OSGeo Journal

The Journal of the Open Source Geospatial Foundation

Volume 1 / May 2007

In This Volume

OSGeo
Your Open Source Compass

Integration Studies

Geospatial Processing via Internet on Remote Servers – PyWPS

PyWPS and Embrio

Jáchym Čepický and Lorenzo Becchi

Document OGC 05-007r4[1] *describes a process for offering geospatial operations over networks using Web Services. This paper introduces an example implementation – PyWPS. Even if the original target of PyWPS was to make modules of GRASS GIS accessible from Internet applications, in general it is possible to use any command-line oriented tool or any tool which has bindings to Python Programming language. With the help of PyWPS we can perform time consuming calculations on the server side, as well as build your real WebGIS application, running in a web browser. Let us describe how it works and if PyWPS could fit your needs.*

OGC Web Processing Service

Since the OGC Web Processing Service (WPS) standard is still relatively new and not as well known as, for example, its cousin the Web Map Service (WMS), we start by providing a brief overview of the standard here. The basic unit of the WPS is the *process* – a geospatial operation, with inputs and outputs of a defined type. The client communicated with the server with the help of three types of requests. The request can be sent to the server via HTTP GET with parameters provided as Key-Value Pairs (KVP) or via HTTP POST, with parameters supplied in a XML file. There are three types of requests that can be sent to the server:

GetCapabilities – Server responds with XML, describing server provider, fees, general description and giving a list of processes, prepared to be performed.

DescribeProcess – Server responds with XML, which describes concrete inputs and outputs type, so the client is able to formulate the *Execute* request.

Execute – Client requests the execution of a geospatial operation, with all required input data – the server process runs and informs the client (the user) of its progress.

The following are some illustrative examples of these requests. Let us assume we would like to per-

[1] OGC Web Processing Service (WPS): http://www.opengeospatial.org/standards/requests/28

form line-of-sight calculation from defined x and y coordinates on a raster file, which can be obtained from a remote server. The process name will be *visibility*.

Basic usage of OGC Web Processing service

First we need to find out which calculations the server offers (see this link [2]) From the resulting XML it is clear that the process *visibility* is available on the server and the abstract tells us that it does what we would like. In the second step we need to find out what kinds of input and output the process requires or will send back (see this link[3])

- *x* of type *LiteralValue*
- *y* of type *LiteralValue*
- *maxdist* – maximum distance from the observer. Type *LiteralValue*, minimal allowed value is 0, maximal 5000 meters.
- *observer* – observer height. Type *LiteralValue*, minimal allowed value is 0, maximal 50 meters.
- *dem* – type *ComplexValue* – raster map of digital elevation model, on which the visibility should be calculated.

Now we can formulate the input request, send it to the server and look forward to the calculation results[4] The server will download the input digital elevation model, perform the line-of-sight calculation and return the resulting raster image back to the client.

Introduction to PyWPS

PyWPS is a relatively new project, started in April 2006. The original project goal was to make the connection between UMN MapServer and GRASS GIS as easily possible so that we could build a real WebGIS application able to perform, for example, interpolation of raster data or various digital elevation model analysis. Time has shown that even though GRASS GIS is a powerful tool, it is not necessarily the best or the only choice for all possible tasks. The design of PyWPS has changed so that it could be used without GRASS GIS in the background, but with any other tool or just with Python itself.

PyWPS is an implementation of OGS's Web Processing Service standard as defined in the document OGC 05-007r4. Currently, the complete standard is not yet supported, but around 95% of the standard is implemented and usable.

The project is built on a simple CGI script hoping to make the life of WebGIS coders as easy as possible. It provides functions to:

- Parse all input and create all output
- Perform basic validation of the input, like insuring type of LiteralValue input or maximum file size for the ComplexValue input, etc.
- Create and remove on-the-fly generated temporary files and directories, like temporary GRASS locations and mapsets and other files created as part of the calculation.
- Run other useful operations

The coder has to do only one thing: define the processes input and output, which is basically a script in the Python programming language. The process is one class *Process*, with one mandatory method *execute*, in which the calculation is provided. Input and output data are defined in a similar way, in a complex dictionary structure[5]:

```
{
    'Identifier':'maxdist',
    'Title': 'Maximal distance',
    'Abstract':'Maximal distance of visibility',
    'LiteralData': {
        'values':[ [0.,5000] ],
    },
    'dataType': type(0.0),
},
{
    'Identifier': 'dem',
    'Title': 'Digital elevation mode'
    'Abstract': 'Raster map with elevation model',
    'ComplexValueReference': {
        'Formats':["image/tiff"],
    }
},
```

The *execute()* method can then use e.g. GRASS modules directly:

[2]http://pywps.ominiverdi.org/cgi-bin/wps.py?service=WPS&request=GetCapabilities

[3]http://pywps.ominiverdi.org/cgi-bin/wps.py?service=WPS&request=DescribeProcess&version=0.4.0&identifier=visibility

[4]http://pywps.ominiverdi.org/cgi-bin/wps.py?service=WPS&version=0.4.0&request=Execute&identifier=visibility&datainputs=x,602829.1875,y,4925326.875,maxdist=2000,observer=1.2,dem,http://somewhere/?some&service

[5]Note that this has been replaced by new methods *Add*Input()* in current *svn* version

```
def execute(self):
    # importing dem
    os.system("r.in.gdal in=%s out=dem" %\
                (self.datainputs['dem']))
    # setting region according to dem file
    os.system("g.region rast=dem")
    # lines-of-sight module
    os.system("r.los input=dem output=output \
            coordinate=%s,%s max_dist=%f \
            obs_elev=%d" % \
            (self.datainputs['x'],
            self.datainputs['y'],
            self.datainputs['maxdist'],
            self.datainputs['observer']))
    # exporting raster map
    os.system("r.out.gdal in=output out=out.tif")
    # setting the output value
    self.dataoutputs['output'] = "out.tif"
    return
```

As the source code is the best documentation, around ten example processes are distributed together with the PyWPS source, so the user can get a general picture of the process definition. There is also on-line and offline documentation available, which tries to describe the installation process and setup of your own processes.

Recently, a new class has been defined, which provides easy definition of process inputs and outputs as well as better support for GRASS GIS modules. Any web interface will be able to track progress of data imports derived directly from the `r.in.gdal` module. This improvement will be available in the next release of PyWPS.

Further development

The first "stable" version of PyWPS with the version number 1.0.0 was released in November 2006. Currently, the PyWPS development team is planning to release version 2.0.0 soon, with added functionality and a few bug fixes. Common effort in the development goes in three directions:

- Implementation of the complete OGC WPS standard
- Making the application as secure as possible, to avoid server compromise
- Making life of the process-coder as easy as possible

The PyWPS development team would also like to start discussion in the geospatial communities about defining process metadata. The OGC WPS standard defines the input types from the process point of view, however it does not say anything about input (or output) type from the user (interface) point of view. For example, coordinate x is of type *LiteralValue* but this does not describe x as a coordinate. So it could be useful to setup this input value with a mouse click in the map window rather then with the keyboard in some input form field. If you want to have a closer look at PyWPS, feel free to visit the PyWPS project page[6] where the source code as well as links to projects already using PyWPS are available.

Using ka-Map & PyWPS to create a GRASS WEB GIS

Ominiverdi[7] became involved with developing PyWPS after its initial release. Our first goal was to create a FOSS Web Client to access GRASS functions. PyWPS was not the first attempt to do a connection between GRASS and the Web but none of the others were based on open standards, making it impossible to create a shared platform.From the beginning we started planning two different projects: **Embrio** and **Wuiw**.

Embrio

This is the first implementation we had in mind: use of PyWPS to let UMN MapServer and its MapScript API to interact with GRASS.

Another important goal was a Web rich client and that's why we decided to base our efforts on ka-Map. Ka-Map uses MapScript to access the powerful set of features in UMN MapServer and offers a Google Maps-like navigation experience. The output of PyWPS, once the Process returns a map, can be in GeoTiff, for raster output, and in GML for vector output. Both formats are natively accessible by UMN MapServer and can easily be coupled with an existing map environment. In this way a request for the *Visibility* module can return a GeoTiff that is kept by ka-Map and inserted in the actual map coordinates system. Then a style, that can be an SLD, is applied to raster values and a temporary cache is created on the fly while tiles are sent back to the client. The temporary cache is related to the *sessionId*.

This system allows different modules to run in parallel and to compare their outputs using the ka-Map interface. Layer opacity control and layer verti-

[6]PyWPS project page: http://pywps.wald.intevation.org
[7]Ominiverdi website: http://www.ominiverdi.org

cal position can be useful to understand the resulting output. Even the query function is synchronized if the output is queryable. The *Visibility* module can output the incident angle with the point of view, the query function can return the value for each pixel clicked.

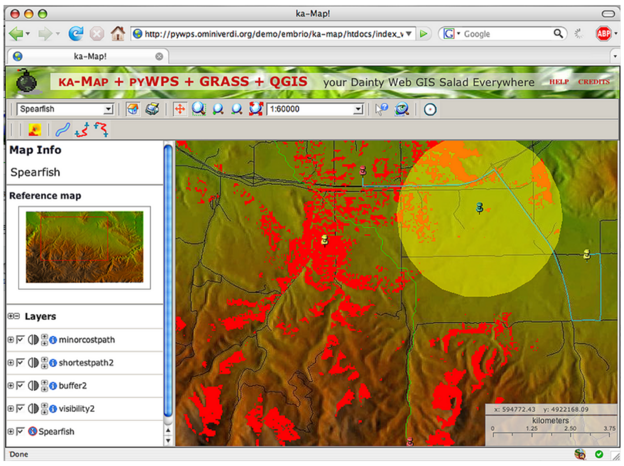

Figure 1: Screenshot of the most recent Embrio interface

Wuiw

Wuiw is still more of a concept than an application. It's intended to offer a Javascript API that connects the WPS to data resource as OWS and render the output independently from any mapserver application.

We are still evaluating current limits of the WPS draft that are not yet relating the input definition with a complex type definition. The first idea has been to use Metadata information to create this kind of interaction but there is the risk of developing a parallel dialect that will loose the benefits of standard interoperability.

We hope that the WPS path to version 1.0 will create a comfortable solution to this limitation.

Further development

It is obvious that WPS, PyWPS, Embrio and Wuiw all have short histories. Even if most things are still to be done, the beginning is promising. Regarding Embrio's future, we plan to improve interaction with the WPS script process feedback and show a process progress bar. Many more GRASS modules can be developed and we hope to receive help from the GRASS community to achieve this.

Another important goal is to add more interaction with CLI (command line interface) accessible applications, including R, GDAL/OGR, etc. for geo statistics, format conversion and many, many other functions.

We also hope to add an AJAX CLI to interact with a protected system interacting through WPS.

Licenses

It important to note that this project uses many different software packages with at least two different licenses.

- GNU/GPL: GRASS, PyWPS, R
- MIT/BSD: UMN MapServer, MapScript, ka-Map, Embrio

Resources

- OGC Web Processing Service [8]
- PyWPS home page: [9]
- Last Embrio live example: [10]
- Embrio home page [11]

Jáchym Čepický
http://les-ejk.cz
jachym AT les-ejk cz

Lorenzo Becchi
http://ominiverdi.org
lorenzo AT ominiverdi com

[8]http://www.opengeospatial.org/standards/requests/28
[9]PyWPS Home page: http://pywps.wald.intevation.org
[10] Live example:
http://pywps.ominiverdi.org/demo/embrio/ka-map/htdocs/index_wps_qgis.html
[11]http://pywps.ominiverdi.org/

Acknowledgements
Various reviewers & the GRASS News Project

The *OSGeo Journal* is a publication of the *OSGeo Foundation*. The base of this newsletter, the LaTeX 2ε style source has been kindly provided by the GRASS and R News editorial board. All articles are copyrighted by the respective authors. Please use the OSGeo Journal url for submitting articles, more details concerning submission instructions can be found on the OSGeo homepage.

Newsletter online: http://www.osgeo.org/journal

OSGeo Homepage: http://www.osgeo.org

Mail contact through OSGeo, PO Box 4844, Williams Lake, British Columbia, Canada, V2G 2V8

ISSN 1994-1897