ZOO Project: The Open WPS Platform

G. Fenoy [a, *], N. Bozon [b], V.Raghavan [c]

[a] GeoLabs sarl, Futur Building I 1280, avenue des Platanes 34970 Lattes, FRANCE. gerald.fenoy@geolabs.fr
[b] 3LIZ sarl, 361 rue J-F. Breton B.P. 5095 34196 Montpelllier, FRANCE. nbozon@3liz.com
[c] Graduate School for Creative Cities, Osaka City University, 3-3-138 Sugimoto, Sumiyoshi-ku, Osaka 558-5858, JAPAN. raghavan@media.osaka-cu.ac.jp

KEY WORDS: Web Processing Service, Open Geospatial Consortium, Geographic Information Systems, Open Source Geospatial.

ABSTRACT:

This paper aims to present the ZOO Project, which is a new open source implementation of the Open Geospatial Consortium's (OGC) Web Processing Service (WPS), released under the term of the MIT/X-11 license. Based on a robust server-side C language Kernel (named ZOO Kernel), ZOO Project proposes a new approach to develop, handle and chain standardized GIS-based Web services. A brief review of WPS and existing implementations will be first proposed in order to detail the ZOO Project development background and goals. Then, the ZOO itself will be presented, focussing on its assets and limitations, formost to highlight the new opportunities provided by such a platform. The ZOO Kernel and its architecture will be first examinated, before further explanations on the proposed method for Web services creation. The ZOO JavaScript API that provides an easy way to orchestrate and chain Web services will be then presented through technical ramblings on server-side JavaScript support into ZOO Kernel. Both Kernel and API are illustrated and documented through different Web service code snippets. Some visual examples of client-side interactions are also presented.

1. INTRODUCTION

1.1 Research context

Progress of geographic information systems (GIS) and the more systematic use of the Open Geospatial Consortium Webservices (OWS) has led to a variety of available technologies and methods to store and spread GIS data over the Internet. Standardization of spatial data and metadata have become crucial in the context of collaborative Web GIS development, but also due to specific directives or policies regarding data use and sharing, such as the INSPIRE directive for the European context. The quiet recent but very fast development of new Web GIS techniques (partly due to new opportunities offered by Web 2.0 and Cloud Computing), is leading to a growing public and governmental awareness on the necessity of using standards for Web-based spatial data infrastructures (SDI).

Numerous tools are available today to strore and spread spatial data over the Internet, through the Web Map Service (WMS) and Web Feature Service (WFS). Many open source or proprietary GIS solutions now supports such standards and these methods have become very popular. By contrast, only a few solutions are available for processing such data through WPS.

Some WPS fundamentals and a review of the existing implementations are recalled in the first section, in order to explain the ZOO Project development context, its differences, assets and limitations. Second and third section then present the ZOO Kernel and API architecture, and aim to detail the method for setting up Web Services through simple examples.

1.2 Web Processing Service (WPS)

WPS is one of Open Geospatial Consortium's most recent interoperability standards. It was first proposed under version 0.4 in 2005, and some improvements were added in version 1.0.0 which was released in 2007, after requests for public comments and huge collaborative efforts by adopters and dedicated working groups.

WPS is designed to standardize the way that GIS algorithms are made available on the Internet. It specifies a mean for a client to request the execution of a spatial calculation from a service. It intends to automate geoprocessing by employing geospatial semantics in a service-oriented architecture (SOA). WPS supports simultaneous processes via the HTTP GET and POST method, as well as Simple Object Access Protocol (SOAP) and Web Services Description Language (WSDL), thus providing freedom to the Web Service developer to choose the most suitable method regarding his implementation needs.

There are three key requests that can be submitted to a WPS server. *GetCapabilities*, which first generates a metadata file as a XML document that describes the available processes on the server-side. *DescribeProcess*, which then provides more detailed information of a specific process such as the necessary input data, the targeted output data format, as well as the service title and short abstract.

Once all the necessary supplied parameters are gathered from *DescribeProcess* request, the processing task can be submitted to the server by the *Execute* request. The latter can answer directly to the client by returning the created output, or store the results as Web accessible resources, in an *ExecuteResponse*.

* Corresponding author.

This XML document contains the URL location of future status document update, and notifies the accepted and rejected processing tasks. Thus, the client can request the next *ExecuteResponse* document until the process completion is indicated by the "*ProcessSucceeded*". While the *Execute* request is active, the progress of a process can be followed continuously with the "*ProcessAccepted*", "*ProcessStarted*", "*ProcessSucceeded*" and "*ProcessFailed*" statements. In case of "*ProcessSucceeded*", the *ExecuteResponse* includes either the raw generated outputs or some URL indicating the physical location where from the output data could be accessed.

This short presentation of WPS shows that it is massively based on XML and the three key requests along with their respective responses, which cover the essential part of the OGC's specification (Michael, Ames, 2007).

1.3 Existing WPS implementations

As it was already said, WPS specification is rather new and implemented only by a few GIS library, and so supported by a very few GIS clients. One can note that most of the available WPS implementations are using Java language, and this can be proved by the following list:

- **Deegree** framework is a Java-based environment which implements most of OGC standards and has implemented WPS quite early.
- **WPSint** is a Java plug-in for Spring GIS software and implements WPS 0.4.0 in a Java/JEE application framework.
- **52° North WPS** is also written in Java, as a plug-in for Java Tomcat servlet container, and interacts with other OWS standards. It also includes a UDig software client to interact with 52° North WPS.
- **GeoTools** and **GeoServer** also from the Java world are actually working to implement WPS 1.0.0 (Holmes, 2009)

PyWPS is the only Python-based WPS implementation and provides an environment to create Web Processing Services in Python. It also proposes a native support for GRASS-GIS python scripting. This project was started in 2006 and is now starting the Open Source Geospatial Foundation's (OSGeo) incubation process.

Other proprietary implementions may have been carried out, but not published, so the versions of WPS, the protocols and the languages they are using are still unknown. However, one can notice that the Environmental Systems Research Institute (ESRI) is actually working on implementing WPS support into future version of Arc GIS Server (Fee, 2008).

This review of existing WPS compliant products shows that a small number of open source projects are actively building the WPS definition and implementation in close collaboration with the OGC, and that it is a relatively new research field regarding the geospatial Web. WPS 2.0.0 is now announced. We can also notice that every available WPS solution is language dependant, meaning that Web Services must be coded in Java or in Python, and this is a huge limitation to the use of WPS.

As an open source project too, ZOO is in line with this technical research context, and wants to propose new opportunities that promotes WPS and make it simpler. Despite it is the younger project, it supports several programming languages and provides an original Apache-based approach to setup reliable and powerful WPS servers. The ZOO's architecture and functioning are described in the two next sections.

2. ZOO KERNEL

2.1 Presentation

ZOO Kernel is the heart of the ZOO Project. It is a server-side C Kernel which makes it possible to create, manage and chain WPS 1.0.0 compliant Web Services, by loading dynamic libraries and handling them on-demand. Thus, it can easily connect to geospatial libraries and scientific models, but also with the common cartographic engines and spatial databases.

ZOO Kernel is written in C language, but Web Services can be programmed in C, Python, Java, Fortran, PHP and JavaScript. This multi-language support is convenient for developers and allows above all to use existing code to create new Web Services. Open source GIS libraries or specific code (spatial based or not) can so be ported server-side with very little modifications. Some examples are given in section 2.3.

2.2 Architecture

ZOO Kernel basic architecture is detailed in this section. Internal mechanisms based on the concept of *Service Provider* and the adopted grammar for configuration are first explained. The supported programming languages and their respective dependencies are also listed.

ZOO Service Provider

A ZOO services provider is a couple of a Services Shared Objects (SSO) and one metadata ZOO configuration file (.zcfg) per provided service. The ZOO Configuration file contains all the metadata information about the service provider and the latter contains the source code of the related services**.**

The Services Provider was conceptualized this way in order to facilitate the GetCapabilities and DescribeProcess Requests. Indeed, ZOO Kernel only have to parse the ZCFG file using a specific Flex and Bison parser to answer to this two kind of requests. Using Flex and Bison, a grammar was then defined a for the configuration file which will make the ZOO Kernel able to check if all required inputs were provided in the request. If yes, a well formed Exception XML Document as expected and defined by the OGC's WPS standard will be produced. Flex and Bison are used by lot of software such as MapServer, but a new grammar had to be defined. A serious near-future alternative is to use in the the YAML format to easily define the service metadata.

Multi-languages

ZOO Services can be written natively in C and Python language. The Python interpreter was embedded into ZOO Kernel which allows to use existing Python libraries as ZOO Services.

PHP (embedded version), Java, Fortran and JavaScript are optional languages and compilation options must be defined at compilation, and specific dependencies installed (PHP embedded, Java SDK, G77 and SpiderMonkey).

This variety of supported languages allows the WPS Web Service end-developer to choose his preferred language and above all to use existing code and turn it into WPS. Web

Services coded in different languages can also be chained in a standardized way.

2.3 Using OSGeo librairies as WPS using ZOO Kernel

The ZOO Project initial idea was to build a platform able to connect the numerous and good OSGeo libraries together and to use them as Web Services. The Web Services creation was logically first test with the GDAL/OGR library (Warmerdam, 1999), in order to perform basic vector and raster WPS from a stable C library.

GDAL/OGR

As GDAL/OGR is coded in C, the corresponding Web Services were written in the same language, in a rather simple way. Indeed, as ZOO Kernel is able to load dynamic libraries, only a few modifications were needed in the original code. This can be confirmed for example by looking at the well known *ogr2ogr* code and the corresponding .zcfg file on the ZOO Project Trac (Fenoy, Bozon, Raghavan, 2010). Using the OGR code base again, we could have also setup a WPS Service for single and multiple geometries spatial operations (Fenoy, Bozon, Raghavan, 2010[2]). Same work was done using the GDAL code base in order to implement the *gdalgrid* and *gdaltranslate* capabilities as WPS. Those Web Services allow to convert, reproject and process both vector and raster data online in a standardized way.

GRASS GIS

Some other experiments are actually being carried out to communicate with GRASS GIS, which provides advanced open source GIS processing algorithms. (Neteler, Mitasova, 2008). GRASS 7 now provides a WPS process description exporter, which returns XML documents describing the GRASS functions (Gebbert, 2009). This is very useful for our tentative connection to GRASS, and ZOO Kernel can take advantage of such GRASS outputs. Thus, a GRASS XML to ZOO configuration file (.zcfg) converter was developed (Gebbert, 2010), allowing ZOO Kernel to understand the numerous GRASS function through WPS. Then, a GRASS module starter was also developed in Python to call the desired function and the corresponding .zcfg in a generic way (Gebbert, 2010[2)]. These Python scripts are actually callable as ZOO Web Services and several successful tests have been carrried using the *r.add*, *r.div*, *r.mult* and *r.sub* functions (Gebbert, 2010[3)]. This very promising work must now be fully tested and integrated as ZOO Services, and the other GRASS functions could be supported soon.

The use of GDAL/OGR and the future support for GRASS GIS are showing that ZOO Kernel can use existing libraries as standardized Web Services, with minimum modifications of the original codes. Future work and development plans are based on integrating other open source GIS libraries, but also on working with non-GIS libraries (but useful when communicating with GIS), mainly for statistics and document management.

2.4 Client-side interaction examples

Let us now explain how such WPS Web Services can be called and exploited from a client-side webmapping application based on OpenLayers library (Schmidt, 2006). Figure 1 shows a WMS layer (used as input data) on which the user can select a polygon by click and then apply a buffer process on it, by calling the OGR-based Web Service cited above. Any other single-geometry vector operation can be performed such as convex hull, boundary or centroïd. The geometries are quiet simples and light so the results can be rendered using GeoJSON.

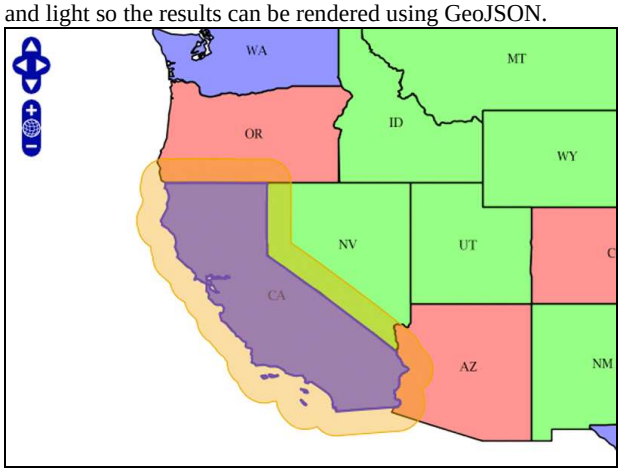

Figure 1. Example buffer output

Once the buffer is shown on the map, the user can then select another polygon and perform a multi-geometry operation. Figure 2 shows the result of an intersection process between the previous calculated buffer and the second selected polygon.

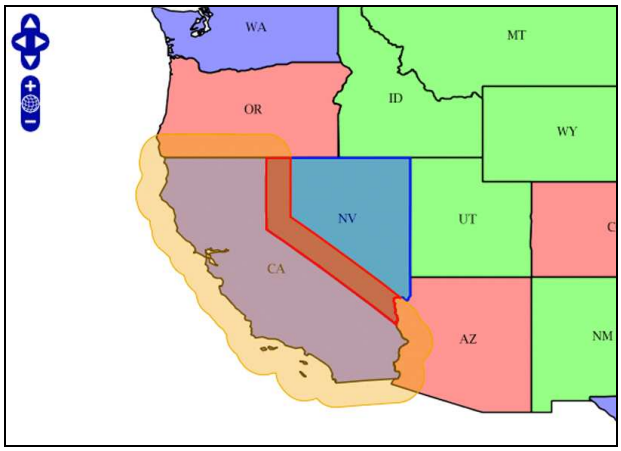

Figure 2. Example intersection output

Last client-side interaction example is based on the GDAL *ExtractProfile* function which allows to get the *z* value of any raster layer. Using an OpenLayers client once again, we could set up a specific control using the GDAL Web Service with a GTOPO30 DEM layer and a GeoJSON line string to generate JavaScript elevation profiles on the fly.

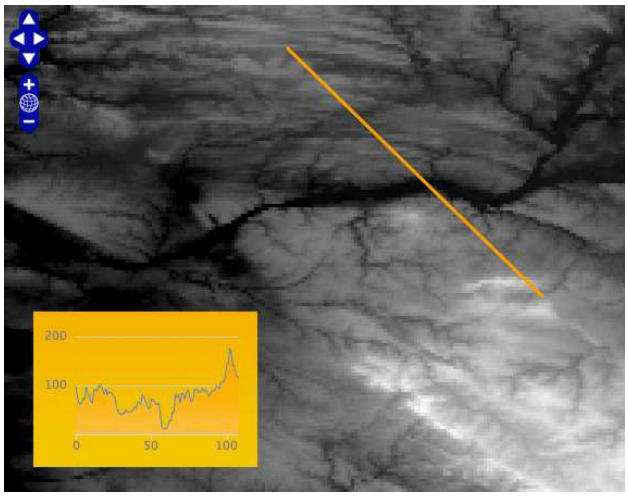

Figure 1. Example elevation profile

These client-side examples are available on the ZOO Project website and prove that ZOO Web Services can be requested from a traditional Web GIS client. However, these are proof-of-concept implementations and ZOO does not provide any WPS oriented client-side library yet.

3. ZOO API

ZOO API is a concise server-side JavaScript library designed to simplify the WPS processes creation and chaining. It is based on the ZOO Kernel JavaScript support and the Mozilla foundation JavaScript engine, Spider Monkey (Mozilla, 2010). The API allows to orchestrate WPS services using specific method and offers the ability to add logic and controls in the WPS chaining. It also uses a Proj4js (Adair, 2007) adaptation for server-side reprojection, allowing to easily convert processes outputs into common vector formats (GML, KML, GeoJSON, etc) when needed.

3.1 Server-side Javascript for WPS

Several projects are based on server-side JavaScript using the Mozilla SpiderMonkey or Rhino JavaScript engines. It was relevant to do the same in the case of ZOO Kernel for two main reasons. First, it can be compiled with optional JavaScript support, thus allowing to develop Web Services with another language. Then, the ZOO API provides ready-to-use JavaScript functions for handling WPS HTTP requests, querying available WPS Web Services, defining input/output flows in WPS chaining and converting WPS outputs into several vectorial formats.

3.2 Classes

ZOO API is first composed of several general classes dedicated to WPS requests construction such as *ZOO.String*, *ZOO.Request* and *ZOO.Bounds* (D'Hont, 2010). A *ZOO.Projection* class is also available and linked to the Proj4js source code for handling any projection defined by the Spatial Reference cartographic projections directory. *ZOO.Feature* and *ZOO.Geometry* classes, along with their respective subclasses, allow to handle the different types of vector data. Finally, a generic *ZOO.Process* class was developed to setup input/output, call and chain available WPS processes.

4. CONCLUSION

ZOO Project technical context, architecture and development goals were presented in this paper. The ZOO Kernel logic and its multi-languages capabilities were detailed, and illustrated through different achieved and on-going Web Services examples. The ZOO API and the assets of server-side JavaScript for WPS were also presented. Future research will deal with ZOO Kernel internal enhancements , as well as working on other integrations of Open Source libraries.

References from Journals:

Cepicky, Becchi, 2007. OSGeo Journal, may 2007. Geospatial Processing via Internet on Remote Servers – PyWPS

Michael, Ames, 2007. OSGeo Journal, may 2007. Evaluation of the OGC Web Processing Service for Use in a Client-Side GIS

References from Books:

Neteler, Mitasova, 2008, Springer
Open Source GIS: A GRASS GIS Approach
.

References from websites:

Holmes,2009. OpenGeo Blog
http://opengeo.org/products/coredevelopment/geoserver/wps

Cepicky, 2009. PyWPS offical Website
http://pywps.wald.intevation.org/

Fee, 2008. James Fee Blog
http://www.spatiallyadjusted.com/2008/07/30/the-esri-2008-uc-qa

Fenoy, Bozon, Raghavan, 2010. ZOO Project website
http://zoo-project.org/trac/browser/trunk/zoo-services/ogr/ogr2ogr/service.c

Fenoy, Bozon, Raghavan, 2010[2]. ZOO Project website
http://zoo-project.org/trac/browser/trunk/zoo-services/ogr/ogr2ogr/service.c

Gebbert, 2009. GRASS GIS wiki WPS section
http://grass.osgeo.org/wiki/WPS

Gebbert, 2010. GRASS XML to ZOO .zcfg
http://code.google.com/p/vtk-grass-bridge/source/browse/trunk/WPS/ZOO_Project/GrassXMLtoZCFG.py

Gebbert, 2010[2]. ZOO GRASS GIS support
http://code.google.com/p/vtk-grass-bridge/source/browse/trunk/WPS/ZOO_Project/ZOOGrassModuleStarter.py

Gebbert, 2010[3]. ZOO GRASS support tests
http://code.google.com/p/vtk-grass-bridge/source/browse/#svn/trunk/WPS/Testing/Python/GrassAddons

Mozilla Foundation, 2010. SpiderMonkey JavaScript engine
http://www.mozilla.org/js/spidermonkey/

Adair, 2007. Proj4js official website
http://proj4js.org/

D'Hont, 2010. ZOO API on ZOO Project Trac system.

http://www.zoo-project.org/trac/browser/trunk/zoo-api/js/ZOO-api.js

Acknowledgments:

Authors would like to thank F.Warmerdam for creating and maintaining GDAL/OGR. Thanks also To Soeren Gebbert and Markus Neteler for their active support in the GRASS GIS integration into ZOO Project.