Spatial Statistics on the Geospatial Web

Matthias Hinz
University of Münster/ Institute for Geoinformatics
Weseler Strasse 253
Münster, Germany
matthias.hinz@uni-muenster.de

Daniel Nüst
52°North GmbH
Martin-Luther-King-Weg 24
Münster, Germany
d.nuest@52north.org

Benjamin Proß
52°North GmbH
Martin-Luther-King-Weg 24
Münster, Germany
b.pross@52north.org

Edzer Pebesma
University of Münster/ Institute for Geoinformatics
Weseler Strasse 253
Münster, Germany
edzer.pebesma@uni-muenster.de

Abstract

The Geospatial Web provides data as well as processing functionality using web interfaces. Typical examples of such processes are models and predictions for spatial data, known as spatial statistics. Such analyses are written by domain experts in scripting languages and rarely exposed as web services. We present a concept of script annotations for automatic deployment in server runtime environments and demonstrate it with an implementation based on the open standards and open source components OGC Web Processing Service and R.
Keywords: Geoprocessing, R, spatial statistics, WPS, Geospatial Web

1 Introduction

The Geospatial Web comprises standards and software for a wide variety of applications, ranging from typical GIS (geographic information systems) functionality, via emergency management and monitoring use cases to distributed workflows [27]. Functions include data access and visualization, but also workflows are published as web-accessible resources. Web processing is a well-known concept for distributing processing capabilities on the web, and the OGC Web Processing Service (WPS) [24] standard defines a service interface for this [28].

Spatial statistics is a branch of statistics that focuses on choosing and fitting models and predicting observables from spatial data, and focuses on cases where the spatial locations matter. Spatial statistics adopts different types or representations of spatial and spatio-temporal data, for example points, lines, polygons, grids and trajectories [17]. A substantial part of the applied spatial statistics community works with R [2,21], a language and environment for statistical computing [19]. Other GIS applications use tools for spatial data analysis written in Python scripting language [25]. This means that complete analyses can be communicated, and reproduced, by scripts, along with input data.

Web processes in the Geospatial Web can use a wide variety of input and output data in different data models and encodings (such as GML) and formats (such as NetCDF, XML, but also service interfaces such as WFS). How a processing backend actually carries out the analysis is typically invisible to the user. The motivation for web-based processing comes, on the one hand, from scarcity, for example of computational power, bandwidth or knowledge. Mainstream technologic developments such as grid and cloud computing provide an infrastructure to outsource processes to high performance environments and facilitate the integration of heterogeneous resources [9] and real-time data handling. On the other hand, publishing processes implies sharing capabilities without dependencies on the scripting environment and allows for an open scientific discourse that includes researchers who are not domain-experts or experienced programmers in a particular language [19].

With this work, we investigate how scripted geoprocesses can be integrated into a standardized web service interface. As a solution we describe a generic annotation concept. We apply it to the example of R scripts published in an OGC WPS. It may later on be applied to other scripting languages (e.g. Python) and server environments as well.

2 Related Work

R provides several packages for web service interactions namely Rserve [26], websockets [12], Shiny [23], RWebServices [14] and RevoDeployR [22]. They execute either code blocks or R scripts with defined inputs and outputs. Naturally they can provide geospatial functions which is demonstrated by INTAMAP [18] or pyWPS and

Rpy2[1] [3,13]. However, the lacking feature is the missing explicit support for geospatial data and simple usage patterns.

Standardized geoprocessing service interfaces guarantee vendor independence and additional value through reusability in a service oriented infrastructure, which includes decoupling as well as chaining of service instances into workflows. Several OGC WPS implementations exist both on the server and client side [1,3,6,5,7]. Giuliani et al. promoted WPS based mediation as an approach to distributed computing, namely on grids and clouds. A WPS based mediator can achieve interoperability between different heterogeneous resources, particularly services and pieces of software that act as processing backends, and client applications. Thereby it provides scalable functionalities and is potentially independent to middleware [9].

3 Annotations for Script-based Processes

3.1 Motivation

Script-based geoprocessing addresses various user groups. It involves users and programmers of a scripting environment who want to publish their tools and analyses as web services and share it with other researchers in a cloud environment, to make the tools part of the model web. The web service achieves not only technical interoperability but contextual interoperability, as it mediates between the scripting environment and various client applications. It makes the processing component usable for researchers who do not know the scripting environment. Researches can use such a component within the environment they are familiar with, for instance a GIS application such as ArcGIS or Quantum GIS, that creates analysis tools from web processes. A script-based process may also become part of complex and automated geoprocessing workflows that are composed as service chains. Script programmers may work together with web developers in order to build sophisticated client/server applications using geoprocessing. Thereby web developers develop web service interfaces while script programmers can deploy their script on server side — neither do script programmers need expertise in web development nor do web developers need expertise in scripting.

3.2 Annotations

Scripts collect tasks written in a certain programming language for a specific software environment. As such they are a powerful means to conduct automated processing of data in a reproducible way: all steps are documented. Since scripts are a typical way for analyzing data in many different scientific domains, a generic and seamless solution to deploy scripts on web services opens up the possibility to share and re-use scripts. The goal of annotations is to make special coding for a specific server runtime environment obsolete and to reduce the workload for deploying scripts to uploading a single file – the annotated script. The idea has similarities with practices such as documentation comments for the Javadoc tool [15], where (only) user documentation is generated from source code comments.

A fully automated deployment of script-based processes is not feasible because on the one hand, scripting languages normally use weak typing, and therefore in- and output which is to be exposed must be defined by the script author. Furthermore the type could change at runtime, but a web service interface requires metadata to facilitate discovery and integration by clients, e.g. clear statements of possible inputs, outputs, and resources.

An alternative would be defining conventions, e.g. on parameter naming. It was dismissed because existing scripts should be reusable without significant code changes, because such conventions might not work across scripting languages, and because overloading of variable names with metadata is not practical.

Our annotations serve two purposes: (i) as technical instructions for the web service on how to handle the scripts with respect to requirements, input data and output data, and (ii) as a source for generated process descriptions. The process description is critical for users to decide about fitness for use of the offered functionality and provides a means to abstract from specific runtime environments. It should be noted that users may not know much about the server technology or scripting language actually used. From their perspective, the metadata must suffice to comprehend how the process works.

Annotations are not part of the script instructions so they must occur in source code comments. That has also the advantage that the runtime environment can process the annotations independent of the script environment. In fact, annotations can occur anywhere, but for better comprehension they should be placed close to the code they detail. To improve readability we chose short names for the annotations.

In our examples, we use the hash symbol (#) to start each line, because it is a common comment character in scripting languages, and facultative indentations for clarity.

The annotation syntax is simple: each annotation is a combination of a name and content delimited by a colon (:). The content is a comma separated sequence of fields, which are key and value delimited by an equals sign (=). A semicolon marks the end of an annotation.

The first annotation type provides a general *process description* and occurs once in a script. It is used as followings:

```
# des: id = process, title= “my script”,
#       abstract = “analyze 42 things”,
#       author= “me”;
```

The mandatory *id*-field sets the identifier of the script and must be used by the web service interface to identify the process provided by the script. The fields *title*, *abstract* and *author* are self-explaining and provide a brief textual description of the process and the creator.

The annotations *in* and *out* can occur multiple times in a script. The former is used to declare the *inputs* that should be loaded into the script environment before the script is executed; the latter is used to declare *outputs* that the execution environment should retrieve from the script workspace after the script has been executed and provide to the requesting web client. They are used as follows:

```
  # in: id = myFactor, type = integer,
  #     title = “numerical factor”,
```

[1] http://wiki.rsg.pml.ac.uk/pywps/R-cran

```
#     abstract = "the number to be used
#            for factorization",
#     value = 1,
#     minOccurs = 0, maxOccurs = 1;

# out: id = myResult, type = string,
#     title = "factorized output",
#     abstract = "output number as text in
#     scientific notation (a x 10^b)";
```

Apart from the descriptive and optional fields *abstract* and *title*, they contain the mandatory fields *id* and *type*. *Id* in this case specifies the name of the variable within the script's workspace that contains the input or output value. In case the input or output is a file, the corresponding variable contains the name of the file. The file itself must be stored in the working directory of the script.

The *type* argument declares the data type. Its value is a well-known key that corresponds to (i) a primitive data type, i.e. one of integer, double, boolean or (character) string, or (ii) complex data types and formats, for example list, object and file. The *input* annotation also contains an optional field *value* to declare literal default values.

The optional fields *minOccurs* and *maxOccurs* explicitly state how many times an input can or must occur. A given default value implies *minOccurs=0*. If *maxOccurs* is larger than one, the input can be a list of values.

Some fields are optional and do not have be declared. If the order is strictly followed then the key can be omitted. This results in shorter annotations at the cost of readability. A minimal version of the previous input example is

```
# in: myFactor, integer;
```

For many analyses, it makes sense to store files permanently on a server rather than replicating them for every execution of a script in a working directory. For this purpose we introduced a *resource* annotation that lists the names of necessary resource files:

```
# res: file1.data, file2.zip, file3.txt, ...;
```

These files are managed by the execution environment, e.g. they could be stored in a pre-defined folder.

In addition to the annotations, we introduced the two control flags *off* and *on*. All script lines that follow the flag *off* must be ignored by the execution environment until the annotation, *on* is found:

```
  # off;
  script.part.to = be_ignored_on_server()
  # on;
  script.part.to = run on the server()
```

This allows scripts to be written in a way that they run both locally and on the server. A more flexible way of interaction between server and scripting environment is to insert metadata from the server environment into the script environment. As an example, a boolean variable that indicates a server session allows programmers to write code that is only executed on the server. It may be implemented as

```
if(exists("server") && server == TRUE){
srv.metadata = get server url();}
```

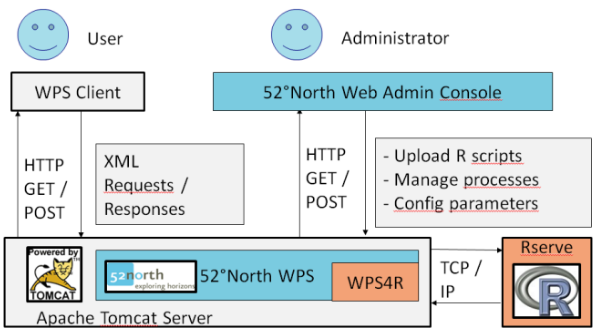
Further adaption to specific server environments is possible by prepending an optional namespace such as *myRuntime* delimited by a dot character (.) to the annotations, so that one script can be annotated for different web servers:

```
# myRuntime.des: id = …
```

4 Application Example: WPS4R

Our annotation framework has been implemented in the WPS4R module for the 52°North WPS [1] and R. The system architecture and user roles are shown in Figure 1. Annotated R scripts are deployed by WPS administrators using an upload form in the Web Admin Console. This console is used to manage the repository of scripts, set configuration parameters such as the script and resource folder, and contain invalid annotations. Rserve [26] can be started automatically on the same machine or a remote instance must be configured.

Figure 1: WPS4R System Architecture



The application example illustrates how WPS client applications utilize the process description generated from the annotations to prepare user interfaces, such as wizards or forms, and how input and output data are seamlessly integrated in the client application.

We deployed an annotated R script on a WPS and executed it from Quantum GIS [20] using its WPS client plugin [5]. The script carries out an inverse distance weighted (IDW) interpolation and depends on the R packages sp [2], gstat [16] and rgdal [11,8]. It requires two inputs: a shapefile containing the measurements and their locations, and a string with the attribute that is interpolated. The EPSG code of the source data is optional input.

The complete script is shown in Figure 2. First, the shapefile is loaded and a target grid covering the observations' bounding box is created. Then, the interpolation function is called. It creates an R data structure which is exported as a GeoTIFF. The script contains annotations for process metadata (*wps.des*), process in- and output (three *wps.in* and one *wps.out*), and execution control (the flags *wps.off* and *wps.on*).

Figure 2: Annotated R script for inverse distance weighted interpolation using WPS

```r
# wps.des: id = idw, title = "Inverse distance weighted interpolation",
#   abstract = "A simple interpolation carried out by R, uses packages gstat, sp and rgdal",
#   author = "Matthias Hinz";

# wps.in: points, application/x-zipped-shp, "Observations",
#   "The point observations and measurements to be interpolated";

# wps.in: attributename, string, "Attribute name",
#   "Name of the attribute to be interpolated, musst match the observations";

# wps.in: epsg, integer, "EPSG code of the observations reference system",
#   "Optional input due to file transfer issues", minOccurs=0;

library(gstat); library(sp); library(rgdal);
# retrieve input from work directory:
#   wps.off;
  points <- "LocalShapeFile.shp"
  attributename <- "LocalAttributeName"
#   wps.on;
set_ll_warn(TRUE)
points=readOGR(points,sub(".shp","",points))
if(exists("epsg")){
  proj4string(points) <- CRS(paste("+init=epsg:",epsg,sep=""))
}

# create target grid:
range=bbox(points)[,"max"] - bbox(points)[,"min"]
cellsize= rep(max(range) / 200, 2)
cells.dim = ceiling(range / cellsize) + 30
cellcentre.offset = bbox(points)[,"min"] - 15 * cellsize
g=GridTopology(cellcentre.offset, cellsize, cells.dim)
raster=SpatialGrid(g, proj4string(points))

#runs interpolation and builds data frame:
form=formula(paste(attributename,"~ 1"))
idw=gstat::idw(form,points,raster)
idw=idw["var1.pred"]

# parse output, return filepath:
output=writeGDAL(idw,"output.tif", drivername="GTiff")
result=paste(getwd(),output,sep="/")

# wps.out: result, image/geotiff, "Interpolated predictions",
#   "A raster file that contains predictions for the observed area";
```

The following XML chunks are part of the process description. Listing 1 shows the process identifier, which is the *id* field of the *des* annotation prefixed with *org.n52.wps.server.r.*. Title and abstract are mapped underneath. Furthermore the description points to metadata such as the source script and the R session info, which contains information about the installed R version and loaded packages. Resources can be downloaded for full reproducibility and transparency.

Listing 1: Excerpt of process description

```xml
[…]
<ProcessDescription statusSupported="true"
  storeSupported="true" wps:processVersion="1.0.0">
  <ows:Identifier>org.n52.wps.server.r.Idw</ows:Identifier>
  <ows:Title>Inverse distance weighted
      interpolation</ows:Title>
  <ows:Abstract>A simple interpolation carried out by R,
      uses packages gstat, sp and rgdal</ows:Abstract>
  <ows:Metadata xlin:title="R Script used for this process"
     xlin:href="http://localhost:8080/wps/R/scripts/Idw.R"
  />
  <ows:Metadata xlin:title="Resource Directory URL"
     xlin:href="http://localhost:8080/wps/R/resources" />
  <ows:Metadata xlin:title="R Session Info"
    xlin:href="http://localhost:8080/wps/R/sessioninfo.jsp"
  />
  <DataInputs>
[…]
```

The XML chunks in show two inputs of the process: the shapefile *points*, whose files must be provided in a zip archive, and the *attributename*. Here the mapping from the *in* annotation to metadata elements is visible.

Listing 2: Excerpt of process description for inputs

```xml
[…]
  <DataInputs>
     <Input minOccurs="1" maxOccurs="1">
       <ows:Identifier>points</ows:Identifier>
       <ows:Title>Observations</ows:Title>
       <ows:Abstract>The point observations and measurements
           to be interpolated</ows:Abstract>
       <ComplexData>
         <Default>
           <Format>
             <MimeType>application/x-zipped-shp</MimeType>
           </Format>
         </Default>
         <Supported>
           <Format>
             <MimeType>text/xml;subtype=gml/3.0.0</MimeType>
             <Encoding>UTF-8</Encoding>
             <Schema>http://schemas.opengis.net/gml/3.0.0/ba
             se/feature.xsd</Schema>
              </Format>
              <Format>
    […]
        <Input minOccurs="1" maxOccurs="1">
          <ows:Identifier>attributename</ows:Identifier>
          <ows:Title>Attribute name</ows:Title>
          <ows:Abstract>Name of the attribute to be
          interpolated, musst match the
          observations</ows:Abstract>
          <LiteralData>
            <ows:DataType ows:reference="xs:string" />
            <ows:AnyValue />
          </LiteralData>
        </Input>
      </DataInputs>
    […]
```

The following XML chunk in Listing 3 shows the output of the process based on the output annotation.

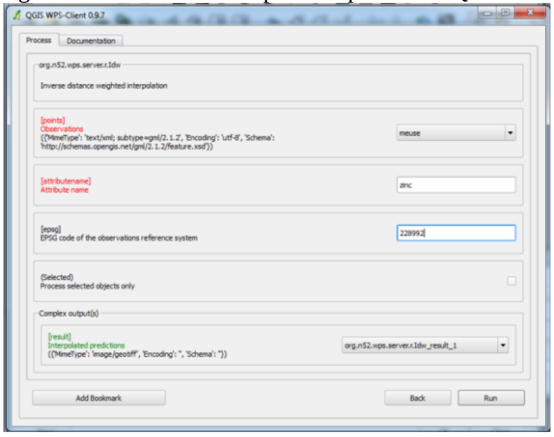
Listing 3: Excerpt of process description: process outputs.

```xml
[…]
<ProcessOutputs>
   <Output>
     <ows:Identifier>result</ows:Identifier>
     <ows:Title>Interpolated predictions</ows:Title>
     <ows:Abstract>A raster file that contains
          predictions for the observed area</ows:Abstract>
     <ComplexOutput>
       <Default>
         <Format>
           <MimeType>image/geotiff</MimeType>
         </Format>
       </Default>
       <Supported>
         <Format>
         <MimeType>application/image-ascii-
            grass</MimeType>
           <Encoding>base64</Encoding>
         </Format>
[…]
```

In Quantum GIS, we imported a shape file based on the meuse data set from the sp package. After starting the WPS wizard the user must fill in the WPS URL and then select the interpolation process from a list of available processes. In the next step the client opens the request form displayed in Figure 6, where the user must enter the input parameters.

By clicking the “Run” button, the client submits an Execute request to the WPS server. The WPS4R module loads the required resources in a temporary directory and executes the annotated R script. After the script is finished, the WPS reads the output from the working directory and returns it to the client. The process result is shown in Figure 4 as a color coded raster overlaid with the input point observations.

Figure 3: Form for the interpolation process in Qantum GIS



QGIS WPS-Client 0.9.7

Process | Documentation

org.n52.wps.server.r.Idw

Inverse distance weighted interpolation

[points]
Observations
({'MimeType': 'text/xml; subtype=gml/2.1.2', 'Encoding': 'utf-8', 'Schema': 'http://schemas.opengis.net/gml/2.1.2/feature.xsd'})
meuse

[attributename]
Attribute name
zinc

[epsg]
EPSG code of the observations reference system
228992

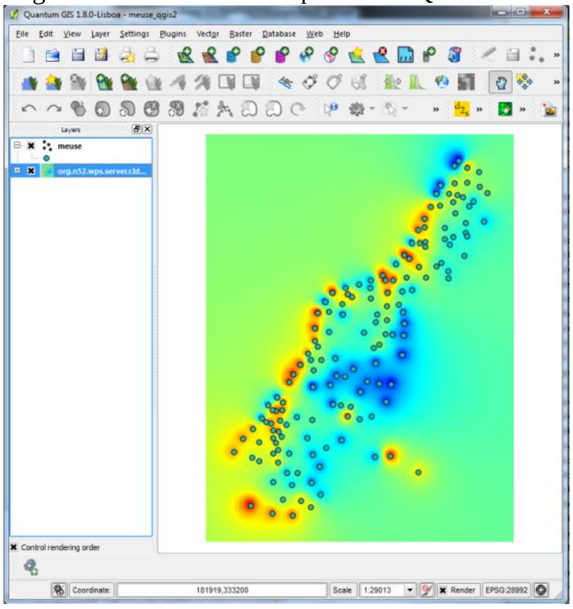
(Selected)
Process selected objects only

Complex output(s)

[result]
Interpolated predictions
({'MimeType': 'image/geotiff', 'Encoding': '', 'Schema': ''})
org.n52.wps.server.r.Idw_result_1

Add Bookmark | Back | Run

Figure 4: Result of the interpolation in Quantum GIS


Quantum GIS 1.8.0-Lisboa - meuse_qgis2
File Edit View Layer Settings Plugins Vector Raster Database Web Help
Layers
meuse
org.n52.wps.server.r.Id...
Control rendering order
Coordinate: 181919,333200
Scale 1:29013
Render
EPSG:28992

5 Discussion

Once an R script can be deployed as a process, statistical computing can be integrated in various GIS clients, service-chains and workflows because of the coupling with WPS. We see a demand of R users to deploy their spatial analyses on the web and a demand of GIS users to enhance their software's functionality with state of the art geostatistics. The main simplification of this solution is that web service development is no longer a technical obstacle for domain scientists, who more typically write R scripts.

We showed that annotating scripts provides a practical solution to making spatial statistics available on the Geospatial Web. Our application example shows how an existing script can be modified without altering its functionality using a simple set of discovery and process metadata. The related work section presents alternatives to expose specifically R scripts; however, none of these reach the interoperability of an OGC WPS implementation. If the interface definition is automated then the granularity is fixed, for example script functions are mapped to web service endpoints. By using annotations the granularity can be chosen by the user and whole workflows as well as simple mathematical functions can be exposed. Also, the annotations define a working API between runtime environment and scripting language. Changes to the script, e.g. written in R, do not require programming in another language.

We discovered the following limitations. The annotations could not be completely mapped to the WPS specification: the author field could not be inserted into the process description. Here one can see improvements of the standard with respect to transactional processes which require information about the creators. Furthermore the Quantum GIS wizard does not map all information from the process description, i.e. it lacks input and output descriptions but only shows the titles.

We did not map complex objects of R to input or outputs, because most of them can be serialized by R itself to interoperable file types. The class SpatialPointsDataFrames from the sp-package for instance can be converted to/from vector and raster formats by using rgdal.

The concepts of WPS and R differ substantially. The WPS targets interoperability for geospatial processes to make web-based processing independent from implementation of server or client components, essentially making processes independent of the execution environment, while R provides both a lingua franca for writing statistical analysis processes as well as the execution environment. For R users, the R language itself and using common packages for data models ensures interoperability. These packages have not been officially standardized but have become de-facto standards through good functionality, maintenance, stability and are often the improved by community feedback. Therefore sharing R processes is easy between R users, but the integration of R algorithms in a standardized process such as defined by the OGC WPS is a challenge.

6 Conclusion and Future Work

The main contribution of this work is a specification of annotations for the most important process metadata to be used within scripting languages for publication of functionality as standardized web services. It is complemented by a comprehensive description of the WPS4R framework, which implements the annotation concept, and demonstrated using existing web processing client and server software.

Our specification depends neither on R nor on WPS, as scripts written in any prevalent computer language can be annotated and likewise integrated into different server runtime environments. Thus it will be useful to test other middleware and client projects and to extend use cases with more data formats and types. However, even this single combination of one scripting language and one processing server already addresses a large user base by allowing R users to share geospatial analyses with other non-R users who work with geospatial data using GIS and/or web services. Furthermore our implementation fosters an open approach to processing in the geospatial web because it facilitates process sharing and reproducibility: all software is open source [21,1] and the process description points to all resources to run the same code in other runtime environments.

Concrete plan for further verification of this approach is the transfer of the INTAMAP functionality [18] to WPS4R. Also the usability can be improved by providing an R extension package for interacting with the server runtime environment from within annotated scripts; by providing an R extension package to upload and execute scripts on a transactional WPS; or by providing tools for writing and validation of annotations in the respective scripting language. Usability will be a key point for adoption by the scientific community.

Open questions comprise measures of *security*. In our example we addressed security issues by access limitation, i.e. uploading R scripts requires administrator rights. Sandboxes may be used to prohibit changes to the server runtime environment using and security mechanisms against illegal operations should be explored. Blacklists for instance provide a basic safeguard, although they can be penetrated by skilled programmers. Further work includes supporting *complex use cases* involving versions of scripts, rights, and provenance information. Here we see potential to extent the minimalistic approach of only a few functionally required metadata elements by evaluating metadata standards such as Dublin Core [4] or ISO 19119 [10].

References

[1] 52°North GmbH. (2012) 52°North WPS. [Online]. http://52north.org/communities/geoprocessing/wps/index.html

[2] Roger S. Bivand, Edzer J. Pebesma, and V. Gómez-Rubio, Applied spatial data analysis with R.: Springer, 2008.

[3] J. Cepicky, "PyWPS 2.0.0: The presence and the future," in Proceedings Geoinformatics FCE CTU 2007, Prague, Czech Republic, 2007.

[4] DCMI Usage Board. (2012) DCMI Metadata Terms. [Online]. http://dublincore.org/documents/dcmi-terms/

[5] H. Düster. (2013) QGIS Web Processing Client. [Online]. http://kappasys.org/cms/index.php?id=10

[6] G. Fenoy, N. Bozon, and V. Raghavan, "ZOO-Project: the open WPS platform," Applied Geomatics, pp. 1-6, 2012.

[7] J. Fitzke, K. Greve, M. Müller, and A. Poth, "Building SDIs with Free Software-the deegree project," in Proceedings of GSDI-7, Bangalore, India, 2004.

[8] GDAL Development Team. (2013) GDAL - Geospatial Data Abstraction Library, Version 1.9.2. [Online]. http://www.gdal.org

[9] Gregory Giuliani, Stefano Nativi, Anthony Lehmann, and Nicolas Ray, "WPS mediation: An approach to process geospatial data on different computing backends," Computers & Geosciences, vol. 47, pp. 20-33, 2012.

[10] ISO/TC 211. (2005) ISO 19119:2005 Geographic Information Services.

[11] Timothy H. Keitt, Roger Bivand, Edzer Pebesma, and Barry Rowlingson. (2012) rgdal: Bindings for the Geospatial Data Abstraction Library. [Online]. http://CRAN.R-project.org/package=rgdal

[12] Bryan W. Lewis and Jeffrey Horner. (2012) websockets: HTML 5 Websocket Interface for R. [Online]. http://CRAN.R-project.org/package=websockets

[13] Walter Moreira, Gregory R. Warnes, and Laurent Gautier. [Online]. http://rpy.sourceforge.net/

[14] Martin T. Morgan, Nianhua Li, Seth Falcon, and Robert Gentleman. (2007) Enabling R packages for web or grid services. [Online]. http://www.bioconductor.org/packages/2.12/bioc/vignettes/RWebServices/inst/doc/EnablingPackages.pdf

[15] Oracle Corporation. (2013) How to Write Doc Comments for the Javadoc Tool. [Online]. http://www.oracle.com/technetwork/java/javase/documentation/javadoc-137458.html

[16] Edzer J. Pebesma, "Multivariable geostatistics in S: the gstat package," Computers & Geosciences, vol. 30, pp. 683-691, 2004.

[17] Edzer Pebesma, "spacetime: Spatio-Temporal Data in R," Journal of Statistical Software, vol. 51, no. 7, pp. 1-30, 2012.

[18] E. Pebesma et al., "INTAMAP: the design and implementation of an interoperable automated interpolation web service," Computers & Geosciences, vol. 37, no. 3, pp. 343-352, 2011.

[19] Edzer Pebesma, Daniel Nüst, and Roger Bivand, "The R software environment in reproducible geoscientific research," Eos, Transactions American Geophysical Union 93, vol 16, pp. 163-164, 2012.

[20] Quantum GIS Development Team. (2013) Quantum GIS Geographic Information System. [Online]. http://qgis.osgeo.org/

[21] R Core Team. (2013) R: A Language and Environment for Statistical Computing. [Online]. http://www.R-project.org/

[22] Joseph B. Rickert. (2010) R for Web-Services with RevoDeployR. Version 1.0. [Online]. http://info.revolutionanalytics.com/RevoDeployR-Whitepaper.html

[23] RStudion and Inc. (2012) shiny: Web Application Framework for R. [Online]. http://CRAN.R-project.org/package=shiny

[24] Peter Schutt. (2007) OpenGIS Web Processing Service. [Online]. http://www.opengeospatial.org/standards/wps

[25] Lauren M. Scott and Mark V. Janikas, "Spatial Statistics in ArcGIS," in Handbook of Applied Spatial Analysis., 2010, pp. 27-41.

[26] Simon Urbanek, "Rserve -- A Fast Way to Provide R Functionality to Applications," in Proceedings of the 3rd International Workshop (DSC 2003), 2003.

[27] Pheisheng Zhao and Liping Di, Geospatial Web Services: Advances in Information Interoperability.: IGI Global, 2011.

[28] Peisheng Zhao, Feng Lu, and Theodor Foerster, "Towards a Geoprocessing Web," Computers & Geosciences, vol. 47, pp. 1-2, 2012.