PUBLICAMUNDI

SCALABLE
REUSABLE
OPEN
GEOSPATIAL
DATA

REPORT FOR

DELIVERABLE D4.2

1 INTRODUCTION

This report provides an overview of the technical characteristics and functionality of deliverable D4.2 “Web Processing Services”. Its purpose is to provide a short introduction to the ZOO-Project software and to present the major functionalities and improvements introduced by the PublicaMundi project.

The reader is encouraged to visit the software’s repository (https://github.com/PublicaMundi) to receive:

- Up-to-date versions of the software, along with documentation targeted to developers
- Detailed information regarding all development effort (commits, activity, issues)
- Instructions regarding the installation of the software and its dependencies

2 ENHANCMENTS OF THE WPS PLATFORM

2.1 GENERAL ENHANCMENTS

The release 1.5.0 of ZOO-Project was published on July 3rd 2015, introducing various important updates into the ZOO-Kernel implementation and into the ZOO-Client as part of Work Package 4 (WP4).

The first major update of the ZOO-Kernel was the introduction of the WPS 2.0 support including the Dismiss extension. Furthermore, another component was improved in this release: the ZOO-Client, a JavaScript API which can be used from a client to interact in an easy manner with WPS servers. The API was updated to support WPS 2.0.

Additionally, some WPS 2.0 issues regarding the metadata inheritance and cataloging capability were tackled with the Process Profiles Registry implementation. This implementation also can be used with WPS 1.0 as implemented by ZOO-Kernel. The metadata inheritance does not depend on the version of WPS; nevertheless, the information about the inheritance is only exposed to WPS 2.0.0 clients.

With the implementation of metadata inheritance, a new set of WPS Services is available, targeted to the registry browsing, giving the capability to the WPS Client to browse the different levels of metadata hierarchy. The service function (*GetFromRegistry*) parses the proper metadata file corresponding to the level of the metadata (*generic* or *implementation*) and returns the corresponding document (*GenericProcess* or *Process*).

Finally, a shared library (*lib_zoo_service*) was implemented providing the capability to modify the C API *without having to rebuild* the whole set of services. This way, when the C API is modified internally, there are no perceived changes, making the maintenance of a large collection of services easier. Administrators only need to install the new version of the *lib_zoo_service* to take advantage of all the new function definitions

2.2 SCALABILITY ENHANCEMENTS

During this period, we identified and successfully addressed several bottlenecks affecting the scalability of the ZOO-Kernel.

The most important bottleneck affected GetStatus requests, introduced in WPS 2.0, but also available in WPS 1.0. In previous versions of the ZOO-Kernel, this mechanism relied on specific Operating System (OS) resources (*shared memory*) which cannot be shared between various hosts. In fact, this

hard limitation also implied a low number of concurrent requests even when running on a same host (OS limit). Once this limit was reached, the ZOO-Kernel continued to accept incoming requests, but the client could not have access to the status of a service running asynchronously.

This bottleneck was tackled by rewriting the code handling asynchronous requests, and by adding the capability to use the filesystem or a database backend on which the ZOO-Kernel relies for all the concurrency handling. The new implementation was made publicly available in the 1.5.0 release of the ZOO-Project.

This new ZOO-Kernel release addresses one more bottleneck: the second version of WPS introduced the Dismiss request, which should permit the WPS client to request the server to stop the running service, and remove any temporary or resulting files created during the execution of a service running asynchronously. It was previously observed that for the GetStatus requests, only one host was able to effectively stop the running service and to remove the files created by the WPS service (the host which effectively executes the Service).

Further, we introduced a new implementation of the ZOO-Kernel where all the asynchronous requests pass through a *RabbitMQ messaging system*, thus giving the capability to the ZOO-Kernel to be more restrictive in the way it accepts new services. As a result, the only limit in the number of asyncthonous requests that can be executed comes from the number of process the OS supports. Another issue we addressed concerned the parsing of all service configuration files for every GetCapabilities request. This process was very intensive for the ZOO-Kernel, and optimized by creating a metadata catalog which caches metadata information during the ZOO-Kernel launch time.

As a result, the kernel now acts as a standalone server. It starts by parsing all the ZCFG files, the location where the specific service metadata is stored, and then fork the process by a number of times defined in the main configuration file of the server, creating two different pools of server instances. The first pool uses a UNIX domain socket where all the *synchronous* requests are forwarded to Apache. The second pool handles asynchronous requests, which are forwarded to the RabbitMQ message queue. Once all the service configuration files (ZCFG files) are parsed, their metadata become available in the metadata registry for all instances of the ZOO-Kernel, thus speeding up the GetCapabilities and DescribeProcess requests.

In a similar manner, the Execute requests use the cached metadata information by removing the latency of parsing the metadata files. As

presented earlier, at the initialization step, the ZOO-Kernel will fork a certain number of times. One pool of servers will be used to handle synchronous requests and another pool for asynchronous requests. In a situation where a server responsible for an asynchronous request fails for any reason, the main ZOO-Kernel will notice that the forked instance disappeared and will create a new instance to make sure that the number of running instances is the same with the number defined in the configuration file.

Another important improvement is improved *resiliency*, in case the main server instance is restarted. A WPS request is removed from the queue only when the service run was *complete*, no matter if the service run successfully or failed. Even after a reboot, the server will be able to run all the requests which are still present in the queue. Moreover, the RabbitMQ queues can be accessed by various hosts, so when a host is in reboot state, one can expect that another host will treat the requests in the queue. Indeed, if the main ZOO-Kernel is able to create new RabbitMQ queues, we can start up more kernels in a mode which is handling only the synchronous or asychronous requests, so we end up with multiple ZOO-Kernel instances running on various hosts that can access the same message queue. This leads to a much improved scalability of the ZOO-Kernel.

2.3 NEW WPS SERVICES

With the 1.5.0 version of the ZOO-Project a big number of new services are released. One may notice that for the new services, it is not required *to add any Service specific code*.

The goal of the ZOO-Project, from its inception, was to support as much external programs without requiring the end-user to have knowledge of the external programs. So rather than directly implementing new services, we implemented *new internal code* to support direct integration of external programs.

In this manner, we focused in supporting the *OrfeoToolBox (OTB)* which is mainly designed for image processing, but also offers some vector operations. We decided to first focus on OTB due to the way it deals with "OTB applications", offering a cataloging system giving the capability to a C program to browse its activated capabilities. Obviously, we did not abandon the ZOO-Kernel internal mechanisms and it still requires to have access to configuration files (ZCFG) defining the metadata information relative to a specific service. In this case though, rather than expecting the

administrator to have the responsibility to write the configuration file as it should be for any new service implementation, the metadata definition files are automatically generated by a tool named "otb2zcfg" which is now included in the ZOO-Project code. Once the ZCFG files are generated, all the corresponding applications can be accessed as a WPS service through the ZOO-Kernel server. There is no need for any source code for the OTB service to run. Instead of loading a shared library, the OTB serviceType is supported in the ZCFG file, making the ZOO-Kernel be aware that it should load an OTB application. This means, that any OTB developer can create a new application and expose it directly as a WPS service without more effort than running the otb2zcfg converter to automatically create the corresponding ZCFG files.

An important addition to the OTB support is to use the internal OTB mechanisms to access the run-time information of an OTB application. The GetStatus request can provide up-to-date information of the ongoing services, so those are used directly from the ZOO-Kernel and forwarded to the WPS server instances. This way, messages provided by the OTB applications can be accessed through any WPS server.

The new OTB support adds more than 70 new services to the ZOO-Project. In the following table we present a brief list of services and their short description.

Service Name	Description
BandMath	Performs a mathematical operation on single band images
BinaryMorphologicalOperation	Performs morphological operations on an input image channel
BlockMatching	Performs block-matching to estimate pixel-wise disparities between two images
ClassificationMapRegularization	Filters the input labeled image using Majority Voting in a ball shaped neighborhood.
CompareImages	Estimator between 2 images.
ComputeConfusionMatrix	Computes the confusion matrix of a classification
ComputePolylineFeatureFromImage	This application compute for each studied polyline, contained in the input VectorData, the choosen descriptors.
ConcatenateImages	Concatenate a list of images of the same size into a single multi-channel one.
DimensionalityReduction	Perform Dimension reduction of the input image.
EdgeExtraction	Computes edge features on every pixel of

	the input image selected channel
GrayScaleMorphologicalOperation	Performs morphological operations on a grayscale input image
KMeansClassification	Unsupervised KMeans image classification
KmzExport	Export the input image in a KMZ product.
LineSegmentDetection	Detect line segments in raster
Segmentation	Performs segmentation of an image, and output either a raster or a vector file. In vector mode, large input datasets are supported.
LSMSSegmentation	Second step of the exact Large-Scale Mean-Shift segmentation workflow.
MeanShiftSmoothing	Perform mean shift filtering
OpticalCalibration	Perform optical calibration TOA/TOC (Top Of Atmosphere/Top Of Canopy). Supported sensors: QuickBird, Ikonos, WorldView2, Formosat, Spot5, Pleiades, Spot6. For other sensors the application also allows to provide calibration parameters manually.
OrthoRectification	This application allows to ortho-rectify optical images from supported sensors.
Quicklook	Generates a subsampled version of an image extract
Rasterization	Rasterize a vector dataset.
Rescale	Rescale the image between two given values.
Smoothing	Apply a smoothing filter to an image
SOMClassification	Unsupervised Self Organizing Map image classification.

A first example of using the OTB support is to perform an *unclassified KMeans* on a small extract of a Landsat 8 scene by invoking the KMeansClassification service. It requires the following 3 inputs parameters to be set:

- in: the input image file
- out: the output image pixel type (uint8, uint16, int16n, int32, float or double)
- nc: the number of classes

Using the following url we may invoke the KmeansClassification service; the resulting ResponseDocument will provide urls to access the resulting resources, while the resulting tiff file is shown later in this document.

```
http://localhost/cgi-
bin/zoo_loader.cgi?request=Execute&service=WPS&version=1.0.0&Identif
```

```
ier=OTB.KmeansClassification&DataInputs=in=Reference@xlink:href=http
://geolabs.fr/dl/Landsat8Extract.tif;out=float;nc=25&ResponseDocumen
t=out@mimeType=image/tiff@asReference=true;outmeans@mimeType=text/pl
ain@asReference=true
```

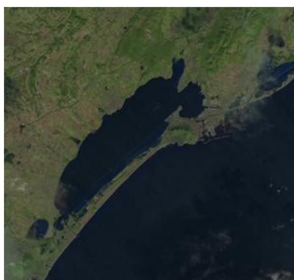
A second sample use case of the OTB support is to apply an *anisotropic diffusion smooth filter* to the same extract of a Landsat 8 scene. For this purpose, we will invoke the Smoothing service:

- in: the input image file
- out: the output image pixel type
- type: the smoothing type (mean, gaussian or anidif)
- type.mean.radius: the smoothing radius value in pixels (used in case the mean type was selected)
- type.anidif.timestep: the Anisotropic Diffusion equation time step (used in case the anidif type was selected)
- type.anidif.nbiter: the number of iterations which controls the sensitivity of the conductance term (used in case the anidif type was selected).

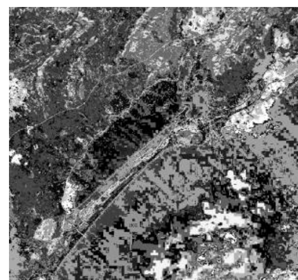
To Run the Smoothing service, setting the type.anidif.timestep parameter to 0.1 and type.anidif.nbiter to 5, we may use the following url:

```
http://localhost/cgi-
bin/zoo_loader.cgi?request=Execute&service=WPS&version=1.0.0&Identif
ier=OTB.Smoothing&DataInputs=in=Reference@xlink:href=http://geolabs.
fr/dl/Landsat8Extract.tif;out=float;type=anidif;type.mean.radius=2;t
ype.anidif.timestep=0.1;type.anidif.nbiter=5&RawDataOutput=out@mimeT
ype=image/tiff
```

Both input and result of the first and second use cases are presented below.



Input Landsat image



KMeans Classification

Smoothed image

The second platform that was integrated in the ZOO-Kernel is SAGA-GIS. As in the OTB case, the specific “SAGA” serviceType was added to the ones supported by ZOO providing the ZOO-Kernel all the information needed to execute a SAGA-GIS program.

In fact, for supporting SAGA-GIS, more than one executables are needed: some utilities import the data in the format expected by SAGA-GIS and then the main executable handles the main processing. In similar way, once the

main application has been executed, some export utilities are needed from SAGA-GIS to export the results in the expected format.

For instance, when the default output of a SAGA-GIS program is a shapefile but the WPS client asked for a GML output, then the ZOO-Kernel will invoke the specific “io_ogr” command to request the output to be converted into the expected output format. In case of a raster file, then the “io_gdal” will be used.

The implemented SAGA-GIS support offers more than 300 services dealing with both Raster and Vector input and output. Furthermore, the SAGA-GIS support in ZOO offers the capability to deal with LAS files, which are a popular topic nowadays. The ZOO-Kernel has been modified to support this kind of files despite the lack of any standards available yet for it. The support of LAS files in SAGA-GIS is using the old libLAS library which should now be replaced by the new LASlib.

The metadata files (ZCFG) corresponding to the SAGA-GIS services are stored in various directories corresponding to the libraries included in the SAGA-GIS software. These libraries give access to the different services and are identified by a number. In SAGA-GIS, we may invoke an application by using the following command: “saga_cmd ta_lighting 0”, meaning calling the tool 0 from the ta_lighting (*ta stands for Terrain Analysis*) library, in the ZOO-Project, this correspond to accessing the SAGA.ta_lighting.0 service.

We briefly list here the supported libraries.

- grid_analysis
- grid_calculus
- grid_filter
- grid_tools
- grid_visualisation
- imagery_classification
- imagery_photogrammetry
- imagery_segmentation
- imagery_svm
- imagery_tools
- pointcloud_tools
- shapes_grid
- shapes_lines
- shapes_points
- shapes_polygons
- shapes_tools
- sim_ecosystems_hugget
- sim_erosion
- sim_fire_spreading
- sim_hydrology
- statistics_grid
- statistics_kriging
- statistics_points

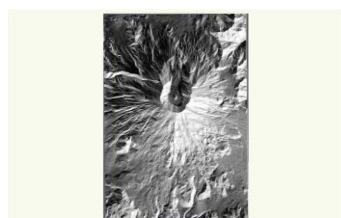
- statistics_regression
- ta_channels
- ta_compound
- ta_hydrology
- ta_lighting
- ta_morphometry
- ta_preprocessor
- ta_profiles
- ta_slope_stability

The spectrum covered by the SAGA-GIS libraries is so large that it is difficult to summarize them. In the following, we present the resulting dataset of SAGA.shapes_points.12 and SAGA.shapes_points.16 which correspond respectively to convexhull/minimum rectangle envelope and Thiessen polygons computation.

Convexhull and rectangle envelope

Thiessen polygons

In the three figures bellow we present results of raster processing using the following SAGA-GIS services: SAGA.lightning.0, computing hillshade, SAGA.hyrdology.2, computing catchment areas and SAGA.tin_tools.1 generating Triangle Irregular Network (TIN).



Hillshade

Catchment areas

TIN edges

Additionally, the last enhancement to the WPS platform was to improve the level of support provided for GRASS GIS. Initially, it was planned to integrate directly GRASS to the ZOO-Kernel but we decided to enhance the

already available WPS-GRASS-Bridge by giving it the capability to upgrade to the new stable GRASS 7.

The old WPS-GRASS-Bridge code relied on an old version of PyXB to generate ZCFG files. So the first goal was to generate the ZCFG file by a new method. This resulted in the publication of a specific shell script developed in conjunction to an XSL transformation file which is used to extract relevant metadata from the DescribeProcess document which can be generated from the GRASS applications directly. By combining the generic option --wps-process-description provided by GRASS programs with the XSL file, one is able to generate the corresponding ZCFG file. Even if it is possible to generate a ZCFG file individually for a specific grass command, the ZOO-Project provides a simple shell script which is able to generate the ZCFG files for every service.

In contrast to the previous platforms supported by the ZOO-Kernel, the GRASS implementation does not expose any catalogue of programs available which can be used to generate all the ZCFG files. So the implemented script method will simply check for all commands available in the GRASS binary directory and will generate the corresponding ZCFG file. The ZCFG files generated by the grass2zcfg tool take into account all the available options for installed GRASS programs.

The WPS-GRASS-Bridge was initially made to work with all the WPS implementations. Our enhancements only focused on the ZOO-Project support and we have not tested other WPS implementations. The shell script created to generate the zcfg files is available directly from the ZOO-Project repository. The grass2zcfg tools are not available in the 1.5.0 release, but they will be integrated in a future release. The grass2zcfg tool produces hundreds of new metadata files for services for vector and raster processing and it produces the required python scripts for each GRASS binary.

Thanks to the work made within this period of the PublicaMundi project, the ZOO-Project integrated more services than ever before. Those services are invoked by the ZOO-Kernel directly through APIs transparently, like OTB, SAGA-GIS and GRASS. Third party applications can be integrated by using the ZCFG generator presented in this section to keep the list of services synchronized with the installed software on a production system.