Useful extensions to the OGC Web Processing Service based on a Python client/server implementation

Ag Stephens (ag.stephens@stfc.ac.uk), Stephen Pascoe (stephen.pascoe@stfc.ac.uk) and Philip Kershaw (philip.kershaw@stfc.ac.uk).
Science & Technology Facilities Council, Didcot, Oxon, United Kingdom.

1. INTRODUCTION TO WPS

Web Processing Service (WPS) is an Open Geospatial Consortium (OGC) standard currently at version 1.0.0 (OGC, 2011). Compared to other OGC services WPS is still relatively new but has the potential to act as an important component in the web service landscape. The WPS interface consists of three main methods: (i) **GetCapabilities** - an XML document outlining the processes deployed within the WPS; (ii) **DescribeProcess** - details of the inputs/outputs for a given process; (iii) **Execute** - a request to execute a job accompanied by the process ID and a set of input parameters.

WPS provides a generic framework suitable for implementing web services that cannot be delivered by the commonly adopted Web Mapping Service (WMS), Coverage Service (WCS) and Feature Service (WFS). Its capability for asynchronous job management makes WPS a good candidate for managing offline processes, scheduling and service-chaining when handling large requests lasting minutes, hours or even days.

2. THE COWS WEB PROCESSING SERVICE

The Centre for Environmental Data Archival (CEDA: http://ceda.ac.uk) manages holdings of atmospheric and earth science data approaching a petabyte in volume. This archive requires sophisticated tools to allow users to discover and request the data they need. Requests often result in many gigabytes of output. Having already developed the CEDA OGC Web Services (COWS: http://cows.ceda.ac.uk) framework it was a logical progression to build the COWS WPS in the same Python-based software stack.

The COWS WPS framework (http://cows.ceda.ac.uk/cows_wps.html) builds upon a host of open source tools to allow fast prototyping and deployment of web services as WPS "processes". New processes can be added by the simple addition of a configuration file, describing inputs and outputs, and a Python wrapper module that interfaces the processing code with the WPS framework. The framework integrates with CEDA Security middleware (Kershaw *et al*, 2011), enabling calls to the service, and more importantly to a specific process, to be intercepted and checked against an XML-based authorisation policy. Dual OpenID and PKI-based authentication schemes are supported.

The COWS WPS is operationally deployed behind the UK Climate Projections User Interface (http://ukclimateprojections-ui.defra.gov.uk) and within CEDA. Deployment at other institutions is expected as part of the upcoming G8-funded ExArch project.

Features of WPS and useful extensions

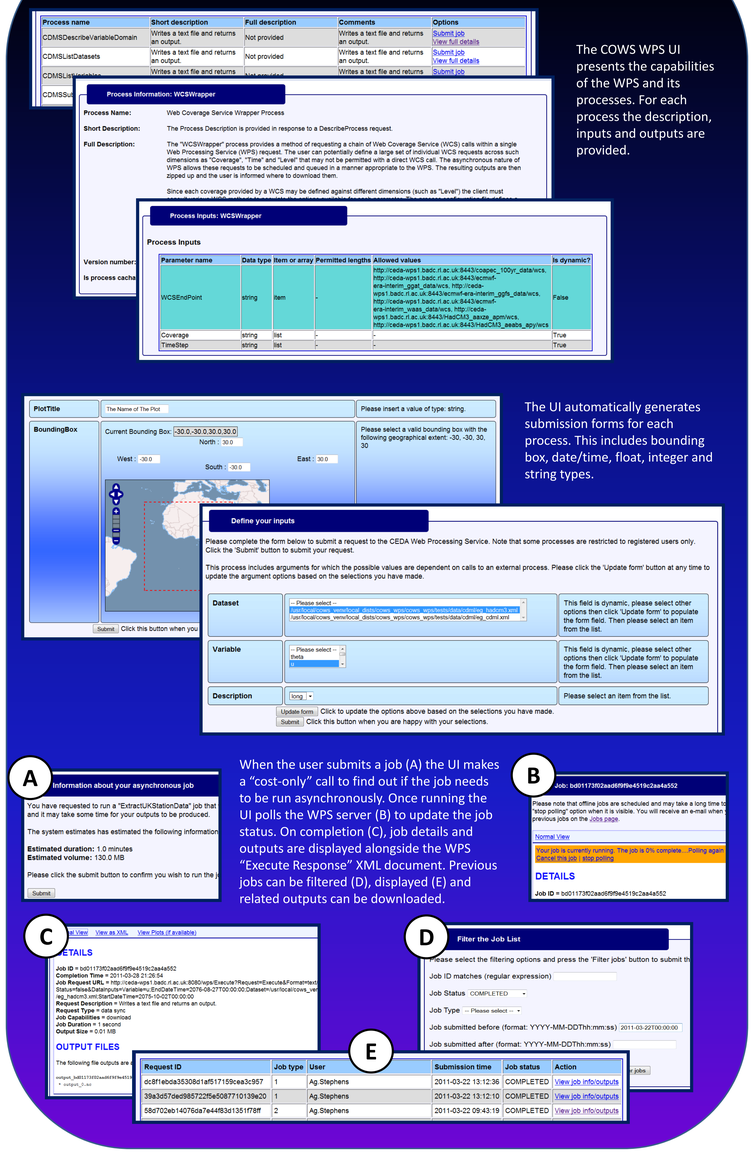
WPS v1.0.0 features

- A web service interface, using POST or GET.
- Asynchronous reporting and control of jobs.
- A defined XML interface for responses, including exceptions.
- A common format for passing arguments to the server.
- Job status interrogation.

Extensions in the COWS WPS

- Ability to inform users via e-mail when a job has completed (or failed).
- Integration with CEDA Security middleware.
- Coupling to a web-client to auto-generate process submission forms and interrogate current and previous jobs.
- A configuration mechanism to add new processes via a simple configuration file and a single Python interface module.
- Connection to a parallelised processing back-end using Sun Grid Engine.
- Make a "cost-only" request to estimate the job size and duration without executing the process.
- Zip up output files and report details of their contents in the XML response.

The COWS WPS – viewed from the client



The COWS WPS UI presents the capabilities of the WPS and its processes. For each process the description, inputs and outputs are provided.

The UI automatically generates submission forms for each process. This includes bounding box, date/time, float, integer and string types.

When the user submits a job (A) the UI makes a "cost-only" call to find out if the job needs to be run asynchronously. Once running the UI polls the WPS server (B) to update the job status. On completion (C), job details and outputs are displayed alongside the WPS "Execute Response" XML document. Previous jobs can be filtered (D), displayed (E) and related outputs can be downloaded.

3. WPS CLIENT: COWS WPS UI

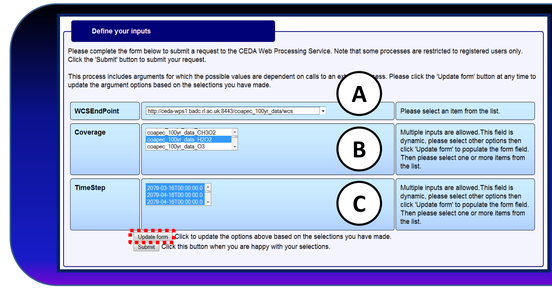
A web-client has been recently developed that is currently tightly-coupled to the COWS WPS server implementation. The client, the COWS WPS UI, presents human-readable HTML views of the "Capabilities" and "Process Description" XML documents. Job management tools enable searching and filtering old jobs, viewing outputs, and cancelling a job currently running.

One of the most powerful aspects of the UI is its ability to automatically generate a submission form for a given process based on the input parameters defined in the configuration file. This web-form can generate an interactive map for bounding box selection and appropriate input elements for other typed selections such as date-time and integer. By extending the list of known types beyond those listed in the WPS specification a rich client application can be produced. For general development of web services this provides rapid deployment of web-clients without the need to produce bespoke interfaces for each new process. Put simply, the client/server coupling provides a framework to make any piece of useful code available as a secured web service with a web front-end for managing user interactions.

4. SERVICE-CHAINING WITH WPS

The WPS 1.0.0 specification assumes that all input parameters to a given process are static. Experimental work to investigate service-chaining with OGC WCS has identified that a more flexible approach could be useful.

Many WCS implementations do not have the capacity for asynchronous interactions and may therefore require repeated access for batch requests. Placing the WPS in front of a WCS allows the WPS to manage a set of WCS calls within a single request. In this configuration, the WCS parameters reflected in its "GetCapabilities" and "DescribeCoverage" responses are made visible to the WPS Process Descriptions (returned from requests to the "DescribeProcess" method) by the use of "dynamic parameters". These dynamic parameters are not available to the WPS client until certain service calls have been made. Such parameters are not static and therefore cannot be represented in the "DescribeProcess" document returned from a standard WPS.



Service-chaining example

In this example the COWS WPS UI generates a form for the "WCSWrapper" process. When the page loads the user can view the (A) "WCSEndPoint" options. These are listed in the process configuration file. Once a WCSEndPoint is selected the user clicks the "Update form" button to load the available "Coverages" (B). The Coverages are extracted from a call to "GetCapabilities" at the WCSEndPoint. When the user selects a Coverage and clicks "Update form" the "TimeSteps" (C) are loaded. Once all selections have been made the user can click "Submit" to make a request to the WPS. The WPS then calls the WCS "GetCoverage" method at the WCSEndPoint with appropriate Coverage and TimeStep parameters.

5. CONCLUSIONS

Due to its asynchronous job management and generic capability for deploying processes WPS has the potential to become a major part of the future OGC service landscape. The COWS WPS is a Python-based tightly-coupled client/server implementation that builds on the standard to provide a useful and flexible framework for rapid development and deployment of new processes. Investigations into service-chaining have shown that future versions of the standard may benefit from a more dynamic approach to defining input parameters.

References:

The Open Geospatial Consortium (OGC) Web Processing Service, (2011). Available from: http://www.opengeospatial.org/standards/wps
Kershaw, Philip *et* al., (2011). *Building a Web Based Environment for the Intercomparison of Distributed Environmental Science Data: Challenges in Access Control and Security*. EGU2011-7579, Poster XL234. (Visible: Thurs 7 Apr from 17:30 - 19:00).