UncertWeb Processing Service: Making models easier to access on the web

Richard Jones, Dan Cornford, and Lucy Bastin

Computer Science Research Group

Aston University

Birmingham, UK

Abstract

Models are central tools for modern scientists and decision makers, and there are many existing frameworks to support their creation, execution and composition. Many frameworks are based on proprietary interfaces, and do not lend themselves to the integration of models from diverse disciplines. Web based systems, or systems based on web services, such as Taverna and Kepler, allow composition of models based on standard web service technologies. At the same time the Open Geospatial Consortium has been developing their own service stack, which includes the Web Processing Service, designed to facilitate the executing of geospatial processing - including complex environmental models. The current Open Geospatial Consortium service stack employs Extensible Markup Language as a default data exchange standard, and widely-used encodings such as JavaScript Object Notation can often only be used when incorporated with Extensible Markup Language. Similarly, no successful engagement of the Web Processing Service standard with the well-supported technologies of Simple Object Access Protocol and Web Services Description Language has been seen. In this paper we propose a pure Simple Object Access Protocol/Web Services Description Language processing service which addresses some of the issues with the Web Processing Service specification and brings us closer to achieving a degree of interoperability between geospatial models, and thus realising the vision of a useful 'model web'.

1 Introduction

The development and use of models has been key to the successes of science in improving our understanding of, predictions for and decisions about the world around us. Increasingly models are being used in a policy context, where it is often necessary to integrate models from different domains to form a more holistic picture of the overall system. Such model integration raises several challenges, both philosophical and practical. These are considered in Bastin et al. (2011) which focusses strongly on the issue of uncertainty management. Here we focus more closely on the practical issues which arise from exposing geospatial models on the web (Geller and Turner, 2007) to facilitate their access and subsequent integration.

Many models, particularly those of environmental systems, are geospatial in nature. The geospatial community has been gradually making a transition from standalone applications to service-oriented architectures. This transition has been driven by the Open Geospatial Consortium (OGC), an organisation responsible for defining a number of standards for web service interfaces and data representation. These interfaces include the Web Feature Service (WFS) for serving geographical features, the Web Coverage Service (WCS) for serving raster coverages, and the Web Processing Service (WPS) for exposing geospatial processing functionality (including complex environmental models) over the web. In addition to these service standards, it is necessary to support a set of information models to represent geospatial objects and observations. To achieve this, the Geography Markup Language (GML) (OGC 07-036, 2007) and Observations & Measurements (O&M) (OGC 10-025r1, 2011) conceptual models and schemas are defined. The use of such standardised interfaces and information models can lead to increased interoperability, as organisations are able to interact and share data in a common manner.

Outside the OGC community, there are a number of widely used specifications for web services (Pautasso et al., 2008). These include Simple Object Access Protocol (SOAP) and Web Services Description Language (WSDL), two complementary standards for exchanging messages between services and describing those services. These specifications were developed within the World Wide Web Consortium (W3C) and are extensively used across the web. Since the WPS shares a similar purpose to these standards, the lack of in-

tegration between WPS and SOAP/WSDL requires further exploration. The aim of this paper is to review the objectives and nature of the WPS specification and to demonstrate a SOAP/WSDL based implementation for exposing models to the web, which fulfils the functional requirements of a WPS, but is compatible with existing well-supported web technologies, and addresses some of the limitations of the WPS standard. This framework is being used within the UncertWeb project[1], and can integrate with existing modelling frameworks, such as Taverna and Kepler, which are compatible with more widely adopted web standards.

The paper firstly introduces web service technologies, including the WPS and those from outside the OGC community and explains the motivation for developing an alternative solution to the WPS. The design and implementation of this alternative, which integrates geospatial processes with existing standards, is then detailed. A use case based on crop yield modelling is employed to demonstrate the practicality of the framework in a real world example, this also being used as motivation throughout the paper. Findings are then evaluated and conclusions are given.

2 Web service technologies

Web service technologies have been developing for many years, and continue to evolve. The WPS specification was developed after SOAP/WSDL technologies were standardised, and these are both reviewed below. Other architectural patterns such as RESTful approaches are considered, and more recent trends in web based systems are identified.

2.1 WPS

Within the geospatial community, the OGC WPS standard is widely used for exposing processes on the web. The standard aims to facilitate the publishing, discovery, and client binding of geospatial processes (OGC 05-007r7, 2007). It defines an Extensible Markup Language (XML) based client-server communication protocol with three main operations:

[1]http://www.uncertweb.org/

GetCapabilities to return metadata about the service, including the processes offered;

DescribeProcess to retrieve information about a specific process, including its inputs and outputs;

Execute which includes any required inputs and parameters, and returns the output(s) of the process.

There are benefits of using the specification, such as having standardised mechanisms for passing data by reference, and requesting asynchronous execution of a process. Without these mechanisms, interoperability would be reduced as each service may have a different approach to providing these features. A more in depth review of the limitations of the current WPS specification is given in Section 3.3.

2.2 SOAP/WSDL

The W3C defines two standards which are considered to be essential technologies for deploying and describing a web service: SOAP and WSDL (Louridas, 2008). SOAP is an XML-based protocol for exchanging messages between systems, commonly used in network services (Box et al., 2000). WSDL is an XML specification for describing these network services (Christensen et al., 2001). Compared to the relatively new WPS standard, SOAP and WSDL both have the associated benefits of being mature specifications, with a wide range of tool and community support.

Using SOAP gives the advantage of a standard header element, which can be used for transferring infrastructure information such as security, reliability and routing. A standard fault element is also specified, making a clear distinction between a normal response from a process or an exception. WSDL focuses on providing a technical description of a service, listing operations, inputs, outputs, and payload types (OGC 08-009r1, 2008), with the aim of allowing services to describe themselves.

2.3 Representational State Transfer (REST)

REST is an alternative architectural style to SOAP/WSDL introduced by Fielding (2000). RESTful approaches to web services are based on the transfer of representations of re-

sources, rather than a focus on operation calls as with other architectures. RESTful services are increasingly popular on the web due to their simplicity and full use of HTTP verbs. A common usage pattern is GET for retrieving, POST for creating, and PUT for updating resources.

The geospatial community has made a move to adopt RESTful services for data, with the OGC Web Map Tile Service (WMTS) standard detailing a full REST interface (OGC 07-057r7, 2010). Similar interfaces for the WCS have also been proposed (Mazzetti et al., 2009). This may be caused by the inherent ease of mapping data objects to resource URIs, and operations on those data objects to HTTP verbs. In contrast, existing examples of REST processing services are limited. Foerster et al. (2011) describes an implementation of a RESTful web processing service. Although it demonstrates how such a service can be exposed using REST, it remains to be seen whether this approach offers benefits for a processing service over other architectures. In the approach proposed in Foerster et al. (2011) metadata and process execution are exposed as resources which can be accessed using HTTP verbs (a simple approach, but one which is inconsistent with the OGC model). The request/response nature of processes means that the full range of HTTP verbs is only utilised when dealing with asynchronous processing jobs; a job is created with an HTTP POST request and queried with subsequent GET requests which ultimately, return the result when it becomes available. The job and its results can then be removed from the service using an HTTP DELETE request. It remains unclear, however, how inputs are specified and how processes themselves should be represented as resources — a matter identified by the paper as a key concept for REST. The example poses the question of whether the REST architecture is appropriate for every use case.

2.4 JavaScript Object Notation (JSON)

There has been a recent shift towards web browser based applications. These have many benefits over traditional desktop applications, such as not requiring local installation, and being able to instantly update without the user being aware. Traditionally, XML has been the primary data-interchange format used in web service architectures. However, the shift towards browser based applications has driven an increase in popularity of JSON,

a lightweight data-interchange format (Crockford, 2006).

JSON is based on a subset of JavaScript, the client-side programming language which forms much of the basis for these web applications. While JSON can be natively parsed into JavaScript objects, XML requires specific parsers for each XML-based language, making it more complex to process to a web application developer compared with JSON. This fact is becoming clear after the popular websites Twitter and Foursquare deprecated parts of their XML Application Programming Interfaces (APIs) in late 2010.

3 Motivation

3.1 UncertWeb

UncertWeb is an EC funded project aiming to 'uncertainty-enable' the model web (Geller and Turner, 2007). The model web is a concept based on the notion that models, as well as data resources, can be published to the web, discovered and invoked within complex web-based workflows (Geller and Turner, 2007). UncertWeb aims to develop mechanisms, standards, tools, and test-beds for uncertainty propagation in web service workflows. UncertWeb has several use cases, including a scenario based on crop and yield modelling in the UK, working with the Food and Environment Research Agency (FERA). As the model web is currently only a concept, UncertWeb has the challenge of building these components, in addition to uncertainty-enabling them.

Within the model web concept, and thus within UncertWeb, a model is defined as a process which takes a set of inputs, and produces a set of outputs. However this naive view of models alone is not sufficient to fully describe a geospatial model of a real physical system. A model also embodies knowledge, typically contained within the experience and expertise of the model owner / creator, or relevant community, which is very difficult to fully capture in an automated manner. Such knowledge includes information about the appropriateness of running the model in different configurations and at different resolutions, about the stability of the model at different parameter settings, and about its applicability within certain usage scenarios. We speculate that currently such information cannot be well expressed in a form which can be understood and reasoned

with by machines, and thus we posit that this contextual information should be provided to model users in textual, unstructured form. However, models do normally have well defined requirements on their inputs, and specification of their outputs in terms of the data types supported. These inputs and outputs are often simple scalars, vectors and arrays, but can also have more complex structures such as vector spatial models. Many current models are rather inflexible in this respect, requiring specialised domain- and even model-specific data structures to be provided as inputs. The semantics of describing model inputs therefore remains a challenging problem. Models also come in many flavours in terms of computational complexity, from the highly computationally-demanding numerical weather prediction and climate models, to very simple statistical models of input/output relationships for example an empirical crop yeild model that uses only soil and weather data, learned on historical data, rather than physically meaningful processes and mechanisms.

Building the model web presents many challenges for interoperability. The vision is to have a number of models from different providers and organisations, and for a user to be able to select one of those models and use it, either by itself or in a workflow. For this to be achievable, models must be able to communicate with each other in a standard way, meaning that a set of suitable information models for representing data, and service interfaces for defining interactions, is required. One solution being investigated in the UncertWeb project is the Composition as a Service (CaaS) approach (Bigagli et al., 2011), where, instead of requiring a single service interface specification, the CaaS component can mediate between services with different interfaces. The CaaS is envisaged as allowing a user to specify in an implementation–independent language which services they wish to use, and how they want the services to be orchestrated in the workflow. The CaaS is then responsible, through the use of adapters for each different (data and interface) type, for building requests and parsing responses to each service. The services described in this paper will be used alongside traditional WPS to evaluate the mediation and workflow orchestration capabilities of the UncertWeb CaaS.

3.2 Processing requirements for the model web

We have identified several desirable features for a web-based processing service:

- Easy of use. The services (models) will be used by modellers who are not necessarily familiar with the specifications and standards used within the geospatial community.
- Client / library support. If existing clients and supporting libraries are available, the barrier to use is much lower.
- Process development. Processes will potentially be developed and deployed by the model owner who may not be familiar with geospatial interface and encoding standards.
- Workflow support. The UncertWeb project is heavily reliant on model workflows, making compatibility with associated standards and tools important.
- Reference passing. When dealing with large geospatial datasets, passing resources by reference is required.
- Asynchronous processing. Geospatial processes can take hours to complete, making asynchronous execution desirable.
- Dicovery and usage. For describing processes, inputs, and outputs to enable the potential for discovery and automated service consumption.

Although this list of requirements has been gathered specifically for the UncertWeb project, they are relevant to the majority of cases of exposing models on the web. These requirements will form the basis of our assessment for the current version of the WPS, and the new processing service described in this paper.

3.3 WPS shortcomings

While the WPS specification has valuable features such as standard ways of controlling process execution and providing metadata, there are several drawbacks to the specification. Many of these drawbacks increase complexity for the service consumer, presenting

an immediate barrier to use. The interface provided by a WPS is designed to be generic, and allows the publishing of any process, geospatial or otherwise. A process can therefore use any data type for an input or output. If a client fully supports a data type, it should be possible to parse, modify, generate, extract and in many cases visualise the information carried by the data. Supporting any data type makes client development extremely difficult, as it will only be possible to fully support a limited number of data types, and subsequently a limited number of services. It is argued within the WPS specification that profiles should be developed to support use, and that the WPS is only an abstract model of a web service. However, profiles require consumers to develop specific clients, which negates some of the interoperability benefits of using such a specification. In practice there are very few existing profiles, and no mechanism for managing these.

Usability issues are even more prevalent when considering the description mechanism for model inputs and outputs. In the process description returned by a WPS, a type can only be described with a MIME reference and an XML schema URL. This leaves a lot of ambiguity when consuming a service, since an XML schema could, and typically does, contain several elements. Some current WPS implementations rely on using the URL to handle data, but do not consider that the same type may have different schema locations. This also poses a problem when using a large schema, such as GML version 3. Many services advertise processes where the input must be described using a GML 3 schema URL, but in reality the service will not be able to understand every element in that schema.

As a concrete example, consider exposing an agricultural field-use simulator as a web service. In its simplest form the model has three inputs: the field areas as real valued measurements, field type classification as text observations, and a crop transition probability matrix, which arises because of the Markov structure of the crop transition probabilities assumed in the model. Our use case model workflow contains just such a field-use simulator, which will be further discussed in Section 5. With the WPS, we are only able to specify that the former two inputs conform to the O&M schema, which encompasses observations of a variety of types, and itself is a conceptual model for which a profile must be developed. The third input is more complex still, but is typical of

the sorts of inputs that are needed by models, which are often quite specialised to their domain of application. With the current WPS standard a user has three options for determining the exact data type to use:

1. Prior knowledge (that is, they know what the model needs before even seeing the model description);

2. Use of the plain text input descriptions, which are generally not machine readable;

3. Sending a request and hoping that, if incompatible, the service sends information to help correct the error, although there is nothing in the service description to suggest that this should occur.

Client development is also restricted by the tool and library support available for the WPS. Compared to the more widely used web service standards, such as SOAP and WSDL, support for WPS is limited. Although some tools are available, they lack functionality compared to equivalents outside the geospatial world. Workflow standards and software such as the Business Process Execution Language (BPEL), Taverna and Kepler are all compatible with services described using WSDL, and do not contain any built-in support for WPS.

With the WPS specification being this generic, misuse is possible and common. Under the specification, it is valid for a service owner to expose a process with a data type simply described as being any XML. Whilst the owners of the service know what that input should consist of, interoperability with external users is significantly affected. A consumer should be able to send a valid request to the service without having to rely on additional information provided outside of the interface itself. For a service owner wishing to only expose a process internally, it is unclear whether the advantages of implementing the WPS specification outweigh the effort required. There is a common misconception that merely complying with a standard ensures immediate interoperability of services. However, interoperability also requires common data types (information models), concrete descriptions of service interfaces, and the tools to be able to support these information models and services.

The latest version of the WPS specification is currently at 1.0, and work on version 2.0 is taking place within the OGC. Although initial plans for version 2.0 aim to solve some of the issues mentioned here (OGC 09-184, 2009), the standardisation process is lengthy, making it impractical to rely on implementing an updated standard within the timescales of the UncertWeb project.

3.4 WPS integration with other technologies

A key motivation for developing an alternative service interface was the lack of software and community support for WPS. Using tools such as Apache Axis and Microsoft Visual Studio, it is possible to quickly deploy a usable SOAP/WSDL web service from existing code. These tools are also able to generate code from WSDL documents, making it easier to integrate applications with web services. A large amount of workflow software is compatible with, and in some cases relies on, services described using WSDL.

WPS adoption by the open source community in the geodomain has been partially successful. Service frameworks such as 52N WPS[2], PyWPS[3], and the ZOO Project[4] have been developed to implement the standard, and GRASS GIS[5] has support for generating WPS compliant process descriptions. However, client support is still limited. For example, uDig[6] may be able to send requests to a WPS with the appropriate plugin, but it can only support a very limited number of formats for the inputs and outputs of these services. If a format is unsupported, the user is unable to use the process. A uDig plugin for orchestrating workflows has also been developed (Schäffer and Foerster, 2008), but is restricted to only WPS instances and shares the same data format issues as the client plugin.

The popularity of SOAP/WSDL services has driven several efforts to integrate these standards with WPS. Although detail regarding WSDL implementation was originally missing from the specification, the OGC later detailed possible solutions (OGC 08-009r1,

[2]http://52north.org/communities/geoprocessing/

[3]http://pywps.wald.intevation.org/

[4]http://www.zoo-project.org/

[5]http://grass.fbk.eu/

[6]http://udig.refractions.net/

2008). These are based on providing a WSDL document either to describe the whole WPS instance, or on a per-process basis. Due to the generic nature of the WPS, each instance can only be described by WSDL in an abstract manner. The schema for the *Execute* request document only specifies that a process has a number of inputs and outputs with any data type. No mechanisms are provided for obtaining a concrete schema for an individual process. Instead, the *DescribeProcess* operation defined by the WPS specification must be used to discover the exact input formats required for a process. Due to the complexity of this usage pattern, the benefits of using WSDL are drastically reduced. If the field-use simulator example is exposed as a WPS and then included in a Taverna workflow, a WSDL document is required. When using a generic WSDL document, a user only knows that a number of inputs are required, and is unaware, for example, that there are three inputs and that these inputs consist of measurements and a transition matrix. To discover more information about the process, a *DescribeProcess* request must be separately issued, and the *Execute* request document must be built manually. A concrete WSDL document, by contrast, can be imported and the required inputs and data types immediately identified. Taverna then provides graphical interfaces for setting these inputs.

When creating a WSDL document or schema on a per-process basis, request and response messages which do not validate against WPS schemas must be defined. This is required because the WPS schemas only define a request as having a number of *Input* and *Output* elements, with no data type information. It is impossible to add specificity whilst maintaining compatibility with the base schema. The benefits of using WPS are therefore reduced, as interoperability provided by following the fixed message structures detailed in the specification is lost. Each provider may have a different pattern for creating the WSDL messages, as there are no specific guidelines or specifications to follow on this.

There have been attempts to automatically derive WSDL documents from WPS instances (Sancho-Jiménez et al., 2008), relying on proxy services to convert a WPS process description to WSDL. In practice, these approaches generate documents with inadequate message structure descriptions. Since the WPS describes a type with only a MIME reference and XML schema URL, input and output data types cannot be described in enough

detail, because WSDL requires an element, or set of elements, from within a schema to be specified for each input or output. As a WPS process description does not contain this information, the majority of proxy based services use the XML *anyType* element. This reduces the potential for usability and interoperability since the human (or machine) process consumer has no information about the actual data type without further querying the service through the *DescribeProcess* operation.

Another proxy approach was developed in the early stages of the UncertWeb project, as part of a number of experiments with BPEL. To solve the issue of inadequate message descriptions, custom tags were added to the metadata elements for each process input or output on the WPS instance. The proxy service parses and interprets these tags to generate a concrete WSDL document. Whilst this approach solves this issue of inadequate descriptions, the non-standard nature of the metadata tags makes it infeasible for adoption on a wider scale. With a proxy-based solution, there is additional effort for the developer, and overhead in running an additional service.

If the benefits of WPS over other widely used standards were clearer, the effort and overheads of manually creating WSDL documents and developing proxy services would be worthwhile. These solutions maintain compatibility with both WPS and WSDL clients. However, it is currently difficult to see why a combined WPS and proxy approach would be favoured over simpler standalone SOAP/WSDL services, which can be generated using frameworks that take existing code and convert it to usable web services.

The SOAP and WSDL standards were defined by W3C in 2000 and 2001, and version 1.0 of the WPS specification was finalised in 2007. With these existing standards well established at that point, it is unclear why the OGC chose to implement a custom protocol. It becomes even more unclear when considering that the features within OGC specifications have equivalents within SOAP/WSDL. For example, SOAP has a common fault element, as does the OGC. To maximise reusability, it would have been possible to use the established SOAP fault element, rather than require users of the OGC WPS standards to implement new tools. The OGC could perhaps argue that the aim was to ensure consistency across their web service stack, which defines a common pattern for all service interfaces, including WFS, WCS and WPS. We argue that in the context of

standards uptake consistency within a service stack is less important than consistency across the World Wide Web.

With SOAP/WSDL or JSON, creating clients is straightforward. You can either use the previously mentioned code-generating tools, use a generic client, or develop a custom one. The compact nature of JSON allows requests and responses to be generated and parsed with ease, especially in JavaScript. In comparison, the WPS is complicated for the consumer. To understand how to issue and build requests the specification must be read, as message descriptions are not provided in a standard format such as XML schema. The WPS usage pattern is complex, and involves: listing process identifiers with *GetCapabilities*, retrieving a full description of the process they wish to use with *DescribeProcess*, and finally using *Execute* for the actual processing request. Whilst this pattern is familiar to users within the OGC world, it may seem unnecessarily complex for those outside the domain.

In a rapidly-evolving web environment, it is important to support current technologies. This priority is even more critical for geospatial communities when you consider the growing interest in location, with smartphones and many other devices becoming location-aware. The barrier for entry is lowered if current technologies are supported, as more tools and support will be available. To give one example, the current lack of JSON support in WPS can lead to longer implementation times for developers of web applications. The lengthy and detailed standardisation process for OGC standards can mean a long wait of several years before useful features such as this are added to a specification, although it is relatively simple to implement such support in a non-standardised way. This is a common problem when comparing standardisation and popularisation as mechanisms for developing a shared understanding or a community model; in practice some aspects of both are required. This paper aims to make a contribution to future development of geospatial processing on the web.

4 The framework

To provide an alternative service interface based on existing standards, we developed a generic Java-based framework for exposing processes on the web. The framework has two interfaces: one based on SOAP/WSDL, and the other on JSON. Although it is currently possible to use JSON in SOAP and WPS interfaces, these require embedding or linking to JSON documents from within XML. Using two data exchange formats in this manner adds complexity, especially when combining the lightweight JSON format with heavyweight XML, requiring multiple parsing mechanisms. Therefore, we created the JSON-based interface to provide data exchange in a single, lightweight format. Our main aim for the service framework was to make it as easy as possible for both users of the processes, and process developers. Each component of the system is designed to be extensible, essentially creating a 'pluggable' architecture.

[Figure 1 about here.]

Using the framework at its simplest, a developer needs only to extend the abstract process class, as shown in Figure 1. The subclass must implement methods for returning a list of input and output identifiers, their data type, and the process outputs given a set of inputs. Each input and output has a data type specified by a Java class. The framework is responsible for automatically selecting the appropriate XML or JSON encoding depending on this class, whether that be GML, O&M, GeoJSON, or some other format. For example, if a process has an output with a data type specified as a JTS[7] Point, the built in GML encoding would be selected. In the JSON interface, GeoJSON encoding would be used. Currently, the framework can parse and encode data conforming to the UncertWeb GML and O&M profiles described in Section 5. In addition to these vector formats, support is included for raster NetCDF data.

If no appropriate representation exists, the developer must implement an encoding class themselves. If the developer does not wish to use the built in encoding classes, they are able to override them with their own. An encoding class for XML should specify what

[7] http://www.vividsolutions.com/jts/

classes are supported, where the schema can be found, and a mapping from Java class to schema type, along with methods for parsing and generating the XML.

A standard feature of the service is reference passing. For any complex input or binary data, it is possible to specify a URL referencing the data rather than embed it in the request. The service will then load the URL and parse the data using the encoding classes. In cases where large binary data will be used, and it may not be feasible to load the complete file into a Java object representation, a process developer can request to skip the encoding classes and simply be passed the URL to the data. It is also possible to ask the service to return a reference to the output data rather than embed the data in the response. In these cases, the data is currently stored by the service and returned as a URL. However, it would also be possible to support standard data access services such as the WFS and the Sensor Observation Service (SOS).

Utilising the information kept within encoding classes, the framework is able to automatically generate a WSDL document. For a given input or output, the encoding class knows which schema element the data type Java class should refer to, and the relevant schema URL. The generated document is therefore fully-specified, with process-specific concrete schema elements. A consumer of the document is aware of each input and output, its data type, and whether or not it is required, without any additional calls to the service.

In addition to exposing a concrete WSDL document, the service uses a fixed pattern for process requests and responses. In the SOAP/WSDL interface, each input or output has an element, the name of which is the identifier. That element can either contain a simple value, a complex value, or a reference. Simple values are primitives such as a floating-point numbers and strings. Complex values are structures consisting of several nested values, usually representing objects with multiple properties, such as elements within GML and O&M. A reference is given by a URL, which could locate a file containing, for example, binary coverage data on a web server, or some data on a web service. Listing 1 demonstrates this fixed pattern in a request document for a process with three inputs; one simple, one complex, and one reference. The same approach is adopted in the JSON interface. This fixed message pattern allows a generic usage scenario, similar

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/" xmlns:ps="http
    ://www.uncertweb.org/ProcessingService">
  <soap:Header />
  <soap:Body>
    <ps:ExampleProcessRequest>
      <ps:A>0.444</ps:A>
      <ps:B>
        <gml:Point xmlns:gml="http://www.opengis.net/gml/3.2" gml:id="point1">
          <gml:pos srsName="http://www.opengis.net/def/crs/EPSG/0/4326">52.87 7.78</gml
              :pos>
        </gml:Point>
      </ps:B>
      <ps:C>
        <ps:DataReference href="http://uncertws.aston.ac.uk/data/example_point.xml"
            mimeType="text/xml" />
      </ps:C>
      <ps:RequestedOutputs>
        <ps:O reference="true">
      </ps:RequestedOutputs>
    </ps:ExampleProcessRequest>
  </soap:Body>
</soap:Envelope>
```

Listing 1: An example SOAP request to the processing service

to that of the WPS.

All exceptions caught within the framework are returned to the user as standard SOAP fault elements. Each element contains a human-readable explanation, further specific error information, and a code with the intention to help find the source of the exception - either client or server. In the case of the JSON interface, there is currently no specification similar to SOAP available. We therefore created our own structure to mimic the information contained in a SOAP fault. If a developer wishes to indicate a fault produced by a process, they can throw an instance of *ServiceException*.

Whilst the framework is functional, it is currently an early version, and missing some features. Geospatial processes are often performed on large data sets, creating execution times ranging from seconds to hours. With synchronous process execution, it may be infeasible for a user to wait for a response, or the request may simply time out. These issues stress the importance of asynchronous process execution, a standard feature in the WPS specification, but currently missing from the framework. This feature will be prioritised in future development of the framework, in addition to support for more raster formats and utilising SOAP security to restrict access to processes.

5 Use case

To test the service framework, a workflow was built as part of the FERA UncertWeb use case. The workflow, as introduced in Section 3.3 and shown in Figure 2, is composed of a set of models for predicting land-use and crop yield response to climatic and economic change. It currently consists of two models, although will be further extended in the future. The first, Land Capability Classification System (LCCS) is used to calculate the probability of crop transitions within fields, given a set of historical crop rotation data. The other, LandSFACTS, is the previously-mentioned field-use simulator. LandSFACTS was developed by the Macaulay Land Use Research Institute (Castellazzi et al., 2010), and simulates crop allocations for a period of five years, based on the transition probabilities which form the outputs of the LCCS and a set of rules and constraints to ensure for example the correct treatment of genetically modified crops, or other external constraints.

[Figure 2 about here.]

For each of the models, a process was deployed using the framework. The models had been previously developed and were supplied as R and C++ source code. As neither models were accessible through a web service interface, the processes on the framework were implemented as wrappers. The wrappers are responsible for taking the parsed web service inputs, converting them to a suitable format for the existing model, executing the model, then converting the outputs to the required format. These steps will often be required, as formats commonly used in a web service environment, such as O&M, are

unlikely to be supported by many existing models. This mapping of intrinsic model data types to data types supported in the modelling framework is a signficant challenge for exposing models on the web, and interoperability more generally.

The GML and O&M encoding standards are able to model a wide range of objects from within geographic systems, and can describe geographic data sets. At their base, generic level, the standards are difficult to use. For example, the schema for O&M states that the result of an observation can be of any type, which is impossible to implement. A profile of a schema restricts the types that can be used. Within the UncertWeb project, we have developed profiles for both GML and O&M. These restrict the elements available in the GML schema to widely-used primitives, such as points, lines, polygons and raster grids, and the result in an O&M observation to boolean, categorical, text, uncertainty, discrete numeric, reference, and measurements. These restricted profiles make it possible to guarantee support for these types within services and clients, and a set of fully-featured Java APIs have been developed to facilitate usage[8].

In our web processing framework, support for the UncertWeb GML and O&M profiles is included, and was appropriate for the majority of inputs and outputs in our example workflow. However, there are occasions when no elements in these standards are able to represent some of the data required or produced by the workflow. An example of this is the crop transition matrix output from the land capability model. We considered adding a transition matrix type to our profiles, but to keep the profiles more general, it was decided to keep it separate. The flexibility of the framework allowed us to develop our own encoding class for this type.

[Figure 3 about here.]

To demonstrate the workflow, a simple JavaScript web client was developed. Using the JSON interface provided by the processing service framework, we were able to orchestrate the workflow entirely with JavaScript. Requests and responses are generated and parsed, calling each service in turn and displaying appropriate visualisations once processing is completed. Several parameters can be adjusted within the web client, allowing users to control the behaviour of the workflow. Once the user clicks the submit

[8]http://www.uncertweb.org/software/utilities

button, the client reads the parameter values in the fields presented to the user. A JSON request to the LCCS process is built, and sent asynchronously to the service. When the response is received, several tables displaying the transition matrices are shown to the user. These output transition matrices are added to a JSON request for the LandSFACTS process, in addition to the field areas and type classification. This request is again, sent asynchronously to the service. After the response has been received, the simulations are parsed. For each field, a colour indicator is placed on a map provided by Google Maps. This indicator represents the simulated crop in that field for the selected year. The user is able to select the sample, simulation and year from which they wish to display the simulation. The processes can also be orchestrated with Taverna, making it possible to extend the workflow by adding other processes and functionality if required.

6 Evaluation

The developments within this paper successfully achieve the goal of creating a model execution service framework which can support widely used web technologies. By automatically selecting appropriate data types and generating a service description, model owners can focus on process functionality rather than the more technical aspects of exposing their models on the (geo)web. A challenge still exists in connecting models to a web service interface. This is caused by models being written in a variety of languages. Our wrapper approach goes some way in solving this, but still requires development effort, as for the majority of models specific wrappers will have to be developed.

The implementation of the crop allocation model workflow provided a basis for a usability test based on the requirements in Section 3.2. The outcomes from this test are summarised in Table 1. Exposing the existing models on the web was generally as simple as overriding methods in a class. The extensible nature of the framework allowed a custom encoding format to be integrated in the same simplistic manner as a process. The benefits of providing a concrete WSDL document could be immediately seen by composing the workflow with Taverna, and generating usable Java code with Apache Axis. By utilising the JSON interface, a JavaScript web client was developed without the

need for parsing and generating code, as required by XML. Further usability tests will be performed upon releasing the framework to a wider community. Both the framework and WPS support reference passing. Unlike the WPS, the framework is unable to support asynchronous process execution. This could be a serious limitation when dealing with long-running processes. However, this will be implemented in a future version.

[Table 1 about here.]

An additional drawback of the framework, which relates to a limitation of WSDL, is the lack of metadata attached to the service interfaces. OGC WPS can provide additional metadata to assist service discovery and usability. This metadata can be generic information such as a description of a process input, or geospatially specific information such as the supported resolutions for a coverage input. Metadata like this would facilitate automated or semi-automated workflow composition. Initial steps are being taken to solve this limitation by supplementing the generated WSDL document with metadata annotations. These annotations, described in Table 2, are tag based and share a common dictionary with all models in the UncertWeb project. An alternative approach could be to create an additional operation for retrieving metadata on a per-process basis. In either case, it is extremely important that the approach is standardised to make it useful to the wider community.

[Table 2 about here.]

7 Conclusions

With WSDL describing each process on the service in a standard manner, interoperability at the process level is achieved. A compatible client is able to parse the description and present inputs to a user, build a request, and execute a process. Workflow tools and software can orchestrate multiple services described with the approach we propose here, without the need for process-specific code. The built-in support for the UncertWeb GML and O&M profiles, which are rather generic in character, encourages the use of interoperable formats for commonly used data, such as geospatial primitives and observations.

As the framework is not formally standardised, it is only able to reach a certain level of interoperability. The use of fixed message structures could help to achieve a higher level, but only if they were standardised and adopted. While, clearly, a specification like WPS can greatly facilitate interoperability, the current version presents many complexity and usability issues, often leading to process implementations which actually hinder interoperability. Integrating with popular web technologies and focusing on usability of the WPS will enable the specification will be more useful and attractive to a wider community.

Several issues require further attention to make the developed framework into a more complete model integration tool. First, support for existing community modelling frameworks should be considered. This might take the form of a set of tools to help model developers and users add support for their models to the framework. Secondly a more complete support for different data types would be helpful; however we speculate that this may evolve with the framework as more models are added. Thirdly, support for security and asynchronous processing could be added to the framework to allow a more flexible and secure distributed computational environment. Future work should also further consider the semantic annotation of models and their inputs to facilitate improved discovery and machine mediation between model components.

Acknowledgements

The research leading to these results has received funding from the European Union Seventh Framework Programme (FP7/2007-2013) under grant agreement n [248488]. We acknowledge discussions with Christoph Stasch and other members of the UncertWeb team in defining the UncertWeb profiles and service annotations. Jill Johnson and John-Paul Gosling provided modelling expertise to the use case demonstrated in this paper. We are grateful to two reviewers, whose constructive suggestions helped us improve the paper.

References

L. Bastin, D. Cornford, R. Jones, G. B. Heuvelink, C. Stasch, E. Pebesma, S. Nativi, P. Mazzetti, and M. Williams. Managing uncertainty in integrated environmental modelling frameworks: The UncertWeb framework. *Environmental Modelling and Software*, in review, 2011.

L. Bigagli, M. Santoro, V. Angelini, P. Mazzetti, and S. Nativi. Service frameworks for modelling resources. Technical report, National Research Council of Italy, 2011.

D. Box, D. Ehnebuske, G. Kakivayam, A. Layman, N. Mendelsohn, H. F. Nielsen, S. Thatte, and D. Winer. *Simple Object Access Protocol (SOAP) 1.1.* W3C Note. W3C, May 2000. URL `http://www.w3.org/TR/soap11`.

M. Castellazzi, J. Matthews, F. Angevin, C. Sausse, G. Wood, P. Burgess, I. Brown, K. Conrad, and J. Perry. Simulation scenarios of spatio-temporal arrangement of crops at the landscape scale. *Environmental Modelling & Software*, 25(12):1881–1889, 2010. ISSN 1364-8152. doi: 10.1016/j.envsoft.2010.04.006.

E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. *Web Services Description Language (WSDL) 1.1.* W3C Note. W3C, March 2001. URL `http://www.w3.org/TR/wsdl`.

D. Crockford. JSON: The fat-free alternative to XML. In *Proceedings of XML 2006*, 2006. URL `http://www.json.org/fatfree.html`.

R. T. Fielding. *Architectural Styles and the Design of Network-based Software Architectures.* PhD thesis, University of California, Irvine, 2000. URL `http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm`.

T. Foerster, A. Bruehl, and B. Schaeffer. RESTful Web Processing Service. In *Proceedings of the 14th AGILE International Conference on Geographic Information Science*, 2011.

G. Geller and W. Turner. The model web: a concept for ecological forecasting. In *Geoscience and Remote Sensing Symposium, 2007. IGARSS 2007. IEEE International*, pages 2469–2472, July 2007. doi: 10.1109/IGARSS.2007.4423343.

P. Louridas. Orchestrating Web Services with BPEL. *IEEE Software*, 25:85–87, March 2008. ISSN 0740-7459. doi: 10.1109/MS.2008.42. URL `http://dl.acm.org/citation.cfm?id=1345866.1345902`.

P. Mazzetti, S. Nativi, and J. Caron. RESTful implementation of geospatial services for Earth and Space Science applications. *International Journal of Digital Earth*, 2(1 supp 1):40–61, 2009. doi: 10.1080/17538940902866153. URL `http://dx.doi.org/10.1080/17538940902866153`.

OGC 05-007r7. *OpenGIS Web Processing Service*. OpenGIS Standard. Open Geospatial Consortium Inc., Wayland, USA, 2007.

OGC 07-036. *OpenGIS Geography Markup Language (GML) Encoding Standard*. OpenGIS Standard. Open Geospatial Consortium Inc., Wayland, USA, 2007.

OGC 07-057r7. *OpenGIS Web Map Tile Service Implementation Standard*. OpenGIS Standard. Open Geospatial Consortium Inc., Wayland, USA, 2010.

OGC 08-009r1. *OWS 5 SOAP/WSDL Common Engineering Report*. OGC Discussion Paper. Open Geospatial Consortium Inc., Wayland, USA, 2008.

OGC 09-184. *Improve specification of complex data input/output formats in process description*. OGC Change Request. Open Geospatial Consortium Inc., Wayland, USA, 2009.

OGC 10-025r1. *Observations and Measurements - XML Implementation*. OpenGIS Standard. Open Geospatial Consortium Inc., Wayland, USA, 2011.

C. Pautasso, O. Zimmermann, and F. Leymann. Restful web services vs. big web services: Making the right architectural decision. In *17th International World Wide Web Conference (WWW2008)*, pages 805–814, Beijing, China, April 2008 2008. URL `http://www2008.org/`.

G. Sancho-Jiménez, R. Béjar, M. A. Latre, and P. R. Muro-Medrano. A Method to Derivate SOAP Interfaces and WSDL Metadata from the OGC Web Processing Service Mandatory Interfaces. In *Proceedings of the ER 2008 Workshops (CMLSA,*

ECDM, FP-UML, M2AS, RIGiM, SeCoGIS, WISM) on Advances in Conceptual Modeling: Challenges and Opportunities, ER '08, pages 375–384, Berlin, Heidelberg, 2008. Springer-Verlag. ISBN 978-3-540-87990-9. doi: http://dx.doi.org/10.1007/978-3-540-87991-6_44. URL `http://dx.doi.org/10.1007/978-3-540-87991-6_44`.

B. Schäffer and T. Foerster. A client for distributed geo-processing and workflow design. *Journal of Location Based Services*, 2(3):194–210, 2008. doi: 10.1080/17489720802558491. URL `http://www.tandfonline.com/doi/abs/10.1080/17489720802558491`.

List of Figures

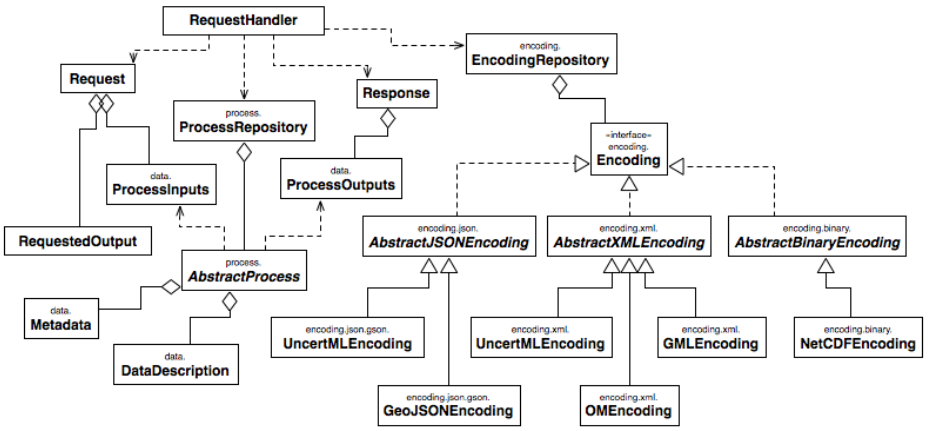

Figure 1: A simplified overview of the processing service framework class architecture.

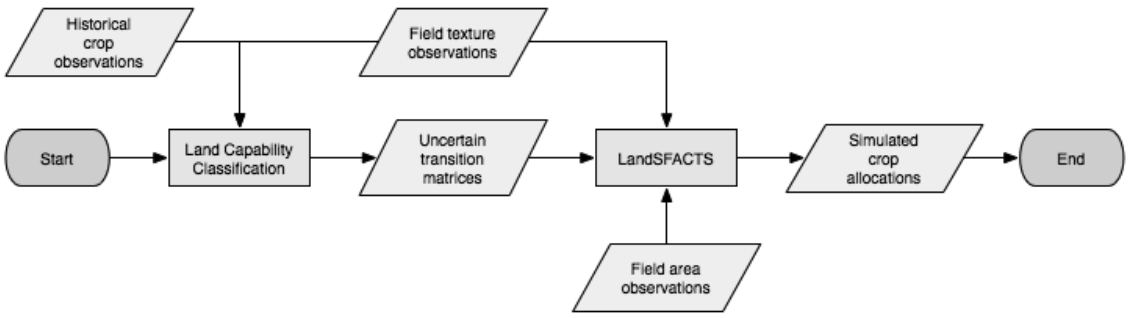

Figure 2: A diagram of the FERA land-use and crop yield response workflow.

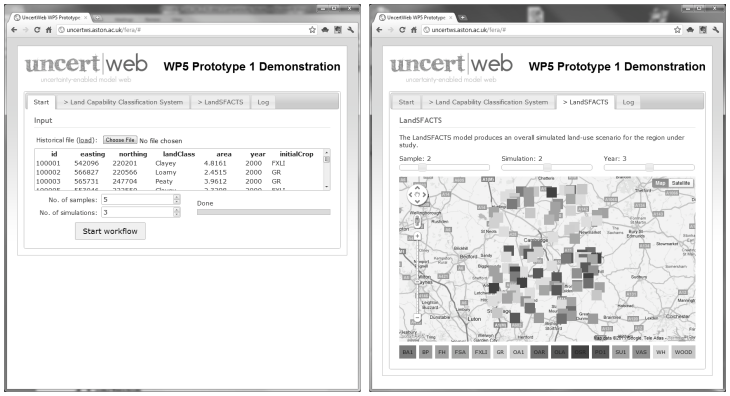

Figure 3: Screenshots from the workflow demonstration web client.

List of Tables

Table 1: A summary evaluation comparison in the context of the use case.

	UncertWeb Processing Service	**WPS 1.0.0**
Standardisation	Not formally standardised, making it difficult to encourage adoption. Fixed structure documents are used, creating the potential to standardise.	Formally standardised, ensuring a level of compatibility and support within the OGC world.
Service description	Fully described with concrete schema elements. For example, the user is aware that the field areas should be given as an O&M measurement collection.	Described with an abstract schema document. For example, the user only knows the field areas are O&M — this could be text, boolean, measurement, or various other types of observation.
Client generation	Java client code was generated using Apache Axis. User does not need to see any XML.	The 52N WPS Java client can be used, but a user is still required to view and understand the ProcessDescription XML document.
Web demonstration client	Used the JSON-based interface to develop a simple client. Responses parsed to native JavaScript objects.	Client would require full use of XML.
Orchestration with Taverna	Supported out of the box. Able to create a workflow without dealing with XML directly. Some usability problems with O&M — Taverna cannot parse the schema properly due to the number of elements.	Orchestration with Taverna requires creating a WSDL file for the WPS instance. Request and response documents then have to be manually created.
Metadata	Annotated tags. Can be parsed by clients and services within UncertWeb.	Free text, potential for annotated tags.

Table 2: The UncertWeb dictionary of process and parameter metadata tags.

Process	
description	A textual description of the process, input, or output.
Spatial	
spatial-resolutions	Supported resolutions of the raster layers.
spatial-support-types	Indicates whether the support of the cell value is the centre of the object or the average value of the complete object (typically a grid).
spatial-crss	Supported spatial reference systems.
spatial-geometry-types	Types of geometry supported.
spatial-domain	Extent of the spatial domain supported.
Temporal	
temporal-resolutions	Indicates the temporal support of the values of the variable.
temporal-support-types	Indicates whether the support of the cell value is the centre of the object or the average value of the complete object (typically a time instant)
temporal-domain	Extent of the temporal domain supported.
Variable	
variable-phenomena	Phenomena identifier. For example, the observedProperty URI in case of O&M data.
variable-uncertainty-types	UncertML[9] type of uncertainty information.
variable-units-of-measure	Units of measure of the variable.