. SSL) to ensure the security of MyProxy information in the WPS Process Execution Get Request.

5 Implementation

A Grid-enabled WPS service has been implemented within the OGC's OWS-6 activity as a proof-of-concept using an atmospheric particle-tracing 'trajectory service'[11] and deployment on the UK National Grid Service[12], which is explicitly mentioned as a target Grid infrastructure in the OGC-OGF MoU. The implementation of this Grid-enabled WPS is fully complaint with the OGC WPS specification 1.0.0, and uses Python as the underlying programming language and Pylons[13] as the integrated web development framework. At an architectural level, it depends on a number of other services and components (Figure 3) to enable invocation and controlling of Grid-based processes through standard WPS requests.

[10] http://grid.ncsa.uiuc.edu/myproxy/

[11] http://badc.nerc.ac.uk/community/trajectory/

[12] http://www.ngs.ac.uk

[13] http://pylonshq.com/

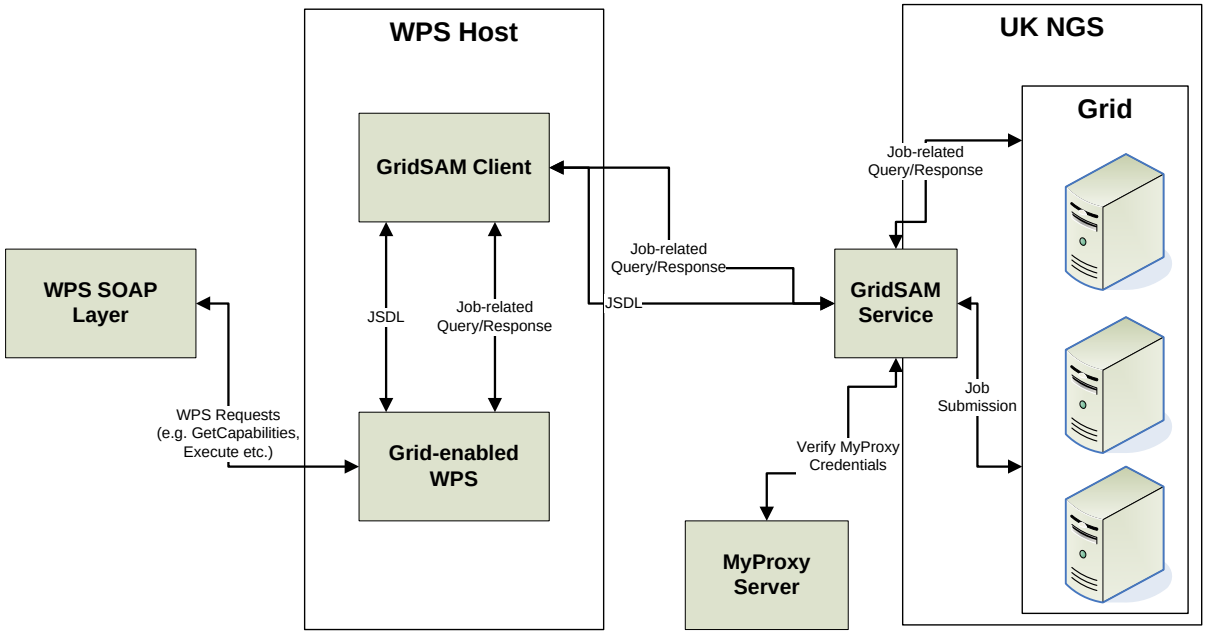

Figure 3: An Architectural View of the Grid-enabled WPS

- **GridSAM Client**

The Grid-enabled WPS liaises with a GridSAM client deployed on the machine that is also hosting the WPS to submit jobs to the UK NGS. The GridSAM client receives the WPS-generated JSDL job description and submits it to the GridSAM server. The WPS also interacts with the GridSAM client for checking the status of a job on the Grid; the job status result is used by the WPS to produce and record WPS-specific process status (for example ProcessCompleted, ProcessSucceeded, etc.)[14].

- **GridSAM Service**

The aforementioned GridSAM client sends a JSDL job description to the UK NGS GridSAM service (installed on the Oxford node of NGS), which then executes the requested job on the NGS. This service also receives queries (for example job status check, etc.) from the GridSAM client and responds to them by liaising with the Grid with which it is associated. The GridSAM service is also responsible for retrieving and verifying users' Grid certificates using the MyProxy credentials received along with the JSDL job description from the GridSAM client.

- **WPS SOAP / Proxy Layer**

There is also a WPS SOAP/Proxy layer that provides a SOAP wrapper outside the network firewall to proxy SOAP requests through HTTP GET requests to the Grid-enabled WPS. The SOAP interface provided by the SOAP wrapper conforms to the WPS specification 1.0.0. In addition, this layer is also used for forwarding other HTTP GET

[14] Standard GridSAM status codes are mapped onto WPS status codes.

requests to the Grid-enabled WPS, such as request for downloading process output and status check request for a process.

6 Conclusions & Future Work

Grid computing provides efficient means of executing resource-intensive processes, and thus should be beneficial to the Geospatial community that has an increasing need for performing highly complex geo-processing operations involving large geographical datasets. The WPS-Grid profiling approach presented in this paper demonstrates the feasibility of integrating Grid capability within standard geo-processing web services, such as the Web Processing Service. However, there is considerable scope for further work in this area, for example, refactoring OGC web services around a resource-oriented view of data within the Grid infrastructure using technologies such as the Web Services Resource Framework (WSRF)[15]. It may also be useful to look into replacing GridSAM as the grid middleware for consuming JSDL documents and job management on the Grid, with any middleware conforming to the 'HPC Basic Profile'[16], subject to associated restrictions on the allowable scope of JSDL. Moreover, future evolution of the WPS-Grid profile will need to align with best practice for adding security to other OGC service interfaces, possibly by harmonising the Grid Security Infrastructure (GSI) and the current OGC approaches to security.

[15] http://www.globus.org/wsrf/

[16] http://www.ogf.org/documents/GFD.114.pdf