Appl Geomat (2013) 5:19–24
DOI 10.1007/s12518-011-0070-0

ORIGINAL PAPER

ZOO-Project: the open WPS platform

Gérald Fenoy · Nicolas Bozon · Venkatesh Raghavan

Received: 3 January 2011 / Accepted: 30 November 2011 / Published online: 11 January 2012
© Società Italiana di Fotogrammetria e Topografia (SIFET) 2012

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 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 given in order to detail the ZOO-Project development background and goals. Then, the ZOO itself will be presented, focusing on its advantages and limitations, foremost to highlight the new opportunities provided by such a platform. The ZOO-Kernel and its architecture will be first examined, before further explanations on the proposed method for Web services creation are given. Then the ZOO JavaScript API that provides a new way to orchestrate and chain Web services through the server-side JavaScript will be presented next. Both Kernel and API are illustrated and documented through different Web service code snippets that are available online. Some visual examples of client-side interactions are presented.

G. Fenoy (✉)
GeoLabs SARL,
Futur Building I 1280, Avenue des Platanes,
34970 Lattes, France
e-mail: gerald.fenoy@geolabs.fr

N. Bozon · V. Raghavan
Graduate School for Creative Cities, Osaka City University,
3-3-138 Sugimoto, Sumiyoshi-ku,
Osaka 558-5858, Japan

N. Bozon
e-mail: nicolas.bozon@gmail.com

V. Raghavan
e-mail: raghavan@media.osaka-cu.ac.jp

Keywords Web processing service · Open geospatial consortium · ZOO-Project · Open source geospatial foundation

Introduction

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 (CEU 2007) 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 technologies), is leading to a growing public and governmental awareness on the necessity of using standards for Web-based Spatial Data Infrastructures (SDI).

Numerous applications are available today to store and spread spatial data over the Internet, using Web Map Services (WMS), Web Feature Services (WFS) and Web Coverage Services (WCS). Many open source or proprietary GIS solutions now supports standards to access or modify the data (Bocher 2009), but only a few are available for processing such data through Web Processing Service (WPS).

Some WPS fundamentals and a review of the existing implementations are introduced in the first section, in order to explain the ZOO-Project development context, its goals, advantages and limitations. Sections 2 and 3 then present the ZOO-Kernel and ZOO-API architecture, and aim to detail

the method for setting up Web Services through simple examples.

Web processing service

WPS is one of the most recent interoperability standards published by Open Geospatial Consortium (OGC). It was first proposed under version 0.4 in 2005 (OGC 2005), and some improvements were added in version 1.0.0, which was released in 2007 (OGC 2007a, b).

WPS is designed to standardize the way that GIS algorithms are made available through 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 the Simple Object Access Protocol (SOAP) and Web Services Description Language (WSDL).

There are three mandatory requests that can be submitted to a WPS server: *GetCapabilities*, *DescribeProcess* and *Execute*. GetCapabilities first provides an Capabilities document containing important metadata information about the WPS Server instance (as other OGC Web Services listed before) and a list of the services available on the server-side presented as a unique identifier, a title and a short description. Next, DescribeProcess provides more detailed information on one or more services, containing the necessary input data, the targeted output data format, as well as a title and a short abstract for each of these entries.

Once all the necessary parameters are gathered from the DescribeProcess request, the processing task can be submitted to the server using the Execute request. The latter can answer directly to the client by returning the created output or storing it as a Web accessible resource. The result provided could be an ExecuteResponse document or the data itself depending on the presence in the request of the RawDataOutput or ResponseDocument parameter. As the service could require time to produce an answer, WPS servers should be able to return directly an ExecuteResponse document without processing data. In such a case, the processing task will be executed as a background task and the Server will provide an ExecuteResponse document containing the URL location of future status document update, and notifies if the processing task was accepted or rejected (e.g., if the maximum number of running services was reached). Thus, the client can request the next ExecuteResponse document until the process completion was indicated by the “ProcessSucceeded” or “ProcessFailed” node. While the Execute request is active, the progress of a process can be followed continuously using the “percentCompleted” attribute of the “ProcessAccepted” node or the potential “ProcessSucceeded” and “ProcessFailed” statements. In case of “ProcessSucceeded”, the ExecuteResponse includes either the generated outputs or the URL indicating the physical location where from the output data could be accessed.

This short presentation of WPS shows the three key requests along with their respective responses, which cover the essential part of the OGC’s specification.

Existing WPS implementations

As already noted, the WPS specification is rather new and implemented only by a few GIS applications, and subsequently is supported by very few GIS clients. One can note that most of the available WPS implementations are using the Java language:

- **Deegree** framework is a Java-based environment which implements most of OGC standards and has implemented WPS quite early (Fitzke 2004).
- **WPSint** is a Java plug-in for Spring GIS software and implements WPS 0.4.0 in a Java/JEE application framework (Schut 2005).
- **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 an UDig software client to interact with 52° North WPS (Foerster 2007).
- **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 WPS in Python. It also proposes a native support for GRASS-GIS python scripting. This project was initiated in 2006 and is now in the Open Source Geospatial Foundation’s (OSGeo) incubation process (Cepicky 2009).

Other proprietary implementations may have been carried out, but details are not published; hence, 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 adding WPS support to Arc GIS Server (Sankaran 2011).

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 has now been announced. We can also notice that every WPS solutions is language-dependent, meaning that Web Services on the server have to be coded in the same language as the Server core code, so in Java or in Python. This is a limitation to the use of WPS.

As an open source project too, ZOO-Project aims to provide an alternative implementation by proposing new opportunities that promotes WPS and make it simpler to implement. Despite being a nascent project, it supports

several programming languages and provides an original approach to setup reliable and powerful WPS servers. The ZOO-Project's architecture and functioning are described in the next sections.

ZOO-Kernel

Background

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

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

Architecture

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

ZOO-Services Provider

A ZOO-Services Provider is a couple of Services Shared Objects (SSO) and one metadata ZOO configuration file (.zcfg) per provided service. The ZOO Configuration file contains all the metadata information and obtains the name corresponding to the service's identifier. The SSO is the file containing the function corresponding to the related process.

The Services Provider was conceptualized this way in order to simplify the treatment for the GetCapabilities and DescribeProcess Requests. Indeed, ZOO-Kernel only has to parse the ZCFG file to answer to those two requests. The parser implemented currently use Flex and Bison technologies. Flex is a tool for generating scanners: programs which recognized lexical patterns in text (http://flex.sourceforge.net/). Bison is a general-purpose parser generator that converts a grammar description for a context-free grammar into a C program to parse that grammar (http://www.gnu.org/software/bison/). A specific syntax was then defined for the configuration file which will make the ZOO-Kernel able to check if all required inputs were provided in the request. If an error occurs due to a wrong configuration file or a missing parameter in the request, an Exception XML Document as defined by the OGC's WPS standard will be produced. Flex and Bison are widely used by the open source geospatial community but suffer from the need to define a new grammar for reading such a file. A more robust near-future alternative currently being considered is to use the YAML format (http://www.yaml.org/) to more easily define the service metadata.

Regarding the SSO file, its format depends on the language used to implement the service. For compiled languages such as C and FORTRAN the file will be a shared library, for interpreted such as Perl, Python, PHP and JavaScript the file will be the script to run using the specific interpreter, for Java Language it is a Class file to run in the Java Virtual Machine (Lindholm and Yellin 1999). This SSO file will be dynamically loaded on demand by the ZOO-Kernel. If all the required inputs were specified in the request, the internal function corresponding to the service will then be executed. When running a service from an SSO, ZOO-Kernel will pass by reference inputs, outputs and an internal environment as arguments to the corresponding function, using an internal representation specific to each supported language. Passing values by reference to the Service function ensures that each modifications of this values made in the function's body can be then accessed from the ZOO-Kernel after the call. The service function should then return an integer value representing the processing success or failure statement. On success, ZOO-Kernel will get the resulting value from the outputs argument passed earlier to the function to return the result in the desired format.

Multi-languages

ZOO-Services can be written natively in C and Python language. The Python interpreter was embedded into ZOO-Kernel that allows use of existing Python modules as ZOO-Services.

PHP (embedded version), Java, FORTRAN and JavaScript are optional languages and compilation options must be defined, after specific dependencies have been installed (PHP embedded, Java SDK, F77 and SpiderMonkey).

This variety of supported languages allows the WPS Web Service end-developer to choose his/her 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.

Using OSGeo libraries from WPS Services with the ZOO-Kernel

The ZOO-Project initial idea was to build a platform able to connect the numerous OSGeo libraries together and to use

them as Web Services. The Web Services creation was first tested 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. Indeed, as ZOO-Kernel is able to load dynamic libraries, only a few modifications were needed in the original code. This is illustrated by looking at the well known ogr2ogr code and the corresponding .zcfg file on the ZOO-Project Trac (Fenoy 2010a). Using the OGR code base again, we have also setup a WPS Service for single and multiple geometries spatial operations (Fenoy 2010b). Similar work was applied to the GDAL code base in order to implement the gdalgrid and gdaltranslate capabilities as WPS. Those Web Services allow users to convert, reproject and process both vector and raster data online in a standardized way.

GRASS GIS

Some other experiments have been carried out to interact with GRASS GIS, which provides advanced GIS processing algorithms (Neteler 2008). GRASS 7 now provides a WPS process description exporter, which returns XML documents describing the GRASS functions (Gebbert 2009). This is very useful as a bridge to GRASS, and ZOO-Kernel can take advantage of this GRASS outputs. Thus, a GRASS XML to ZOO configuration file (.zcfg) converter was developed (Gebbert 2010a), 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 2010b). These Python scripts are actually callable as ZOO Web Services and several successful tests have been carried using the r.add, r.div, r.mult and r.sub functions (Gebbert 2010c).

The use of GDAL/OGR and the support for GRASS GIS shown that ZOO-Kernel can use existing libraries as standardized Web Services, with only small 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 statistic analysis and document management.

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 the OpenLayers library (Schmidt 2006). Figure 1 shows a WMS layer (used as input data) on which the user can select a polygon by clicking and then apply a buffer process on it, by calling the OGR-based Web Service cited above (Fenoy 2010b). Other single-geometry vector operation can be performed such as centroïd, convex hull, boundary or simplify. The geometries are quiet simples and light, hence, the results can be rendered using GeoJSON.

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 previously calculated buffer and the second selected polygon.

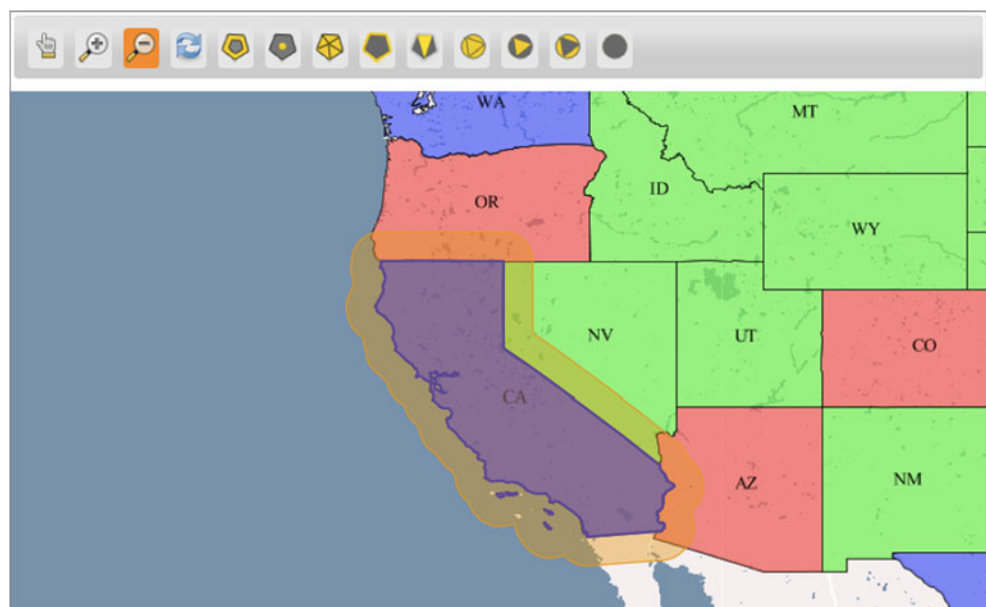

Fig. 1 Example buffer output, using OpenLayers Metacarta WMS and TOPP United States WMS/WFS

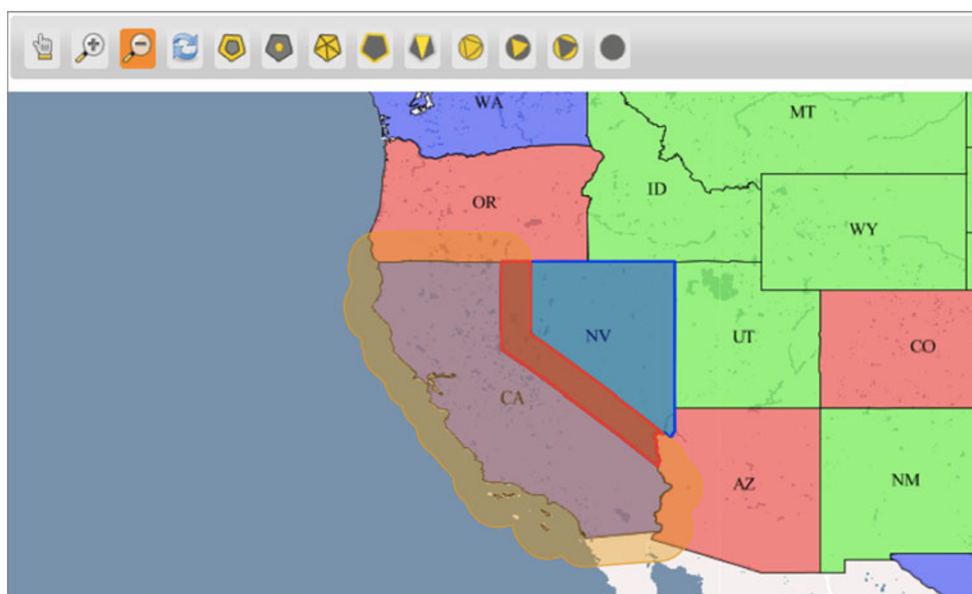

Fig. 2 Example intersection output

The last client-side interaction example is based on the GDAL ExtractProfile function, which allows us to obtain 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 LineString to generate JavaScript elevation profiles on the fly (Fig. 3).

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 client-side library capable to discover and interact with WPS Services yet.

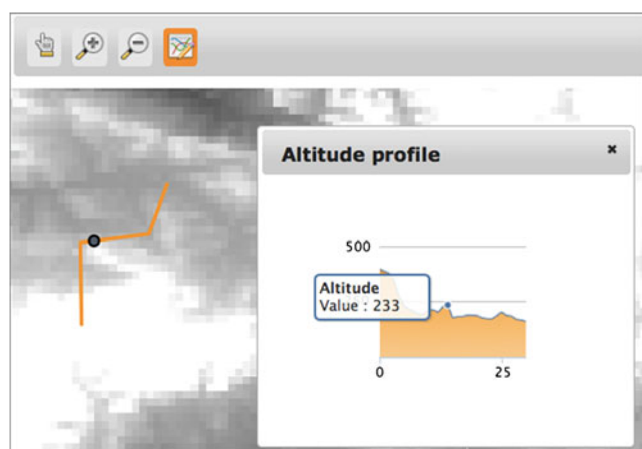

Fig. 3 Example elevation profile using GTOPO30 data set from U.S. Geological Survey

ZOO API

The 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, SpiderMonkey (Mozilla 2010). The API allows orchestration of WPS services using specific methods and offers the ability to add logic and controls in the WPS chaining. It also uses a Proj4js (Adair 2007) adaptation for server-side coordinate re-projection. ZOO-API is based on OpenLayers allowing to easily convert processes outputs into common vector formats (GML, KML, GeoJSON, etc.) when needed.

Server-side Javascript for WPS

Some recent projects provide 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, because JavaScript programming language is very popular and second, it allows users to chain services together to build more complexe services. This is possible by adding the ZOO-API which is a set of 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 vector formats.

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 cartographic projections library. ZOO.Feature and ZOO.Geometry classes, along with their respective subclasses, allow handling the different types of vector data. Finally, a ZOO.Process class was developed to setup input/output, call and chain available WPS processes.

Conclusion

As presented in this article, the ZOO-Project offers an alternative WPS implementation that supports multiple programming languages and simplifies the development of new services as independent modules. Nevertheless, ZOO-Kernel still lack the WSDL and SOAP support which is required for WPS Servers. A solution will be soon integrated in the upcoming enhancement effort to add this specific support. These enhancement efforts will also address necessary developments for compliance with the future WPS 2.0.0 standard specifications. The ZOO-API provides to services developers a way to build complex services using already existing ones by chaining them and adding logic in the chain. Future research will deal with ZOO-Kernel internal enhancements such as YAML support to replace our own grammar for the ZOO Configuration File. Another goal is to provide more services using other popular Open Source libraries such as libLAS (http://www.liblas.org) and Orfeo Toolbox (http://www.orfeo-toolbox.org).

Some promising experiments were already made for integrating the ZOO-Kernel as an XPCOM component. The XPCOM technology is coming from the Mozilla Foundation and is used to produce desktop applications that provide the capability to communicate from a user interface with services implemented in various supported programing languages through the JavaScript Engine. For instance, Mozilla FireFox is an application based on this technology. Thus, once ZOO-Kernel will be used as an XPCOM component, Services will be able to run locally or remotely and developers will have to develop a service only once. Here, the goal is obviously not to redefine the way of interaction with local services. Instead, the key idea is to preserve the definitions of WPS requests and to utilize them by avoiding the use of the HTTP protocol when services are used locally.

Acknowledgments The authors thank Fank Warmerdam for developing and maintaining GDAL/OGR. Thanks also to Soeren Gebbert and Markus Neteler for their active support in the GRASS GIS integration into the ZOO-Project.

References

Adair (2007) Proj4js official website http://proj4js.org
Bocher E (2009) Geospatial free and open source software in the 21st century. In Proceedings of the First Open Source Geospatial Research Symposium, 2009, LNGC. Springer, Heidelberg, in press.
Cepicky J (2009) PyWPS offical Website http://pywps.wald.intevation.org
CEU (2007) Directive 2007/2/EC of the European Parliament and of the Council of 14 March 2007 establishing an Infrastructure for Spatial Information in the European Community (INSPIRE) http://eur-lex.europa.eu/LexUriServ/LexUriServ.do?uri=OJ:L:2007:108:0001:0014:EN:PDF
D'Hont R (2010) ZOO API on ZOO-Project Trac system. http://www.zoo-project.org/trac/browser/trunk/zoo-api/js/ZOO-api.js
Fenoy G (2010a) ZOO-Project website http://zoo-project.org/trac/browser/trunk/zoo-services/ogr/ogr2ogr/cgi-env/Ogr2Ogr.zcg
Fenoy G (2010b) ZOO-Project website http://zoo-project.org/trac/browser/trunk/zoo-services/ogr/ogr2ogr/service.c
Fitzke J (2004) Building SDIs with free software — the Deegree Project. In: Proceedings of GSDI-7, Bangalore, India
Foerster T (2007) http://52north.org/maven/project-sites/wps/52n-wps-webapp/index.html
Gebbert S (2009) GRASS GIS wiki WPS section http://grass.osgeo.org/wiki/WPS
Gebbert S (2010a) GRASS XML to ZOO .zcfg http://code.google.com/p/vtk-grass-bridge/source/browse/trunk/WPS/ZOO_Project/GrassXMLtoZCFG.py
Gebbert S (2010b) ZOO GRASS GIS support http://code.google.com/p/vtk-grass-bridge/source/browse/trunk/WPS/ZOO_Project/ZOOGrassModuleStarter.py
Gebbert S (2010c) ZOO GRASS support tests http://code.google.com/p/vtk-grass-bridge/source/browse/#svn/trunk/WPS/Testing/Python/GrassAddons
Holmes C (2009) OpenGeo Blog http://opengeo.org/products/core development/geoserver/wps
Lindholm T, Yellin F (1999) Java virtual machine specification, 2nd ed. Addison-Wesley Longman Co.
Mozilla (2010) SpiderMonkey JavaScript engine, http://www.mozilla.org/js/spidermonkey/
Neteler M (2008) Springer, Open Source GIS: A GRASS GIS Approach
OGC (2005) Web Processing Service. OGC Discussion Paper, Document Reference Number 05- 007r4, Version 0.4.0
OGC (2007a) Web Processing Service. OpenGIS Standard, Document Reference Number 05-007r7, Version 1.0.0
OGC (2007b) OWS-4 Workflow IPR. Hrsg. OGC. RefNum OGC 06–187; Version 1.0.0; 2008-03-11 Status: internal OGC Discussion Paper
Sankaran S (2011) http://proceedings.esri.com/library/userconf/devsummit11/tech/tech_53.html
Schmidt C (2006) http://www.openlayers.org
Schut P (2005) http://wpsint.tigris.org/
Warmerdam F (1999) http://www.gdal.org