Processing geospatial operations via Internet on remote servers – PyWPS

PyWPS and Embrio

Jáchym Čepický and Lorenzo Becchi

Document OGC 05-007r4 describes the way, how geospatial operations should be offered in networks using Web Services. This paper introduces one of it's implementation – PyWPS. Even if the original target of PyWPS was to make modules of GRASS GIS accessible from Internet, in general it is possible to use any command-line oriented tool or tool, which has bindings to Python Programming language. With help of PyWPS we can perform time consuming calculations on the server side, as well as build your real WebGIS application, running in web browser. Let us look, how it works and if PyWPS could fit your needs.

OGC Web Processing Service

Since OGC Web Processing Service (WPS) standard is still relatively new and not so known as for example it's cousin Web Map Service (WMS) is, we would like to give a brief overview of the standard on this place.

Basic unit in the WPS is the process – geospatial operation, with inputs and outputs of defined type. Client is communicating with the server with help of three types of requests. The request can be sent to the server via HTTP GET with parameters sorted like KVP (Key-Value Pairs) or via HTTP POST, with parameters sorted in XML file.

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

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

Execute – Client requests the performance of some geospatial operation, with all required input data. Depending on, if the server is supposed to run the process in the background and if it is supposed to store the resulting data on the server, resulting XML or e.g. GML file is returned immediately after the request is obtained or first after the calculation is performed.

Let us introduce some illustrative examples of these requests. Let us assume, we would like to perform line-of-sigh calculation from defined x and y coordinates on some raster file, which can be obtained on 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:

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

From the resulting XML it is clear, that the process *visibility* is available on the server and abstract tells us, that it does what we would like. In the second step, we need to find out, what kind of input and outputs does the process need or will send back:

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

- *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* – ComplexValue – raster map of digital elevation model, on which the visibility should be calculated.

Now we can formulate the input request, sent it to to server and looking forward to calculation results.

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

Server will download the input digital elevation model, perform the lines-of-sight calculation and return resulting raster image back the the client.

Introduction to PyWPS

PyWPS is relatively new project (it's development started in April 2006). Original project target was, making the connection between (UMN) MapServer and GRASS GIS easy possible, so that we could build real WebGIS application, which would be able to perform for example interpolation of raster data or various digital elevation model analysis. Time has shown, that even if GRASS GIS is powerful tool, it does not necessary have to be the best or the only one program for all possible tasks. Design of PyWPS has changed, so it can be used without GRASS GIS in the background, with any other tool or just with Python itself.

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

It is a simple CGI script, which is trying to make life of WebGIS coder as easy as possible.

- It parses all inputs and creates all outputs
- It performs basic control of the input, like type of LiteralValue input, maximum file size for the ComplexValue input and similar.
- It creates and removes on-the-fly generated temporary files and directories, like temporary GRASS locations and mapsets and temporary generated files created as part of calculation.
- And other useful operations

The coder has to do only one thing: he has to define his process with inputs and outputs, which is basically a script in 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 on similar way, it is a complex dictionary structure[1]:

```
{
    '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:

```
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 PyWPS source, so the user can get general picture about the process definition. There is also on-line and offline documentation available, which is trying to describe the installation process and setup of own processes.

Currently, new class has been defined, which provides easy definition of process In- and Outputs as well as better support for GRASS GIS modules. Any web-interface will be able to track e.g. progress of data importing, derived directly from the `r.in.gdal` module. This improvement will be available in next release of PyWPS.

Further development

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

- Implementation of OGC WPS standard to maximal degree
- Making the application as secure as possible, to avoid server compromitation

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

- Making life of the process-coder as easy as possible.

The PyWPS development team would also like to start discussion across geospatial communities about process metadata definition. 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 type of *LiteralValue* but this does not say anything about, that *x* is coordinate so it could be useful to setup this input value with mouse click in the map window rather then with keyboard in some input form field.

If you want to have closer look at PyWPS, feel free to visit PyWPS project page at `http://pywps.wald.intevation.org`, 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

As far as PyWPS has been published we, Ominiverdi.org, started developing on it.

Our first goal was to create a FOSS Web Client to access GRASS functions. PyWPS has not been the first attempt to do a connection between GRASS and the Web but none of the others was based on open standards causing the impossibility 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 to interact with GRASS.

An other important goal was a Web rich client and that's why we decided to base our effort on ka-Map. Ka-Map use mapscript to access the powerful set of features of UMN Mapserver and offers a Google Maps alike navigation experience.

The output of PyWPS, once the Process returns a map, can be in geo Tiff, for raster output, and in GML for vector output. Both formats are natively accessible by UMN Mapserver then easy to be coupled with an existing map environment.

In this way a request for the Visibility module can return a geo Tiff 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 allow to run different module in parallel and to compare their outputs using the ka-Map interface. Layer opacity control and layer vertical position can be useful to understand the resulting output. Even the query function is synchronised if the output is query-able; es: Visibility module can output the incident angle with the point of view, the query function can return the value for each pixel clicked.

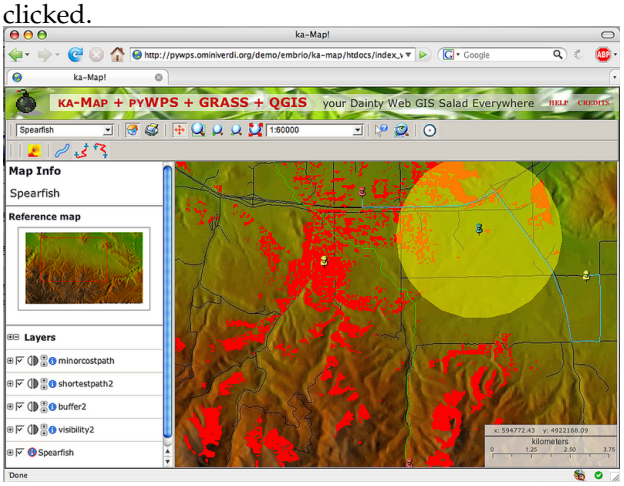

Figure 1: Screenshot of the most recent Embrio interface

Wuiw

WUIW is still a concept more than an application. It's intended to offer a Javascript API that connect 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 informations to create this kind of interaction but the risk is to develop a parallel dialect that will loose the standard interoperability.

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

Further development

It's easy to see that WPS, PyWPS, Embrio and Wuiw have all young histories. Even if most has still to be done the beginning is strongly promising. Specifying Embrio future, we imagine to develop a decent interaction with WPS script process feedback and show a process progress bar. Many more GRASS modules can be developed and we hope to find the help of the GRASS community for this.

Another important goal is to add many other interactions with CLI (command line interface) accessible applications. We are actually thinking at R, GDAL/OGR, ecc. for geo statistics, format conversion and many many other functions.

We are even dreaming to add an AJAX CLI to interact with a protected system passing from WPS.

Licenses

It important to note that this project use many different softwares and each one has its licence.

- GNU/GPL: GRASS, PyWPS, R
- MIT/BSD: UMN Mapserver, Mapscript, kaMap, Embrio

Resources

Last Embrio live example: http://pywps.ominiverdi.org/demo/embrio/ka-map/htdocs/index_wps_qgis.html

Embrio home page: http://pywps.ominiverdi.org/subversion/trunk/web/

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

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