

Treball de Final de Grau/Màster / Trabajo de Final de Grado/Màster

TÍTOL / TÍTULO / TITLE

WEB PROCESSING SERVICES FOR FORESTRY AND ENVIRONMENTAL
APPLICATIONS

Autor/a / Autor/a/ Author: CONSTANCIO AMURRIO GARCÍA

Director/a / Director/a/ Supervisor: ÓSCAR BELMONTE FERNÁNDEZ
Tutor/a o supervisor/a / Tutor/a o supervisor/a/ Co-supervisors:
CHRISTOPH STASCH, MÁRIO CAETANO

Data de lectura / Fecha de lectura/ Date of Thesis Defense:

MARCH, 6th 2014



Resum / Resumen/ Abstract:

Nowadays spatial processing on the web is becoming a requirement for more and more web applications. The use of processes helps to find solutions to a wide range of spatial problems and extends the common functionality of Web GIS. There are many open source technologies that can be implemented in each component of a Web GIS application. Forestry and environmental problems, with their strong territorial implications, are especially suitable to be analysed applying these technologies.

In order to create an application with spatial processes, we propose a framework with a layered service-based architecture. It is layered because its structure is divided in a set of functional layers: the user layer (geoportal or client), the service layer (inside the server) and the data layer (spatial database). The access and processing of spatial data is accomplished through adequate service standards of OGC (Open Geospatial Consortium): Web Map Services (WMS), Web Feature Services (WFS), Web Coverage Services (WCS) and Web Processing Services (WPS).

We implement a complete forestry – related application from scratch that offers access, visualization, querying and processing of spatial data and an active user interaction. The key of the application is WPS. Additionally, other processing solutions (like making queries with the spatial database) are discussed.

In brief, this work presents an overview of the current technology and possible solutions for integrating spatial processes on the web and proposes some guidelines to implement them in a fully working system.

Paraules clau / Palabras clave/ Key words: Web GIS Applications, Forestry Applications, Environmental Applications, Spatial Process, Geographical Information System, Web Services, Web Processing Service, Web Map Service, Web Feature Service, Web Coverage Service, LiDAR, GeoServer, Postgresql, Postgis, OpenLayers, HTML, CSS, Ext JS, GeoExt, JavaScript, Java.

Masters Program in **Geospatial Technologies**



WEB PROCESSING SERVICES FOR FORESTRY AND ENVIRONMENTAL APPLICATIONS

Constancio Amurrio García

Dissertation submitted in partial fulfilment of the requirements
for the Degree of *Master of Science in Geospatial Technologies*

WEB PROCESSING SERVICES FOR FORESTRY AND ENVIRONMENTAL APPLICATIONS

Dissertation supervised by

PhD Óscar Belmonte Fernández

PhD Christoph Stasch

PhD Mário Caetano

March 2014

ACKNOWLEDGMENTS

I would like to express my gratitude to PhD Óscar Belmonte, PhD Christoph Stasch and PhD Mário Caetano for the time dedicated to helping me and to reviewing this document.

Special thanks to all my colleagues for making me spend this great time during last year and a half. Thanks to all our teachers in Spain, Germany and Portugal. Of course thanks also to Dori Apanewicz for the continuous support.

Finally, thanks to all the family for the personal and economical support during this time.

WEB PROCESSING SERVICES FOR FORESTRY AND ENVIRONMENTAL APPLICATIONS

ABSTRACT

Nowadays spatial processing on the web is becoming a requirement for more and more web applications. The use of processes helps to find solutions to a wide range of spatial problems and extends the common functionality of Web GIS. There are many open source technologies that can be implemented in each component of a Web GIS application. Forestry and environmental problems, with their strong territorial implications, are especially suitable to be analyzed applying these technologies.

In order to create an application with spatial processes, we propose a framework with a layered service-based architecture. It is layered because its structure is divided in a set of functional layers: the user layer (geoportal or client), the service layer (inside the server) and the data layer (spatial database). The access and processing of spatial data is accomplished through adequate service standards of OGC (Open Geospatial Consortium): Web Map Services (WMS), Web Feature Services (WFS), Web Coverage Services (WCS) and Web Processing Services (WPS).

We implement a complete forestry – related application from scratch that offers access, visualization, querying and processing of spatial data and an active user interaction. The key of the application is WPS. Additionally, other processing solutions (like making queries with the spatial database) are discussed.

In brief, this work presents an overview of the current technology and possible solutions for integrating spatial processes on the web and proposes some guidelines to implement them in a fully working system.

KEYWORDS

Web GIS Applications	GeoServer
Forestry Applications	Postgresql
Environmental Applications	Postgis
Spatial Process	OpenLayers
Geographical Information System	HTML
Web Services	CSS
Web Processing Service	Ext JS
Web Map Service	GeoExt
Web Feature Service	JavaScript
Web Coverage Service	Java
LiDAR	

ACRONYMS

- AJAX** – Asynchronous JavaScript and XML
- API** – Application Programming Interface
- AVP** – Attribute Value Pairs
- BBOX** – Bounding Box
- CGI** – Common Gateway Interface
- CRS** – Coordinate Reference System
- CRUD** – Create, Read, Update and Delete.
- C/S** – Client / Server
- CSS** – Cascading Style Sheets
- CSV** – Comma-Separated Values
- CSW** – Catalogue Services for the Web
- DEM** – Digital Elevation Model
- DOM** – Document Object Model
- ECMA** – European Computer Manufacturers Association
- EFDAC** – European Forest Data Centre
- EFFIS** – European Forest Fire Information System
- EPSG** – European Petroleum Survey Group
- GEO** – Group on Earth Observations
- GEOS** – Global Earth Observation System of Systems
- GIS** – Geographic Information System
- GML** – Geography Markup Language
- GUI** – Graphical User Interface
- HTML** – HyperText Markup Language
- HTTP** – HyperText Transfer Protocol

IGN – National Geographic Institute

IIS – Internet Information System

IP – Internet Protocol

JAI – Java Advanced Imaging

JSON – JavaScript Object Notation

JSP – JavaServer Pages

JSPTL – JavaServer Pages Template Library

JVM – Java Virtual Machine

LIDAR – Light Detection and Ranging

MIME – Multipurpose Internet Mail Extension

OGC – Open Geospatial Consortium

OSM – OpenStreetMap

POM – Project Object Model

REST – Representational State Transfer Protocol

SDI – Spatial Data Infrastructure

SLD – Styled Layer Descriptor

SOA – Service Oriented Architecture

SQL – Spatial Query Language

TCP – Transmission Control Protocol

UOM – Units of Measure

URI – Uniform Resource Identifier

URL – Uniform Resource Locator

WCS – Web Coverage Service

WFS – Web Feature Service

WFS-T – Web Feature Service Transactional

WMS – Web Map Service

WPS – Web Processing Service

XHR– XMLHttpRequest

XML – Extensible Markup Language

WWW – World Wide Web

INDEX

Acknowledgements.....	II
Abstract.....	III
Keywords.....	IV
Acronyms.....	V
1. Introduction	1
1.1. Overview	1
1.2. Thesis objectives.....	7
1.3. Dissertation organization	8
2. State of the Art.....	9
3. Literature review	14
4. Application design	18
4.1. Basics of Web GIS technology	18
4.1.1. Introduction	18
4.1.2. HTTP protocol.....	18
4.1.3. HTML, CSS and JavaScript	21
4.1.4. Server and Client technologies	21
4.1.5. Data exchange formats	22
4.1.6. Web Services	23
4.1.6.1. Web Map Service (WMS).....	23
4.1.6.1.1. Introduction.....	24
4.1.6.1.2. HTTP request rules.....	24
4.1.6.1.3. HTTP response rules	25
4.1.6.1.4. WMS Operations.....	25
4.1.6.2. Web Feature Service (WFS)	28
4.1.6.2.1. Introduction	28
4.1.6.2.2. WFS Operations	29
4.1.6.3. Web Coverage Service (WCS).....	30
4.1.6.3.1. Introduction.....	30
4.1.6.3.2. WCS operations	30
4.1.6.4. Web Services summary	31

4.2. Architecture.....	32
4.2.1. Introduction	32
4.2.2. Architecture Principles	32
4.2.3. Components.....	34
4.2.4. Server – Client Workload distribution.....	36
4.2.5. Architecture summary	37
4.3. Scenario description	37
4.3.1. Introduction	37
4.3.2. Study area.....	38
4.3.3. Features.....	39
4.4. Implementation of the application	41
4.4.1. Technologies.....	41
4.4.1.1. Server technology: GeoServer.....	42
4.4.1.1.1. Introduction	42
4.4.1.1.2. GUI and Functionality.....	42
4.4.1.1.3. Performance and caching	44
4.4.1.1.4. Security considerations	45
4.4.1.2. Database technology: Postgresql / Postgis	46
4.4.1.3. Client technology: OpenLayers.....	48
4.4.1.3.1. Introduction	48
4.4.1.3.2. Procedure to set the application up.....	49
4.4.1.3.3. Coordinate Reference System	50
4.4.1.3.4. Third party mapping APIs.....	51
4.4.1.4. Other technologies	52
4.4.1.4.1. Ext JS	52
4.4.1.4.2. GeoExt	54
4.4.1.4.3. DualMaps	55
4.4.1.4.4. OpenWeatherMap	56
4.4.2. General organization of the application.....	56
4.4.3. Data.....	57
4.4.3.1. Data directory structure.....	57
4.4.3.2. List of layers.....	58
4.4.3.2.1. Base layers	58
4.4.3.2.2. Operational Layers	59
4.4.3.3. Data CRS.....	59
4.4.4. Web Services implementation	60
4.4.4.1. Web Map Service implementation.....	60
4.4.4.2. Web Feature Service implementation	61
4.4.4.3. Web Coverage Service implementation	61

4.4.5. Implementation of the functionality	61
4.4.5.1. Layers Panel.....	62
4.4.5.1.1. Layers Panel description	62
4.4.5.1.2. Layers Panel implementation	62
4.4.5.2. Map Display Panel	63
4.4.5.2.1. Map Display Panel description.....	63
4.4.5.2.2. Map Display Panel implementation	64
4.4.5.3. Toolbar	65
4.4.5.3.1. Toolbar description	65
4.4.5.3.2. Toolbar implementation	67
4.4.5.4. Geo-Processes Panel	69
4.4.5.4.1. Geo-Processes Panel description.....	69
4.4.5.4.2. Geo-Processes Panel implementation.....	70
5. Web Processing Services	74
5.1. Introduction.....	74
5.2. Basics about WPS. The standard	75
5.2.1. WPS operations	75
5.2.1.1. GetCapabilities WPS operation.....	75
5.2.1.1.1. WPS operation request	75
5.2.1.1.2. WPS GetCapabilities response.....	76
5.2.1.2. WPS DescribeProcess	77
5.2.1.2.1. WPS DescribeProcess operation request.....	77
5.2.1.2.2. WPS DescribeProcess response	79
5.2.1.2.3. Types of data inputs and process outputs	80
5.2.1.3. WPS Execute.....	81
5.2.1.3.1. Execute request parameters	81
5.2.1.3.2. WPS exceptions	82
5.3. Implementation of the processes. Results	82
5.3.1. Introduction	82
5.3.2. WPS clients implementations.....	83
5.3.3. Guide for creating and using WPS	85
5.3.4. GeoServer WPS.....	86
5.3.4.1. Using existing processes: Request Builder.....	86
5.3.4.2. Creating new Processes.....	87
5.3.4.2.1. Java-based Processes.....	87
5.3.4.2.2. Scripting-based Processes	89
5.3.5. Developed WPS Processes	89
5.3.5.1. Process 1: Basic Raster Statistics	89

5.3.5.2. Process 2: Buffers	92
5.3.5.3. Process 3: Vegetation description	95
5.3.5.4. Process 4: Road description	99
6. Discussion and conclusions	106
6.1. Discussion	106
6.2. Main conclusions.....	107
7. Further work.....	110
8. References	113

Appendix A. Data.

Appendix B. Code.

INDEX OF FIGURES

Fig. 1.- INSPIRE technical architectural overview (INSPIRE, 2007).....	3
Fig. 2.- Viewshed calculation in Terrasit. Comunidad Valenciana SDI.....	10
Fig. 3.- Buffer tool in Cartoweb (Comunidad Valenciana).....	10
Fig. 4.- View of OTALEX WPS Client.....	11
Fig. 5.- Route calculation using WPS in CartoCiudad	12
Fig. 6.- Spatial Analysis tool of IGN.....	13
Fig. 7.- European Forest Data Centre Viewer	17
Fig. 8.- Example of a GET HTTP request header information.....	20
Fig. 9.- GetCapabilities request response example.....	27
Fig. 10.- GetMap request response example	28
Fig. 11.- GetFeatureInfo request response.....	28
Fig. 12.- WFS POST request.....	29
Fig. 13.- WFS Server response.....	30
Fig. 14.- Architecture of the application. Source: Author.	34
Fig. 15.- Study area. Source: Author.	39
Fig. 16.- GeoServer Administrative Interface	42
Fig. 17.- PostGIS geometry hierarchy (OGC, 2010).....	46
Fig. 18.- Processing with PostGIS. Source: Author.	48
Fig. 19.- Definition of EPSG: 3857.....	51
Fig. 20.- Structure of the GUI using Ext JS BorderLayout. Source: Author.	53
Fig. 21.- DualMaps example	55
Fig. 22.- OpenWeatherMap example	56
Fig. 23.- View of layers panel	63
Fig. 24.- Some images of the editing process.....	64
Fig. 25.- Transparency slider.....	71
Fig. 26.- LiDAR data integration in OpenLayers.....	72
Fig. 27.- Client – Server interactions in a WPS (Schaeffer, 2008).....	75

Fig. 28.- Response of a GetCapabilities document.....	76
Fig. 29.- Simple process description in the capabilities document.....	77
Fig. 30.- Example of DescribeProcess response.....	78
Fig. 31.- Decision tree for creating and using WPS. Source: Author.....	86
Fig. 32.- WPS Request Builder	87
Fig. 33.- Polygon to calculate statistics	92
Fig. 34.- Tools and Results panel	92
Fig. 35.- Buffer tool configuration	95
Fig. 36.- Buffer around a polygon geometry	95
Fig. 37.- National Forest Map	98
Fig. 38.- Vegetation description result table.....	99
Fig. 39.- Structure of the road description process. Source: Author.....	100
Fig. 40.- Roads and polygon to calculate the statistics	105
Fig. 41.- Road statistics result	105
Fig. 42.- General overview of the application	105

INDEX OF TABLES

Table 1.- WMS GetCapabilities request parameters (OGC, 2006)	25
Table 2.- WMS GetMap request parameters (OGC, 2006)	26
Table 3.- WMS GetFeatureInfo request parameters (OGC, 2006)	27
Table 4.- WCS GetCoverage request parameters (OGC, 2006)	31
Table 5.- WPS GetCapabilities request parameters (OGC, 2007)	76
Table 6.- WPS DescribeProcess request parameters (OGC, 2007)	78
Table 7.- WPS response (OGC, 2007)	79
Table 8.- WPS types of data (OGC, 2007)	80
Table 9.- WPS Execute parameters (OGC, 2007)	82
Table 10.- Possible exceptions in an Execute WPS request (OGC, 2007)	82

1. INTRODUCTION

1.1. OVERVIEW

Since the normalization of the Internet use in the early 1990s, fundamental changes have happened in the way businesses are done. In this regard, Geographic Information Systems (GIS) are not an exception. The article “*What is Web 2.0*” (O’Reilly, 2005) explains three main characteristics that define the recent evolution of the web, what the author calls Web 2.0. First of all, the user-generated content. Nowadays, the **workflow starts at the bottom** (users) and comes up to construct the web. Second, the consideration of the web as a platform in which actions are performed using **web services**. Finally, in the Web 2.0 there is a **rich user experience**: the look of the new web applications helps to an easy understanding of their use and calls to the interaction. All these characteristics can be applied to GIS, opening up a world of possibilities. In this document **Web GIS** is understood basically as the use of GIS on the web.

In parallel to web development, **users** have also evolved. As they make use of the web, they also **request more advanced services**, demand higher quality information and become more exigent. In the GIS field, this process has gone together with the publication of web cartography, which increases every year. Nowadays we have a variety of data available for the user in the national and regional Spatial Data Infrastructures (SDIs). The publication usually follows the standards for web services of Open Geospatial Consortium (OGC).

Visualization of geospatial data on the web is accomplished and supported efficiently thanks to the use of Web Map Services (WMS). But what happens when the user wants to go further, and perform analysis? In this case the situation changes drastically. The development of **Web Processing Services** (WPS), services that allow processing on the web, is not so widely spread. The data published by the Spanish SDI illustrates this situation: at this moment (January 2014), 1754 WMS are available but there are only 7 WPS (SDI, 2014). Therefore there is a pending challenge in offering services not only for accessing geospatial data, but also for processing them.

Due to its territorial and geographical connections, forestry and environmental sectors have used GIS techniques from the origin of this technology. The use and analysis of geospatial data is a basic step that is implemented in every project within these fields. The triple function of forestry / environmental sector, that includes natural, economic and social perspectives, normally with an important geographical component, is very suitable to be studied using GIS. In terms of availability of information, the effort made by the Public Administrations in last years has been decisive. But again there is a lack of high quality processing services that allow the user to go a step further and perform some advanced analysis on the existing data.

Bringing processing capacity to the user has some implications if we want to obtain a good user experience. First of all, the processes should be **easy to use**. The audience can be very broad, including those users that want information about forestry / environment but may not know about GIS. The Graphical User Interface (GUI) should hide the system complexity and present only the necessary options to set the tools up. Next, users should **interact with the underlying system**. That means the user is qualified to get different results depending on his or her own inputs. Finally, the system has to **adapt to the user**: a citizen (i.e. a forest land owner) that needs an answer to a simple question or a scientist that wants to implement a complex model, etc.

Web GIS technologies are an engaging option, with many benefits: they do not require installation, the administrators do the updates (the user does not have to worry about that), users can enjoy the potential of the web (as, for example, in sharing contents or user interaction) and the usability is equivalent to desktop-based applications.

The technical basic structure for Web GIS applications is based in a **communication process between a web client and a web server**. Typically a client starts the communication sending a request to a server, the server processes the information and returns an answer back to the client. This is accomplished through the Hypertext Transfer Protocol (HTTP) and this architecture is called client / server (C/S). In this context, **web services** provide the functions that were traditionally deployed in

desktop-based systems: discovery of data, access and visualization, transformation and processing. In order to achieve interoperability, these different services (like map services, data services or analytical services) are implemented using OGC standards.

This project focuses in **developing a Web GIS application** totally based in standard OGC services. The application is used as a client to perform analysis on the web, using Web Processing Services. Its architecture is based on INSPIRE SDI directive (INSPIRE EU Directive, 2007), specifically the INSPIRE Technical Architecture Overview Document (INSPIRE EU Technical Architecture Document, 2007). This document defines a **layered architecture** for Web GIS applications: the **presentation layer**, with the GUI; the **service layer**, that includes these types of web services: registry, discovery, view, download, transference and invoke spatial data services and finally the **data layer**, that manages the spatial datasets, service metadata and registers (Fig. 1).

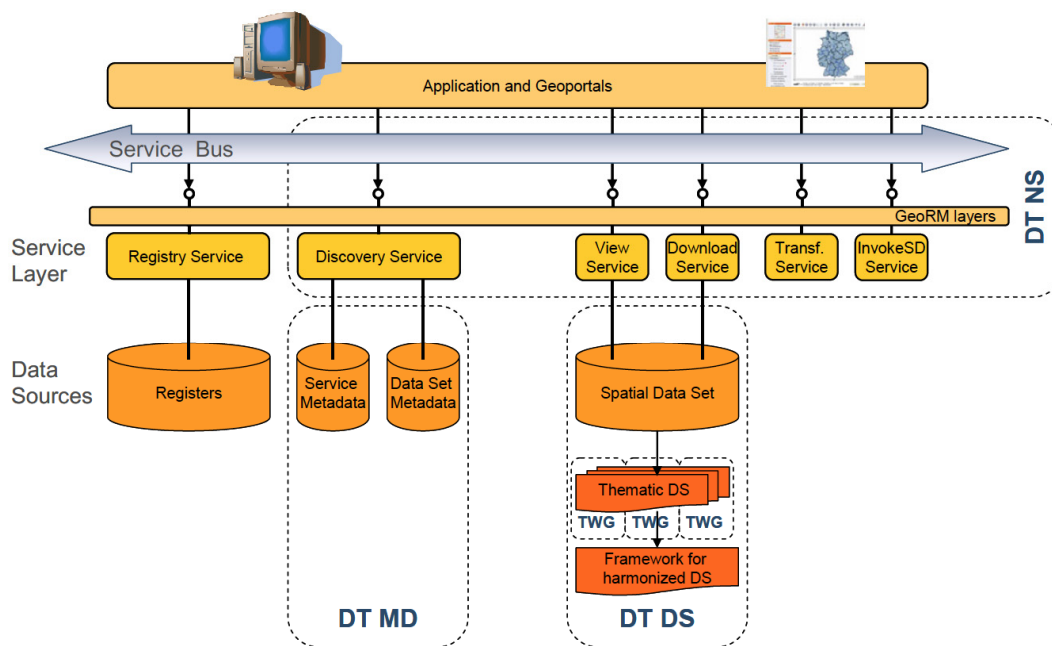


Fig. 1.- INSPIRE technical architectural overview (INSPIRE, 2007)

In our implementation, the presentation layer consists in the creation of a complete GUI deployed on the client side, which is supposed to be a web browser. The client accesses an HTML document that executes JavaScript code, showing up the complete

GUI. The entire client – server logics is managed using **OpenLayers**¹, a powerful mapping library. Services included in the service layer are stored in a geographical server (**GeoServer**²). Finally, the data is stored in a spatial database (**PostGIS**³). As these technologies illustrate, this project develops a Web GIS application through **open-source software**.

The application uses the necessary OGC standards to accomplish the expected GIS tasks. **WMS** for map display, visualization and query; Web Feature Services Transactional (**WFS-T**) to send and retrieve vector data to the spatial database; Web Coverage Services (**WCS**) to receive raster data from the server, and of course **WPS** for processing. WPS is the core of the application and sometimes it needs other standards to supply the input data the process needs, or to store the results. In this document, we refer to **processing** (or geo-processing) as synonymous of **analysis** (or spatial analysis), like the **tasks applied in order to ask questions and derive insight from spatial data**.

It is worth to note that pure WPS use is a good choice to execute geo-processes on the web, but other options are also available. One example is Spatial Query Language views (**SQL views**). PostGIS, the spatial database, has powerful spatial query capabilities through SQL. Based on this, we can create a result (an operation, a new layer, etc.) coming from a query execution. These queries support a wide variety of processing tasks as raster processing, topological analysis, spatial analysis, data transformation, reference system management, etc. Other option is the use of **rendering transformations**. Basically this technology applies a filter function to the geometry (on the server side) before rendering it. That allows some kind of processes “on the fly” thanks to the Styled Layer Descriptor language (**SLD**). This language is intended to modify the symbology of layers. The use of a transformation function allows passing arguments to it, getting as result a change in the final rendering. An example of a rendering transformation could be contour extraction of a vector layer.

¹ <http://openlayers.org> [accessed January 15th 2014]

² <http://geoserver.org> [accessed January 15th 2014]

³ <http://postgis.net> [accessed January 15th 2014]

We can change the contour style (and thus we can show up only a selection of contours) using a SLD transformation function.

Last option for processing is through **scripting**. Thanks to an extension, the server technology manages many different scripting languages like Python, JavaScript or Groovy. It provides a number of extension points, called “hooks”, that are different ways to plug in functionality via a script. For example, there is a hook that provides a way to make scripts runnable as a WPS process.

These different processing options are taken into account in some Sections of this document, including also some examples of them, although pure WPS processes with Java are the ones that are widely studied. WPS standard defines a framework for executing almost any kind of computation. Some useful computations come “built-in” with the server extension functionality. Others have to be programmed. In fact, in GeoServer a WPS process is a Java class with an execute method. This method’s parameters correspond to the WPS parameters, and it returns the output of the process. In order to help the programming process, there is a GIS library called **GeoTools**⁴ that is used by the server to build the processing classes with few lines of code. The application develops 4 different processes, including chaining of existing processes.

LiDAR (Light Detection and Ranging) data defines a point cloud with pulses of the terrain representing not only the surface but also the elements that are on it (like buildings, vegetation, infrastructures, etc.). Every pulse collects the coordinates (including Z), the RGB components of the reflection, the reflection order (first return, second return, etc.), the time and angle, etc. It is a very interesting technology to get complete information about the vegetation (like bush height, tree height, density, coverage, etc.). Thus, the technology has many different uses in forestry. LiDAR data management and processing using WPS is still in development. The nature of these data hinders an easy implementation (like the high data volume), but some works are in progress (Lanig et al., 2009). Nevertheless, to study how to integrate this technology in Web GIS applications, we propose a pilot project that uses **SQL queries to process LiDAR data**. This project is not integrated in the final application

⁴ <http://geotools.org> [accessed January 15th 2014]

implementation, but this document presents a procedure to work with this kind of data using the spatial database.

The different parts of the application are described with full detail in this document. As result, we have a complete explanation of its architecture and implementation, adequate for a wide range of forestry / environmental purposes, which develops from simple to complex processes through open source software.

1.2. THESIS OBJECTIVES

The **main objective** of this work is **to define the architecture and to describe and to realize the implementation of a Web GIS application with spatial processes whose intended use is forestry / environmental related.**

This main objective can be divided into these narrower sub-objectives:

1. To define an **architecture for Web GIS applications** considering as reference the layered structure of INSPIRE Technical Architecture Overview Document (INSPIRE EU Technical Architecture Document, 2007).
2. To make a study of the current **state-of-art of processing on the Web**: what technologies are used, how they are combined together, what are the bottlenecks in processing, what are the guidelines for further works, etc. In general, to have a detailed description of what is the actual situation and what is the presumable future evolution.
3. To define the **architecture for WPS** adapted to forestry / environmental purposes and following the principles of interoperability, modularity and reusability for the different components that participate in the spatial processes.
4. **To implement a Web GIS application** according to the architecture defined in (1) and using open source software.
5. **To implement a set of WPS** according to the architecture defined in (3), using open source software.
6. To study how to **use LiDAR data** into Web GIS applications.

1.3. DISSERTATION ORGANIZATION

The structure of this document consists of eight chapters: *Introduction*, *State of the Art*, *Literature Review*, *Application Design*, *Web Processing Services*, *Conclusions*, *Further Work* and *References*. There are also two appendices: *Appendix A: Data* and *Appendix B: JavaScript code*.

State of the Art chapter makes a revision of a selection of works and applications that are widely used today in Spain and that implement some kind of spatial processing. In most of the cases these works refer to the development of SDIs. *Literature Review* chapter summarizes the most important points of a selection of literature references that have inspired this work.

Application design chapter brings together all the aspects related with both the architecture and the implementation of the application. It defines the basics about web technology (necessary to understand the whole), the scenario description and the architecture definition. Then it moves to the implementation through the description of the technologies, the organization of the application, the data, the coordinate reference systems (CRS) management and the functionality of each component.

Web Processing Services chapter describes the WPS standard through the description of its operations and the detailed study of every implemented process.

Conclusions chapter describes the problems and achievements of this work (including a discussion) and *Further work* chapter makes some comments on things that have not been developed due to time restrictions and that will be interesting to consider in other future works.

Appendix A describes the datasets used in the application and *Appendix B* contains the JavaScript code included in the main HTML file and that performs the main functionality of the application.

2. STATE OF THE ART

During last years, there has been an important increment in the number of Web GIS applications. This situation has evolved jointly with the development of national, regional and local SDIs. Each one normally provides some kind of viewer, where the user can look for the available information, display the data and perform some easy tasks (like measuring or querying the coordinates). This Section presents a brief description of some forestry /environmental applications on the regional (Comunidad Valenciana mainly, where this project takes place) and national (Spain) levels. The reason of this selection is that our project takes place in an area inside Comunidad Valenciana and it is a way to narrow the long list of available applications. The ones presented here are well known by the users because they belong to important institutions and national organizations. The list is not exhaustive. Nevertheless, since WPS and processing on the web is still barely spread as technology, it is difficult to find examples of spatial processing capabilities. What we normally find is an application that has some isolated processing implementation. For each application we make a brief description about what it is, what technology it uses and how spatial processing is implemented.

Terrasit (Comunidad Valenciana)

*Terrasit*⁵ application is the front-end of the regional SDI of Comunidad Valenciana region. It allows loading official cartography published by different regional public administrations. It can load also external WMS and WFS services. It has a complete set of tools for viewing and querying information. The implementation uses open source software: OpenLayers as mapping library and ExtJS and JQuery for the GUI. It has a unique spatial process implementation: viewshed calculation. The user introduces a point somewhere on the map and the server calculates the viewshed from that point. The tool has a list for selecting the base Digital Elevation Model (DEM) to calculate the viewshed from.

⁵ <http://terrasit.gva.es/es/ver> [accessed December 2nd, 2013]

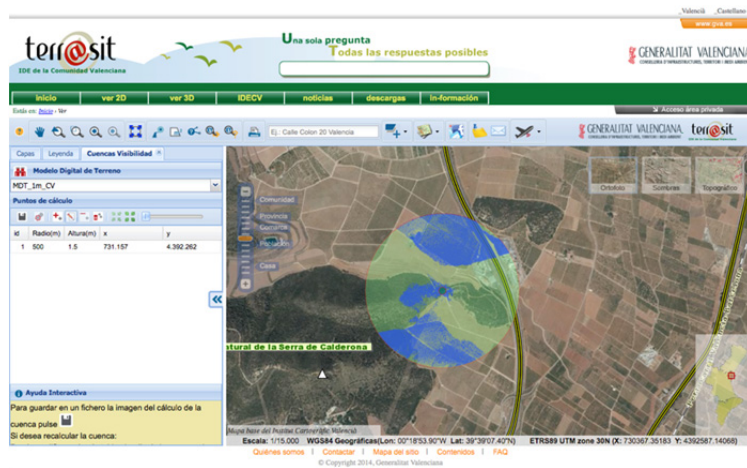


Fig. 2.- Viewshed calculation in Terrasit. Comunidad Valenciana SDI.

Cartoweb (Comunidad Valenciana)

Another interesting public Web GIS project in Comunidad Valenciana is the viewer of the Environmental and Forestry Public Administration (Conselleria de Infraestructuras, Territorio y Medio Ambiente). This project is called Cartoweb⁶. Its functionality is similar to the previous one, but the list of available environmental information is more exhaustive. The application uses the JavaScript Application Programming Interface (API) of ArcGIS. There is only one spatial process (buffer calculation). The user selects any feature of any layer of the map (like a building or a road) and the server returns a user-defined buffer and a list of features within that buffer.

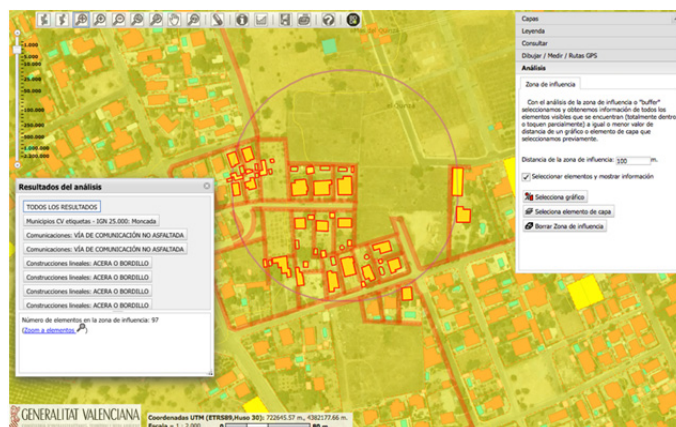


Fig. 3.- Buffer tool in Cartoweb (Comunidad Valenciana)

⁶ <http://cartoweb.cma.gva.es/visor/index.html> [accessed December 2nd, 2013]

OTALEX Project (Junta de Extremadura)

OTALEX⁷ Project is the most complete WPS client in Spain. It is an open source software solution that supports the regional SDI of Extremadura Region. It acts like a viewer, but it has also various powerful tools like editing, WFS management, social networks integration, etc. But maybe its main feature is the WPS client. It is based on 52°N WPS JavaScript client and it contains a list of more than 230 different processes of many types: from data management, conversions, spatial analysis, statistics calculations, etc. The mapping library is OpenLayers and the GUI is programmed with ExtJS. The idea of a project like this is highly interesting, but it has some dysfunctions: the different tools are not well documented and thus it is difficult to know how to use them, the usability is improvable and most of the time the processes do not work properly.

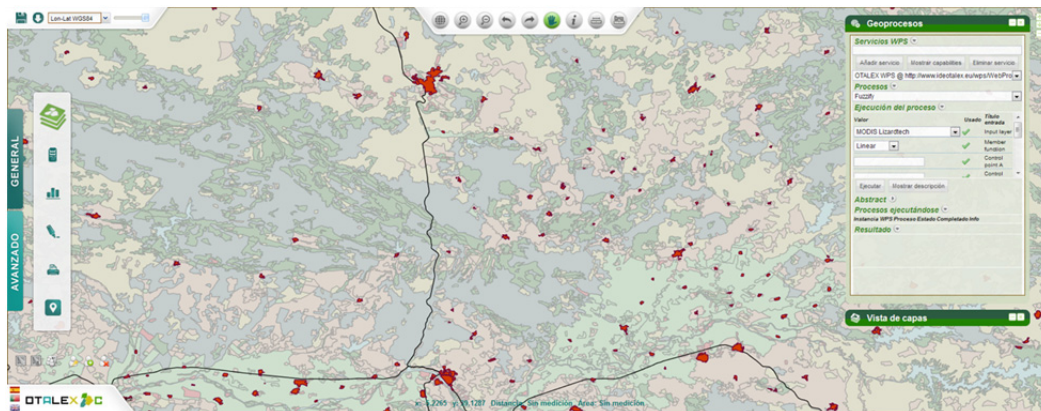


Fig. 4.- View of OTALEX WPS Client

CartoCiudad (IGN)

*CartoCiudad*⁸ is a national collaborative project for production and publication of web services with this information: street network, urban cartography, toponymy, ZIP codes and census districts. The National Geographic Institute (IGN), in coordination with other Public Administrations, leads it. The project has developed a public viewer where a set of OGC services is implemented (including WPS). It uses OpenLayers as mapping library as well. The tool uses OGC WPS specification version number 0.4.0

⁷ <http://ideotalex.eu> [last access December 2nd, 2013]

⁸ <http://www.cartociudad.es/visor/> [last access December 2nd, 2013]

and implements the following processes: route calculation between two or more points, buffer calculation, points of interest finding tool and inverse geocoding tool.

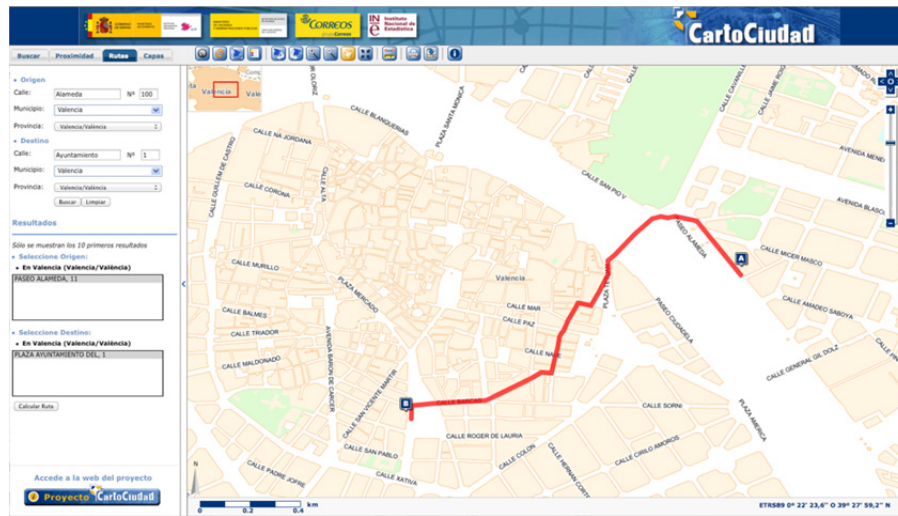


Fig. 5.- Route calculation using WPS in CartoCiudad

Spatial Analysis tool (IGN)

From the existing applications made by the National Administration, this is the most complete one in terms of number of WPS. Its name is Spatial Analysis Tool⁹ and it has been created by IGN, in collaboration with universities and private companies. It connects to services that offer environmental data and perform spatial analysis on that data. To do that the application supports WMS, WCS, WFS and WPS OGC specifications. This is an open source software solution. The mapping library is, again, JavaScript. The list of implemented processes is: query of altitude data, query of slope data, query of aspect data, profiles calculation, visibility maps and spatial analysis statistics. It is worth to highlight that during the testing operations, the application throws multiple errors in different browsers environments, so it is not working properly.

⁹ http://www.idee.es/clientesIGN/analisis_territorial/index.html [accessed December 2nd, 2013]

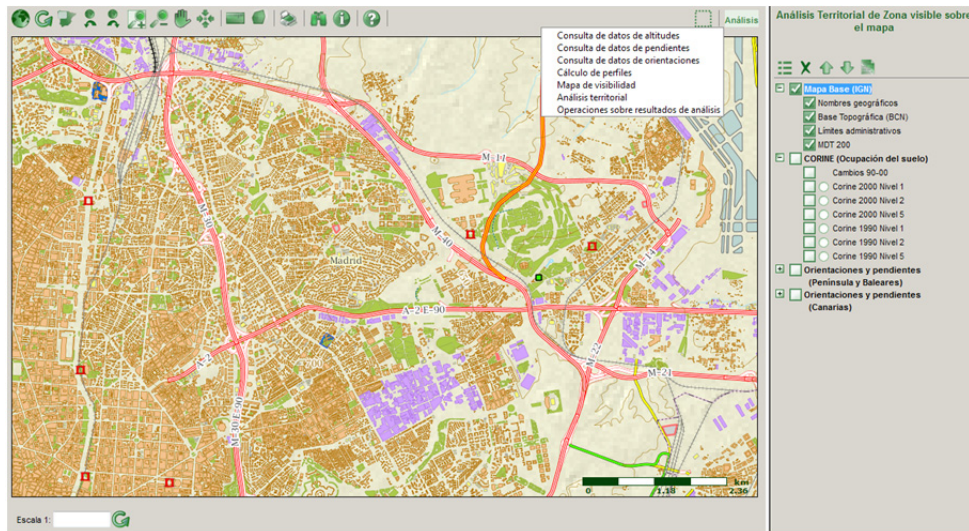


Fig. 6.- Spatial Analysis tool of IGN

This is only a selection of all the currently available Web GIS applications, but it is representative of the state of spatial processes on the web (in the Spanish level). This state of the art shows how there is still big room for improving different aspects like easy access of processes, good documentation, performance, downloading of results, etc. The different solutions are or too limited (only one spatial process) or too wide (many spatial processes coming from adaptations of other existing clients, with execution errors).

3. LITERATURE REVIEW

There are plenty of works and scientific papers related to WPS architectures and implementations. In this Section we present a selection of some interesting references to WPS projects that are directly and indirectly related to forestry or environment.

eHabitat project (Dubois et al., 2013) designs and implements environmental-related spatial processes. The paper basically presents a WPS for computing the likelihood of finding ecosystems with equal properties to those specified by the user. The problem to solve is ecological niche modelling for a given species by introducing a set of environmental variables. They explain the Mahalanobis distance as approach to compute similarity to a reference location. But the interesting part for this work is the WPS architecture and how it is implemented. WPS implementation is done using PyWPS (a lightweight Python based framework). Input data is managed using OGC standards. Boundaries can be either defined by the users or derived from a database of polygons representing protected areas. This is one of the key features of this application: the use of interoperable standards for data access and for process execution eases model chaining and integration. The calculation of the Mahalanobis distance itself is done using the R statistics language. The process finally works as usual, a WPS execute request is sent to the WPS server. Then the results are processed to generate different output formats. The paper presents two use cases: ecological forecasting in the Tassilli n'Ajjer Unesco site and ecological forecasting of birds ranges. As summary, the interest of this paper lies in the proposal of a simple and easily extensible model that uses WPS in combination with other OGC standards. Its versatility allows it to be used within different contexts and workflows. Despite the availability of WPS for modelling, few practical implementations exist and e-Habitat makes a significant addition to the field.

The design and implementation of web modelling, as a combination of chained WPS processes, is a challenging topic. One excellent example is the hydrological modelling processes description made in Díaz et al. (2010). This paper presents the architecture and the implementation for constructing chained WPS in hydrology. These processes are embedded in a distributed geospatial framework based on the INSPIRE SDI

(INSPIRE EU Directive, 2007). Compared to other solutions, this one allows users to interact directly with the underlying hydrological model (changing the settings of the WPS) and allows users to load specific datasets of interest. The architecture is defined using layers: a presentation layer, which manages the user interface and the interactions; a horizontal service layer, that allows the description and implementation of the components (as for example the WPS client to run models); the service layer that include all the OGC services and finally the data layer, which holds the spatial data and the metadata. The service layer implements: discovery services (through an open source implementation of a Catalogue Services for the Web - CSW specification), view services (through MapServer WMS standard, including chart services, necessary in hydrological modelling, and implemented using the OGC WPS interface), download services (through the OGC WFS), transformation services (of coordinates and data, all of them using the WPS interface too) and finally the Processing Services, of course implemented with WPS OGC. Regarding processing services, the paper presents a strategy that starts defining very simple processes, chaining them to construct more complex ones. The more complex the processes are, the less reusable they are. As summary, this paper has been very helpful and some of its assumptions have been used in the present work. The main goal to keep in mind is that processes and models in the web have to adopt a new paradigm for scientist working in a distributed and remote environment in order to reuse and share geospatial resources.

Similar to the previous paper is the work of Granell et al. (2009) about service-oriented applications for environmental models. The topic is the same in both studies, but in the latter the descriptions are more discursive. The AWARE project description is the main goal of the paper. It is a project to offer online geospatial processing services to help monitor and forecast water resources derived from specific quantity and distribution of snowmelt in alpine regions (Granell et al., 2009). The architecture is based on the layer structure explained above, but including in this case the full explanation of every module included in the project. The interest of the approach lies in the definition of an architecture that allows interoperability, reuse and a compromise in the performance of an application for environmental modelling, basing the functionality in a service-oriented framework that uses OGC standard protocols for spatial data.

Finally, there is an article that summarizes the actual situation of web services in forestry (Bastin et al., 2012) in line with INSPIRE and Global Earth Observation System of Systems (GEOSS¹⁰) frameworks. GEOSS belongs to the Group on Earth Observations (GEO), formed by the European Union and other nations and international organizations. Its aim is to build a common interface to link together different observing systems around the world. Forestry is one of the areas of interest inside GEOSS. The article explains how forest data is compiled in a Pan-European level through the European Forest Data Centre¹¹ (EFDAC), which incorporates the European Forest Fire Information System¹² (EFFIS). At a global scale, TREES-3 Action of Joint Research Centre supplies data addressed specifically to study land cover change over time. All this information is disseminated using web services: WMS, WFS, WCS and used within web-based modelling activities (WPS). One example of web client for managing this information is the EuroGEOSS¹³ project Web Map Viewer, which allows the control of some WPS. TREES-3 project validates the information from experts using a browser-based tool. But the interesting elements come with the WPS examples. The study explains three cases: the eHabitat project (described above) and two more. The second is about forest fire monitoring in protected areas: the service provides direct access to EFFIS in the Iberian Peninsula and returns the impact of forest fires in previous user-selected areas. The third is about monitoring forest change. It analyses the extent of a gain, loss or stability of forest areas between years 2000 and 2006 and returns a thematic map with statistical graphs.

¹⁰ <http://www.earthobservations.org/geoss.shtml> [accessed December 2nd, 2013]

¹¹ <http://forest.jrc.ec.europa.eu/efdac/> [accessed December 2nd, 2013]

¹² <http://forest.jrc.ec.europa.eu/effis/> [accessed December 2nd, 2013]

¹³ <http://www.eurogeoss.eu/> [accessed January 14th, 2014]

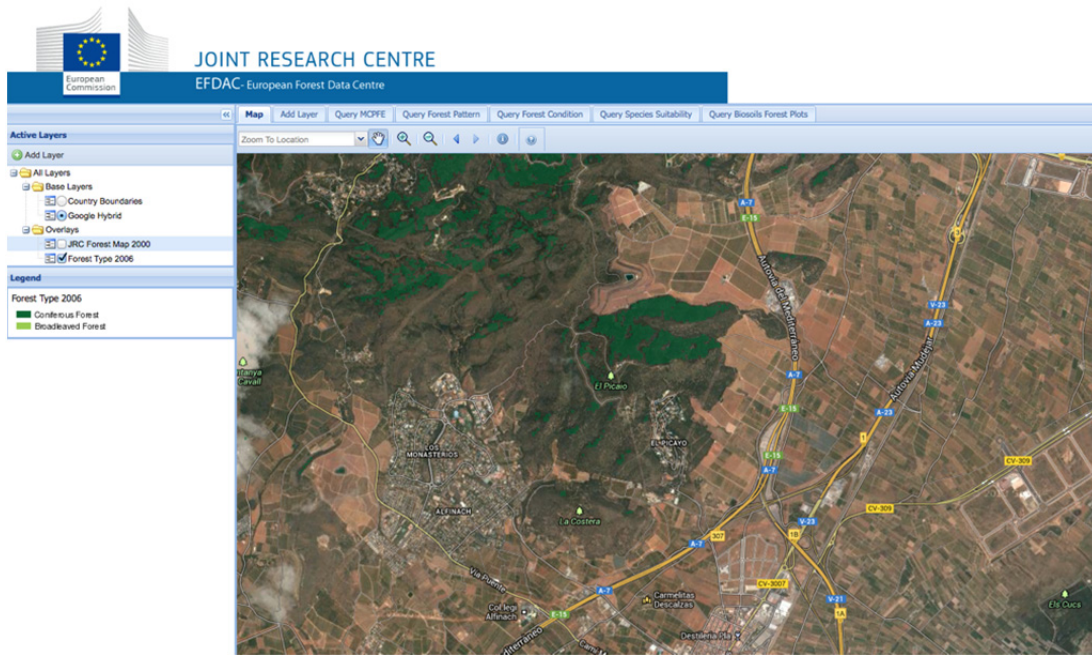


Fig. 7.- European Forest Data Centre Viewer

4. APPLICATION DESIGN

4.1. BASICS OF WEB GIS TECHNOLOGY

4.1.1. INTRODUCTION

This first Section is an introductory part about what Web GIS technology is, which elements a Web GIS application needs, how are these different elements related, etc. The aim is to establish the basic concepts to contextualize the other Sections and make their understanding easier.

In a broad sense, we define **Web GIS as a GIS that uses web technology** (Fu and Sun, 2011). The main principle of this technology is the communication between a Web GIS server and clients of few types (normally web browsers, but also desktop and mobile clients). The communication is performed using the **Hypertext Transfer Protocol (HTTP)** created by the researcher Berners-Lee in the 90s. Understanding this protocol is crucial in the development process and it is explained in detail in next Section.

4.1.2. HTTP PROTOCOL

In a regular communication process through the HTTP protocol, the user starts sending a request to the server (by typing a **Uniform Resource Locator**, or URL) using a client (normally a web browser). The server receives the request and processes it. That normally generates a response back to the client. The client receives the response that is shown to the user.

Technically, HTTP conducts the delivery of network resources. The term resource refers here widely to any element like text, query results, images, etc. that can be delivered. Usually HTTP takes place through what is called **TCP/IP sockets**. A socket is an endpoint of a process communication flow across a computer network. Transmission Control Protocol (TCP) sockets of Internet Protocol (IP) allows the streaming information in an ordered and reliable way between elements inside a

network. Web browsers use the TCP/IP when they connect to servers on the World Wide Web (WWW) in the general structure of a client-server (C-S) model.

Therefore HTTP protocol is the basic element to control communication between server and client. It can be defined as **simple**, **stateless** (after the response, the connection between server and client is typically dropped by means of reducing the load of the server) and **flexible** (allows the transmission of any type of data object, including spatial data).

The **structure of an HTTP transaction** is well defined and it follows a formal request format. Its elements are an initial request line, zero or more header lines, a blank line and an optional message body (Marshal, 2012):

- The **initial request line** has this structure: *METHOD NAME* (space) *PATH TO URL* (space) *HTTP VERSION*. The **METHOD** is the kind of request that the user wants to do to the server. HTTP defines eight methods: GET, POST, HEAD, PUT, DELETE, TRACE, OPTIONS and CONNECT. Usually, for Web GIS applications, both methods GET and POST are used. **GET requests data from a specific resource** (it is used for viewing something), while **POST is used to submit data** to be processed to a specific resource (it is used for changing something). The **PATH TO URL** indicates how to access the resource. URL is a subset of a **Uniform Resource Identifier (URI)**. The **HTTP VERSION** has this format in uppercase: “HTTP/x.x”, where x.x is the version used of the HTTP protocol.
- **Header lines.** These lines provide information about the requests, the responses or the object sent in the message body. HTTP 1.1 defines 46 headers. As stated in the specification (W3C, 1999) these fields act as request modifiers, with semantics equivalent to parameters on a programming language method invocation.
- **Message body.** This part contains a set of data sent after the header lines and completes the HTTP request. One common use of the message body is the indication of where the requested resource is returned to the client.

Once the request has been processed, the server returns the initial response line (also called **status line**) that basically indicates using a code system whether the request was correctly managed or not. The format of the status line is: HTTP/x.x CODE REASON. HTTP/x.x indicates the version of the HTTP protocol. REASON is a text in English describing the status code. CODE field is a three digits integer. The first digit indicates the category of the response:

- 1xx: indicates an information message.
- 2xx: indicates success.
- 3xx: redirects to another URL.
- 4xx: indicates an error on the client side.
- 5xx indicates an error on the server side.

A typical example of the status code is: 200 OK (means that the request succeeded) or 404 Not found (the requested resource does not exist). Web services follow the HTTP protocol to send and receive information between the server and the client. It is helpful to know the structure of the HTTP request in order to interpret the communication flow and the problems that could arise.



Fig. 8.- Example of a GET HTTP request header information

In Web GIS the HTTP requests are usually sent by JavaScript code. When this programming language is used, a common request is **XMLHttpRequest (XHR)**. This API allows sending the requests to the server (coming from the script), and loading the server response back to the script (W3C, 2009). A common use of XHR in Web

GIS is in WFS. The possibility to introduce modifications in the script using XHR can led to security problems. Due to these security restrictions, it is not possible to retrieve information from remote domains via this kind of request. To be able to do that, we need to install a **proxy host**. It is a program that acts like an intermediary between the server and the client. The requests are sent to the proxy and then the proxy forwards the request to the server. This fact is taken into account in our implementation.

4.1.3. HTML, CSS AND JAVASCRIPT

Hypertext Markup Language (HTML) is still the main language for creating web pages and the basic structure for web technologies. The language is basically plain text with a set of tags (markup). These tags define the content, the layout and the formatting information of the web page. A good practice is to separate the content from the layout. To do that, we use **Cascading Style Sheets** (CSS). Plain HTML is not enough to work with Web GIS technologies. In fact, in order to enhance the user experience, JavaScript is now the key element of these applications. That supposes the movement from a web page paradigm to a **web application paradigm**. HTML is still necessary as entry point to the application from a browser and to include the references to the JavaScript code and CSS resources but JavaScript code is now responsible of the application initialization and GUI definition.

4.1.4. SERVER AND CLIENT TECHNOLOGIES

Server technology must comply with HTTP specifications and accordingly knows how to manage the communication flow with clients. There are different kinds of server technologies that can be used as Internet Information Services (IIS), Apache Web Server, Oracle/Sun Java System Web Server, IBM WebSphere Web Application Server, etc. We choose **Apache Web Server 2.2**¹⁴ as Server technology. The reasons are:

- It is free and open-source.
- It is very powerful, flexible and HTTP/1.1 compliant.

¹⁴ <http://httpd.apache.org> [accessed October 21th, 2013]

- It is the most used server worldwide (NetCraft, 2013).
- It manages object relational databases, needed for Web GIS.
- It allows setting up files, like Common Gateway Interface (CGI) scripts (proxy host script).

The use of Apache Server is a necessary but not a sufficient step. We also need a servlet container for Java code. In this regard we use **Apache Tomcat**¹⁵, which implements both Java Servlet and JavaServer Pages (JSP). That means the server-side **programming technology** is **Java**.

From the client side, the software application is the **Web Browser**. Other clients exist, but they are not considered in this work. Web browsers know how to communicate with web server technology, how to display HTML, how to manage CSS styles and how to execute JavaScript code. The browser is a choice of the user, so the application has to be tested in the most used ones (basically Mozilla Firefox, Google Chrome, Apple Safari and Microsoft Internet Explorer). Together all these browsers represent (January 2014) the 96.7 % of browser usage¹⁶. We have already emphasized that JavaScript is the scripting language that is used to create the GUI and to get interactivity. To complement JavaScript there is a set of web development techniques called **AJAX** (Asynchronous JavaScript and XML). The aim of AJAX is to allow an asynchronous user interaction with the application (independent of the communication with the server). Users interactions (that generate HTTP requests) are sent to what is called the AJAX engine. Any response to a user action does not require going to the server and the engine handles it instead. If the engine needs extra information, it makes the requests asynchronously (normally using Extensible Markup Language, **XML**) without delaying the user, improving the user experience this way.

4.1.5. DATA EXCHANGE FORMATS

The two most used data exchange formats for Web GIS are XML and JavaScript Object Notation (JSON).

¹⁵ <http://tomcat.apache.org> [accessed October 21th, 2013]

¹⁶ http://www.w3schools.com/browsers/browsers_stats.asp [accessed February 11th, 2014]

XML is a markup language that encodes documents in both human-readable and machine-readable way. It is defined in the XML 1.0 Specification of W3C (W3C, 2008). The main advantages of XML are its simplicity, its well-defined structure (and thus its easy validation) and the facility to understand the content. The main drawback is the use of closed tags, making parsing with JavaScript not efficient.

JSON interchange format is more efficient to parse than XML because it is lighter. It is also human-readable but uses attribute-value pairs (AVP) instead of tags. It is defined in the European Computer Manufacturers Association ECMA-404 standard (ECMA, 2013). There is a variation of JSON for encoding geographic data structures called **GeoJSON**¹⁷. It is not a standard, but it is useful in Web GIS applications. It can represent Point, LineString, Polygon, MultiPoint, MultiPolygon and GeometryCollection geometries.

4.1.6. WEB SERVICES

Web services, as defined in Fu and Sun (2011), are *programs that run on a web server and expose programming interfaces to other programs on the web*. To fulfil the functions any Web GIS application should have (data visualization, querying, analysis, etc.) we use web services. The scope of this Section is to explain the main content of the different web services standards. This information is basic to understand the application implementation. All the standards are published on the OGC website¹⁸.

4.1.6.1. WEB MAP SERVICE (WMS)

Web Map Service (WMS) standard implementation specification document (OGC, 2006) contains the full detail of this web standard. This service is used for showing the cartography through images.

¹⁷ <http://geojson.org/geojson-spec.html> [accessed January 15th, 2014]

¹⁸ <http://opengeospatial.org> [accessed January 15th, 2014]

4.1.6.1.1. INTRODUCTION

The scope of this service is to produce maps as a digital image, suitable to be displayed on a screen. This means that the map does not contain spatial data itself but an image in one of a set of possible formats (like PNG, GIF or JPEG, among others). To access maps, the user invokes a request using HTML through an URL. But in our architecture the user does not need to write the request because the complexity is hidden and the user only needs to activate and deactivate layers in a user-friendly way. This standard conforms the main way to visualize maps. Additionally, the standard defines two “types” of WMS: basic and queryable. As expected, in the second case the user can query the attributes of the map. Latest version standard is 1.3.0.

4.1.6.1.2. HTTP REQUEST RULES

Both GET and POST methods can be used to request a WMS resource, but GET method is mandatory while POST method is optional. GET requests URLs that are built using reserved characters. These characters are (OGC, 2006):

- ? indicates the start of the query string.
- & defines the separation between parameters in a query string.
- = is the separator between name and value of a parameter.
- , (comma) is the separation between individual values in list-oriented parameters.
- + is a shorthand representation for a space character.

The general structure for the request is a URL prefix (including the schema like “http”, the hostname, port, path, etc.), the mandatory question mark “?” and the optional string with parameters ending in an ampersand “&”. In order to compose a query request, the client appends the mandatory request parameters and any optional parameters as name/value pairs in the form “name=value&”, being the ampersand the separator between different parameters. These rules are applicable to all the standards (not only WMS).

4.1.6.1.3. HTTP RESPONSE RULES

Once the server receives the HTTP request, it sends back a response or a service exception if it is unable to respond correctly. A **Multipurpose Internet Mail Extension** (MIME) **type** accompanies the response object. It is always a file containing text or a map image. The list of output formats is the following: GIF, PNG, JPEG, TIFF, SVG and WebCGM.

4.1.6.1.4. WMS OPERATIONS

The standard defines three operations:

- **GetCapabilities.** It is a mandatory operation that returns the service metadata in a machine-readable and human-readable way. The request parameters can be checked in Table 1.

Mandatory parameters		
Name	Value	Description
SERVICE	WMS	Indicates the service type
REQUEST	GetCapabilities	Indicates the request name
Optional parameters		
Name	Value	Description
VERSION	VersionNumber	Request version
FORMAT	MIME_type	Output format of service metadata
UPDATESEQUENCE	String	Sequence number or string for cache control

Table 1.- WMS GetCapabilities request parameters (OGC, 2006)

The response of this operation is a XML document containing the service metadata. These metadata are (selection of the output): name, title, online resource, abstract, keyword list, contact information, layers (there are as many descriptions as layers), styles, Coordinate Reference System (CRS), bounding box, scale denominators, etc.

- **GetMap.** It is the mandatory operation that returns a map. The mandatory and optional parameters can be checked in Table 2.

Mandatory parameters		
Name	Value	Description
VERSION	1.3.0	The requested version
REQUEST	GetMap	The requested operation name
LAYERS	layer_list	Comma-separated list of one or more map layers
STYLES	style_list	Comma-separated list of one rendering style per layer
CRS	namespace:identifier	The Coordinate Reference System
BBOX	minx,miny,maxx,maxy	Bounding box corners (lower left, upper right) in CRS units
WIDTH	output_width	Width in pixels of map picture
HEIGHT	output_height	Height in pixels of map picture
FORMAT	output_format	Output format of map
Optional parameters		
Name	Value	Description
TRANSPARENT	TRUE/FALSE	Background transparency of map (default=FALSE)
BGCOLOR	colour_value	Hexadecimal red-green-blue colour values for the background colour (default=FFFFFF, white colour)
EXCEPTIONS	exception_format	The format in which exceptions are to be reported by the WMS (default=XML)
TIME	time	Time value for the layer
ELEVATION	elevation	Elevation value for the layer

Table 2.- WMS GetMap request parameters (OGC, 2006)

- **GetFeatureInfo.** This is an optional operation supported only on layers in which the attribute queryable is set to true (“1”). Once the map has been returned using GetMap operation, the user chooses a point on the map to obtain more information. Parameters are in Table 3.

Mandatory parameters		
Name	Value	Description
VERSION	1.3.0	The requested version
REQUEST	GetFeatureInfo	The requested operation name
(Map request part)		Partial copy of the Map Request parameters that generated the map for which information is desired
QUERY_LAYERS	layer_list	Comma-separated list of one or more layers to be queried
INFO_FORMAT	output_format	Return format of feature information
i / X	pixel_column	i / X coordinate in pixels of feature in Map Coordinate

Mandatory parameters		
		System
j / Y	pixel_row	j / Y coordinate in pixels of feature in Map Coordinate System
Optional parameters		
Name	Value	Description
FEATURE_COUNT	number	Number of features (default=1)
EXCEPTIONS	exception_format	The exception format (default=XML)

Table 3.- WMS GetFeatureInfo request parameters (OGC, 2006)

Next figures show some examples of WMS requests for each operation. In Figure 9 there is a GetCapabilities XML response document, where we can see some metadata about the service itself. In Figure 10 the GetMap operation returns a static image of a layer (what is called a tile). Finally Figure 11 shows a GetFeatureInfo operation in a pixel, and the response is a table with the layer attributes.

Request: <http://localhost:8080/geoserver/wms?service=wms&version=1.1.1&request=GetCapabilities>

This XML file does not appear to have any style information associated with it. The document tree is shown below.

```

- <WMT_MS_Capabilities version="1.1.1" updateSequence="518">
- <Service>
  <Name>OGC:WMS</Name>
  <Title>GeoServer Web Map Service</Title>
- <Abstract>
  A compliant implementation of WMS plus most of the SLD extension (dynamic styling). Can also generate PDF, SVG, KML, GeoRSS
</Abstract>
- <KeywordList>
  <Keyword>WFS</Keyword>
  <Keyword>WMS</Keyword>
  <Keyword>GEOSERVER</Keyword>
</KeywordList>
<OnlineResource xlink:type="simple" xlink:href="http://geoserver.sourceforge.net/html/index.php"/>
- <ContactInformation>
  - <ContactPersonPrimary>
    <ContactPerson>Constancio Amurrio Garcia</ContactPerson>
    <ContactOrganization>EnerGIS</ContactOrganization>
  </ContactPersonPrimary>
  <ContactPosition>Chief Engineer</ContactPosition>
- <ContactAddress>
  <AddressType>Work</AddressType>
  <Address/>
  <City>Valencia</City>
  <StateOrProvince/>
  <PostCode/>
  <Country>Spain</Country>

```

Fig. 9.- GetCapabilities request response example

Request: `http://localhost:8080/geoserver/forestry/wms?SERVICE=WMS&LAYERS=forestry%3ALics&FORMAT=image%2Fpng&TRANSPARENT=TRUE&VERSION=1.1.1&REQUEST=GetMap&STYLES=&SRS=EPSG%3A900913&BBOX=-78271.516953126,4735426.7756641,-39135.758476563,4774562.5341406&WIDTH=256&HEIGHT=256`

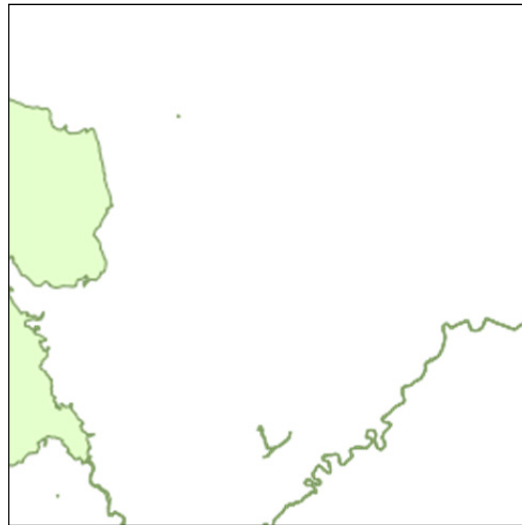


Fig. 10.- GetMap request response example

Request: `http://localhost:8080/geoserver/forestry/wms?SERVICE=WMS&LAYERS=forestry%3ALics,forestry%3AMunicipalities&QUERY_LAYERS=forestry%3ALics,forestry%3AMunicipalities&STYLES=,&SERVICE=WMS&VERSION=1.1.1&REQUEST=GetFeatureInfo&BBOX=-180264.187372%2C4711667.245905%2C7006.531901%2C4751108.752495&FEATURE_COUNT=10&HEIGHT=258&WIDTH=1225&FORMAT=image%2Fpng&INFO_FORMAT=text%2Fhtml&SRS=EPSG%3A900913&X=481&Y=94`

Lics

fid	SITE_CODE	CA	SITE_NAME
Lics.9	ES5233040	COMUNIDAD VALENCIANA	MUELA DE CORTES Y EL CAROIG

Fig. 11.- GetFeatureInfo request response

4.1.6.2. WEB FEATURE SERVICE (WFS)

4.1.6.2.1. INTRODUCTION

Web Feature Service standard implementation specification document (OGC, 2005) describes how this service allows the exchange of vector data over the Internet. Additionally, WFS-T (transactional) supports edit operations. The standard gives access to tasks as querying, styling, editing and downloading vector data. WFS-T performs the basic operations of persistent storage (known as CRUD operations: create, read, update and delete).

4.1.6.2.2. WFS OPERATIONS

This standard is quite vast so we explain a selection of operations that are of interest for our application. The operations available in all WFS versions are **GetCapabilities** (discovery of data), **DescribeFeatureType** (information about an individual feature type), **GetFeature** (downloading a selection of features from the data source), **LockFeature** (prevention of a feature to be edited) and **Transaction** (creation, updating and deleting of features).

As WMS, WFS can use both GET and POST methods and the rules of Section 4.1.6.1.2 are also applicable here. The request encoding can be done using two different methods: XML (used in HTTP POST) and keyword-value pairs (used in HTTP GET). The appropriate MIME type must accompany the response objects.

WFS returns features in a number of formats. The available options are: Geography Markup Language (GML2 and GML3), shapefile (compressed in a ZIP file), JSON and Comma-Separated Values (CSV).

Here there is an example of a WFS-T POST request to create a feature edited by the user and saved in the spatial database. The first figure is the POST request. The second figure shows the server response. In the request we see the coordinates of the geometry that is sent to the server (in this case, a polygon). The response confirms that it has been correctly inserted in the spatial database.

```
XML
<wfs:Transaction xmlns:wfs="http://www.opengis.net/wfs" service="WFS" version="1.1.0" xsi:schemaLocation="http://
  <wfs:Insert>
    <feature:computestatisticspolygon xmlns:feature="http://localhost:8080/thesis/forestry">
      <feature:geom>
        <gml:MultiSurface xmlns:gml="http://www.opengis.net/gml" srsName="EPSG:900913">
          <gml:surfaceMember>
            <gml:Polygon>
              <gml:exterior>
                <gml:LinearRing>
                  <gml:posList>-94043.219477955 473735...9477955 4737350.0874052</gml:posList>
                </gml:LinearRing>
              </gml:exterior>
            </gml:Polygon>
          </gml:surfaceMember>
        </gml:MultiSurface>
      </feature:geom>
    </feature:computestatisticspolygon>
  </wfs:Insert>
</wfs:Transaction>
```

Fig. 12.- WFS POST request

```

<wfs:TransactionResponse version="1.1.0" xsi:schemaLocation="http://www.opengis.net/wfs
//www.opengis.net/ogc" xmlns:gml="http://www.opengis.net/gml" xmlns:xsi="http://www.w3.
<wfs:TransactionSummary>
  <wfs:totalInserted>1</wfs:totalInserted>
  <wfs:totalUpdated>0</wfs:totalUpdated>
  <wfs:totalDeleted>0</wfs:totalDeleted>
</wfs:TransactionSummary>
<wfs:TransactionResults></wfs:TransactionResults>
<wfs:InsertResults>
  <wfs:Feature>
    <ogc:FeatureId fid="computestatisticspolygon.158"></ogc:FeatureId>
  </wfs:Feature>
</wfs:InsertResults>
</wfs:TransactionResponse>

```

Fig. 13.- WFS Server response

4.1.6.3. WEB COVERAGE SERVICE (WCS)

4.1.6.3.1. INTRODUCTION

Web Coverage Service (WCS) standard retrieves geospatial data as coverages (grid files). This functionality is very important for forestry and environmental purposes because raster data is widely used in modelling. It is necessary to clarify the differences of WCS respect WMS. WMS retrieves images that represent the data, while WCS retrieves the original raw values (instead of pictures) and thus those data can be used in spatial processes and interpreted in its complexity (and not just portrayed).

The WCS standard specification implementation (OGC, 2006) clarifies its characteristics.

4.1.6.3.2. WCS OPERATIONS

The three operations of WCS are:

- **GetCapabilities:** as usual, it returns an XML document with the service description and its coverages.
- **DescribeCoverage:** allow clients to request a full description of one or more coverages. The response is an XML file.

- **GetCoverage.** As expected, this operation returns a coverage encoded in a well-known coverage format. The request allows the control of the coverage we want to retrieve (see Table 4).

Mandatory parameters		
Name	Value	Description
IDENTIFIER	name	Defines the coverage to get
FORMAT	format	Format for the output. The formats are both images and georeferenced. Images: JEP, GIF, PNG, Tiff and BMP. Georeferenced: GeoTiff, GTopo30, ArcGrid and GZipped ArcGrid.
Optional parameters		
Name	Value	Description
RANGESUBSET	field	This parameter allows the selection of any of the available fields of the data
BOUNDINGBOX	minx,miny,maxx,maxy	The spatial subset of the data we want

Table 4.- WCS GetCoverage request parameters (OGC, 2006)

Next URL shows a GetMap WCS request example of a Digital Elevation Model (DEM) raster dataset:

http://localhost:8080/geoserver/wcs?SERVICE=wcs&VERSION=1.0.0&REQUEST=GetCoverage&coverage=mde200_V4&format=tiff.

The server returns the coverage, which can be saved to the local system.

4.1.6.4. WEB SERVICES SUMMARY

Web services are the standard-based units that fulfil the tasks a Web GIS is supposed to give. The combination of different services (WMS, WFS, WCS and WPS) defines a complete solution for every environmental or forestry issue. In the implementation, the complexity of the service request is hidden to the user (it is encapsulated). Once the service is created, third parts can use it (they are reusable), favouring the collaboration and exchange of information. Last, these services can be mashed up, creating complex applications that would be difficult to develop otherwise.

4.2. ARCHITECTURE

4.2.1. INTRODUCTION

This Section's aim is to propose a software architecture suitable to be applied in applications where forestry / environmental data and processes are involved. Processes themselves are not explained in detail here, thus they have their own Section (see Section 5). Section 4 includes instead how other web services (except processing) are integrated in the structure.

4.2.2. ARCHITECTURE PRINCIPLES

Web GIS technologies provide nowadays most of the needs of spatial information users. However, an important challenge is still in process to be solved: how to consolidate an efficient use of processing on the web. As a particular type of GIS, Web GIS can perform the normal functions a regular GIS does, with some strengths (and of course also some weaknesses). The general **functions that the application should perform** are as follows:

- **Data visualization.** This is the basic function. The application has to offer a set of data in an accurate way and with adequate symbology. The user needs a set of tools to move around the extent in a comfortable way.
- **Querying.** Displaying the graphical information (mapping) is not enough. Most of the interest comes when the user queries the data attributes. This function consists on answering the question of: "*what is here?*"
- **User active interaction.** To make the application more interesting, the user interacts with the application creating his or her own information through editing.
- **Analysis.** This function is the one that makes Web GIS powerful. The aim is to answer both simple and complex questions and perform spatial processes.

To fulfil these functions, we need web services. Thus, the **first principle** is: **the architecture is Service-Oriented (SOA)**.

Web services have evolved very quickly during the past years, bringing up a set of advantages in computing: use of distributed systems, in which the different components are not physically in the same computer but they work as a whole; the web service is affected neither by the operating system nor the programming language it was used to create it; the client can consume many web services and the web service can be consumed by multiple clients; when the creator of the server makes any update on the web service, it is automatically available when requested by the users; and finally interoperability is easier to achieved through OGC web service standards.

In GIS, web services can be classified according to the functions they provide. **Map services** return maps in image format, identify the attributes and show up a legend. **Data services** allow access and downloading of raw data for vector and raster data types. **Analytical services** allow the users to execute processes on the web.

The main design principle behind SOA is that a **service** is a standards-based, loosely coupled unit **composed of a service interface and a service implementation**. Service interface describes the function capabilities of a service. Service implementation implements what a service should execute (Granell et al., 2009). This approach allows the easy use of the service independently of the technology used to implement it.

The **second principle** is based on the **layered structure** proposed by the INSPIRE Directive (INSPIRE EU Directive, 2007). The technical architecture document (INSPIRE EU Technical Architecture Document, 2007) defines the functionality of a SDI through web services (so it is also a SOA). Our architecture is a simplification of the INSPIRE proposal and it is organized in a set of layers:

1. **Presentation / User layer**. It contains the client (what is called the **geoportal**). It organizes the communication between the user and the services of the system.

2. **Service layer.** It contains the services. The types of them (directly related with the functions of a GIS exposed above) are: discovery services, view services, download services and processing services.
3. **Data layer.** It contains the data / metadata supplied to the services.

There is direct connection between this layered structure and the components of the Web GIS application (see Figure 14). The presentation layer corresponds to the web client component, the service layer corresponds to the web server component and the data layer corresponds to the database component.

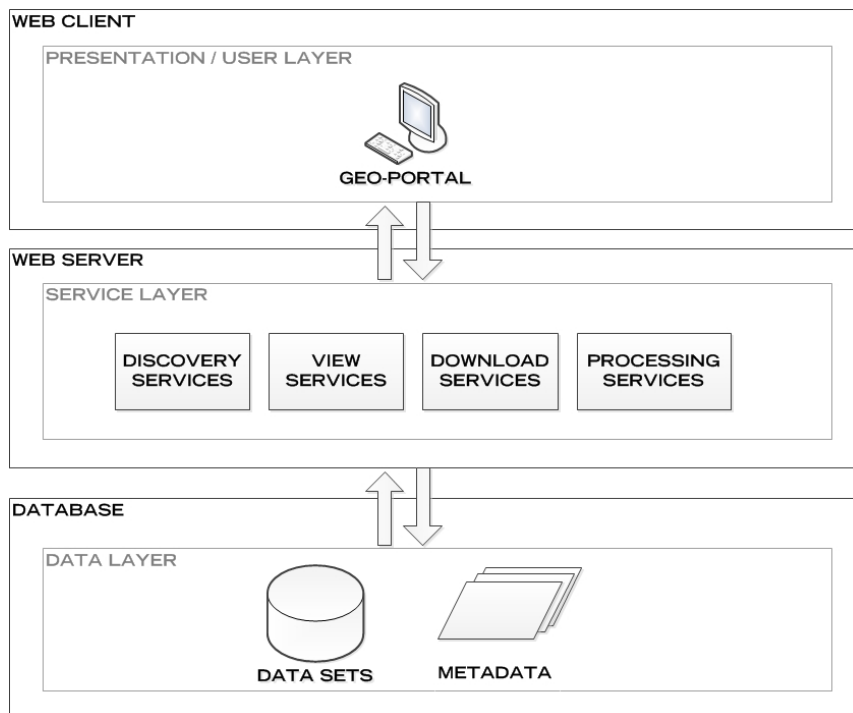


Fig. 14.- Architecture of the application. Source: Author.

4.2.3. COMPONENTS

The architecture has three intimately related components: Web GIS client, Web GIS server and database. **Web GIS client** features are as follows:

- It must provide a friendly and easy-to-use GUI. Learning its functionality has to be almost immediate.

- It must manage different kind of interactions between user and system: interpreting user input events, sending data to the server, receiving back information from the server, showing messages to the user that allow him or her to know whether a process has been correctly executed or not, etc.
- It has to manage spatial data and show it up in a meaningful way.

Web GIS Server features are:

- It must have fully compliance with Open Geospatial Consortium (OGC) standards.
- It must provide support for deploying important data formats and relational databases as PostGIS and shapefiles. The server needs to disseminate, at least, 2D maps. 3D processing is outside the scope of this work.
- It must provide, in addition, support for raster data, absolutely crucial for forestry / environmental purposes.
- It must correctly manage the most common data projections.
- The Web GIS can generate usual format as outputs like JPEG, GIF PNG, PDF, SVG, KML, GeoRSS, GML, GeoJSON, etc.
- In order to improve performance, the Web Server has to manage tile creation.
- It supports layer styling. The symbology will be managed in the server side.
- It must provide good performance, availability of Representational State Transfer (REST) architectural style and a good security system. REST is very important in the daily server management within a production environment.

The **database** is in the basis of the architecture and it has to be spatially enabled. It is not a static container but a powerful processing engine (through SQL). Its features are:

- Store spatial data and a rich set of different formats.
- Complete integrity and possibility to establish complex relationships between data.
- Topology rules support.
- Accessibility for multiple users at the same time.
- High performance almost independently of the volume of data / number of users (in a common situation).
- Security, with backup, recovery and rollback support.

4.2.4. SERVER – CLIENT WORKLOAD DISTRIBUTION

Workload distribution between server and client is a criterion for classifying Web GIS applications. In **thin client** architectures the server does most of the workload and in **thick client** architectures the client does most of it (Gong, 1999). We propose the use of a substantially pure thin client architecture because:

- Processes need server technology to be performed. Web browsers do not have the technical capabilities to execute them by their own.
- No software or plug-in has to be installed in the client-side, making it easier for the user. A regular computer with a sufficient bandwidth is enough to make the application work.

- Normally, forestry / environmental projects have an outstanding geographical extent (i.e. as regional or national projects) and they need various datasets in order to run models. These data come from different sources and can be very complex (as Light Detection and Ranging - LiDAR data). The management of this information from the client side would endanger the system performance.

The reasons exposed above do not pre-set minor client participation. In fact, the client can perform some tasks to speed up the interaction and avoid excessive pressures on the server. Best practices suggest partitioning on the workload and we apply this principle in our architecture. The tasks managed on the client side are: editing of new cartography by the user (these new layers are used as inputs in some processes); query attributes of layers from user click events, showing up a window with those attributes and finally generation of a legend with of active layers in the table of contents.

4.2.5. ARCHITECTURE SUMMARY

We propose a SOA and layer-based architecture to achieve the maximum interoperability and a better user experience. The geospatial services provide functionality in an easy way, decoupling the description from the implementation. Users invoke those services using a geoportal that hides the details of the underlying HTTP requests. The services use open geospatial standard interfaces. The structure lies on a powerful database that connects with the services that are at the same time connected to the client. The server supports most of the workload but some tasks are performed on the client side.

4.3. SCENARIO DESCRIPTION

4.3.1. INTRODUCTION

Before the implementation, and now that the architecture is defined, we make the scenario description. It defines the study area and the features the application provides.

4.3.2. STUDY AREA

The extent of the study area is the province of Valencia (Comunidad Valenciana – Spain) and specifically the region called “Canal de Navarrés”, administrative division that groups together these municipalities: Enguera, Anna, Chella, Bolbaite, Navarrés, Quesa, Bicorp and Millares. The total area is about 117,816 ha. The reasons for choosing this area are:

- Interesting region from the environmental / forestry point of view: it has a Mediterranean landscape with a mix of crops and forests. Different ecosystems exist in a relatively small area.
- From the economical point of view, there is an important amount of forest resources still to be assessed and managed.
- From the social point of view, there is a consolidated associational structure that helps the relationship between the private sector and the public administrations.
- The extent is big enough to check the performance of the application in a real case study.

The geographical extent of the study area is as follows:

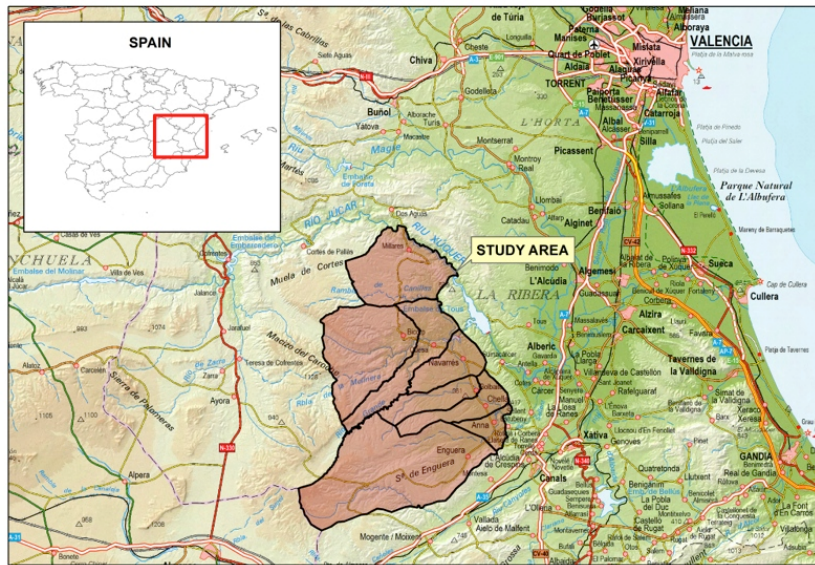


Fig. 15.- Study area. Source: Author.

As general rule, the extent of the layers is the study area but sometimes that extent is bigger (usually the province level). The reasons for this bigger area are the geographical contextualization (often it is necessary to publish a wider extend to prevent clipping geometries in the boundaries) and the use of external WMS (cascading WMS). These external WMS have their own extent and it cannot be changed (only used as is).

4.3.3. FEATURES

The application has two kinds of functionality: general tasks of any application and specific tasks (forestry / environmental related). It must offer an easy-to-use and powerful framework for non-expert users to perform spatial processes. These processes should be of full interest: they should provide answers to different questions, as in these examples:

- A landowner, expert in the management of his or her land but maybe not a GIS expert, wants to know some characteristics of his or her ownership. Some examples of questions are: *“What is the density of roads?”* *“Is it adequate for*

wood extraction?” “Should I need to construct additional roads to improve the productivity?” These are examples of **economy**-related questions.

- A scientist wants to know the rainfall inside an area by drawing a polygon on the map, and getting the average, maximum and minimum values only within that area (**ecology**-related questions)
- A student wants to make photo-interpretation, and needs to mash up some layers and put some transparency to discover some patterns about land use (**ecology** application).
- A hiker wants to know where archaeological sites are in order to visit them (**social** application).

The **features** the application makes available to the user are:

- Easy and **adaptable GUI**, with an unambiguous map visualisation and access to every element.
- Availability of a well-organized **list of layers**, divided into groups according to its content. The list has a set of base layers that defines a general framework for the location and interpretation of the overlays.
- A **map display area** that will occupy most of the screen space.
- A **set of tools**, including those that allow the interaction user – system:
 - Zoom in and zoom out (by one level each click).
 - Zoom in by box.
 - Zoom history.
 - Zoom to Extent (the study area).
 - Pan.
 - Measure length and areas (both partial and total).

- Identify layer attributes.
 - Clean screen tool.
 - Legend tool.
 - Google Street View and Bing Maps (where available) through user click.
 - Check coordinates tool.
-
- **Spatial processes** area. The processes are grouped by type (i.e. basic and forestry processes). There is a short explanation of how each process works in the GUI.
 - **Results** area. Near the spatial processes area, there is a results area where they are printed out (when appropriate).

4.4. IMPLEMENTATION OF THE APPLICATION

This Section's aim is to explain the steps of the implementation (except WPS, see Section 5). We describe the general structure and organization of the application, which pre-processing do we apply to the data, what technology has been used, how the different elements are related to each other and a description of the developed functionality. The steps can be used to “replicate” a new implementation, as a guide.

4.4.1. TECHNOLOGIES

This Section describes and justifies which technology is used for every component of the architecture (server, client and database). Each component has different software options that can be used. In the implementation one of the options is chosen. In addition, other software technologies are also described. They offer extra functionality and help all the application work together properly.

4.4.1.1. SERVER TECHNOLOGY: GEOSERVER

4.4.1.1.1. INTRODUCTION

GeoServer is an OGC compliant open source software server written in Java (GeoServer, 2014). It is a consolidated project used worldwide to publish spatial data through standards. It accomplishes features list in Section 4.2.3. Since it is a Java application, it can be installed in many operating systems. It needs a servlet container in order to make it work (Apache Tomcat).

Compared to other spatial servers, GeoServer has a smooth learning curve and the main reason for that is its intuitive administrative interface. The access to the different configuration options is straightforward (see Figure 16).

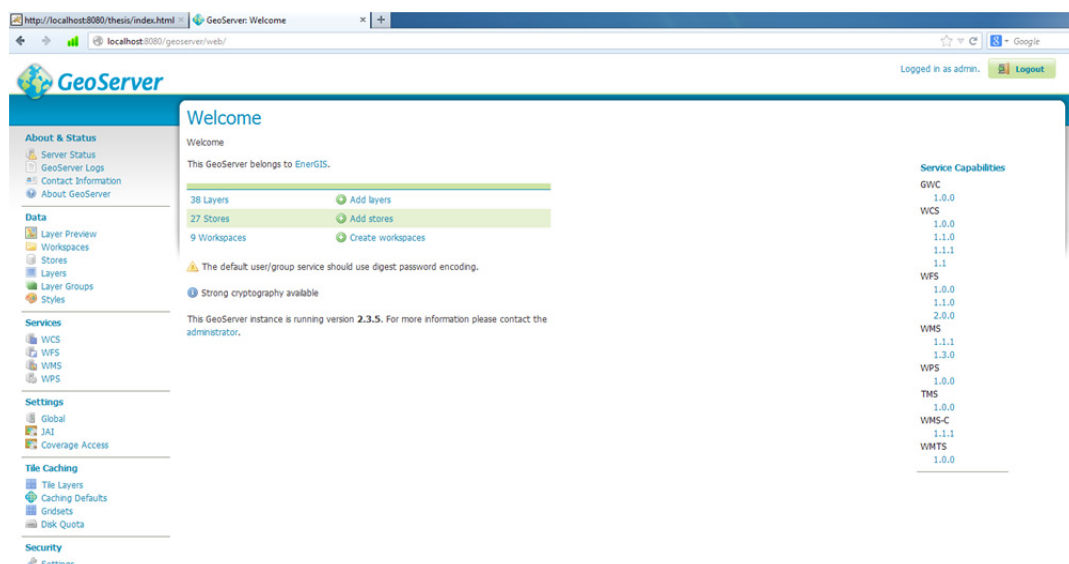


Fig. 16.- GeoServer Administrative Interface

4.4.1.1.2. GUI AND FUNCTIONALITY

GeoServer functionality is divided into six big sections. **About & Status Section** is the “control panel” of the server. It checks the status of the working server to make it run properly or identify possible errors. Some of the information that is checked here is: number of connections, memory usage, Java Virtual Machine (JVM) configuration, fonts, Java Advanced Imaging (JAI) configuration, cache management, logs files and contact information.

Data Section is where we perform data management. There is a set of rules to upload spatial data to GeoServer. The information is organized in a structure with these elements:

- **Workspaces:** used to organize the layers. For this implementation, there is one unique **workspace** called **forestry**.
- **Stores:** connect Geoserver to repositories where the data is located. Each store has to match the format of the spatial data it connects to. There are different types of stores. Some examples are, for vector data: shapefiles, Oracle Databases, PostGIS databases, WFS, etc. For raster: ArcGRID, GeoTIFF, ImageMosaic, etc. Our implementation has 10 stores. An interesting functionality is the connection to an external WMS using a store (**cascading WMS**).
- **Layers:** in GeoServer, a layer holds the metadata information about a feature type. In this Section you can add (register) a new layer or delete it (unregister). In the application there is a **total amount of 15 layers**. Layers are located inside stores.
- **Layer Groups:** used to group together related data. We do not use this functionality in the application.
- **Styles:** the spatial data visualization is controlled in GeoServer through styles, XML files containing a detailed description of how a feature type has to be drawn on a map. GeoServer uses **SLD** markup language. SLD is powerful and versatile language that renders both vector and raster data. Our implementation has a total of **5 styles**. When a style is applied to a layer, the developer can see the result before publishing using the OpenLayers **Layer preview** functionality.

Services Section is the configuration area of OGC services. By default WMS, WFS and WCS are available, but installing an extension WPS is incorporated as well. Some

tasks performed in this panel are, for example, enabling the Transactional functionality of WFS or enabling WPS libraries.

Settings Section contains global setting options (for example, changing the log file location or the character set). JAI is managed here as well. JAI is a library developed by Oracle for advanced image manipulation. GeoServer can run without it, but using it greatly improves performance when working with raster data. As WMS uses image formats, it decreases waiting times. Finally, there is Coverage Access Settings subsection.

Tile caching Section is key for a good performance. Please refer to next Section for more details.

Security Section is explained in Section 4.4.1.1.4.

Demos Section presents a set of tools that make easier the work with standards and spatial processes. The available tools are Demo requests (example requests for GeoServer), SRS List (List of all SRS known to GeoServer), Reprojection console (simple coordinate reprojection tool), WCS and WPS request builder. Specifically, WPS request builder is very useful to compose XML Post requests.

4.4.1.1.3. PERFORMANCE AND CACHING

Fu and Sun (2011) state that *caching involves generating maps and performing queries and other processes in advance rather than performing them at run time and then storing the result for future use*. Caching is necessary to improve the server performance. To do so, GeoServer uses **GeoWebCache**¹⁹. It is a Java web technology used to cache map tiles coming from a variety of sources like WMS. The working process is as follows: it has no sense to process every client request from scratch. This may result in increasing waiting times until the resource is delivered. To avoid this, GeoWebCache saves (catches) map images (also called tiles) as they are requested, acting as a proxy between client and server. When the client requests new calls,

¹⁹ <http://geowebcache.org/> [accessed Jan 16th, 2014]

GeoWebCache checks these calls and returns back to the client pre-rendered tiles if stored (or calls the server to render new tiles if they are not available). Thus, once the tiles are stored at different scale levels, the speed of map rendering increases by many times. **Our application has enabled tiling:** GeoWebCache is automatically configured to activate caching in every new WMS layer. It generates 256x256 pixels tiles. The default tile image formats for both vector and raster layers are PNG and JPEG and they use EPSG: 4326 and EPS: 900913 Coordinate Reference Systems. When a new layer is added to Geoserver, GeoWebCache seeds it. **Seeding** is the process of pre-calculating tiles. It avoids some users experiencing a delay when requesting zoom levels and areas yet not cached. Seeding has a huge impact on performance.

4.4.1.1.4. SECURITY CONSIDERATIONS

Any web application is vulnerable to a wide range of attacks. In a production environment there are many security issues. Although we have developed the application locally, some measures have been introduced:

- **Strong encryption** enabling. GeoServer can store passwords in an encrypted format. We use this encryption functionality.
- **Role-based identification.** Now only the administrator has access to the server. If an unauthorized user wants to change a layer, the server throws a *401 error: Unauthorized.*
- **Setting a proxy.** It is not a good idea to expose Tomcat directly to users. A safer option is to use Apache httpd. To expose GeoServer we have set a proxy on the web server. Clients will point to an alias and their requests will be redirected to Tomcat, more safely deployed.

4.4.1.2. DATABASE TECHNOLOGY: POSTGRESQL / POSTGIS

In order to get a good performance and to benefit from the features of object-relational databases, the implementation is built upon a **Postgresql²⁰ database**, with spatial extension **PostGIS** enabled. Spatial databases store and manipulate spatial objects like if they were any other objects. While databases normally hold strings, numbers, dates, etc., spatial databases hold geographic features. The spatial database is not only the place in the architecture where the data is stored. It provides a simple API (SQL) for both CRUD and analysis operations.

PostGIS stores vector and raster data. The spatial data is organized in a hierarchy. In the case of vector data, the basic **geometry types** are (OGC, 2010):

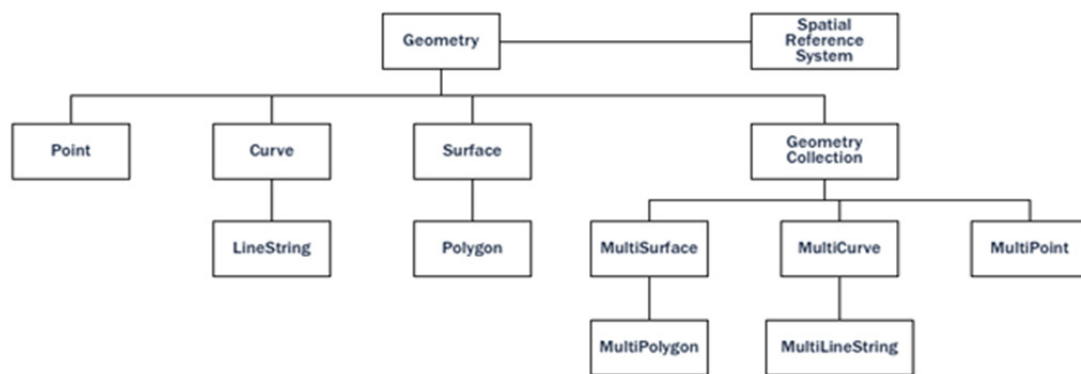


Fig. 17.- PostGIS geometry hierarchy (OGC, 2010)

PostGIS achieves efficiency during querying operations thanks to **spatial indexing**. Every geometry is simplified by its bounding box (the minimum rectangle capable of containing any given feature) and then, using some indexing technologies (as R-trees, Quadtrees, etc.) the database applies the spatial query first for the bounding boxes (process that is computationally efficient). Once a subset of bounding boxes is selected, spatial functions are applied only into the geometries inside the bounding boxes.

²⁰ <http://www.postgresql.org/> [accessed January 16th, 2014]

PostGIS strength is not only spatial data storing, but also the capabilities of a set of **functions** that perform many different tasks. In fact, one of the approaches for working with spatial processes is to use PostGIS functions. Some functions are: construction (building geometries), serialization (outputting geometries in various representations), predicates (testing relationships between geometries), analysis (like distances or areas), accessors (stripping out parts of geometries), builders (like unions, buffers, etc.) and aggregates. The complete list of PostGIS functions can be checked in POSTGIS (2014).

When we use PostGIS as a store (or for CRUD operations), the process is straightforward. The client's API (OpenLayers) has classes to manage these operations, hiding the complexity of the communications with the spatial database. If we use PostGIS as a **processing engine**, the process is a bit more complex:

1. The definition of the SQL is done dynamically using the front-end. In a regular case, the user does not write his or her own queries, but he uses GUI elements (like dropdown lists, radio buttons, etc.) to define them. It is the work of JavaScript to construct the query from the user's input.
2. JavaServer Pages (**JSP**²¹) technology sends the SQL from OpenLayers to PostGIS. Specifically JavaServer Page Template Library (**JSPTL**²²) has scripts to do that: read the parameters from the user, form the SQL query and send the query to the spatial database. This step is somehow a middleware that bridges the gap between the user and the database.
3. PostGIS executes the SQL code and returns the result.
4. Again JSP manages the result (it sends it back to OpenLayers directly or it does some wrapping operations, etc.).
5. OpenLayers shows the results (resulting layers, text or whatever). Sometimes OpenLayers parses the result coming from JSP.

²¹ <http://www.oracle.com/technetwork/java/index-jsp-138231.html> [accessed January 17th 2014]

²² <https://jcp.org/aboutJava/communityprocess/final/jsr052/> [accessed January 17th 2014]

Next figure shows the general structure of this process:

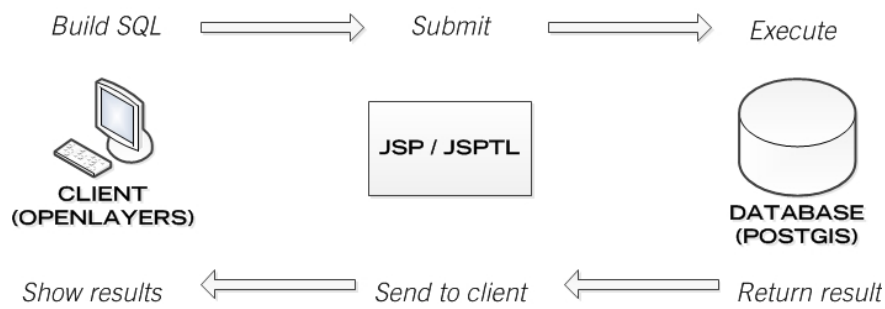


Fig. 18.- Processing with PostGIS. Source: Author.

Although this research is focused on WPS, there are some situations in which this approach can be a good alternative. For example, LiDAR data is hardly manageable using web processing services, but PostGIS can perform tasks on it in a very efficient way.

PostGIS is a good solution but it has a problem to be noticed: it does not exist direct integration for geodetic coordinates (those represented by latitude and longitude). PostGIS assumes that coordinates are Cartesian. If we need geodetic coordinates, we have to use special functions to reproject them into a Cartesian space. For the rest it meets the expectation of most environmental / forestry applications.

4.4.1.3. CLIENT TECHNOLOGY: OPENLAYERS

4.4.1.3.1. INTRODUCTION

In the architecture Section we discussed about the kind of client that is suitable for our application and we concluded that a thin client is an appropriate solution. That means the server performs most of the tasks. Despite that, we also declared that to speed up the application, the client controls some of the tasks (like editing layers, querying results or generating legends). In any case, a client side technology is needed to bridge the server functionality to the user and to build the GUI. This client side technology is OpenLayers. It is an open source JavaScript library for making interactive web maps, viewable in nearly any web browser. Since it is a client side library, it does not need any special server-side installation or settings. The only thing we need to make it

work is the code itself. OpenLayers library adopted version is 2.13. The library gives us access to a set of functionality structured in its API²³.

4.4.1.3.2. PROCEDURE TO SET THE APPLICATION UP

This procedure explains how to consume data from GeoServer using OpenLayers, through an HTML file with a connection to the library:

1. The element that hosts the structure of the application is an HTML file. In our proposal, the use of this HTML is reduced to only a mere container because is JavaScript code that provides functionality using other technologies like ExtJS and GeoExt.
2. In the *<head>* tag of the HTML file we introduce the connection to the OpenLayers API using a *<script>* tag. The library is placed on the server to have a static copy of it. It is not recommended to point directly to the library in the OpenLayers website because of server shut down situations or version changes.
3. We create another script tag that contains all the code of the application. This script could be placed inside or outside the HTML file and it creates a map object from the corresponding OpenLayers class. This object sets up essential elements as projections, extent, zoom levels, layers, scales, etc.
4. The map object is not enough. That object connects to a set of other layer objects. These objects define the way to connect to GeoServer, the web service used (WMS, WFS, etc.) and consequently the data visualization.
5. Once layers are created, we add those layer objects to the map.

²³<http://dev.openlayers.org/releases/OpenLayers-2.13.1/doc/apidocs/files/OpenLayers-js.html> [accessed September 15th, 2013]

6. To manage layers and user events, we introduce the appropriate control objects. Some examples of controls are: navigation, editing toolbar, graticule, scale, zooming, etc.

This is a quite simple explanation of the process. OpenLayers is a complex technology with a steep learning curve. For details of the implementation please check Appendix B.

To solve bugs during the programming process on the client side, we have used **Firebug**²⁴.

4.4.1.3.3. COORDINATE REFERENCE SYSTEM

By default OpenLayers uses CRS EPSG: 4326 (Datum WGS-84, units in degrees). If we wish to use a CRS other than the default one, we need to tell OpenLayers the type of CRS to use by setting its definition (maximum extent, maximum resolution, projection and units). In order to avoid projection problems, we have to set up all layers with the same CRS.

A usual way to give the user a basic navigation framework is by setting up base layers. These layers usually come from third party mapping APIs. These are external APIs that add a complete cartographic context for all the extent of the project. Some of the most known are: Google maps²⁵, Microsoft Bing Maps²⁶, Yahoo²⁷ (now belongs to Nokia) and OpenStreetMap²⁸. For our application, Google maps and OpenStreetMap will be used as base layers (see next Section for details). A common feature of these two APIs is that both use a CRS called **Spherical Mercator**. If we want to mash up layers on top of them, we need to use this coordinate system. The official EPSG code is 3857, but it has some other codes that are still used as EPSG: 900913 and EPSG: 3785. Fortunately OpenLayers provides support for transforming from EPSG: 4326 to EPSG: 900913 or vice-versa.

²⁴ <http://getfirebug.com> [accessed January 17th, 2014]

²⁵ <https://developers.google.com/maps/documentation/javascript/> [accessed January 17th, 2014]

²⁶ <http://www.microsoft.com/maps/choose-your-bing-maps-api.aspx> [accessed January 17th, 2014]

²⁷ <http://developer.here.com> [accessed January 17th, 2014]

²⁸ http://wiki.openstreetmap.org/wiki/API_v0.6 [accessed January 17th, 2014]

If we want to do transformations between other systems, we can adopt one of these two approaches: the backend server handles project transformation or transformations are performed on the client side. If the second option is chosen, we need another library called Proj4js²⁹.

```
PROJCS["WGS 84 / Pseudo-Mercator",
  GEOGCS["WGS 84",
    DATUM["WGS 1984",
      SPHEROID["WGS 84",6378137,298.257223563,
        AUTHORITY["EPSG","7030"]],
      AUTHORITY["EPSG","6326"]],
    PRIMEM["Greenwich",0,
      AUTHORITY["EPSG","8901"]],
    UNIT["degree",0.0174532925199433,
      AUTHORITY["EPSG","9122"]],
    AUTHORITY["EPSG","4326"]],
  PROJECTION["Mercator_1SP"],
  PARAMETER["central_meridian",0],
  PARAMETER["scale_factor",1],
  PARAMETER["false_easting",0],
  PARAMETER["false_northing",0],
  UNIT["metre",1,
    AUTHORITY["EPSG","9001"]],
  AXIS["X",EAST],
  AXIS["Y",NORTH],
  EXTENSION["PROJ4","+proj=merc +a=6378137 +b=6378137 +lat_ts=0.0
+lon_0=0.0 +x_0=0.0 +y_0=0 +k=1.0 +units=m +nadgrids=@null +wktext
+no_defs"],
  AUTHORITY["EPSG","3857"]]
```

Fig. 19.- Definition of EPSG: 3857

4.4.1.3.4. THIRD PARTY MAPPING APIS

The use of third party mapping APIs is a basic step to construct **map mashups**. This term refers to an application that combines various different data sources and functionality together (Fu and Sun, 2011). The existence of a formal API simplifies immensely the integration process. The idea is to bring together the resources that these external APIs have and present the combination in an application.

Google maps is maybe the best known mapping application worldwide. It has a powerful API and high quality cartography, so it is included in our application. Some important things to take into account are:

- There are some legal restrictions in its use. By the time of this writing (January 2014), Google states that if the application generates more than 25,000 map

²⁹ <http://trac.osgeo.org/proj4js/> [accessed January 17th, 2014]

loads each day for more than 90 consecutive days, the API would not be free of charge (Google, 2014).

- We use API version 3. It implies some changes respect version 2:
 - There is no need to include any key as in version 2.
 - The names of layers are different. We will introduce HYBRID, TERRAIN, SATELLITE and ROADMAP layers.
 - These layers are configured by default as Spherical Mercator.
- OpenLayers has the *Layers.Google* class that connects with the different Google layers. The following script tag (introduced in the *<head>* tag of the HTML document) activates the Google API functionality:

```
<script src="http://maps.google.com/maps/api/js?sensor=false"></script>
```

OpenStreetMap (OSM) is a free, wiki-styled map of the world driven by user contributed content. These crowdsourced data is available under an open database license. In this case we do not need to introduce any API address and any type property in our code to enjoy its functionality.

4.4.1.4. OTHER TECHNOLOGIES

4.4.1.4.1. EXT JS

Ext JS³⁰ is a JavaScript library for building interactive web applications. It facilitates the creation of GUIs with a professional look, making the use of HTML not necessary. The learning curve is gentle and the documentation is very complete. Although last version is 4.2 we use version 3.4. The justification is that there is a modified version for geographical purposes (GeoEXT) based in 3.x versions (see next Section). Ext JS is proprietary software, but it is also available under the GPLv3 license for open source projects.

³⁰ <http://www.sencha.com/products/extjs/> [accessed January 17th 2014]

The application content is structured using what Ext JS calls a **Container Layout**. It controls the size and position of the different components. There are many types of layouts. The one selected for the application is called **BorderLayout**. It is a multi-pane that supports multiple nested panels and defines some expanding and collapsing regions: NORTH, SOUTH, WEST, EAST and CENTER. Each region is typically a container or panel that holds all the components for that Section of the GUI (see Figure 20).

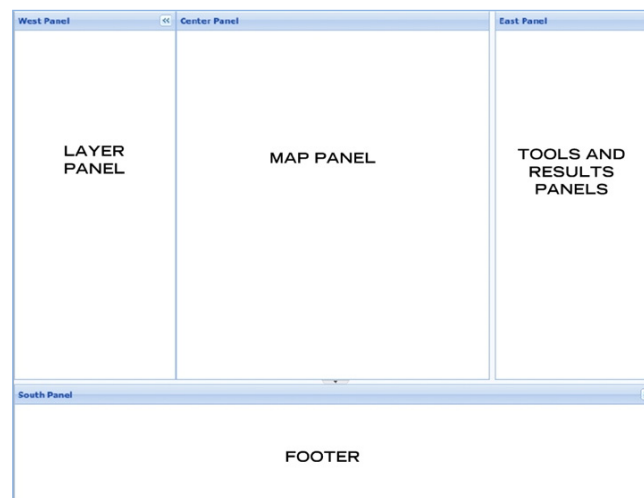


Fig. 20.- Structure of the GUI using Ext JS BorderLayout. Source: Author.

Furthermore, ExtJS provides some other functionality (some other elements are explained in the next Section):

- **Panel:** it is a container that has specific functionality as structural component. It is the basic building block of the application. Panels can have child elements. A panel may also contain bottom and top toolbars and have built-in functionality for being collapsible, expandable or closable if desired.
- **TabPanel:** this kind of panel inherits the properties and functions of the panel class defined above. It is a very useful panel because it organizes content as separate tabs. This kind of panel is used in the application to organize the different spatial processes (one process by tab).

- **TreePanel**: it provides tree-structured representation of data and consequently is suitable to show the layers list in the application.
- **Button**: this component creates simple buttons. They have a handler that executes a JavaScript function when an event is fired (as for example by clicking on it).
- **ComboBox**: this component is a modern version of the traditional HTML dropdown list. We can select an item from the list in order to get some action or to configure some input. We use it to select layers as inputs in spatial processes.
- **MessageBox**: it generates different styles of message boxes. It is used when the system needs to output something to the user.
- **Window**: it is a specialized panel intended for use as an application window. Windows are floated, resizable and draggable by default. They can be maximized, restored to the previous size or minimized. In the application they are used in different situations, like for showing up the attributes of the layers or the legend.

4.4.1.4.2. GEOEXT

GeoExt³¹ is a JavaScript library that brings together Ext JS with OpenLayers. The functionality used in our application is:

- **LegendPanel**: it is a panel showing legends of all layers that are active in a layer store. That means only layers that are rendered in the map panel are shown in the LegendPanel. This component uses an Ext JS window.
- **Popup**: when a feature of any layer is clicked, a popup with the attributes of that feature shows up (thanks to the operation `getFeatureInfo` of WMS standard). In fact, popups are a specialized window that supports anchoring to

³¹ <http://geoext.org> [accessed January 17th 2014]

a particular location of the MapPanel. This class extends Ext.Window class explained before.

- **Action:** this component is used to execute most of the toolbar's tools. The settings that the component needs include the icon to be shown, whether it belongs to a toggle group (like a toolbar) and how to handle the event when it occurs.

In general there are many ways to solve the same problem combining Ext JS and GeoExt. For the implementation details please check Appendix B.

4.4.1.4.3. DUALMAPS

DualMaps³² is a JavaScript library that combines Google Maps, Google Street View and Microsoft Bing Maps (including bird eye) in one embeddable control. It is free and easy to use. With few lines of code you get a popup window once the user clicks somewhere on the map, and you can see the combination of these three resources in just one place.

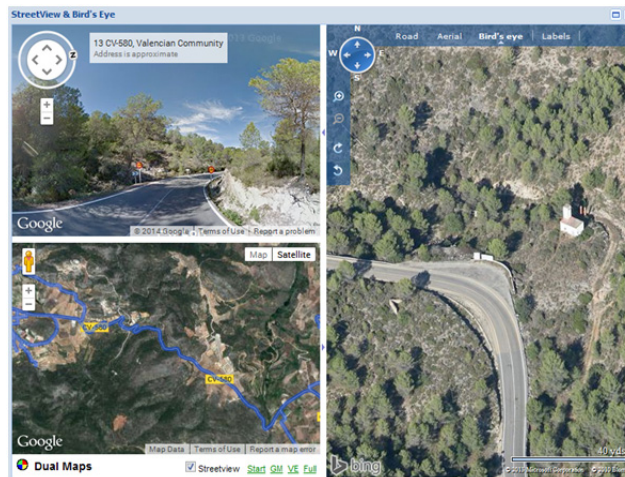


Fig. 21.- DualMaps example

³² <http://www.mapchannels.com/DualMaps.aspx> [accessed November 18th 2013]

4.4.1.4.4. OPENWEATHERMAP

OpenWeatherMap³³ is a JavaScript library that provides free weather data and forecast for the whole world. By including the library in the HTML document and introducing some code we get access to the following weather maps: clouds cover, precipitation (qualitative), sea level pressure, wind speed, temperature (qualitative), snow precipitation and current weather (with a symbol and current maximum and minimum temperature).

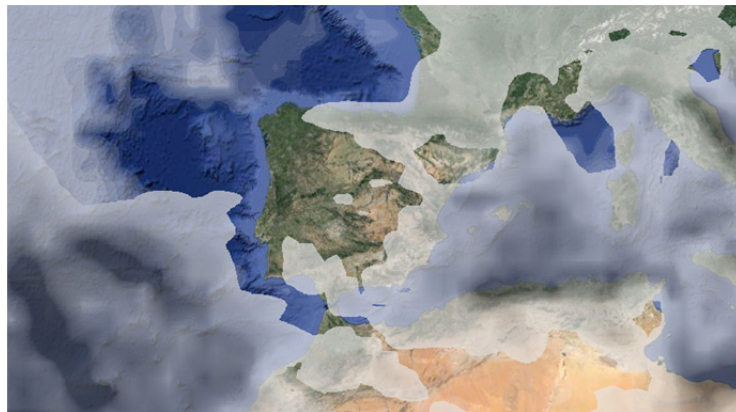


Fig. 22.- OpenWeatherMap example

4.4.2. GENERAL ORGANIZATION OF THE APPLICATION

The work is developed using a **local host environment** to avoid the costs of a real server. The following folder and files structure helps the maintenance of the different elements and keeps a good organization. It has a logical distribution of files depending on its functionality:

- appName: **thesis** (root folder)
 - **cgi** (folder): contains the proxy.cgi file with the proxy configuration.
 - **ext** (folder): contains the elements of the Ext JS 3.4 library.
 - **geoext** (folder): contains the elements of the GeoExt 1.1 library.
 - **openlayers** (folder): contains the elements of the OpenLayers 1.13.1 library.

³³ <http://openweathermap.org> [accessed November 18th 2013]

- **resources** (folder):
 - **css** (folder): contains the external CSS files used in the project.
 - **data** (folder): contains a set of sub-folders with the cartography used by the server.
 - **dualmap** (folder): contains the elements of the Dual Map library.
 - **openweathermap** (file): contains the JavaScript file of OpenWeatherMap library version 1.3.4.
 - **img** (folder): contains the images used in the application like the button icons or the top banner.
 - **proj4js** (folder): contains the file of the Proj4js library, included for managing the different CRS.
- **index.html** (file): the file that is opened by the client and where all the code is located. It is the **entry point** to the application.

This structure is placed inside the WEBAPPS / ROOT folder of the server. This way the access is done through this link: <http://localhost:8080/thesis/index.html>

4.4.3. DATA

4.4.3.1. DATA DIRECTORY STRUCTURE

Every Web GIS application needs a data folder. Its name is **Data Directory** and its structure is built automatically with the GeoServer installation. Once created, to change the structure we use the dynamic interface explained in Section 4.4.1.1.2. The content of this folder is:

- A set of **xml files** (global.xml, logging.xml, wcs.xml, wfs.xml, wms.xml and wps.xml) stores the service information and global settings.
- **Workspaces** directory has as many folders as workspaces. Within a workspace, there are as many folders as datastores. Within a datastore, there are as many folders as layers. At the bottom of this structure, there is the

metadata about the layer (3 xml files): layer.xml, coverage.xml and featuretype.xml.

- **Data** directory is where the spatial data actually exists. This folder benefits portability. If we create another instance of GeoServer, it will look for information in this folder.
- Other directories: **demo**, **geosearch** (KML files), **gwc** (GeoWebCache), **layergroups**, **logs**, **palettes**, **security**, **styles**, **templates**, **user_projections** and **www**.

4.4.3.2. LIST OF LAYERS

The application uses different datasets. In **Appendix A** there is a record for every layer with its description, type, how the server presents it (type of service), original scale, production institution, production date, a thumbnail and other important metadata. Data is organized in two big groups: **base layers** and **operational layers**. Base layers contextualize the navigation through basic cartography like topographic maps and orthophotomaps. Operational layers are specific layers very related with the scope of the project.

4.4.3.2.1. BASE LAYERS

The base layers introduced in the application are:

- OpenStreetMap.
- Google Maps Satellite Layer.
- Google Maps Terrain Layer.
- Google Maps Hybrid Layer.
- Google Maps Roads Layer.

For a full description please check Appendix A.

4.4.3.2. OPERATIONAL LAYERS

The operational layers introduced in the application are:

- Municipalities.
- Natural Areas of Communitarian Interest.
- Public Forest Boundaries.
- Digital Elevation Model 200 m.
- Contours every 5 m.
- Average yearly rainfall (in tenths of mm).
- Roads.
- Archaeological sites.
- Actual weather.
- Actual cloud cover.
- Actual qualitative rain intensity.
- Actual qualitative temperature.
- National Forestry Map.

For a full description please check Appendix A.

4.4.3.3. DATA CRS

OpenStreetMap and Google maps base layers require the use of EPSG: 3857 CRS (see Section 4.4.1.3.3). To avoid projection problems between different layers, we follow this process:

1. Before uploading the cartography to GeoServer, the layers are transformed from their original CRS (normally ETRS-89 30 N) to Pseudo-Mercator (using a Desktop GIS, Quantum GIS³⁴).

³⁴ <http://www.qgis.org> [accessed November 18th, 2014]

2. Once inside GeoServer, Pseudo-Mercator is selected as the Native CRS and also as the Declared CRS. This step is performed during the publishing process configuration.
3. The tiles CRS creation is set to Pseudo-Mercator.
4. Finally, in the JavaScript file, the CRS of the map is set to Pseudo-Mercator as well.

4.4.4. WEB SERVICES IMPLEMENTATION

In our SOA, layers are published through different types of web services. The choice of the web service type depends on the use we want for the layer. If visualization is required, WMS is used. If we want to use the layer to perform CRUD operations, we use WFS. Finally, if we need a raster as an input in a process, we need WCS. Next paragraphs describe the different service configurations.

4.4.4.1. WEB MAP SERVICE IMPLEMENTATION

WMS configuration uses the *OpenLayers.Layer.WMS* class to publish data through this standard. The output format is image/png (to get transparency when needed). The *isBaseLayer* property separates, using a Boolean, between base layers and operational layers. To change the layer transparency using a slider, the *transparent* property of the WMS is set to true. Finally, the *visibility* property decides if a layer is shown on the map by default or not. In our case, only municipalities layer (together to a base layer) is loaded by default (this way the waiting loading times are lower). The layers published with WMS are: municipalities, public forest boundaries, natural areas, archaeological sites (cascading WMS), roads, contours, rainfall, forest map and DEM. It is worth to note that Google Layers and OSM are published using their own OpenLayers classes (*OpenLayers.Layer.Google* and *OpenLayers.OSM*).

4.4.4.2. WEB FEATURE SERVICE IMPLEMENTATION

This standard is used only once, to store the input polygons of the user in the spatial processes. The layer is called *computestatisticspolygon* and uses the *OpenLayers.Layer.Vector* class. This class needs strategy and protocol classes. **Strategy class** is used to control how requests to the server are set up and then what to do with the data returned from the server. We use two strategies: **Save strategy** and **BBOX strategy**. The former is used to store the data to the server. The latter is used to send a request to the server that asks only for data within the viewable map extent. The **protocol class** controls how vector layer communicates with the source of data. It is here where we create the *OpenLayers.Protocol.WFS* object. The creation of a *Layer.Vector* object with the appropriate configuration allows the use of WFS-T within the application.

4.4.4.3. WEB COVERAGE SERVICE IMPLEMENTATION

Although the application does not have any raster download functionality, WCS is used as input in some spatial processes. The output format is image/tiff. The WCS version we use is 1.1.1. In a regular process, the raster is sent to the server using a POST request, where it is used as input. WCS configuration needs an *OpenLayers.Bounds* object to narrow the raster's extent.

4.4.5. IMPLEMENTATION OF THE FUNCTIONALITY

The functionality fulfils different types of tasks, from very basic (like navigation, zooming, panning, etc.) to more complex (basically the spatial processes). This Section systematically explains the complete functionality of the application, with some details about the key points of the programming process. It does not try to detail every line of code (please check Appendix B) but how the different technologies are put together and how they work. This description is organized following the panel structure of Fig. 20.

4.4.5.1. LAYERS PANEL

4.4.5.1.1. LAYERS PANEL DESCRIPTION

This panel shows the layer organization. It has a **tree structure** for quick access to every element. Layers are divided into two groups: Base Layers and Overlays. For making the access simpler, overlays group has a set of other groups inside depending on the topic of the layers. Some examples of groups are: Protected Areas, Land Property, Climatology, etc. In the actual definition of the map object, a base layer has to be always active in order to make the navigation coherent. Then the user can activate an unlimited number of overlays. To maintain this structure, drag and drop is not available. The different layers can be activated just using the “tick” next to their name, or the radio button in the case of Base Layers. By clicking on the top right arrow button, this panel collapses, making bigger the area devoted to the map.

4.4.5.1.2. LAYERS PANEL IMPLEMENTATION

The panel uses an Ext JS **TreePanel object**. It has this structure:

- A **loader**, necessary to allow the access to the content.
- A **root** that connects to the **tree configuration**. The tree configuration defines the structure of the tree panel and it is a JavaScript array.
- Some **other configuration options**, like disabling drag and drop, or the place where the panel must be rendered.

The tree configuration indicates the type of each node (i.e. if it is a base layer container or not), if it is expanded by default, the layers it has to show, the name that should appear in the screen for each level, etc. For more information please check Appendix B (look for Tree Panel code comment).

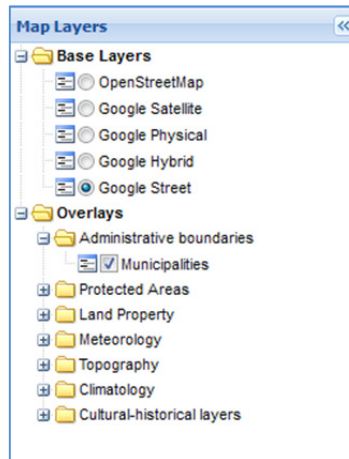


Fig. 23.- View of layers panel

4.4.5.2. MAP DISPLAY PANEL

4.4.5.2.1. MAP DISPLAY PANEL DESCRIPTION

This panel fills most of the screen area and it is not collapsible. It shows the map and permits most of the user – application interaction. In the top left of the panel there are two simple controls to zoom in and zoom out. By default this panel allows panning, dragging and dropping. Also by default the map is zoomed to the extent of the study area and the municipalities layer is uploaded. Google Maps Road layer is the default base layer. In the top right side there are some buttons for editing tasks:



The functionality of these buttons is (from left to right):

- The first button allows a polygon’s drawing in any area of the map by clicking in each of the vertices we want to draw. Once this button is selected, the cursor changes its shape to a small circle, indicating that editing is ready. To finish the polygon the user has to make a double click and an orange semi-transparent polygon appears.
- The double arrow moves the polygon’s vertices. If this button is clicked and the polygon is selected, a set of small circles appears, allowing the movement of each individual vertex. During the editing process the polygon appears in blue.

- The third button deletes any existing polygon by clicking on it.
- The fourth button (the floppy disk) saves the changes to the spatial database. Once we decide the polygon we want, clicking on it will store the element in the PostGIS database. Every time a change is done (like a deletion or a creation) this button must be clicked. When the user tries to store to the database, the system displays a message indicating that all was ok (see Fig. 24).

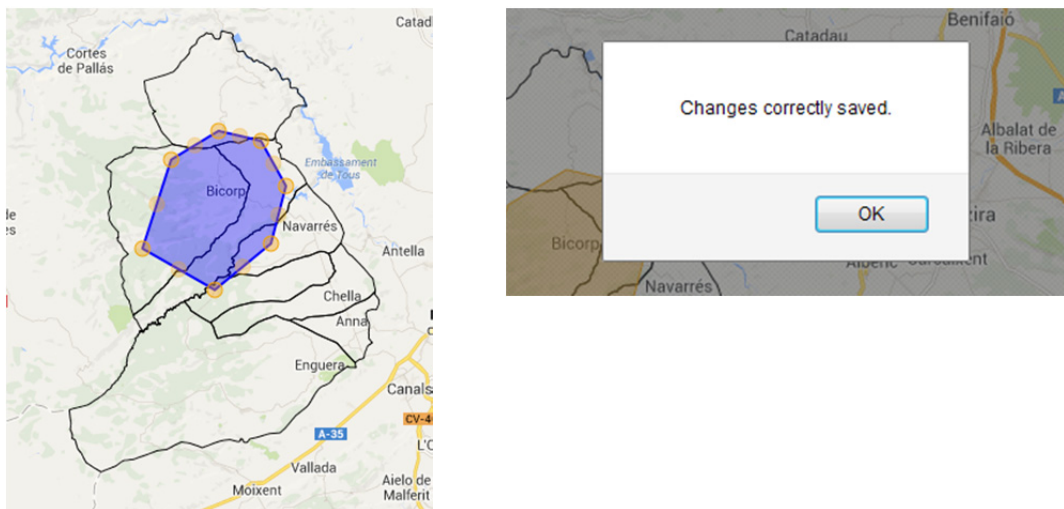


Fig. 24.- Some images of the editing process

4.4.5.2.2. MAP DISPLAY PANEL IMPLEMENTATION

The complete GUI is structured using an Ext JS Viewport component. It is a specialized container representing the viewable application area. The map display panel is placed in the CENTER region of the border layout of this Viewport component. Each region has an xtype property, which indicates the type of the content that will be introduced inside each panel. In this case the xtype is “gx_mappanel”, which has these properties:

- **Layers:** an array with the list of variables that store the layers’ references.

- **Centre:** an *OpenLayers.LonLat* object that is used to centre the map when loaded.
- **Zoom:** an integer indicating the default zoom level.
- **Map:** the variable that stores the *OpenLayers.Map* object.
- **Tbar:** the toolbar that is shown in this panel. Its components are explained in next Section.

The functionality of each editing button is defined in a variable (one variable per button). Each of these variables store an *OpenLayers.Control* object:

- To draw the polygons, *OpenLayers.Control.DrawFeature*.
- To modify the polygons, *OpenLayers.Control.ModifyFeature*.
- To delete features, an ad-hoc class has been created (see Appendix B).
- To save features, a function is used. This function uses the *saveStrategy* of WFS object for results storage.

The four buttons are grouped in an *OpenLayers.Panel* object and the icon images are the ones that come by default in the CSS OpenLayers library files. This editing toolbar is an example of the distribution of workload between server and client. All these tasks are performed in the client side, except when the user clicks on “save” and then the information is sent and stored on the server.

4.4.5.3. TOOLBAR

4.4.5.3.1. TOOLBAR DESCRIPTION

This toolbar contains basic (and some advanced) functions to achieve a good user experience and to present some typical tools that all Web GIS applications should have. These are (from left to right):



- **Zoom in by box:** the user draws a rectangle defining the extent to zoom in. This is a toggle button. It means that once the user has finished using this tool, he or she has to click again to deactivate it.
- **Zoom out:** it zooms out by one level.
- **Previous Zoom / Next Zoom** (next two buttons). These tools allow moving in the zoom history. When the user makes a pan or zoom, the extent is recorded.
- **Zoom to Extent:** the predefined extent is the study area (the municipality layer). If the user clicks on it, the zoom level and the extent are changed to fit that area.
- **Measure length.** The user can measure dynamically partial or total lengths. By clicking the starting point and the end point of a segment, the distance appears on the screen. Double clicking returns the total length value of all segments (next to the right side of the toolbar). The units change from meters to kilometres if the measure is over 1000 m (so 1 km). The values are presented with two floating points positions. This tool is used as toggle button.
- **Measure area.** This tool works in a similar way to the measure length tool. It returns partial and total area, both in square meters or hectares (if we are over 10000 m², so 1 ha). This tool is used as toggle button.
- **Identify tool.** As expected, this tool shows the attributes of the active layers in a popup window. This window contains the attributes that correspond to the clicked feature and it has minimizing, maximizing, anchoring, moving and closing actions. If more than one layer is active, the application shows up a unique window with multiple tables. This tool is used as toggle button.
- **Clear tool.** As the different tools are used, the screen starts to show up windows, measurements, etc. In order to clear all that content there is a clear tool (the rubber).

- **Dual View.** As stated in Section 4.4.1.4.3, this tool expects the user to click somewhere in the map and shows up a window with a combination of Google Maps SATELLITE, Bing Maps (Bird Eye) and Google Street View (if available). The user can enjoy the combination of these technologies in one click. The window has moving, maximizing and closing actions. This tool is used as toggle button.
- **Show legend.** It creates a dynamic legend panel. The layers included are those that are visible in the map display panel. The window can be maximized, minimized, moved and closed.

4.4.5.3.2. TOOLBAR IMPLEMENTATION

From the programming point of view, the toolbar is a JavaScript array that stores a list of variables. Each variable stores an Ext JS Action. The toolbar code is in Appendix B. It is worth to highlight some interesting aspects of this implementation:

- Zoom in by box is a **GeoExt.Action** component stored in a variable that connects to the **ZoomBox** OpenLayers Control. The handler of the Action component manages the way the cursor is shown depending on the user clicks.
- Zoom out code is much easier than the previous one. It is just an **Ext.Button**. The handler of the button is a function that calls the built-in *OpenLayers.zoomOut* map function when clicked.
- Zoom previous and Zoom next: the navigation history is stored in the *OpenLayers.NavigationHistory* control. These zoom tools are a **GeoExt.Action** that calls the property “previous” and “next” of the control (that restores the previous or next state managed by the control), permitting the forwards and backwards movements.
- Zoom to Extent is similar to Zoom out code, but in this case the handler triggers the **setCenter** OpenLayers function. The code is also stored in a variable with an **Ext.Button**.

- Measure length. This tool is more complex. First of all, we define the styles for the points and lines that the user will draw on the map. For doing that we use the **addRules** function to the *OpenLayers.Style* object and then apply it to an *OpenLayers.StyleMap* object. The *OpenLayers.Control.Measure* object performs the length measurements. An ad-hoc function changes the units if we are over 1000 m and prints out the result next to the toolbar.
- Measure area. The structure is exactly the same that for the measure length tool. The main change is the argument of the *OpenLayers.Control.Measure* object, which takes in here an *OpenLayers.Handler.Polygon* instead of an *OpenLayers.Handler.Path* as before.
- Identify tool. This tool is also relatively complex. The main parts of it are:
 - The first step is to store the results of the querying in a variable. To do so, we use the *OpenLayers.Control.WMSGetFeatureInfo*.
 - The second step consists in creating the popup using the *GeoExt.Popup* component. It is placed inside a function that sets up its characteristics.
 - Finally, the control defined in the first step is called using a **GeoExt Action**. This action is executed when the button is active and the user clicks somewhere on the map. It also controls the look of the cursor.
- Clear tool uses an **Ext.Button** component to “clear” all the content created by the calculate length and calculate area tools. It also closes any opened popup. To do that it uses the *popup.close()*, *length.cancel()* and *area.cancel()* functions.
- Dual View functionality needs also three steps to work. The first step consists in creating the window (using the **Ext.Window** component), the second step is creating an ad-hoc class that connects with the DualMaps API. Finally, a

GeoExt.Action executes the functionality when the user clicks and also controls the look of the cursor.

- Show Legend needs a window whose component is a **GeoExt.LegendPanel**. It automatically reads from the layer store the active layers and imports their legends from GeoServer. The functionality is executed once a user clicks on the button (again an **Ext.Button**).

4.4.5.4. GEO-PROCESSES PANEL

4.4.5.4.1. GEO-PROCESSES PANEL DESCRIPTION

The application needs a place to present the spatial processes and to show the results of them. To do so, we use the EAST region of the border layout of the Viewport. This collapsible panel has 2 main parts: the **tools list** panel and the **results** panel:

- The tools list is composed by various vertically collapsible panels that divide the spatial processes in categories:
 - **Basic tools**: tools that are not necessarily forestry or environmental – related, but that help to get some insight from the spatial data.
 - **Basic statistics**. Calculates the average, minimum and maximum value inside the area drawn by the user for the selected raster. It is implemented using a WPS.
 - **Raster transparency**. Implements classic layer transparency to help photointerpretation.
 1. The user selects a layer he or she wants to apply the transparency for.
 2. The user moves the slider left to right to change de transparency value dynamically.
 3. The layer with the selected transparency appears on the map.

- **Buffers.** Implements classic buffer calculation around user-defined geometries. It is programmed using WPS.
- **Forestry tools:**
 - **Vegetation description.** Returns a summarized description of the vegetation inside a user-defined polygon. Internally the tool checks the National Forestry Map and gets the names of each different cover. Then it draws a list and returns it to the user. It is programmed using WPS.
 - **Roads description.** Returns some interesting descriptors about the roads within an area defined by the user: total area length (in meters), total road density (in m/ha) and a classification of the road density in low, medium or high depending on the previous result. It is programmed using WPS.

There is a process that has not been implemented in the application but that has been studied independently: the use of LiDAR data to calculate tree heights. In next Section there is a description of the process done to perform this kind of analysis.

4.4.5.4.2. GEO-PROCESSES PANEL IMPLEMENTATION

Basic statistics, buffers, vegetation description and roads description processes are described in Section 5. This Section explains only the processes where WPS is not involved (raster transparency process and LiDAR tree height).

Raster transparency functionality has a quite straightforward implementation. Ext JS has a default xtype called slider that implements for us a classical slider. What this slider does is controlled thanks to a listener function. In this case, the function takes two arguments: the element and the transparency value. The changes are controlled by the *setOpacity* OpenLayers method.

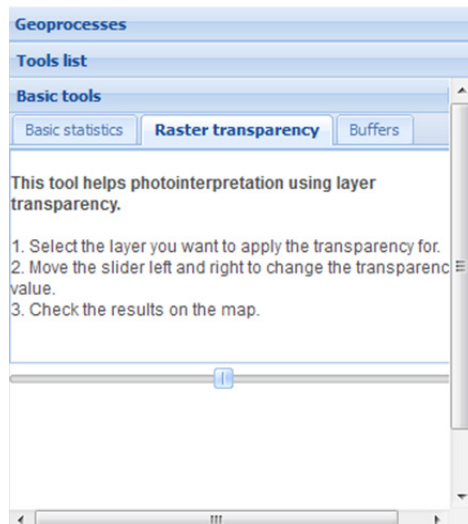


Fig. 25.- Transparency slider

LiDAR tree height. The basics about LiDAR data are explained in the introduction. The process of obtaining tree heights is as follows: the first step is getting a dataset. The National Aerial Orthophotography Program (PNOA³⁵) of National Geographic Institute is in process of collecting LiDAR data of Spain. We have downloaded some data in a forest area of Enguera (Valencia). These data are loaded in the PostGIS spatial dataset. To do that, we need the appropriate extension turned on: pointcloud³⁶. Additionally we need the point data abstraction library (PDAL³⁷) to manipulate the spatial point cloud data. This library has many functions and helps us to read and write different data formats, translate between them, generate the bounds and generate a grid from the points. We use it to chip the raw data into tiles and write the point cloud to the database. Once the data cloud is inside, we can use queries to know more about this data, and to make some processing. The first important problem that appears in here is that PDAL library built in 64 bits systems is not directly supported (there are not binary files for its installation), so the integration with our development framework (that is 64 bits) is not done.

With the data in a database, the second step consists on putting the LiDAR data on the map using a GeoServer installation. The server can only map PostGIS geometries, so we need to convert the LiDAR data to PostGIS geometry data. Then we can see the

³⁵ http://www.ign.es/PNOA/vuelo_lidar.html [accessed 27th January 2014]

³⁶ <https://github.com/pramsey/pointcloud> [accessed 27th January 2014]

³⁷ <http://www.pdal.io> [accessed 27th January 2014]

data (applying some styles to make the look more attractive). This data can be accessed also from OpenLayers.

The final step is checking the trees height. The National Forestry Map defines some polygons with homogenous vegetation types. Within the selected area, we upload polygons in which the tree height is supposed to be more or less constant. This statement is true in regular structure forests (forests with the same age). Irregular structured forests are not studied (this is the second big problem, calculating individual tree heights in irregular structured forests is not performed in this test). To calculate the height we use a SQL query that checks the patches that intersect with the vegetation polygon, turn those patches into individual points and filter those points using the vegetation polygon outline. What we get is the tree elevation, but we want to tree heights. To change from one to the other we need to calculate the elevation of the ground and subtract it from the trees elevation. This calculation is done using the same LiDAR data (the last pulses return the ground elevation). The result is the height of the trees using SQL queries.

In figure 26 there is an example of the look of LiDAR data in OpenLayers. Different ranges of green and yellow depict different surface element heights. Unfortunately, this image is not obtained form the application itself, but form an alternative OpenLayers/GeoServer installation due to system incompatibility of PDAL library.

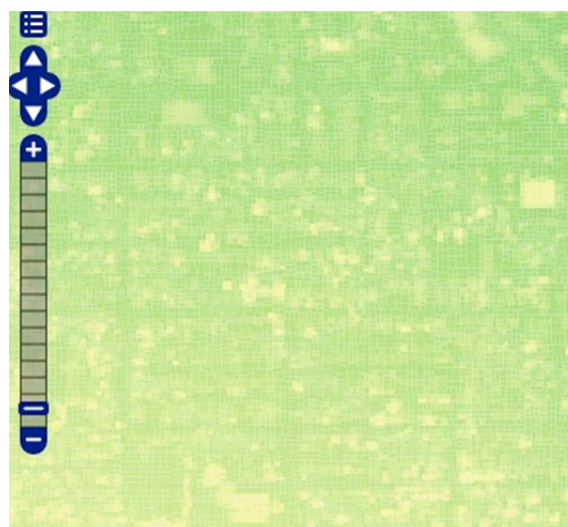


Fig. 26.- LiDAR data integration in OpenLayers

This is an easy example that shows the potential of the use of LiDAR data within Web GIS applications. There is a big room for future improvement and it would be highly useful to provide systems that can manage data and processes in Web GIS frameworks in an easy way.

5. WEB PROCESSING SERVICES

5.1. INTRODUCTION

The benefits that Web GIS development has brought to foster the use of spatial information systems are undeniable. Today it is possible to distribute interactive maps to anyone with an Internet connection. Maybe one of the most powerful and useful features of any GIS is the possibility to execute processes. Spatial web processing offers a wide range of possibilities. This Section details one of these options: the WPS.

According to the standard's document definition (OGC, 2007), WPS is “*a standardized interface that facilitates the publishing of geospatial processes and the discovery of and binding to those processes by clients.*” WPS defines rules for standardizing how requests (inputs) and responses (outputs) should be managed when executing spatial processes.

By the means of this document, a spatial process (or geo-process) is any algorithm, calculation or model that operates on spatially referenced data. It is worth to highlight that WPS can be used for almost any kind of computation, but in this work we only admit spatial processes. The kind of processes that can be managed varies from simple calculations (like an addition of two spatially referenced set of numbers) to more complex models (for example the ones used in remote sensing for defining the vegetation characteristics).

WPS can identify which spatial data are required to perform the calculation / model, to execute them and finally to manage the output coming from that calculation / model. WPS is addressed for both vector and raster data. That means WPS maintains a strong relation with other standards like WFS and WCS. Usually inputs and outputs are taken from or converted to one of these services. Once the WPS is created, the client must input the data and execute the process. In this step, the underlying process is hidden to him or her, so he or she does not have to know nothing about how the process works internally. The chosen service provider exposes a web accessible

process and the client only has to use the defined GUI and wait the process to be executed.

5.2. BASICS ABOUT WPS. THE STANDARD

5.2.1. WPS OPERATIONS

WPS standard (OGC, 2007) specifies three different mandatory operations. The interactions between them are in the next figure (Schaeffer, 2008).

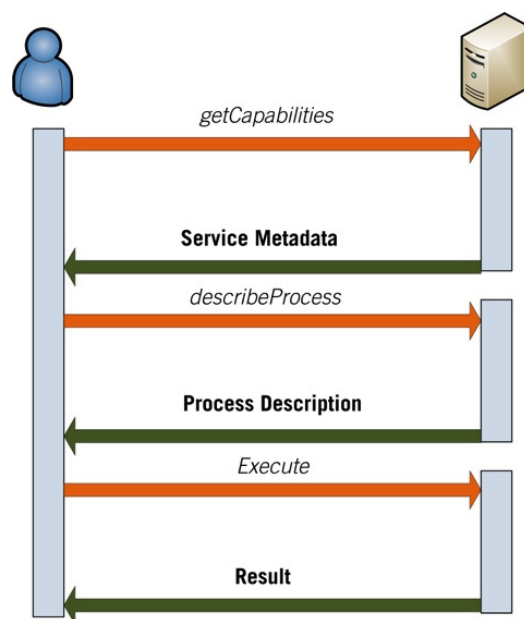


Fig. 27.- Client – Server interactions in a WPS (Schaeffer, 2008)

5.2.1.1. GETCAPABILITIES WPS OPERATION

As usual, this mandatory operation retrieves the service metadata document in XML format, with a brief description of all the processes that are implemented by that service.

5.2.1.1.1. WPS OPERATION REQUEST

The request for the capabilities document is very similar to the other standards we have already studied (see Table 5).

Mandatory parameters		
Name	Value	Description
SERVICE	WPS	Indicates the service type
REQUEST	GetCapabilities	Indicates the request name
Optional parameters		
Name	Value	Description
ACCEPTEDVERSIONS	VersionNumber	Version accepted by the WPS
LANGUAGE	Language	Language used in the service

Table 5.- WPS GetCapabilities request parameters (OGC, 2007)

Here there is an example of a GetCapabilities request in our GeoServer configuration, using a KVP encoding. The response is a XML with the metadata and processes that can be used within the WPS.

<http://localhost:8080/geoserver/ows?service=wps&version=1.0.0&request=GetCapabilities>

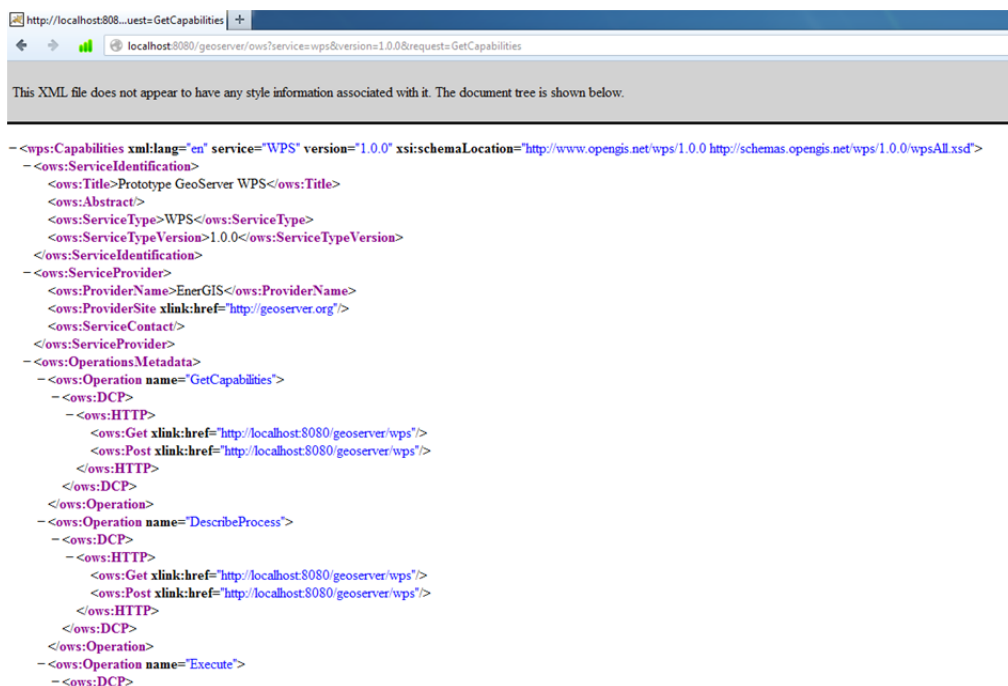


Fig. 28.- Response of a GetCapabilities document

5.2.1.1.2. WPS GETCAPABILITIES RESPONSE

The structure of the XML response is as follows:

- Service identification: title, service type, version.
- Service provider: name, site, contact.
- Operations metadata: information about the operations implemented by the server (GetCapabilities, DescribeProcess and Execute).
- ProcessOfferings: list of the processes with a brief description of each one. The document presents the identifier of the process, the title and a short abstract with its description. In the next figure there is an example of a WPS process description within this capabilities document.

```

- <wps:Process wps:processVersion="1.0.0">
  <ows:Identifier>JTS:area</ows:Identifier>
  <ows:Title>Area</ows:Title>
  - <ows:Abstract>
    Returns the area of a geometry, in the units of the geometry. Assumes a Cartesian plane, so this process is only recommended for non-geographic CRSes.
  </ows:Abstract>
</wps:Process>

```

Fig. 29.- Simple process description in the capabilities document

If the server encounters an error during the execution of the operation, it will return an exception report.

The use of this first request is to have an idea of what the WPS process does and how the processes are called. Once we have found the process we need, we move into the next request (DescribeProcess).

5.2.1.2. WPS DESCRIBEPROCESS

This operation gives a full description of a process, including inputs and outputs parameters and formats. The response document is returned in XML.

5.2.1.2.1. WPS DESCRIBEPROCESS OPERATION REQUEST

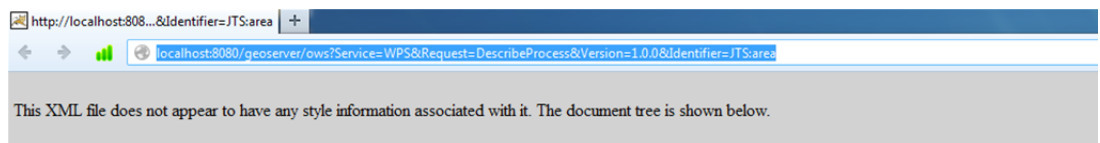
The list of parameters to make a DescribeProcess request is shown in Table 6.

Mandatory parameters		
Name	Value	Description
SERVICE	WPS	Indicates the service type
REQUEST	DescribeProcess	Indicates the request name
VERSION	VersionNumber	Number of the version of the WPS (normally 1.0.0, but also 0.4 can be found)
IDENTIFIER	String	Name of the process we want to describe
Optional parameters		
Name	Value	Description
LANGUAGE	Language	Language used in the service

Table 6.- WPS DescribeProcess request parameters (OGC, 2007)

An example of a DescribeProcess request and its response is as follows (using KVP encoding):

http://localhost:8080/geoserver/ows?Service=WPS&Request=DescribeProcess&Version=1.0.0&Identifier=JTS:area



```

- <wps:ProcessDescriptions xml:lang="en" service="WPS" version="1.0.0" xsi:schemaLocation="http://www.opengis.net/wps/1.0.0 http://schemas.opengis.net/wps/1.0.0/wps.xsd" statusSupported="true" storeSupported="true">
- <ProcessDescription wps:processVersion="1.0.0" statusSupported="true" storeSupported="true">
  <ows:Identifier>JTS:area</ows:Identifier>
  <ows:Title>Area</ows:Title>
  - <ows:Abstract>
    Returns the area of a geometry, in the units of the geometry. Assumes a Cartesian plane, so this process is only recommended for non-geographic CRSes.
  </ows:Abstract>
  - <DataInputs>
  - <Input maxOccurs="1" minOccurs="1">
    <ows:Identifier>geom</ows:Identifier>
    <ows:Title>geom</ows:Title>
    <ows:Abstract>Input geometry</ows:Abstract>
  - <ComplexData>
  - <Default>
    - <Format>
      <MimeType>text/xml; subtype=gml/3.1.1</MimeType>
    </Format>
  </Default>
  - <Supported>
  - <Format>
    <MimeType>text/xml; subtype=gml/3.1.1</MimeType>
  </Format>
  - <Format>
    <MimeType>text/xml; subtype=gml/2.1.2</MimeType>
  </Format>
  - <Format>
    <MimeType>application/wkt</MimeType>
  </Format>
  - <Format>
    <MimeType>application/gml-3.1.1</MimeType>
  </Format>
  </ComplexData>
  </DataInputs>
</ProcessDescription>
</wps:ProcessDescriptions>

```

Fig. 30.- Example of DescribeProcess response

5.2.1.2.2. WPS DESCRIBEPROCESS RESPONSE

The XML response presents the information in Table 7. This table help us to interpret how the process should be built considering the needed inputs and the expected outputs.

Type	Elements	Description
General metadata	-	-
Identifier	-	-
Title	-	-
Abstract	-	-
Data Inputs	Identifier	Each input needs an identifier. It is crucial to know every input name because we refer to it from the client side.
	Title and abstract	-
	minOccurs and maxOccurs	Minimum and maximum number of times that this parameter may be present. The normal situation in which the parameter is required once is expressed like this (XML encoding): <code><Input minOccurs="1" maxOccurs="1"></code> . In the case of an optional parameter it is: <code><Input minOccurs="0" maxOccurs="1"></code> .
	InputFormChoice	Identifies the type of the input and provides supporting information. The types are (see next Section): ComplexData, LiteralData and BoundingBox Data. For each type of data we have: <ul style="list-style-type: none"> Default (mandatory): identifies the default format, encoding and schema for the input. For example, if the process needs a tiff image, it is represented like this: <pre> <Default> <Format> <MimeType>image/tiff</MimeType> </Format> </Default> </pre> Supported: combination of format, encoding and/or schema supported by the input or output. maximumMegabytes: the maximum file size, in MB, for this inputs. If this size is exceeded, the server will return an error.
Data outputs	-	The output description has the same elements that the input description.

Table 7.- WPS response (OGC, 2007)

5.2.1.2.3. TYPES OF DATA INPUTS AND PROCESS OUTPUTS

OGC standard (OGC 2007) defines the types of inputs and outputs that can be used in WPS. These data types are LiteralData, ComplexData and BoundingBox data. Please check Table 8 for the details about the types of data.

Type	Definition	Elements	Description
LiteralData	It can be any character, string, float, date, etc. described as primitive datatype in the W3C XML Schema standard (W3C 2001).	Units of Measure (UOM)	Sometimes the literal value has UOM that are necessary to state, such as when measuring distances, areas, etc. An OGC document (OGC, 2002) explains how to use UOM and gives a set of recommendations.
		DefaultValue	When more than one allowed value is provided, it is necessary to indicate the default value. An example of this is the default distance for a buffer calculation.
		AllowedValues	Indicates a finite set of values and ranges allowed for the input and contains an ordered list of all values and/or ranges.
ComplexData	This data type is used to describe vector, raster or any other data. An element of this data type may contain a MimeType / Encoding / Schema describing the XML content, but it is not compulsory.	Embedded	Embedded inside the XML content. For example, a GML.
		URL reference	URL reference to the data or service that provide the data. This is the usual option.
BoundingBox Data	BBOX is used to define a bounding box area, for example, to narrow the extent of a raster input (and thus decreasing the server workload). Full information about how to define a BBOX data can be found in the Web Services Common Specification (OGC, 2007). Using KVP encoding, the BBOX definition has this structure: <i>&bboxInput=LowerCornerLongitude, LowerCornerLatitude, UpperCornerLongitude, UpperCorneyLatitude, CRS_URI</i> .		

Table 8.- WPS types of data (OGC, 2007)

5.2.1.3. WPS EXECUTE

It runs a specific process using the required input parameters and returning the expected output values if the process executes correctly. As stated before, inputs can be included directly in the Execute request, or referenced from web accessible resources (i.e. other web services). The **outputs** are returned in a **XML response document**, either embedded in this document or stored as web accessible resources. In this last case, the XML includes a URL for each stored output. For a single output, the server can return a raw output without wrapping it in the XML response document.

Processes are normally time-consuming (in fact, this is one of WPS bottlenecks). The standard gives the possibility to get the response immediately (before process execution is finished), returning a URL from which the response document can be retrieved once the execution is complete. This is called asynchronous WPS, but GeoServer does not support it yet. That means **all the processes we develop in this application are synchronous.**

5.2.1.3.1. EXECUTE REQUEST PARAMETERS

The list of request parameters for the Execute operation is in Table 9.

Mandatory parameters		
Name	Value	Description
SERVICE	WPS	Indicates the service type.
REQUEST	Execute	Indicates the request name.
VERSION	VersionNumber	Number of the version of the WPS (normally 1.0.0, but also 0.4 can be found).
IDENTIFIER	String	Name of the process we want to describe.
Optional parameters		
Name	Value	Description
DATAINPUTS	contentType	Encoding and schema of the input.
	href	Reference to web-accessible resource to be used as input.
	method	Identifies the HTTP method. Default is GET.
	Header	HTTP request headers needed by the service.

	Value	Description
	Body	The content of this element is used in the body of the HTTP request.
	BodyReference	Reference to a remote document to be used as the body of the HTTP request message.
RESPONSEFORM	-	The response type, either raw data or XML document.

Table 9.- WPS Execute parameters (OGC, 2007)

5.2.1.3.2. WPS EXCEPTIONS

The possible exceptions that can be returned when performing the Execute operation are presented in Table 10.

Exception	Description
MissingParameterValue	The request does not include a parameter value.
InvalidParameterValue	The value is introduced but it is invalid.
NoApplicableCode	When there is not a code to determine the error.
NotEnoughStorage	The server does not have enough space available to store outputs related to the request.
ServerBusy	The server is too busy to accept and queue the request.
FileSizeExceeded	The input parameter file size is too big.
StorageNotSupported	We want to store the output but the server has not configured this option.
VersionNegotiationFailed	The server does not support the service version for complex referenced data.

Table 10.- Possible exceptions in an Execute WPS request (OGC, 2007)

5.3. IMPLEMENTATION OF THE PROCESSES. RESULTS

5.3.1. INTRODUCTION

Implementing WPS in a production environment requires taking some technical decisions. To make WPS work, we need a **WPS client** (a framework to create and manage WPS processes). Next Section explains some existing options. Our application uses GeoServer WPS extension combined to OpenLayers WPS classes. Choosing the client is not enough. We have to define which strategy is used to create and manage the processes. For example, GeoServer WPS comes with a set of built-in processes that accomplish some common tasks. But if we want to **deploy our own**

processes, we need programming. In the case of GeoServer, **Java** programming language is used. But there are alternatives, like **scripting** solutions³⁸.

If the **processes** are complex, we need to decide about their **granularity**. Service granularity refers to the size of a service in terms of the amount of functionality carried out (Haesen et al., 2008). There is a wide range of options here: we can group various processes together to create a unique coarse-grained WPS or we can use individual chained fine-grained processes. We will discuss which option do we choose in the WPS spatial process description (see Section 5.3.5).

The last important decision regarding WPS is how to **manage the process on the client side**. OpenLayers API has support for managing WPS. We have created a JavaScript function for each process. This function creates an *OpenLayers.Format.WPSExecute* object. This object contains the definition of the execute request for that process and it is sent to the server using an *OpenLayers.Request.POST* object. This last POST object manages the output (normally the response document), performing the necessary extra tasks like parsing the resulting files or saving the result back to the server. For more information about the code please check Appendix B.

5.3.2. WPS CLIENTS IMPLEMENTATIONS

Our solution uses **GeoServer WPS extension combined with the WPS API OpenLayers classes**, but there are other different options regarding WPS clients. These possible clients implementations are:

- **PyWPS**³⁹: it has an environment for programming geo-functions and models. Its main advantage is that it has been written with the native support of GRASS GIS⁴⁰. This software is a free and open source GIS suited for geospatial data management and analysis, image processing and spatial

³⁸ <http://geoscript.org> [accessed 20th January 2014]

³⁹ <http://pywps.wald.intevation.org> [accessed 20th January 2014]

⁴⁰ <http://grass.osgeo.org> [accessed 20th January 2014]

modelling, among others. PyWPS accesses to GRASS moduli via web interfaces. The programming language is **Python**.

- **52° North**⁴¹: it is a German company that acts as a non-profit organization. It has an open source software initiative that defines a network of partners from research, industry and public administration. Its scope is to foster innovation in the field of Geoinformatics. The company has created the 52° North WPS. Its main features are: java-based, supports all the operations of the WPS standard specification, it can use GRASS and SEXTANTE⁴² processes and the results can be stored as simple web accessible resources.
- **Zoo**⁴³. Provides a framework to create and chain WPS. It has three parts: Zoo Kernel that makes possible to manage and chain web services coded in different programming languages; Zoo Services, an example library; and Zoo API, a server-side JavaScript API that chains ZOO services.
- **Deegree**⁴⁴: this project is an open source solution not only for WPS but also for other OGC standards and for spatial data infrastructures and geospatial web in general. It does not have additional features compared to the other technologies, but it is interesting because of its holistic approach.
- **GeoServer WPS**. WPS functionality of GeoServer comes with an extension. The main advantage of this implementation is that, if we are using GeoServer in our application, the integration process with data is direct and easier. The data is stored on the server (as the WPS process), so the input reading and output recording is straightforward. In next Sections we will see some examples of this direct reading of data from GeoServer. Other advantages are the availability of some default libraries that perform usual tasks (like JTS⁴⁵) and a WPS request builder that helps in the XML request construction (that can be tricky if we want process chaining).

⁴¹ <http://52north.org/communities/geoprocessing/wps/> [accessed 20th January 2014]

⁴² <http://www.sextantegis.com> [accessed 20th January 2014]

⁴³ <http://www.zoo-project.org> [accessed 20th January 2014]

⁴⁴ <http://www.deegree.org> [accessed 20th January 2014]

⁴⁵ <http://www.vividolutions.com/jts/JTSHome.htm> [accessed 20th January 2014]

5.3.3. GUIDE FOR CREATING AND USING WPS

The general rules for using and creating WPS processes in our application are:

- We use GeoServer as WPS server-side technology. This server-side technology is complemented through the use of OpenLayers WPS API classes.
- To solve any problem, we split it in individual smaller problems in order to assign these small problems to a possible process. We check if the process exists in the GeoServer libraries first. Our approach is to solve the problem, if possible, using already programmed processes.
- If the process does not exist or cannot be constructed with the libraries, we look for external WPS processes offered by reliable institutions.
- If we do not find it, we develop it using or Java or scripting technology. These technologies will access the GeoTools library.
- If we still have problems to build our process, we use other WPS clients from Section 5.3.2 and try to integrate them with OpenLayers.
- Finally, sometimes WPS will not be the best option and we will use SQL processing or rendering transformations.

Next figure shows, with a decision tree, how to apply the preceding guide:

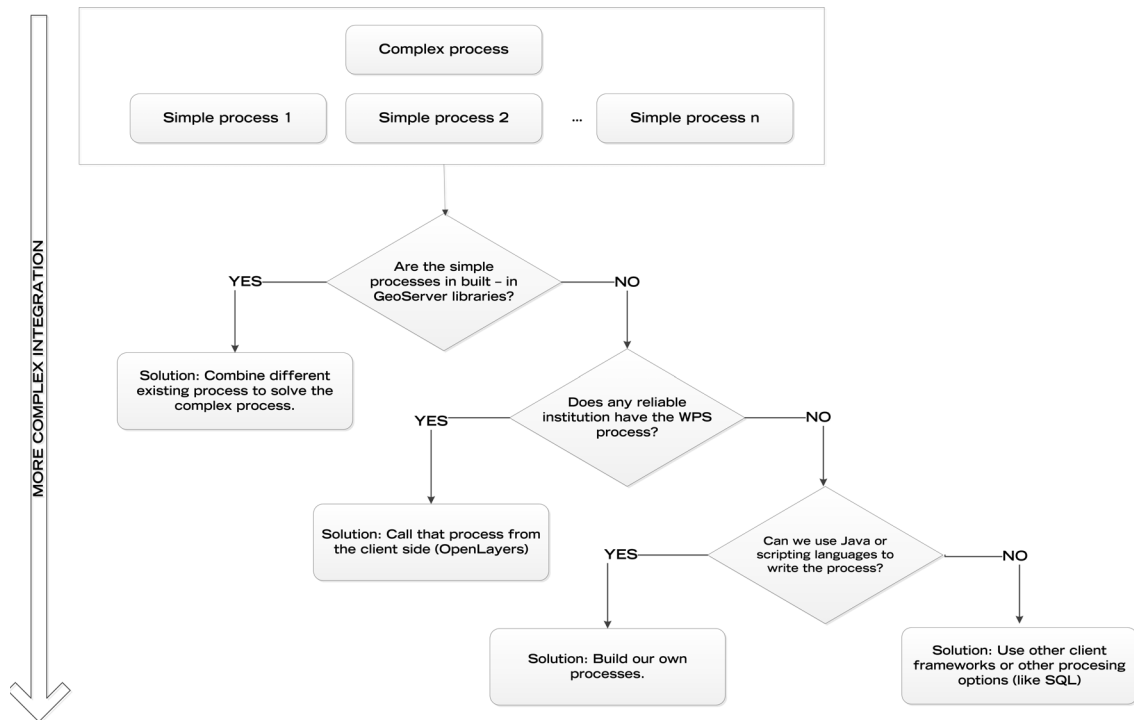


Fig. 31.- Decision tree for creating and using WPS. Source: Author.

5.3.4. GEOSERVER WPS

5.3.4.1. USING EXISTING PROCESSES: REQUEST BUILDER

GeoServer has a few built-in processes stored in different libraries. They are included in the WPS extension. The processes establish internal connections to WFS and WCS for reading and writing data. This is actually the main benefit of working completely inside GeoServer.

To build the POST requests, GeoServer WPS has a useful tool: the **request builder**. It is a functionality that constructs XML POST requests of simple and complex (chained) processes using a GUI. It is used for process testing or to see how the POST request looks like. The process selection is done using a dropdown list.

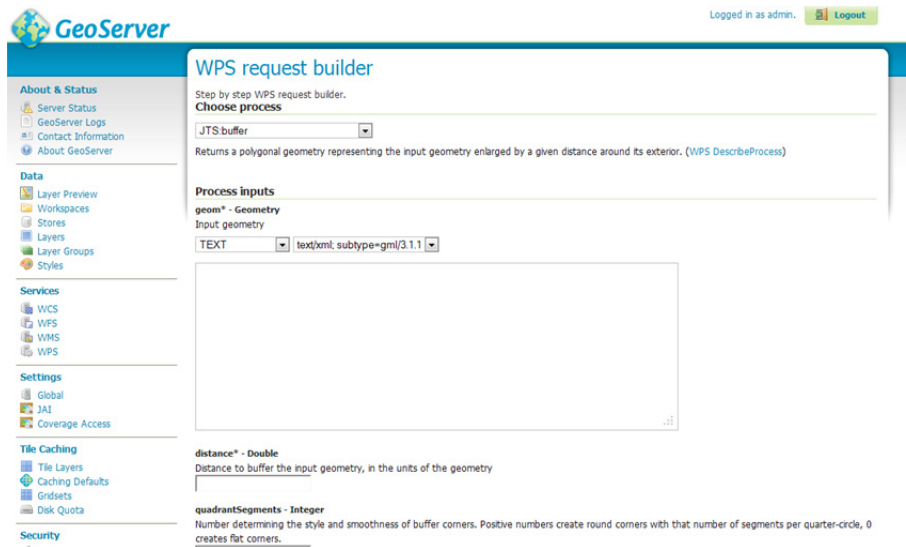


Fig. 32.- WPS Request Builder

The process (or process chaining) is called from OpenLayers using the *OpenLayers.POST* object. The combination of processes can originate very complete and complex processes. A non-exhaustive list of available built-in processes is: area, boundary, buffer, centroid, collectGeometries, contains, contour, cropCoverage, crosses, densify, difference, distance, envelope, getX, getY, heatmap, intersects, length, overlap, poligonize, rasterZonalStatistics, reproject, simplify, splitPolygon, touches and union.

5.3.4.2. CREATING NEW PROCESSES

5.3.4.2.1. JAVA-BASED PROCESSES

Creating processes from scratch is not easy. This Section describes the basics, but it is a complex field with many different working possibilities.

GeoServer can run custom WPS processes written in Java. **A process is a Java class with an execute method.** This method's parameters are the WPS parameters and the method's output is the output of the process. This Java class can contain also metadata (names and descriptions for the process and parameters). To start creating the Java process class, GeoTools API is the basic reference. It has features to make the process easier (like exception handling or conversion between XML representations and Java

objects). The procedure of creating, deploying and running a Java process is as follows:

1. **Creation of a Java process.** **Maven**⁴⁶ is a software project management tool based on the Project Object Model (POM). It manages the project build and it is used to create our Java Process. Specifically the Maven Archetype plugin⁴⁷ creates the template for the Java process. It generates the project directory structure, the POM file (describing the process) and some Java artifacts.
2. Once the project structure is created using Maven, we introduce the adequate **dependencies** to the pom.xml file. These dependencies introduce GeoTools library or other necessary libraries (this step depends on the process we want to implement).
3. Once we have the project definition and dependencies, we create the **Eclipse**⁴⁸ **project using Git**⁴⁹. Eclipse is an Integrated Development Environment (IDE) used to develop applications in Java and other programming languages. Git is a control system, a tool used for developers to track and monitor the changes of any software project.
4. In Eclipse we can **create the necessary Java classes** that define the project. In a regular process, two types of classes are created: auxiliary classes with the spatial processing code and another class with the execute method. The **metadata** is created using **Java annotations**.
5. To deploy the process back to Geoserver we need to use the dependency injection mechanism⁵⁰ of the Spring Framework⁵¹. This is a tricky process, but in practice it consists on creating an xml file in the structure that defines the project configuration as a Spring bean (assigning an id and a process class to it). A Git command builds the project and generates a JAR file that has to be

⁴⁶ <http://maven.apache.org> [accessed 20th January 2014]

⁴⁷ <http://maven.apache.org/archetype/maven-archetype-plugin> [accessed 20th January 2014]

⁴⁸ <https://www.eclipse.org> [accessed 20th January 2014]

⁴⁹ <http://git-scm.com/> [accessed 20th January 2014]

⁵⁰ http://en.wikipedia.org/wiki/Dependency_injection [accessed 20th January 2014]

⁵¹ <http://projects.spring.io/spring-framework/> [accessed 20th January 2014]

copied to the GeoServer *WEB-INF/lib* directory. Restarting GeoServer will allow the process to be available as any other built-in process.

5.3.4.2.2. SCRIPTING-BASED PROCESSES

Scripting processes are easier to create and deploy than Java processes. The scripting programming languages most frequently used are Python and JavaScript. Once the script is written, it is placed in the geoserver *<data_dir>/scripts/wps/* directory. The availability of the process is immediate (no need to restart GeoServer) but to get this functionality we have to install first the Scripting Extension⁵².

5.3.5. DEVELOPED WPS PROCESSES

5.3.5.1. PROCESS 1: BASIC RASTER STATISTICS

This tool calculates basic statistics of a raster. The user edits a polygon (dynamically, on the map) and selects a raster from a dropdown list. Then he or she clicks the Calculate Statistics button to run the process. If it is executed correctly, he or she receives a text message in the result panel with the maximum, minimum and average value for that raster within that polygon. The functionality is very basic, but still powerful as it answers questions like: “*What is the maximum altitude of my area?*” or “*What is the average rainfall in my land property?*”, etc.

The characteristics of this spatial process are:

- This is the simplest process we have created, as it is a **unique built-in GeoServer process** of GS Library. That means we do not program any process from scratch here, and in this case there is not process chaining.
- The process request is done from OpenLayers, according to the user parameters (selected raster and drawn polygon).
- This is a **fine-grained process**, totally reusable in other implementations.

⁵² <http://docs.geoserver.org/latest/en/user/community/scripting/installation.html> [accessed 20th January 2014]

The process to obtain the statistics of a raster is called **gs:RasterZonalStatistics**. The DescribeProcess operation returns its characteristics. Here there is a summary to understand which inputs are needed and which are the expected outputs:

- Process **description**. Raster zonal statistics process computes statistics for the distribution of a certain quantity in a set of polygonal zones.
- Data **inputs**. The process manages four inputs, but only two are compulsory.
 - Input n° 1: **data** (compulsory). It is the input raster to compute statistics for.
 - Type: ComplexData.
 - Default format: image/tiff
 - Supported formats: application/arcgrid
 - Input n°2: **band** (optional). Source band used to compute statistics.
 - Type: LiteralData.
 - Default Format: integer
 - Input n° 3: **zones** (compulsory). Zone polygon features for which to compute statistics.
 - Type: ComplexData.
 - Default format: text/xml;wfs-collection/1.0
 - Supported formats: text/xml;wfs-collection/1.1, application/json, application/wfs-collection-1.0, application/wfs-collection-1.1, application/zip.
 - Input n° 4: **classification** (optional). Raster whose values will be used as classes for the statistical analysis. Each zone reports statistics partitioned by classes according to the values of the raster. Must be a single band raster with integer values.
 - Type: ComplexData.
 - Default format: image/tiff.

- Supported formats: application/arcgrid.
- Data **outputs**. Only one output is generated, the **statistics** file.
 - Type: Complex output.
 - Default format: text/xml;wfs-collection/1.0.
 - Supported formats: text/xml;wfs-collection/1.1, application/json, application/wfs-collection-1.0, application/wfs-collection-1.1, application/zip.

The implementation consists in a POST request sent to GeoServer using an **OpenLayers.Request.POST** object (see Appendix B). When the user clicks the Compute Statistics button in the GUI, a function is fired. The function is called **obtainStatisticsProcess**. It defines an **OpenLayers.Format.WPSExecute()** object. On this object we write the request that finally is sent using the Request.POST. In the WPSExecute object we set the inputs and outputs:

- Raster input comes in the process as a WCS.
- Polygon input is read from the WFS-T.
- From the available output options, we have chosen JSON. The reason is that it is very easy to parse using the **JSON.parse** JavaScript function. Once the result is parsed, it is returned to the user in html format and shown up in the Results panel.

In the next figures there are some examples of the use of this tool:

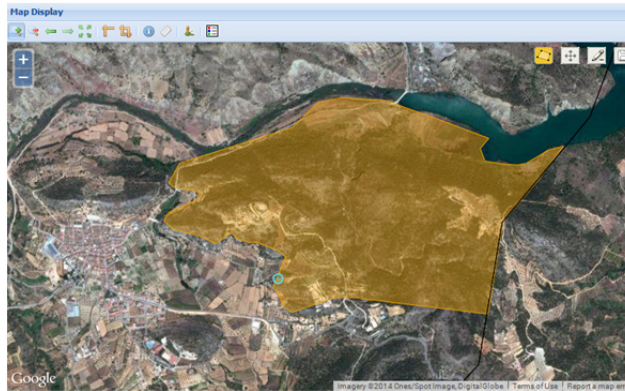


Fig. 33.- Polygon to calculate statistics

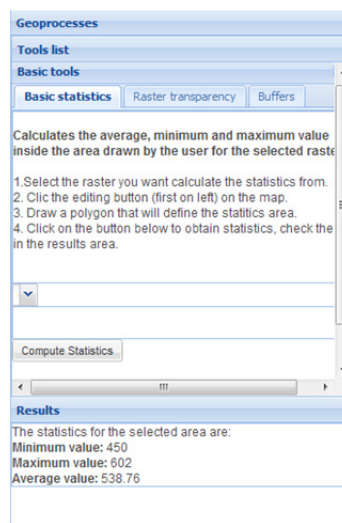


Fig. 34.- Tools and Results panel

5.3.5.2. PROCESS 2: BUFFERS

This tool is used to calculate buffers from the geometry drawn by the user. He or she draws the geometry, selects the distance and calculates the buffer around it. The result is shown up in the map area but it is not saved to the spatial database. Again, this functionality is very basic but helps to answer some questions like: *“Where do I have to cut the vegetation around this road?”* or *“What is the response time of this helicopter in a wildfire attack?”*.

The characteristics of this spatial process are:

- This is a very simple process, as it is a **unique built-in GeoServer process** of GS Library. That means we do not program any code from scratch here and there is not chaining.
- The request is done from OpenLayers, according to the user parameters (selected layer).
- This is a **fine-grained process**, totally reusable in other implementations.

gs:BufferFeatureCollection is the process to calculate buffers. Here there is a summary to understand which inputs are needed and which are the expected outputs:

- Process **description**. Buffers features by a distance value supplied either as a parameter or by a feature attribute. Calculates buffers based on Cartesian distances.
- Data **inputs**. The process manages three inputs, but only two are compulsory.
 - Input n° 1: **features** (compulsory). Input feature collection.
 - Type: ComplexData.
 - Default format: text/xml;wfs-collection/1.0
 - Supported formats: text/xml;wfs-collection/1.1, application/json, application/wfs-collection-1.0, application/wfs-collection-1.1, application/zip.
 - Input n°2: **distance** (compulsory). Fixed value to use for the buffer distance.
 - Type: LiteralData.
 - Default Format: double
 - Input n° 3: **attributeName** (optional). Attribute containing the buffer distance value.
 - Type: LiteralData.

- Default format: any value.
- **Data outputs.** Only one output is generated, the **result** file.
 - Type: Complex output.
 - Default format: text/xml;wfs-collection/1.0.
 - Supported formats: text/xml;wfs-collection/1.1, application/json, application/wfs-collection-1.0, application/wfs-collection-1.1, application/zip.

The implementation consists in a POST request sent to GeoServer using an **OpenLayers.Request.POST** object (see Appendix B). When the user clicks the Compute Buffers button in the GUI, a function is fired. The function is called **BufferProcess**. It defines an **OpenLayers.Format.WPSExecute()** object. On this object we write the request that finally is sent using the Request.POST. In the WPSExecute object we set the inputs and outputs:

- Features input comes in the process as WFS.
- The user introduces the distance dynamically.
- From the available output options, we have chosen WFS. It is shown up in the map, but it is not recorded in the spatial database. Thus, the results are lost in every session.

In the next figures there are some examples of the use of this tool. First the configuration of the process (by introducing the buffer distance calculation) and second the result (in this case, the area covered by an helicopter in 5 minutes time – 12500 m, from the Enguera base).

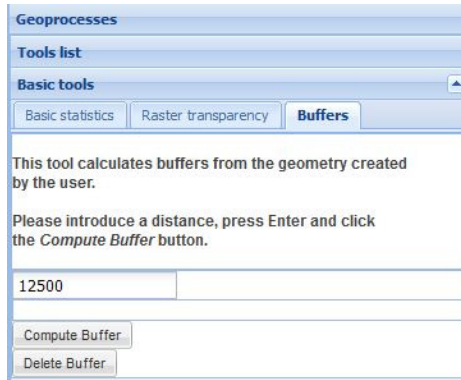


Fig. 35.- Buffer tool configuration

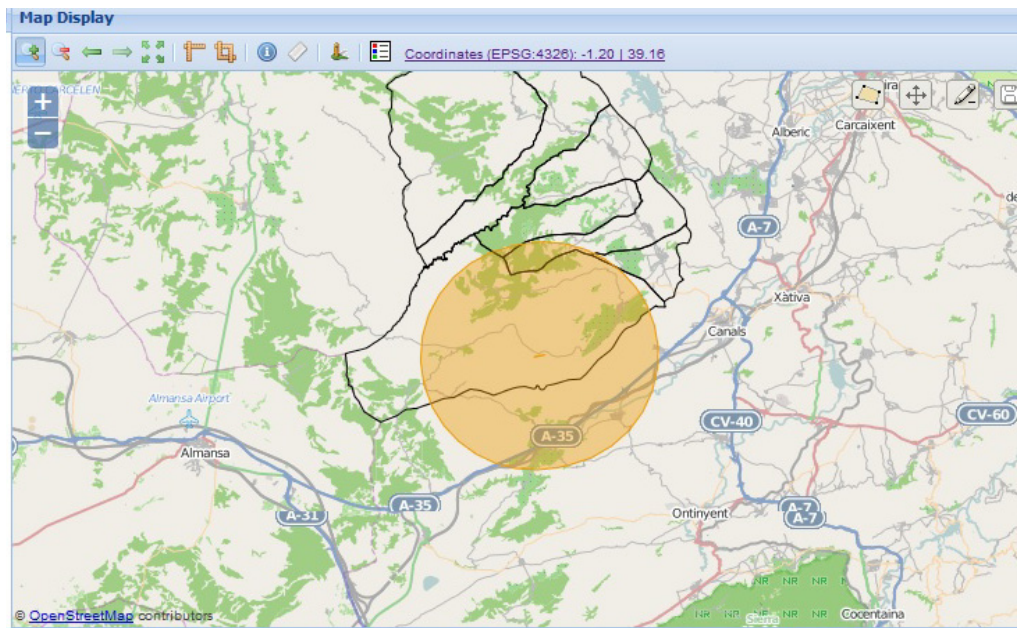


Fig. 36.- Buffer around a polygon geometry

5.3.5.3. PROCESS 3: VEGETATION DESCRIPTION

This tool returns a summarized description of the vegetation inside a polygon defined by the user. Internally the tool checks the National Forestry Map and gets the names of each different cover. Then it returns a table back to the user. The basic question this process answers is: *“What type of vegetation is there inside this area?”*

The characteristics of this spatial process are:

- The process itself is simple, as it is a unique built-in GeoServer process of GS library. However the reusability disappears when the process parses the result, which is data-specific. It has to check the useful data inside the content of a JSON file.
- The request is done from OpenLayers, according to the user parameters (basically the input polygon).
- We can define it as something in the middle between a fine-grained process and a coarse-grained process, because the process is reusable but not its results.

gs:IntersectionFeatureCollection is the process to calculate the vegetation types. Here there is a summary to understand which inputs are needed and which are the expected outputs:

- **Data inputs.** The process manages three inputs, but only two are compulsory.
 - Input n° 1: **first feature collection** (compulsory).
 - Type: ComplexData.
 - Default format: text/xml;wfs-collection/1.0
 - Supported formats: text/xml;wfs-collection/1.1, application/json, application/wfs-collection-1.0, application/wfs-collection-1.1, application/zip.
 - Input n° 2: **second feature collection** (compulsory).
 - Type: ComplexData.
 - Default format: text/xml;wfs-collection/1.0
 - Supported formats: text/xml;wfs-collection/1.1, application/json, application/wfs-collection-1.0, application/wfs-collection-1.1, application/zip.

- Input n° 3: **first attributes to retain** (optional but maximum 2147483647 attributes). First feature collection attribute to include.
 - Type: LiteralData.
 - Any Value.

- Input n° 4: **second attributes to retain** (optional but maximum 2147483647 attributes).
 - Type: LiteralData.
 - Any Value.

- Input n° 5: **intersectionMode** (optional). Specifies geometry computed for intersecting features. INTERSECTION (default) computes the spatial intersection of the inputs. FIRST copies geometry A. SECOND copies geometry B.
 - Type: LiteralData.
 - Allowed values: INTERSECTION, FIRST, SECOND.

- Input n° 6: **percentageEnabled** (optional). Indicates whether to output feature area percentages (attributes percentageA and percentageB).
 - Type: LiteralData.
 - Boolean.

- Input n° 7: **areasEnabled** (optional). Indicates whether to output feature areas (attributes areaA and areaB).
 - Type: LiteralData.
 - Boolean.

- Outputs. Only one output is generated (the **result** file).
 - Type: Complex output.
 - Default format: text/xml;wfs-collection/1.0.
 - Supported formats: text/xml;wfs-collection/1.1, application/json, application/wfs-collection-1.0, application/wfs-collection-1.1, application/zip.

The implementation consists in a POST request sent to GeoServer using an **OpenLayers.Request.POST** object (see Appendix B). When the user clicks the Vegetation Description button in the GUI, a function is fired. The function is called **describeVegetation**. It defines an **OpenLayers.Format.WPSExecute()** object. On this object we write the request that finally is sent using the Request.POST. In the WPSExecute object we set the inputs and outputs:

- Both feature inputs come in the process as WFS. The first one is the Forestry Map and the second one is the polygon drawn by the user.
- From the available output options, we have chosen JSON, which is very easy to parse.

In order to get the vegetation types, the result JSON file is parsed. First we use a loop to go through the file. Every vegetation type is extracted in a JavaScript array. A function removes the duplicates of the array, creating a new array that is ordered and passed to the result panel.

In next figure there is the result of applying this tools to a polygon within the study area:

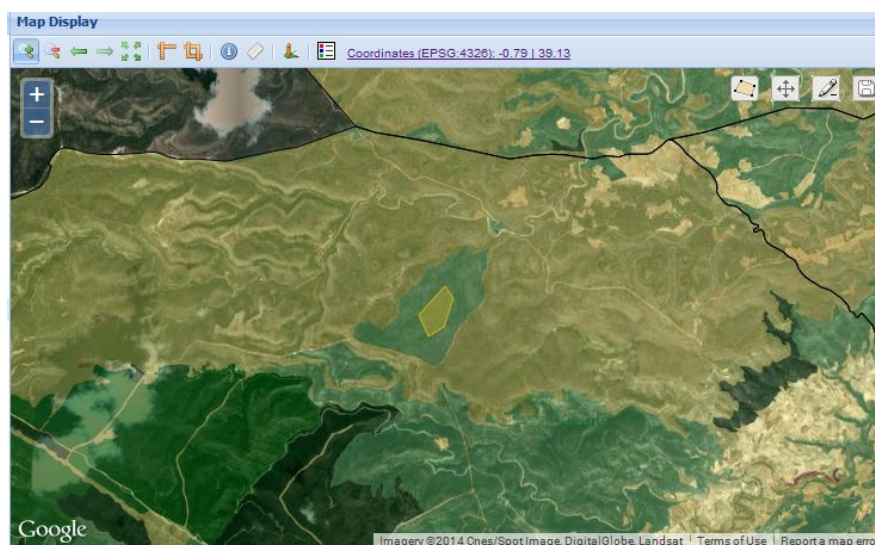


Fig. 37.- National Forest Map

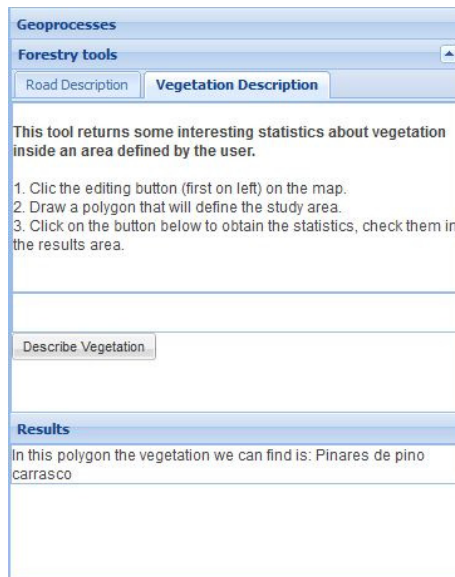


Fig. 38.- Vegetation description result table

5.3.5.4. PROCESS 4: ROAD DESCRIPTION

This tool calculates some basic descriptors about the road network inside a user-defined polygon. The drawing process is identical to the one explained in the raster statistics tool. Once the polygon is drawn and the user clicks the Describe roads button, the POST request is sent to GeoServer and if correctly processed, it returns the total length of roads inside that area, the road density (m/ha) and a simple evaluation about whether that density is low, medium or high. Landowners can use this tool to check if their land has an efficient road network for exploitation purposes.

The characteristics of this spatial process are:

- GeoServer and OpenLayers are the technologies involved but the process is split into different smaller parts.
- The final process is a combination (process chaining on the client side) of built-in GeoServer processes. Although each “small” process is fine-grained, the result (the road description process) is coarse-grained and specific for the problem we want to solve.

- The first process calculates the length of the roads inside the user's polygon. But this cannot be performed directly. First a process intersects the roads inside the polygons and another process converts them in a unique geometry (using a collect geometry process). After that, another process calculates the area. Using the outputs the system can calculate the statistics (see figure):

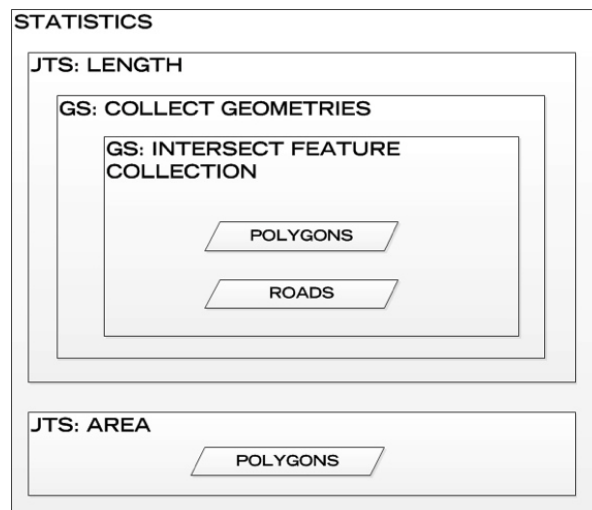


Fig. 39.- Structure of the road description process. Source: Author.

In next paragraphs there is the description of inputs and outputs of every involved process. The first one is **gs:IntersectFeatureCollection** that intersects the roads inside the polygon:

- Process n° 1 **description**. Intersection of Feature Collections. Spatial intersection of two feature collections, including combining attributes from both (if desired).
- Data **inputs**.
 - Input n° 1: **first feature collection** (compulsory).
 - Type: ComplexData.
 - Default format: text/xml;wfs-collection/1.0
 - Supported formats: text/xml;wfs-collection/1.1, application/json, application/wfs-collection-1.0, application/wfs-collection-1.1, application/zip.

- Input n° 2: **second feature collection** (compulsory).
 - Type: ComplexData.
 - Default format: text/xml;wfs-collection/1.0
 - Supported formats: text/xml;wfs-collection/1.1, application/json, application/wfs-collection-1.0, application/wfs-collection-1.1, application/zip.

- Input n° 3: **first attributes to retain** (optional but maximum 2147483647 attributes). First feature collection attribute to include.
 - Type: LiteralData.
 - Any Value.

- Input n° 4: **second attributes to retain** (optional but maximum 2147483647 attributes).
 - Type: LiteralData.
 - Any Value.

- Input n° 5: **intersectionMode** (optional). Specifies geometry computed for intersecting features. INTERSECTION (default) computes the spatial intersection of the inputs. FIRST copies geometry A. SECOND copies geometry B.
 - Type: LiteralData.
 - Allowed values: INTERSECTION, FIRST, SECOND.

- Input n° 6: **percentageEnabled** (optional). Indicates whether to output feature area percentages (attributes percentageA and percentageB).
 - Type: LiteralData.
 - Boolean.

- Input n° 7: **areasEnabled** (optional). Indicates whether to output feature areas (attributes areaA and areaB).
 - Type: LiteralData.

- Boolean.
- **Outputs.** Only one output is generated (the **result** file).
 - Type: Complex output.
 - Default format: text/xml;wfs-collection/1.0.
 - Supported formats: text/xml;wfs-collection/1.1, application/json, application/wfs-collection-1.0, application/wfs-collection-1.1, application/zip.

The second process is **gs:CollectGeometries**. The summary of the DescribeProcess operation is:

- Process n° 2 **description:** Collect Geometries. It collects the default geometries of the input features and combines them into a single geometry collection.
- **Data inputs.**
 - Input n° 1: **Input feature collection** (compulsory).
 - Type: ComplexData.
 - Default format: text/xml;wfs-collection/1.0
 - Supported formats: text/xml;wfs-collection/1.1, application/json, application/wfs-collection-1.0, application/wfs-collection-1.1, application/zip.
- **Outputs.** Only 1 output is generated (the **result** file).
 - Type: Complex output.
 - Default format: text/xml;gml/3.1.1
 - Supported formats: text/xml;gml/2.1.2, application/wkt, application/gml-3.1.1, application/gml-2.1.2.

The third process is the length calculation over the collected geometry (**JTS:length**).

- Process n° 3 **description**: Length. Returns the total length of all line segments in a geometry. Measurement is given in the source units, so geographic coordinates are not recommended.
- **Data inputs**.
 - Input n° 1: geom (compulsory). Input geometry.
 - Type: ComplexData.
 - Default format: text/xml;gml/3.1.1
 - Supported formats: text/xml;gml/2.1.2, application/wkt, application/gml-3.1.1, application/gml-2.1.2.
- **Outputs**. Only one output is generated (the **result** file).
 - Type: Literal output.
 - Double.

The fourth and last process is **JTS:area**, that is applied in the polygon:

- Process n° 4 **description**. Area. Returns the area of geometry, in the units of the geometry. Assumes a Cartesian plane, so this process is only recommended for non-geographic CRS.
- **Data inputs**.
 - Input n° 1: **geom** (compulsory). Input geometry.
 - Type: ComplexData.
 - Default format: text/xml;gml/3.1.1
 - Supported formats: text/xml;gml/2.1.2, application/wkt, application/gml-3.1.1, application/gml-2.1.2.
- **Data outputs**. Only one output is generated (**result** file).
 - Type: Literal output.
 - Double.

The implementation consists in a POST request sent to GeoServer using an **OpenLayers.Request.POST** object (see Appendix B). When the user clicks the Describe Roads button in the GUI, a JavaScript function is fired. The function is called **describeRoadsProcess**. It defines an **OpenLayers.Format.WPSExecute()** object. On this object we write the request that finally is sent using the Request.POST. In the WPSExecute object we set the inputs and outputs:

- In the Intersect Feature Collection process, both inputs (roads and polygon) are WFS. The output from this process is WFS as well.
- In the Collect Geometries process, the input is the WFS coming from the previous process and the output is a GML file.
- This GML file is introduced in the JTS:Length process and we get a number as result from it.
- The polygon surface is calculated separately with another POST request. It has two processes, the CollectGeometries process and the JTS:Area process.
- From the available output options, we have chosen JSON. The reason is that it is very easy to parse using the **JSON.parse** JavaScript function. Once the result is parsed, it is returned to the user in html format and shown up in the Results panel.
- This last request makes the combination of the results of the previous processes. Road density is calculated dividing the total length by the total area. That result is introduced in an if-else clause to assign it to a group. If density is less than 30 m/ha then it is classified as low, if it is between 30 m/ha and 100 m/ha, road density is adequate. Finally, road densities over 100 are considered too high. An HTML file is printed out in the Result panel.

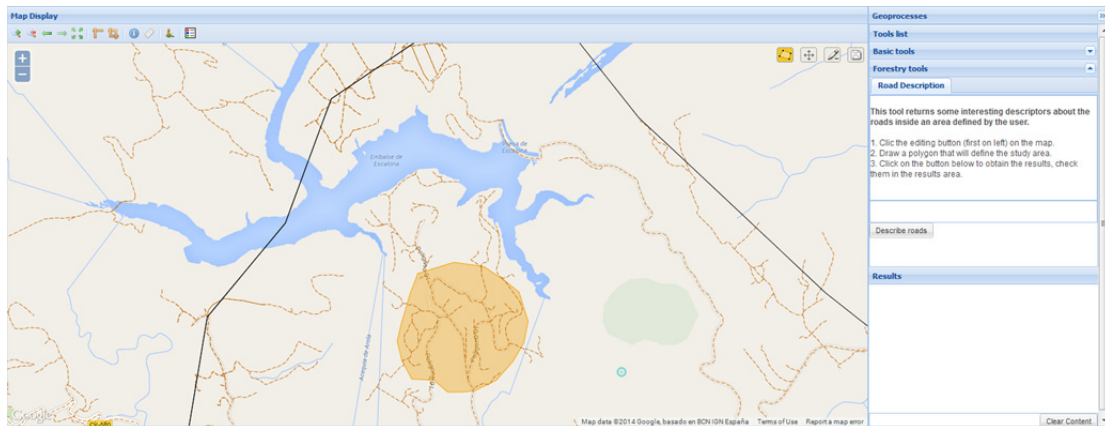


Fig. 40.- Roads and polygon to calculate the statistics

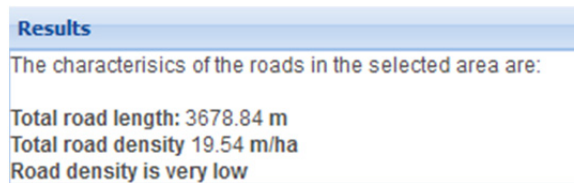


Fig. 41.- Road statistics result

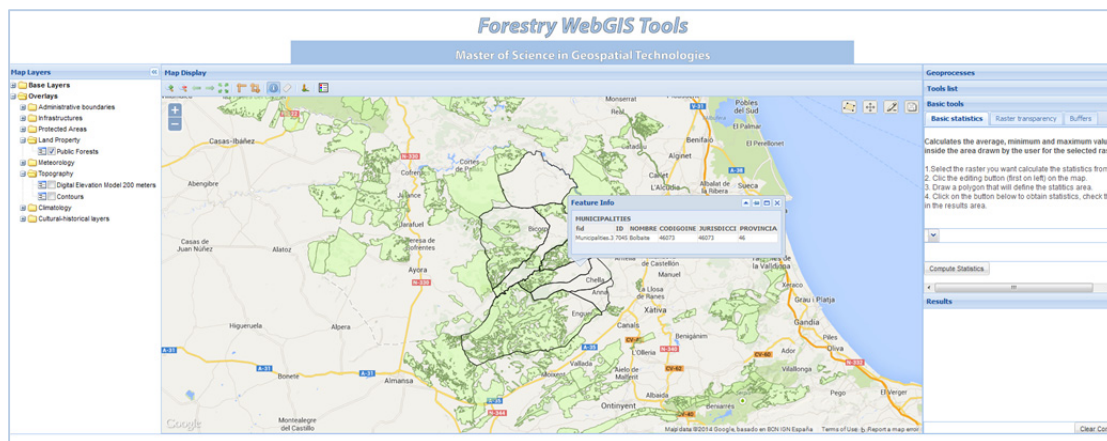


Fig. 42.- General overview of the application

6. DISCUSSION AND CONCLUSIONS

6.1. DISCUSSION

This discussion portrays a summary of the major issues that have appeared during the elaboration of this work. Web GIS technologies are different from other desktop-based developments. They need a server, which has to be functioning to test the code (or remotely or in local host). In the most basic application, there is always a combination of technologies (at least one on the server side and another on the client side). The **integration** of both and with other **technologies** is sometimes **arduous**. These problems are still more intricate when dealing with open source. Normally, the **support and documentation** are **heterogeneous** between technologies. Some are quite complete (at the same level than proprietary software) but others are very limited. Another frequent problem is the different velocity in the versions development: while some projects have a quick evolution, others are partially stopped and thus the user cannot always enjoy a complete functionality (**mismatch of capabilities**). In the Web GIS world, the use of web processes is a relatively new feature and **there are not too many examples** like with other features (visualization, downloading, etc.). That means it is not always easy to find working examples or already implemented processes. All this has made this work more difficult than initially expected.

Regarding the general development of the application, the main problems that have arisen are: how to manage CRS when third-party APIs are involved (like Google or OSM); how to use OpenLayers functionality (it has a very wide API and finding a solution to a problem is not always easy); to select what technology can be used to build the GUI; to decide which is the role and reach of the spatial database; to make a flexible project structure that allows a simple extension of the functionality; how to make the distribution between client and server side; how to create services that can be called from each other (for example a WFS within a WPS) and, in general, how to combine together all the elements to get a good user experience.

Regarding **WPS**, the **generic approach** of the specification (that allows almost any computation) **does not help in the process implementation** because almost every

kind of input and output can be used, and thus is sometimes difficult to know how to use the data. In the case of this development, we have accomplished a long process of test and error in order to construct correctly the different types of POST requests. There is not too much information about this and it has been sometimes a “tinkering” time-consuming process. We have tried different options to use processes on the web: from already existing processes, to Java native developments and scripting. Each of them has its advantages and disadvantages. The use of already existing processes is the easiest option, but it is limited to the libraries that GeoServer has. The use of Java native processes is more complex and when it comes to use existing geoprocessing algorithms (like Sextante) it is still more challenging (the integration is still under development). Geoscripting is a promising solution, but is still in a very early stage (the GeoServer scripting extension is still being tested and it is not offered yet as an official release). Next future will bring us the possibility to use more complex processes (like Sextante or Grass algorithms) as we use today the built-in GeoServer libraries. Regarding LiDAR, the libraries to manage the point cloud are not directly compatible with 64-bits systems, limiting their use. Although these problems, today already almost every process can be constructed and deployed on a Web GIS application, but improvements have to be done to facilitate its use in an efficient way.

6.2. MAIN CONCLUSIONS

The present work goes through the **complete process of designing a Web GIS application**: from the architecture description to its implementation. It proposes a framework to deal with spatial processes in a service-oriented context.

From the technical point of view, the application is the result of a **combination of technologies**. As we said before, due to the different existing possibilities, often it is not evident (and sometimes it is even very complex) to integrate the different parts in a working system. The reasons why this integration is not direct, as stated before, are: documentation is not homogenous, there are different ways to solve the same problem, standards have different versions with different requirements (and not always all the technologies support all the standards), some technologies are not consolidated yet, there is no bibliography about the topic, etc. In fact, there is a

considerable number of scientific references about web processing services, but it has been difficult to find information about their application to forestry or environmental purposes.

This work discusses the principles for creating an open architecture in which web-based spatial processes are involved. While the use of other OGC standards than WPS is extended, processing on the web is still in an early stage. The balance between client and server technologies is often difficult to achieve. Our work presents a **guide to orientate the decision making about which technology to choose** and how to compose the processing service: or by combining different existing small processes on the client side or by creating a coarse-grained big process on the server side. Between these two approaches there is a spectrum of possibilities.

The **way to create the process is not unique**. If the process can be solved using fine-grained existing processes, the solution is easier to achieve (we can call these existing processes from the client side). If the process does not exist, the programming procedure in Geoserver (our server side technology, written in Java) needs the creation of a complex project, with different libraries dependencies like GeoTools. Current trends in spatial web processing are the use of scripting languages, much easier to implement. This work has tried to make an **overview to all these different solutions**. It is worth to highlight that sometimes the use of WPS is not the best option to solve a problem. The architecture lies on a powerful spatial database (Postgresql / PostGIS) whose functionality thought SQL can manage many different processes, to the detriment of WPS (as for the use of LiDAR data).

The implementation demonstrates how it is possible to get a **fully working application using open source software**, with the possibility of displaying, querying and processing spatial data. The main advantage of working inside GeoServer has proven to be the direct use of different standards as input for processes, or to store the results of them back to the server. Other parts of the application, as the toolbar, are very time consuming to program, but they are necessary to provide an acceptable usability level.

What can be learnt from this work is a general structure for using processes on the web: which technologies exist, how they can be used, what kind of processes can be built, which data is suitable to be introduced in a process, which are the current limitations and in general how to relate these technologies together. There are many improvements to develop yet (see next Chapter), but we consider that this work has helped to make a small contribution to the normalization of web spatial processing in next future, specifically in a field that can benefit enormously from them.

7. FURTHER WORK

Web GIS technologies are a very wide field. We have presented a complete working application with few processes, but there are many topics we have not covered that are interesting for forestry / environmental purposes. Next there are ideas for further developments and improvements on the work done so far.

1. **3D Web GIS development.** The use of 3D Web GIS technologies has evolved very quickly in last years. There are plenty of projects for introducing 3D capabilities inside web browsers. A frequently used technology is **WebGL**⁵³, which brings the OpenGL⁵⁴ technology to the web. WebGL needs HTML5 to work and it can be used from JavaScript or Java. An interesting 3D project is **Cesium**⁵⁵, a WebGIS Virtual Globe and Map Engine. It is a JavaScript library for creating 3D globes (and 2D maps as well) in web browsers. Other projects, like **X3DOM**⁵⁶, try to integrate declarative 3D in HTML5, without the need of using any plugin. OGC is also discussing about introducing a **3D Web Service**. There is a draft candidate interface standard (OGC, 2010). It is worth to highlight that theoretically there are not technical limitations to use WPS in 3D architectures. All this topics are outside the scope of this work, but they conform a very interesting approach in spatial processing.
2. **LiDAR data management.** This topic is related with the previous one. The points' cloud of LiDAR data has many different applications and can be very useful for natural resource assessment. There are some examples of LiDAR visualization applications, like Online LiDAR data cloud viewer⁵⁷. Other way to manage LiDAR data is using the PostGIS queries with some extensions enabled (as we did in a small example). But there is room for improvement in using this kind of data, specifically as web services.

⁵³ <http://www.khronos.org/webgl/> [accessed January 21th 2014]

⁵⁴ <http://www.opengl.org> [accessed January 21th 2014]

⁵⁵ <http://cesiumjs.org> [accessed January 21th 2014]

⁵⁶ <http://www.x3dom.org> [accessed January 21th 2014]

⁵⁷ <http://lidarview.com> [accessed January 21th 2014]

3. **Charts.** The application could be improved by introducing a chart module. Charts are a good manner to transmit information and thankfully there are some JavaScript libraries to create charts in an easy way, like for example Highcharts⁵⁸. Some future developments can be terrain profile calculations, graphical statistics, climatic diagrams, etc.
4. **Printing.** It is always helpful to have a printing module in a Web GIS application. A logical implementation for our application could be the GeoServer printing extension that uses MapFish⁵⁹ and GeoExt.
5. **Animation tool.** Time-based layers (like for example land use changes) are better depicted if some kind of animation is used. This functionality is not difficult to implement using the WMS GeoServer animator. It executes a sequence of requests instead of one unique request. This tool has other applications like changing the map extent, or zooming in and out.
6. **Monitoring.** In a production environment it makes sense to monitor the requests like: total time to complete them, origin, timestamp, etc. This information is very important to make the application more efficient and to know better how it is used. If the application is made publicly available, this functionality should be set up.
7. **User symbology configuration.** As it is now, symbology is static. It is configured on the server side (using SLD) and the user cannot change it. It would be a good improvement to give the user the chance to change it according to his or her necessities.
8. **Catalogue Services for the Web (CSW).** In a production environment, to expose a catalogue of geospatial records is essential to discover, browse and query metadata about our data and services. GeoServer CSW extension does

⁵⁸ <http://www.highcharts.com> [accessed January 21th 2014]

⁵⁹ <http://mapfish.org> [accessed January 21th 2014]

this work and it implements two metadata schemes: Dublin Core⁶⁰ and ISO Metadata Profile (OGC, 2007).

9. **Usability tests and error detection.** The next step after completing the application is to establish a testing period, detect errors and get feedback from users to improve the final result.

10. **GeoSearch.** If our project were publicly available, it would be convenient to expose the data to Google's GeoSearch. Thanks to some changes in the server configuration, it is possible to make easier the data finding, by searching directly on Google Maps or Google Earth. The format exposed is KML and search engines are able to crawl it easier.

⁶⁰ <http://dublincore.org> [accessed January 21th 2014]

8. REFERENCES

Bastin, L., McInerney, D., Revez, G., Figueredo, C., Simonetti, D., Barredo, J., Achard, F., San-Miguel-Ayanz, J., 2012. Web Services for Forest Data, Analysis and Monitoring: Developments from EuroGEOSS. EarthZine. <http://www.earthzine.org/2012/07/25/web-services-for-forest-data-analysis-and-monitoring-developments-from-eurogeoss/>. Accessed: January 14th 2014.

Bouwman, Dave., 2005. The GIS Longtail – Google, MSN, Yahoo and ESRI. <http://blog.davebouwman.com/2005/10/11/the-gis-longtail-google-msn-yahoo-and-esri/> Accessed January 17th, 2014.

Díaz, L., Pepe, M., Granell, C., Carrara, P., Rampini, A., 2010. Developing and chaining web processing services for hydrological models. In WebMGS, First International Workshop on Pervasive Web Mapping, Geoprocessing and Services.

Dubois, G., Schulz, M., Skoien, J., Bastin, L., Peedell, S., 2013. eHabitat, a multi-purpose Web Processing Service for ecological modelling. *Environmental Modelling & Software* 41, 123-133.

European Computer Manufacturers Association. Standard ECMA-404. The JSON Data Interchange Format, 2013. Online resource: <http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf> Accessed January 15th, 2014.

Fu, P., Sun, J., 2011. *Web GIS Principles and applications*. ESRI Press. 13, 51, 82.

GeoServer, 2014. User Manual. Overview. Online resource: <http://docs.geoserver.org/stable/en/user/introduction/overview.html> Accessed January 16th, 2014.

Google, 2014. Usage limits and billing for JavaScript API v3. Online resource: <https://developers.google.com/maps/documentation/javascript/usage>
Accessed January 7th 2014.

Gong, J., 1999. Contemporary GIS theory and technology. Wuhan, China: Wuhan University of Surveying and Mapping Science and Technology Press.

Granell, C., Díaz, L., Gould, M., 2009. Service-oriented applications for environmental models: Reusable geospatial services. *Environmental Modelling & Software* 25, 182-198.

Haesen, R., Snoeck, M., Lemahie, W., Poelmans, S., 2008. On the definition of service granularity and its architectural impacts. In: Proc. of International Conference on Advanced Information Systems Engineering (CAiSE'08). LNCS, vol. 5078. Springer, pp. 375-389.

INSPIRE EU Directive, 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). Official Journal of the European Union, L 108/1, Volume 50, April 25th 2007.

INSPIRE Technical Architecture Document - Overview, 2007. 4. Online resource: http://inspire.jrc.ec.europa.eu/reports/ImplementingRules/network/INSPIRETechnicalArchitectureOverview_v1.2.pdf Accessed January 17th, 2014.

Lanig, S., Zipf, A., 2009. Towards Generalization Processes of LiDAR Data based on GRID and OGC Web Processing Services. University of Heidelberg.

Marshal, J., 2012. HTTP made really easy. A practical guide to writing clients and servers. Online resource: <http://www.jmarshall.com/easy/http/> Accessed October 21th, 2013.

NETCRAFT, 2013. April 2013 Web Server Survey. Online resource: <http://news.netcraft.com/archives/2013/04/02/april-2013-web-server-survey.html> Accessed October 21th, 2013.

Open GeoSpatial Consortium, 2006. OpenGIS Web Map Server Implementation Specification. Online resource: http://portal.opengeospatial.org/files/?artifact_id=14416 Accessed January 16th, 2014.

Open GeoSpatial Consortium, 2005. Web Feature Service Implementation Specification. Online resource: https://portal.opengeospatial.org/files/?artifact_id=7176 accessed January 16th, 2014.

Open GeoSpatial Consortium, 2006. Web Coverage Service Implementation Specification. Online resource: http://portal.opengeospatial.org/files/?artifact_id=3837 Accessed January 16th, 2014.

Open GeoSpatial Consortium, 2010. OpenGIS Implementation Standard for Geographic information – Simple feature access- Part 2: SQL option. Online resource: http://portal.opengeospatial.org/files/?artifact_id=25354 Accessed January 17th, 2014.

Open GeoSpatial Consortium, 2007. OpenGIS Web Processing Service. Version 1.0.0. Online resource: http://portal.opengeospatial.org/files/?artifact_id=24151 Accessed January 20th 2014.

Open GeoSpatial Consortium, 2002. OpenGIS Recommendation Paper. Units of Measure. Use and Definition Recommendations. Online resource: http://portal.opengeospatial.org/files/?artifact_id=11498 Accessed January 20th 2014.

Open GeoSpatial Consortium, 2007. OGC Web Services Common Specification. Online resource: http://portal.opengeospatial.org/files/?artifact_id=20040 Accessed January 20th 2014.

Open GeoSpatial Consortium, 2010. Draft for Candidate OpenGIS Web 3D Service Interface Standard. Online resource: http://portal.opengeospatial.org/files/?artifact_id=36390 Accessed January 21th 2014.

Open GeoSpatial Consortium, 2007. OpenGIS Catalogue Services Specification 2.0.2 – ISO Metadata Application Profile. Online resource: http://portal.opengeospatial.org/files/?artifact_id=20555 Accessed January 21th 2014.

O'Reilly, Tim., 2007. What is Web 2.0: Design Patters and Business Models for the Next Generation of Software. Communications & Strategies, 1, p. 17.

POSTGIS, 2014. PostGIS Reference. Chapter 6. Online resource: <http://postgis.net/docs/manual-1.3/ch06.html> Accessed January 17th 2014.

Schaeffer, B., 2008. Towards a transactional web processing service. 4. In: Proceedings of the GI-Days, Münster.

Spanish Spatial Data Infrastructure - SDI, 2014. Service directory. Online resource: <http://www.idee.es/web/guest/directorio-de-servicios> Accessed January 17th 2014.

World Wide Web Consortium. Hypertext Transfer Protocol – HTTP/1.1., 1999. Online resource: <http://www.w3.org/Protocols/rfc2616/rfc2616.html> Accessed October 21th, 2013.

World Wide Web Consortium. XMLHttpRequest., 2012. Online resource: <http://www.w3.org/TR/XMLHttpRequest> Accessed October 21th, 2013.

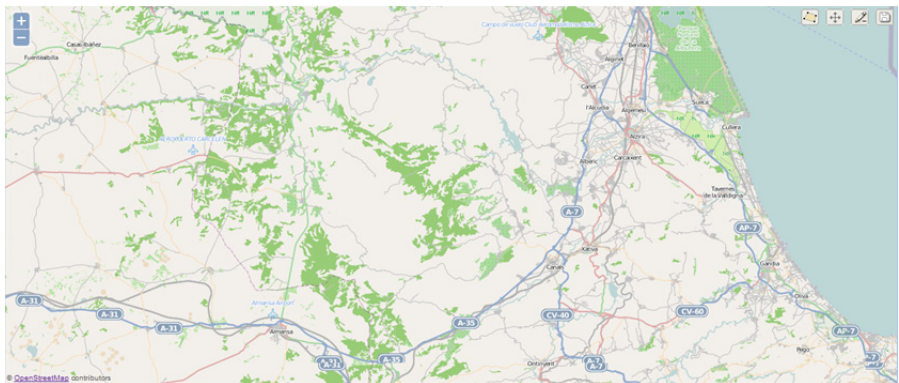
World Wide Web Consortium. Extensible Markup Language (XML) 1.0 (Fifth Edition)., 2008. Online resource: <http://www.w3.org/TR/REC-xml/> Accessed October 21th, 2013.


World Wide Web Consortium, 2001. XML Schema. Part 2 Datatypes. Online resource: <http://www.w3.org/TR/2001/REC-xmlschema-2-20010502> Accessed January 20th 2014.


APPENDIX A: DATA

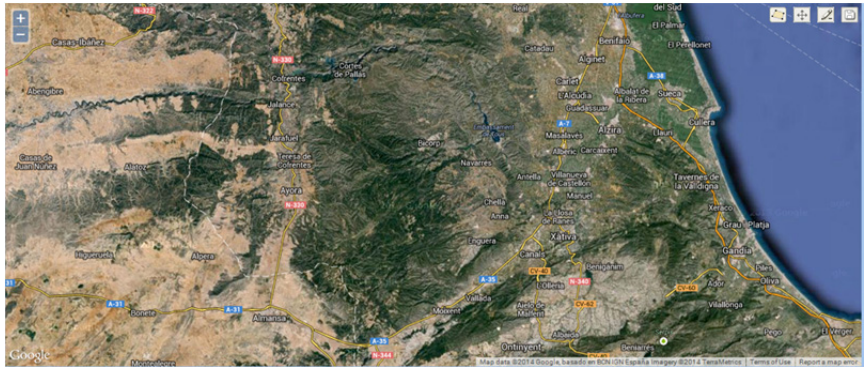
This appendix describes the data used in the application, grouped in base layers and operational layers. We present some basic metadata and a thumbnail of each one.

Base layers

Number	<i>1</i>
Name	<i>OpenStreetMap (OSM)</i>
Description	<i>Cartography of the OSM project. It contextualizes the application acting as a base layer. Its main strength is the existence of points of interest. The spatial elements present in this cartography are: cities, roads, administrative boundaries, spatial areas, hydrology, and place names.</i>
Type	<i>Introduced in the application using the Layer.OSM class of OpenLayers.</i>
Granularity	<i>Variable. The most detailed scale is 1:2115 (zoom level of 18).</i>
Production Institution	<i>Crowdsourcing project.</i>
Date	<i>Depend of the part of the study area. Being updating continuously.</i>
Thumbnail	

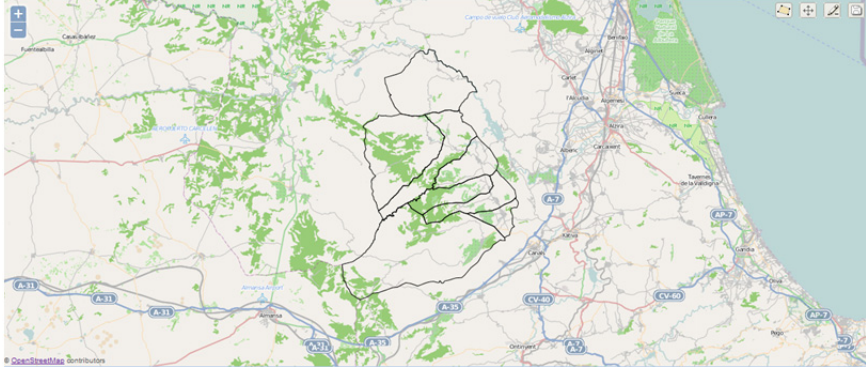
Number	2
Name	<i>Google maps SATELLITE layer.</i>
Description	<i>Google layer with Google Earth satellite images. This is the most important layer for photointerpretation purposes.</i>
Type	<i>Introduced in the application using the Layer.Google class of OpenLayers.</i>
Granularity	<i>Pixel size variable depending on the zone.</i>
Production Institution	<i>Google.</i>
Date	<i>Variable and continuously updated.</i>
Thumbnail	

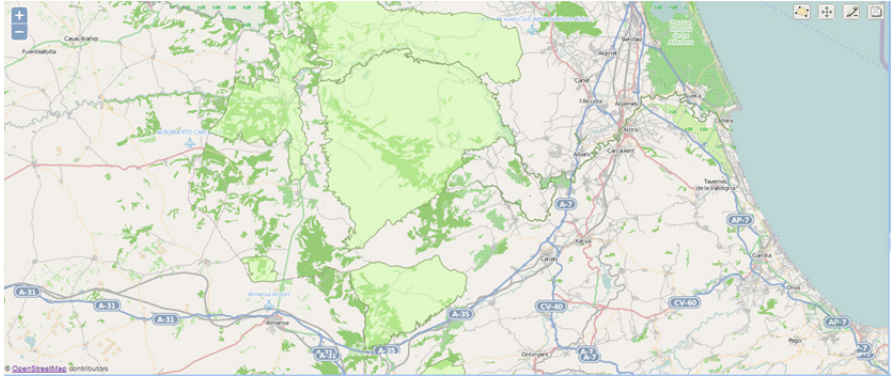
Number	3
Name	<i>Google maps TERRAIN layer.</i>
Description	<i>Google physical map. It has a hill shade model, contours values, roads, hydrological elements and place names.</i>
Type	<i>Introduced in the application using the Layer.Google class of OpenLayers.</i>
Granularity	<i>It depicts contour lines every 100 m.</i>
Production Institution	<i>Google.</i>
Date	<i>Not applicable</i>
Thumbnail	

Number	4
Name	<i>Google maps HYBRID layer.</i>
Description	<i>Google satellite imagery with placenames and roads coming from ROAD layer.</i>
Type	<i>Introduced in the application using the Layer.Google class of OpenLayers.</i>
Granularity	<i>Pixel size variable depending on the zone.</i>
Production Institution	<i>Google.</i>
Date	<i>Variable and continuously updated.</i>
Thumbnail	


Number	5
Name	<i>Google maps ROADS layer.</i>
Description	<i>This is the default layer that is uploaded when the application starts. It is the usual Google base layer, with roads, hydrology elements, place names, cities and natural areas.</i>
Type	<i>Introduced in the application using the Layer.Google class of OpenLayers.</i>
Granularity	<i>It presents various levels of detail depending on the zoom. At the maximum zoom level it shows the street names and numbers.</i>
Production Institution	<i>Google.</i>
Date	<i>Not applicable</i>
Thumbnail	

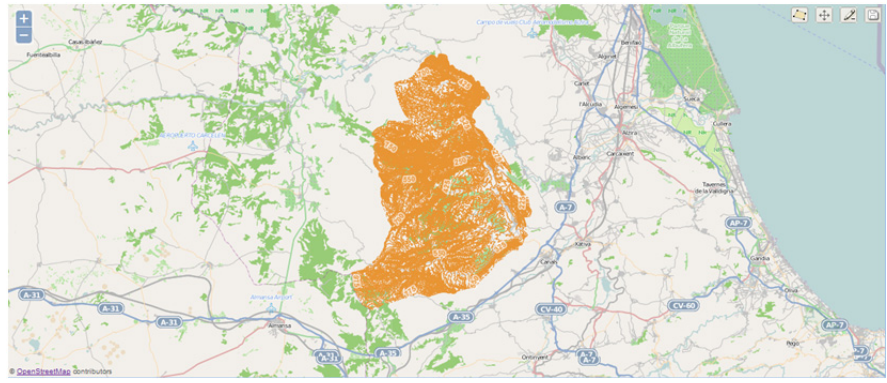
Operational layers


Number	6
Name	<i>Municipalities layer.</i>
Description	<i>Official boundaries of the municipalities inside the study area.</i>
Type	<i>The layer is depicted using a WMS through the OpenLayers.Layer.WMS class.</i>
Granularity	<i>1:25,000</i>
Production Institution	<i>National Geographic Institute.</i>
Date	<i>July 2013.</i>
Thumbnail	

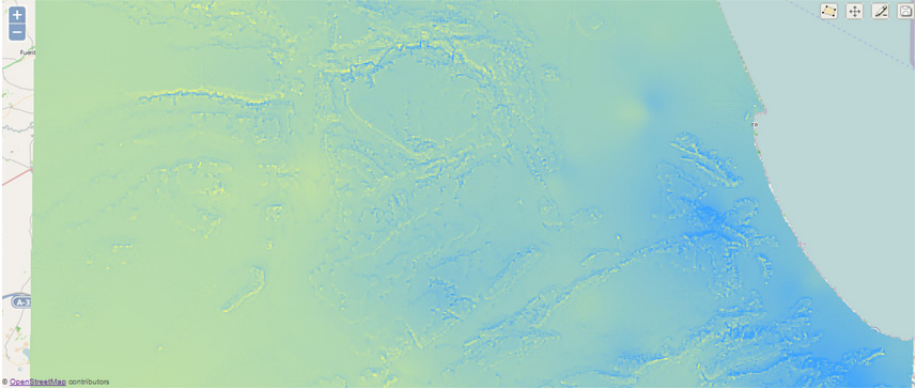
Number	7
Name	<i>LICS layer.</i>
Description	<i>Natural Areas of Communitarian Interest (European Habitat Directive). Name and code of each area.</i>
Type	<i>The layer is depicted using a WMS through the OpenLayers.Layer.WMS class.</i>
Granularity	<i>1:50,000</i>
Production Institution	<i>Spanish Environmental Ministry.</i>
Date	<i>December 2012</i>
Thumbnail	

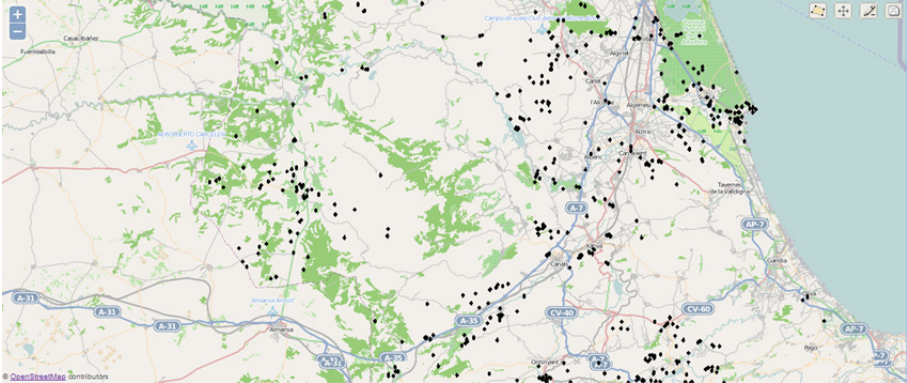
Number	8
Name	<i>MUPS layer.</i>
Description	<i>Public Forests Boundaries. The layer has the code of the forest, its area and its boundary length.</i>
Type	<i>The layer is depicted using a WMS through the OpenLayers.Layer.WMS class.</i>
Granularity	<i>1:10,000</i>
Production Institution	<i>Ministerio de Medio Ambiente, Medio Rural y Marino.</i>
Date	<i>July 2011.</i>
Thumbnail	


Number	9
Name	<i>DEM layer.</i>
Description	<i>Digital Elevation Model. Only an attribute exists: the altitude of each pixel above sea level.</i>
Type	<i>The layer is depicted using a WMS through the OpenLayers.Layer.WMS class.</i>
Granularity	<i>Pixel size of 200 m.</i>
Production Institution	<i>National Geographic Institute.</i>
Date	<i>2009.</i>
Thumbnail	

Number	<i>10</i>
Name	<i>Contours.</i>
Description	<i>Contour lines every 5 m. The SLD style has two levels of detail and shows different contours depending on the zoom level.</i>
Type	<i>The layer is depicted using a WMS through the OpenLayers.Layer.WMS class.</i>
Granularity	<i>Two levels of detail. Base cartography: 1:5,000.</i>
Production Institution	<i>Valencian Cartographic Institute. Topographic Map.</i>
Date	<i>July 2011.</i>
Thumbnail	

Number	<i>11</i>
Name	<i>Meteorology layers</i>
Description	<i>Set of layers with the current meteorology: weather (symbol and maximum and minimum temperatures), cloud cover, qualitative rain intensity and qualitative temperature.</i>
Type	<i>These layers are depicted using a WMS through the OpenLayers.Layer.WMS class.</i>
Granularity	<i>Not applicable.</i>
Production Institution	<i>http://openweathermap.org/</i>
Date	<i>Not applicable.</i>
Thumbnail	

Number	<i>12</i>
Name	<i>Rainfall</i>
Description	<i>Average yearly rainfall (in tenths of mm)</i>
Type	<i>The layer is depicted using a WMS through the OpenLayers.Layer.WMS class.</i>
Granularity	<i>Pixel size of 200 m.</i>
Production Institution	<i>Digital Climatic Atlas of Iberian Peninsula. Barcelona University.</i>
Date	<i>2005.</i>
Thumbnail	

Number	<i>13</i>
Name	<i>Archaeological sites</i>
Description	<i>Point layer indicating the location of archaeological sites in Valencian Community.</i>
Type	<i>Cascading WMS (remote WMS server introduced as a new WMS in GeoServer).</i>
Granularity	<i>1:50,000</i>
Production Institution	<i>Autonomic Government (Generalitat Valenciana).</i>
Date	<i>1998</i>
Thumbnail	

Number	<i>15</i>
Name	<i>National Forest Map</i>
Description	<i>Layer with detailed information about the type of vegetation within each polygon. It defines if its populated by trees and which species are present on them.</i>
Type	<i>The layer is depicted using a WMS through the OpenLayers.Layer.WMS class.</i>
Granularity	<i>1:25,000</i>
Production Institution	<i>National Environment Ministry.</i>
Date	<i>2007</i>
Thumbnail	

APPENDIX B: JAVASCRIPT CODE

```
*****
* WebGIS for forestry and environmental applications.
* January 2014.
* Author: Constancio Amurrio García.
* Master's Degree in Geospatial technologies.
* Univesitat Jaume I / Universität Münster / Universidade Nova Lisboa.
*****/

<!DOCTYPE html>
<html>
<head>

  /*EXTERNAL LIBRARIES AND STYLES*/
  <meta charset="utf-8">
  <script src="http://code.jquery.com/jquery-1.7.min.js" ></script>
  <script src="http://code.jquery.com/ui/1.7.0/jquery-ui.js" ></script>
  <link rel="stylesheet" type="text/css" href="openlayers/theme/default/style.css"></link>
  <script type="text/javascript" src="ext/adaptor/ext/ext-base.js"></script>
  <script type="text/javascript" src="ext/ext-all.js"></script>
  <link rel="stylesheet" href="ext/resources/css/ext-all.css" type="text/css"></link>
  <script type="text/javascript" src="openlayers/OpenLayers.js"></script>
  <script type="text/javascript" src="geoext/lib/GeoExt.js"></script>
  <script type="text/javascript" src="resources/proj4js/lib/proj4js-combined.js"></script>
  <link rel="stylesheet" type="text/css" href="geoext/resources/css/geoext-all-debug.css"></link>
  <script type="text/javascript" src="geoext/ux/StatusBar.js"></script>
  <link rel="stylesheet" type="text/css" href="resources/css/style.css"></link>
  <link rel="stylesheet" type="text/css" href="geoext/resources/css/popup.css">
  <script type="text/javascript" src="openweathermap/OWM.OpenLayers.1.3.4.js"></script>
  <script src="http://maps.google.com/maps/api/js?sensor=false"></script>

  /*JAVASCRIPT APPLICATION CODE*/
  <script type="text/javascript">

    /*PROXY HOST*/
    OpenLayers.ProxyHost = "cgi-bin/proxy.cgi?url="

    /*GLOBAL VARIABLES*/
    var map,polygonStatistics, statisticsMessage, resultWindow,app, layerRuler, infoCtrl,popup, click, dvpopup,
    clipProcess,doc,collectGeometries,clipGeometries, totalLength, totalArea,
    RoadsMessage,lics,legendWindow;
    var controls=[];
    var toolBar=[];

    /*EX-PROFESO CLASS*/
    var DeleteFeature = OpenLayers.Class(OpenLayers.Control, {
      initialize: function(layer, options) {
        OpenLayers.Control.prototype.initialize.apply(this, [options]);
        this.layer = layer;
        this.handler = new OpenLayers.Handler.Feature(
          this, layer, {click: this.clickFeature}
        );
      },
      clickFeature: function(feature) {
        if(feature.fid == undefined) {
          this.layer.destroyFeatures([feature]);
        } else {
          feature.state = OpenLayers.State.DELETE;
          this.layer.events.triggerEvent("afterfeaturemodified", {feature: feature});
          feature.renderIntent = "select";
          this.layer.drawFeature(feature);
        }
      },
      setMap: function(map) {
        this.handler.setMap(map);
        OpenLayers.Control.prototype.setMap.apply(this, arguments);
      },
      CLASS_NAME: "OpenLayers.Control.DeleteFeature"
    });
  /*APPLICATION INITIALIZATION*/
```

```

Ext.onReady(function(){

    /*MAP OBJECT*/
    map = new OpenLayers.Map({
        projection:'EPSG:900913',
        displayProjection:'EPSG:4326',
        zoomMethod: null //In order to avoid weird rendering.
    });

    /*OPERATIONAL LAYERS*/
    /*Municipalities as WMS*/
    var municipalities=new OpenLayers.Layer.WMS(
        'Municipalities',
        'http://localhost:8080/geoserver/forestry/wms?SERVICE=WMS&',
        {layers:'forestry:Municipalities',
        format:'image/png',
        transparent:true},
        {isBaseLayer:false}
    );

    /*MUP as WMS*/
    var mups= new OpenLayers.Layer.WMS(
        "Mups",
        'http://localhost:8080/geoserver/forestry/wms?SERVICE=WMS&',
        {layers:'forestry:Mups',
        format:'image/png',
        transparent:true},
        {visibility:false},
        {isBaseLayer:false}
    );

    /*LICS as WMS*/
    lics= new OpenLayers.Layer.WMS(
        "Comunitary Interest Sites",
        'http://localhost:8080/geoserver/forestry/wms?SERVICE=WMS&',
        {layers:'forestry:Lics',
        format:'image/png',
        transparent:true},
        {visibility:false},
        {isBaseLayer:false}
    );

    /*Archaeological sites as WMS*/
    var archaeologicalSites= new OpenLayers.Layer.WMS(
        "Archeological Sites (External WMS)",
        'http://localhost:8080/geoserver/forestry/wms?SERVICE=WMS&',
        {layers:'forestry:coput_yacimientos_arqueologicos',
        format:'image/png',
        transparent:true},
        {visibility:false},
        {isBaseLayer:false}
    );

    /*ROADS as WMS*/
    var roads= new OpenLayers.Layer.WMS(
        "Roads",
        'http://localhost:8080/geoserver/forestry/wms?SERVICE=WMS&',
        {layers:'forestry:RoadsCanalNavarres3857_3',
        format:'image/png',
        transparent:true},
        {visibility:false},
        {isBaseLayer:false}
    );

    /*CONTOURS as WMS*/
    var contours= new OpenLayers.Layer.WMS(
        "Contours",
        'http://localhost:8080/geoserver/forestry/wms?SERVICE=WMS&',
        {layers:'forestry:contoursCanalNavarres3857',
        format:'image/png',
        transparent:true},
        {visibility:false},
        {isBaseLayer:false}
    );
});

```

```

/*Rainfall as WMS*/
var rainfall= new OpenLayers.Layer.WMS(
    "Rainfall",
    'http://localhost:8080/geoserver/forestry/wms?SERVICE=WMS&',
    {layers:'forestry:rainfall',
    format:'image/png',
    transparent:true},
    {visibility:false},
    {isBaseLayer:false}
);

/*WEATHER - OPENWEATHERMAP*/
/*Weather with symbols*/
var cityWeather = new OpenLayers.Layer.Vector.OWMWeather("Weather",
{visibility:false});

/*Cloud cover*/
var clouds = new OpenLayers.Layer.XYZ(
    "Cloud Cover",
    "http://$${s}.tile.openweathermap.org/map/clouds/${z}/${x}/${y}.png",
    {
        isBaseLayer: false,
        opacity: 0.7,
        sphericalMercator: true,
        visibility:false
    }
);

/*Precipitation*/
var precipitation = new OpenLayers.Layer.XYZ(
    "Precipitation",
    "http://$${s}.tile.openweathermap.org/map/precipitation/${z}/${x}/${y}.png",
    {
        isBaseLayer: false,
        opacity: 0.3,
        sphericalMercator: true,
        visibility:false
    }
);

/*Temperature*/
var temperature = new OpenLayers.Layer.XYZ(
    "Temperature",
    "http://$${s}.tile.openweathermap.org/map/"+"temp"+"/${z}/${x}/${y}.png",
    {
        isBaseLayer: false,
        opacity: 0.5,
        sphericalMercator: true,
        visibility:false
    }
);

/*DEM200 as WMS*/
var dem200= new OpenLayers.Layer.WMS(
    "DEM 200m",
    'http://localhost:8080/geoserver/forestry/wms?SERVICE=WMS&',
    {layers:'forestry:mde200_V4',
    format:'image/png',
    transparent:true},
    {visibility:false},
    {isBaseLayer:false}
);

/*Polygon layer to compute statistics, as WFS-T*/
var saveStrategy=new OpenLayers.Strategy.Save();
polygonStatistics=new OpenLayers.Layer.Vector("Polygons for Statistics",{
    strategies: [saveStrategy, new OpenLayers.Strategy.BBOX()],
    protocol: new OpenLayers.Protocol.WFS({
        version: "1.1.0",
        url: "http://localhost:8080/geoserver/forestry/wfs",
        featureType: "computestatisticspolygon",
        featureNS:"http://localhost:8080/thesis/forestry",
        geometryName:"geom",
        srsName: "EPSG:900913"
    })
});

```

```

    }},
    visibility:true
  });

  /*BASE LAYERS*/
  /*Open Street Maps*/
  var osm= new OpenLayers.Layer.OSM();

  /*Google Layer hybrid*/
  var google_map_hybrid=new OpenLayers.Layer.Google(
    'Google Hybrid',
    {type: google.maps.MapTypeId.HYBRID,numZoomLevels: 22}
  );

  /*Google Layer physical*/
  var google_map_physical=new OpenLayers.Layer.Google(
    'Google Physical',
    {type: google.maps.MapTypeId.TERRAIN,numZoomLevels: 22}
  );

  /*Google Layer satellite*/
  var google_map_satellite=new OpenLayers.Layer.Google(
    'Google Satellite',
    {type: google.maps.MapTypeId.SATELLITE,numZoomLevels: 22}
  );

  /*Google Layer street*/
  var google_map_street=new OpenLayers.Layer.Google(
    'Google Street',
    {type: google.maps.MapTypeId.ROADMAP,numZoomLevels: 22}
  );

  /*OTHER AUXILIAR LAYERS*/
  layerRuler= new OpenLayers.Layer.Vector("Measurements",{
    displayInLayerSwitcher: false
  });

  /*LAYER LIST*/
  var projectLayerList= [
    polygonStatistics,
    dem200,
    rainfall,
    mups,
    roads,
    contours,
    lics,
    cityWeather,
    clouds,
    precipitation,
    temperature,
    archeologicalSites,
    municipalities,
    google_map_street,
    google_map_hybrid,
    google_map_physical,
    google_map_satellite,
    osm,
    layerRuler
  ];

  /*LAYER DATA STORE*/
  var layerDataStore= new GeoExt.data.LayerStore({
    map:map,
    layers:[
    polygonStatistics,
    dem200,
    rainfall,
    mups,
    roads,
    contours,
    lics,
    cityWeather,
    clouds,
    precipitation,

```



```

        temperature,
        archeologicalSites,
        municipalities,
        google_map_street,
        google_map_hybrid,
        google_map_physical,
        google_map_satellite,
        osm,
        layerRuler]
});

/*POPUPS WMS CONFIGURATION*/
infoCtrl = new OpenLayers.Control.WMSGetFeatureInfo({
    url: "/geoserver/forestry/wms?SERVICE=WMS&",
    title: "Identify features by clicking",
    layers: [municipalities,mups,lics,dem200,rainfall,roads,contours],
    queryVisible: true
});

infoCtrl.events.on({
    getfeatureinfo: function (e) {
        if (e.text.length > 687) {
            createPopup(e);
        }
    }
});

function createPopup(e) {
    if (!popup) {
        popup = new GeoExt.Popup({
            title: "Feature Info",
            viewConfig: 'fit',
            margin: 5,
            padding: 5,
            autoScroll: true,
            maximizable: true,
            collapsible: true,
            panIn: false,
            map: map,
            location: map.getLonLatFromPixel(e.xy),
            html: e.text
        });
        popup.show();
        var maxWidth = parseInt(map.getSize().w - map.getSize().w/2);
        var maxHeight = parseInt(map.getSize().h - map.getSize().h/4);
        if (popup.getWidth() > maxWidth) {
            popup.setWidth(maxWidth);
        }
        if (popup.getHeight() > maxHeight) {
            popup.setHeight(maxHeight);
        }
    }
    if (popup) {
        popup.destroy();
        popup = new GeoExt.Popup({
            title: "Feature Info",
            viewConfig: 'fit',
            margin: 5,
            padding: 5,
            autoScroll: true,
            maximizable: true,
            collapsible: true,
            panIn: false,
            map: map,
            location: map.getLonLatFromPixel(e.xy),
            html: e.text
        });
        popup.show();
        var maxWidth = parseInt(map.getSize().w - map.getSize().w/2);
        var maxHeight = parseInt(map.getSize().h - map.getSize().h/4);
        if (popup.getWidth() > maxWidth) {
            popup.setWidth(maxWidth);
        }
        if (popup.getHeight() > maxHeight) {

```

```

        popup.setHeight(maxHeight);
    }
}

map.addControl(infoCtrl);
infoCtrl.activate();

/*TOOLBAR*/
/*ADD STREET MAP AND BIRDEYE. DUALMAPS FUNCTIONALITY*/
function addToDualViewPopup(loc, zoom) {
    if (!dvpopup) {
        dvpopup = new Ext.Window({
            title: "StreetView & Bird's Eye",
            width: 800,
            height: 600,
            maximizable: true,
            modal: true,
            html: '<iframe src="resources/dualmap/map.htm?x=' + loc.lon + '&y=' + loc.lat
                + '&z=' + zoom +
                '&gm=2&ve=5&gc=&bz=0&bd=0&mw=1&sv=1&sh=0&svb=0&svp=0&svz=2&
                svm=1&svf=1" width="100%" height="100%" frameborder="0"
                scrolling="no">/iframe>',
            listeners: {
                close: function () {
                    dvpopup = null;
                }
            }
        });
    }
    dvpopup.show();
}

OpenLayers.Control.Click=OpenLayers.Class(OpenLayers.Control,{
    defaultHandlerOptions: {
        'single': true,
        'double': true,
        'pixelTolerance': 0,
        'stopSingle': false,
        'stopDouble': false
    },
    initialize: function(options) {
        this.handlerOptions = OpenLayers.Util.extend(
            {}, this.defaultHandlerOptions
        );
        OpenLayers.Control.prototype.initialize.apply(
            this, arguments
        );
        this.handler = new OpenLayers.Handler.Click(
            this, {
                'click': this.trigger
            }, this.handlerOptions
        );
    },
    trigger: function (evt) {
        var maploc = map.getLonLatFromViewPortPx(evt.xy);
        var loc = maploc.transform(map.projection,map.displayProjection);
        var zoom = map.getZoom();
        addToDualViewPopup(loc, zoom);
    }
});

click = new OpenLayers.Control.Click();
map.addControl(click);

var dualview = new GeoExt.Action({
    tooltip: "Launch StreetView & Bird's Eye at click location",
    iconCls: "icon-dualview",
    toggleGroup: "navigation",
    id: "DualView",
    pressed: false,
    allowDepress: true,
    control: click,
    map: map,

```

```

        handler: function () {
            Ext.getCmp("map").body.applyStyles("cursor:crosshair");
            var element = document.getElementById("output");
            element.innerHTML = "";
            layerRuler.removeFeatures(layerRuler.features);
        }
    });

    /*IDENTIFY TOOL*/
    var identify = new GeoExt.Action({
        id: "identifyButton",
        tooltip: "Identify Features",
        iconCls: "icon-identify",
        toggleGroup: "navigation",
        pressed: false,
        allowDepress: true,
        control: infoCtrl,
        map: map,
        handler: function () {
            if (Ext.getCmp("identifyButton").pressed === true) {
                Ext.getCmp("map").body.applyStyles("cursor:help");
                var element = document.getElementById("output");
                element.innerHTML = "";
                layerRuler.removeFeatures(layerRuler.features);
            } else if (Ext.getCmp("identifyButton").pressed === false) {
                Ext.getCmp("map").body.applyStyles("cursor:default");
                var element = document.getElementById("output");
                element.innerHTML = "";
                layerRuler.removeFeatures(layerRuler.features);
                if (popup) {
                    popup.close();
                }
            }
        }
    });

    /*ZOOM IN*/
    var zoomIn = new GeoExt.Action({
        id: "zoominButton",
        tooltip: "Zoom In",
        iconCls: "icon-zoomin",
        toggleGroup: "navigation",
        pressed: false,
        allowDepress: true,
        control: new OpenLayers.Control.ZoomBox({
            alwaysZoom: true
        }),
        map: map,
        handler: function () {
            if (Ext.getCmp("zoominButton").pressed === true) {
                Ext.getCmp("map").body.applyStyles("cursor:crosshair");
                var element = document.getElementById("output");
                element.innerHTML = "";
                layerRuler.removeFeatures(layerRuler.features);
            } else if (Ext.getCmp("zoominButton").pressed === false) {
                Ext.getCmp("map").body.applyStyles("cursor:default");
            }
        }
    });

    /*ZOOM OUT*/
    var zoomOut = new Ext.Button({
        tooltip: "Zoom Out",
        iconCls: "icon-zoomout",
        handler: function () {
            map.zoomOut();
        }
    });

    /*ZOOM PREVIOUS*/
    var navHistoryCtrl = new OpenLayers.Control.NavigationHistory();
    map.addControl(navHistoryCtrl);
    var zoomPrevious = new GeoExt.Action({
        tooltip: "Zoom to Previous Extent",
        iconCls: "icon-zoomprevious",

```

```

        control:navHistoryCtrl.previous,
        disabled:true
    });

    /*ZOOM NEXT*/
    var zoomNext= new GeoExt.Action({
        tooltip: "Zoom to Next Extent",
        iconCls: "icon-zoomnext",
        control: navHistoryCtrl.next,
        disabled:true
    });

    /*ZOOM TO EXTENT*/
    var zoomExtentBtn = new Ext.Button({
        tooltip: "Zoom to Initial Extent",
        iconCls: "icon-zoomextent",
        handler: function () {
            map.setCenter(new OpenLayers.LonLat(-0.7782,39.0678).transform('EPSG:4326',
            'EPSG:900913'), 10);
        }
    });

    /*MEASURE TOOLS*/
    var linemeasureStyles = {
        "Point": {
            pointRadius: 4,
            graphicName: "square",
            fillColor: "white",
            fillOpacity: 1,
            strokeWidth: 1,
            strokeOpacity: 1,
            strokeColor: "#333333"
        },
        "Line": {
            strokeColor: "#FF0000",
            strokeOpacity: 0.3,
            strokeWidth: 3,
            strokeLinecap: "square"
        }
    };

    var lineStyle = new OpenLayers.Style();
    lineStyle.addRules([
        new OpenLayers.Rule({symbolizer: linemeasureStyles
    }));

    var linemeasureStyleMap = new OpenLayers.StyleMap({"default": lineStyle});

    var length = new OpenLayers.Control.Measure(OpenLayers.Handler.Path, {
        displaySystem: "metric",
        geodesic: false,
        persist: true,
        handlerOptions: {
            layerOptions: {
                styleMap: linemeasureStyleMap
            }
        },
        textNodes: null,
        callbacks: {
            create: function () {
                this.textNodes = [];
                layerRuler.removeFeatures(layerRuler.features);
                mouseMovements = 0;
            },
            modify: function (point, line) {
                if (mouseMovements++ < 5) {
                    return;
                }

                var len = line.geometry.components.length;
                var from = line.geometry.components[len - 2];
                var to = line.geometry.components[len - 1];
                var ls = new OpenLayers.Geometry.LineString([from, to]);
                var dist = this.getBestLength(ls);
                if (!dist[0]) {

```

```

        return;
    }
    var total = this.getBestLength(line.geometry);
    var label = dist[0].toFixed(2) + " " + dist[1];
    var textNode = this.textNodes[len - 2] || null;
    if (textNode && !textNode.layer) {
        this.textNodes.pop();
        textNode = null;
    }
    if (!textNode) {
        var c = ls.getCentroid();
        textNode = new OpenLayers.Feature.Vector(
            new OpenLayers.Geometry.Point(c.x, c.y), {}, {
                label: "",
                fillColor: "#FF0000",
                fontSize: "14px",
                fontFamily: "Arial",
                fontWeight: "bold",
                labelAlign: "cm"
            });
        this.textNodes.push(textNode);
        layerRuler.addFeatures([textNode]);
    }
    textNode.geometry.x = (from.x + to.x) / 2;
    textNode.geometry.y = (from.y + to.y) / 2;
    textNode.style.label = label;
    textNode.layer.drawFeature(textNode);
    this.events.triggerEvent("measuredynamic", {
        measure: dist[0],
        total: total[0],
        units: dist[1],
        order: 1,
        geometry: ls
    });
}
});

function handleMeasurements(event) {
    var geometry = event.geometry;
    var units = event.units;
    var order = event.order;
    var measure = event.measure;
    var element = document.getElementById("output");
    var hectares;
    var out = "";
    if (order === 1) {
        out += measure.toFixed(2) + " " + units;
    } else if (order === 2 && units === "m" && measure >= 10000) {
        hectares = measure / 10000;
        out += hectares.toFixed(2) + " hectares";
    } else {
        out += measure.toFixed(2) + " " + units + "<sup>2</sup>/" + "sup>";
    }
    element.innerHTML = "&nbsp;&nbsp;&nbsp;" + out;
}
length.events.on({
    "measure": handleMeasurements,
    "measurepartial": handleMeasurements
});

var areameasureStyles = {
    "Point": {
        pointRadius: 4,
        graphicName: "square",
        fillColor: "white",
        fillOpacity: 1,
        strokeWidth: 1,
        strokeOpacity: 1,
        strokeColor: "#333333"
    },
    "Polygon": {
        strokeWidth: 3,
        strokeOpacity: 1,
        strokeColor: "red",

```

```

        fillColor: "red",
        fillOpacity: 0.3
    }
};

var areaStyle = new OpenLayers.Style();
areaStyle.addRules([
    new OpenLayers.Rule({
        symbolizer: areameasureStyles
    })
]);
var areaStyleMap = new OpenLayers.StyleMap({
    "default": areaStyle
});
var area = new OpenLayers.Control.Measure(OpenLayers.Handler.Polygon, {
    displaySystem: "metric",
    geodesic: false,
    persist: true,
    handlerOptions: {
        layerOptions: {
            styleMap: areaStyleMap
        }
    }
});

area.events.on({
    "measure": handleMeasurements,
    "measurepartial": handleMeasurements
});

map.addControl(length);
map.addControl(area);

var measureLength = new GeoExt.Action({
    id: "measurelengthButton",
    tooltip: "Measure Length",
    iconCls: "icon-measure-length",
    toggleGroup: "navigation",
    enableToggle: true,
    pressed: false,
    allowDepress: true,
    control: length,
    map: map,
    handler: function () {
        if (Ext.getCmp("measurelengthButton").pressed === true) {
            Ext.getCmp("map").body.applyStyles("cursor:crosshair");
            var element = document.getElementById("output");
            element.innerHTML = "";
        } else if (Ext.getCmp("measurelengthButton").pressed === false) {
            Ext.getCmp("map").body.applyStyles("cursor:default");
        }
    }
});

var measureArea = new GeoExt.Action({
    id: "measureareaButton",
    tooltip: "Measure Area",
    iconCls: "icon-measure-area",
    toggleGroup: "navigation",
    enableToggle: true,
    pressed: false,
    allowDepress: true,
    control: area,
    map: map,
    handler: function () {
        if (Ext.getCmp("measureareaButton").pressed === true) {
            Ext.getCmp("map").body.applyStyles("cursor:crosshair");
            var element = document.getElementById("output");
            element.innerHTML = "";
            layerRuler.removeFeatures(layerRuler.features);
        } else if (Ext.getCmp("measureareaButton").pressed === false) {
            Ext.getCmp("map").body.applyStyles("cursor:default");
        }
    }
});

```



```

zoom: 10,
map: map,
tbar: toolBar, //Toolbar in top of the map.

}, {
  xtype: 'panel',
  title: 'Geoprocesses',
  region: 'east',
  width: 380,
  collapsible: true,
  autoScroll: true,
  items: [
    {
      xtype: 'panel',
      title: 'Tools list',
      //layout: 'fit',
      autoScroll: false,
      items: [
        {
          xtype: 'panel',
          title: 'Basic tools',
          collapsible: true,
          collapsed: true,
          border: false,
          autoScroll: false,
          items: [
            {
              xtype: 'tabpanel',
              id: 'basictools_statistics',
              border: false,
              activeTab: 0,
              autoScroll: false,
              items: [
                {
                  xtype: 'panel',
                  title: 'Basic statistics',
                  //layout: 'auto',
                  items: [
                    {
                      html: "<br><p><b>Calculates the average, minimum and maximum value<br>inside the area drawn by the user for the selected raster.</b></p><ol><br><li>1. Select the raster you want calculate the statistics in the<br>dropdown list.</li><li>2. Click the editing button (first on left) on the map.</li><li>3. Draw a polygon that will define the statistics area.</li><li>4. Click on the button below to obtain statistics, check them<br>in the results area.</li></ol><br>"
                    },
                    {
                      html: "<br>"
                    },
                    {
                      xtype: 'combo',
                      id: 'selectRasterComboBox',
                      emptyText: "Choose raster...",
                      store: new Ext.data.ArrayStore({
                        fields: ['rasterName'],
                        data : [['DEM'], ['Rainfall']]
                      }),
                      //fieldLabel: 'Please select a raster',

                      displayField: 'rasterName',
                      //valueField: 'rasterName',
                      typeAhead: true,
                      mode: 'local',
                      multiSelect: 'false',
                      triggerAction: 'all',
                      selectOnFocus: true,
                      forceSelection: true,
                      listeners: {
                        'select': function(combo){
                          var
                          combo=Ext.getCmp('selectRasterComboBox')
                          ;
                          alert("Statistics will be calculated on "+combo.getValue()+" raster");
                          if (combo.getValue()=="DEM"){

```



```

        rasterIdentifier="forestry:dem";
        alert("Units are meters");
    } else {
        rasterIdentifier="forestry:rainfall";
        alert("Units are tenths of mm");
    }
    },{
        html:"<br>"
    },{
        xtype:"button",
        text: 'Compute Statistics',
        listeners: {
            'click': function(){obtainStatisticsProcess()}
        }
    }
    ],{
        xtype:'panel',
        title: 'Raster transparency',
        layout:'auto',
        items: [{
            html:"<p><br><p><b>This tool helps photointerpretation using layer transparency.</b></p><ol><br><li>1. Select the layer you want to apply the transparency for.</li><li>2. Move the slider left and right to change the transparency value.</li><li>3. Check the results on the map.</li></ol><br><br></p>"
        },{
            html:"<br>"
        },
        {
            xtype:'combo',
            id:'selectRasterComboBox2',
            emptyText:"Choose raster...",
            store: new Ext.data.ArrayStore({
                fields: ['rasterName'],
                data : [['DEM'], ['Rainfall']]
            }),
            displayField: 'rasterName',
            typeAhead: true,
            mode: 'local',
            multiSelect:'false',
            triggerAction: 'all',
            selectOnFocus:true,
            forceSelection:true,
            listeners:{
                'select': trasparencyRaster=function(){
                    var
                    combo=Ext.getCmp('selectRasterComboBox2');
                    alert("Transparency will be applied to "+combo.getValue()+" raster");
                    if (combo.getValue()=="DEM"){
                        return dem200;
                    } else if (combo.getValue()=="Rainfall"){
                        return rainfall;
                    }
                    else{
                        alert("Not raster");
                    }
                }
            }
        },
        {
            xtype: "gx_opacityslider",
            layer: dem200,
            aggressive:true,
            x:10,
            y:20
        }
    ]
    },{
        xtype:'panel',
        title:'Buffers',
        layout:'auto',

```

```

items: [{
  html: "<p><br><p><b>This tool calculates buffers
from the geometry created<br>by the
user.</p><br>Please introduce a distance, press
Enter and click<br>the <i>Compute Buffer</i>
button. </b></p><br>"
}],{
  xtype:'panel',
  vertical:true,
  items: [
    {
      xtype:'numberfield',
      allowNegative:false,
      boxMaxWidth:5,
      emptyText: "Introduce a distance",
      listeners:{
        specialkey: function (f,e) {
          if (e.getKey() == e.ENTER) {
            distanceValue=this.getValue();
            alert("Buffer of "+this.getValue()+" m will be
calculated");
          }
        }
      },
      {
        html:"<br>"
      },
      {
        xtype:"button",
        text: 'Compute Buffer',
        listeners: {
          'click': function(){calculateBuffers()}
        }
      },
      {
        xtype:"button",
        text: 'Delete Buffer',
        listeners: {
          'click': function(){
            vector_layer.removeFeatures(vector
            _layer.features);
          }
        }
      }
    ]
  }
}],{
  xtype:'panel',
  title:'Forestry tools',
  collapsible:true,
  collapsed:true,
  autoScroll:false,
  border:false,
  items:[{
    xtype:'tabpanel',
    id:'forestrytools_roads',
    border: false,
    activeTab:0,
    items:[{
      xtype:'panel',
      title: 'Road Description',
      layout: 'auto',
      items:[{
        html: "<br><p><b>This tool returns some interesting
descriptors about the roads inside an area defined
by the user.</b></p><ol><br><li>1. Clic the editing
button (first on left) on the map.</li><li>2. Draw a
polygon that will define the study area.</li><li>3.
Click on the button below to obtain the results, check
them in the results area.</li><ol><br><br>"

```

```

    },{
        html:"<br><br>"
    },{
        xtype:"button",
        text: 'Describe roads',
        listeners: {
            'click': function(){
                DescribeRoadsProcess();
            }
        }
    ]
},
{
    xtype:'panel',
    title: 'Vegetation Description',
    layout: 'auto',
    items: [{
        html: "<br><p><b>This tool returns some interesting statistics about vegetation inside an area defined by the user.</b></p><ol><br><li>1. Clic the editing button (first on left) on the map.</li><li>2. Draw a polygon that will define the study area.</li><li>3. Click on the button below to obtain the statistics, check them in the results area.</li><ol><br><br>"
    },{
        html:"<br><br>"
    },{
        xtype:"button",
        text: 'Describe Vegetation',
        listeners: {
            'click': function(){describeVegetation();}
        }
    ]
}
]
}],
autoScroll:true,
height:350
},{
    xtype:'panel',
    title:'Results',
    autoScroll:true,
    border:false,
    items: [{
        xtype:'panel',
        id:'ResultsPanel',
        style:"padding-bottom:185px",
        border:false,
        height:185
    },
    {
        xtype:'panel',
        items:[{
            xtype:"button",
            text: "Clear Content",
            border:false,
            style: 'float:right',
            listeners:{
                'click': function(){refreshPanel();}
            }
        }
    ]
}
}
}],
xtype:'panel',
region: 'south',
height:50

```

```

    }}
  });

  /*ADD CONTROLS TO THE MAP*/
  controls.push(
    new OpenLayers.Control.Navigation({dragPanOptions:{enableKinetic:true}}),
    new OpenLayers.Control.Attribution(),
    new OpenLayers.Control.PanPanel(),
    new OpenLayers.Control.ZoomPanel()
  );

  /*LAYERS TREE*/
  treeConfig={{
    text:"<b>Base Layers</b>",
    expanded:false,
    singleClickExpand:true,
    nodeType:"gx_baselayercontainer"},
    {
      text:"<b>Overlays</b>",
      expanded:true,
      singleClickExpand:true,
      //enableDD:true,
      children:{{
        text:"Administrative boundaries",
        children:{{
          text:"Municipalities",
          nodeType:"gx_layer",
          layer:municipalities
        }}
      }}
    },
    {
      text:"Infrastructures",
      children:{{
        text:"Roads",
        nodeType:"gx_layer",
        layer:roads
      }}
    },
    {
      text:"Protected Areas",
      children:{{
        text:"LIC Areas",
        nodeType:"gx_layer",
        layer:lics
      }}
    },
    {
      text:"Land Property",
      children:{{
        text:"Public Forests",
        nodeType:"gx_layer",
        layer:mups
      }}
    },
    {
      text:"Meteorology",
      children:{{
        text:"Weather",
        nodeType:"gx_layer",
        layer:cityWeather
      }},
      {
        text:"Clouds",
        nodeType:"gx_layer",
        layer: clouds
      },
      {
        text:"Qualitative rain intensity",
        nodeType:"gx_layer",
        layer:precipitation
      },
      {
        text:"Qualitative temperature",
        nodeType:"gx_layer",
        layer: temperature
      }
    },
    {
      text:"Topography",
      children:{{
        text: "Digital Elevation Model 200 meters",
        nodeType:"gx_layer",

```

```

        layer:dem200
    }{
        text:"Contours",
        nodeType:"gx_layer",
        layer:contours
    }
    }{
        text:"Climatology",
        children:[{
            text:"Average yearly rainfall (mm)",
            nodeType:"gx_layer",
            layer:rainfall
        }
    ]
    }{
        text:"Cultural-historical layers",
        children:[{
            text:"Archeological sites (external WMS)",
            nodeType:"gx_layer",
            layer: archeologicalSites
        }
    ]
    }
    }
}

tree=new Ext.tree.TreePanel({
    loader: new Ext.tree.TreeLoader({
        applyLoader:false
    }),
    root:{
        nodeType:"async",
        children:treeConfig
    },
    border:false,
    renderTo: 'ext-gen21',
    rootVisible:false,
    enableDD:false
});

/*HELPER FUNCTIONS*/
/*CONTROLS THAT ALLOW WFS-T*/
var panel = new OpenLayers.Control.Panel({
    displayClass: 'customEditingToolbar',
    allowDepress: true
});

var draw = new OpenLayers.Control.DrawFeature(
    polygonStatistics, OpenLayers.Handler.Polygon,
    {
        title: "Draw Feature",
        displayClass: "olControlDrawFeaturePolygon",
        multi: true
    }
);

var edit = new OpenLayers.Control.ModifyFeature(polygonStatistics, {
    title: "Modify Feature",
    displayClass: "olControlModifyFeature"
});

var del = new DeleteFeature(polygonStatistics, {title: "Delete Feature"});

var save = new OpenLayers.Control.Button({
    title: "Save Changes",
    trigger: function() {
        if(edit.feature) {
            edit.selectControl.unselectAll();
        }
        saveStrategy.save();
        alert("Changes correctly saved.");
    },
    displayClass: "olControlSaveFeatures"
});

/*ROUND FUNCTION*/

```

```

function roundNumber(number, digits) {
    var multiple = Math.pow(10, digits);
    var rndedNum = Math.round(number * multiple) / multiple;
    return rndedNum;
}

/*REFRESH PANEL FUNCTION*/
function refreshPanel(){
    resultWindow.destroy();
}

panel.addControls([save, del, edit, draw]);
map.addControl(panel);
map.addControl(panel);

****SPATIAL PROCESSES****
/*PROCESS1 - RASTER STATISTICS. Calculate the average/min/max values of a raster using a
user-defined polygon*/
function obtainStatisticsProcess(){
    var format= new OpenLayers.Format.WPSExecute();

    var doc= format.write({
        identifier: "gs:RasterZonalStatistics",
        dataInputs:{
            identifier:'data',
            reference:{
                mimeType:"image/tiff",
                href: "http://geoserver/wcs",
                method: "POST",
                body: {
                    wcs: {
                        identifier:'forestry:mde200_V4',
                        version:"1.1.1",
                        domainSubset: {
                            boundingBox: {
                                projection:
                                'http://www.opengis.net/gml/srs/epsg.xml#3857',
                                bounds: new OpenLayers.Bounds(-200931.8344047,
                                4681799.186142525,32948.35238090271,
                                4948113.428142309)
                            }
                        },
                        output: {format:'image/tiff'},
                    }
                }
            }
        },
        identifier:'zones',
        reference:{
            mimeType:"text/xml; subtype=wfs-collection/1.1",
            href: "http://geoserver/wfs",
            method: 'POST',
            body:{
                wfs:{
                    version:'1.1.0',
                    outputFormat: "GML2",

                    featureNS: "http://localhost:8080/thesis/forestry",

                    featureType:'forestry:computestatisticspolygon',
                    geometry:'geom'
                }
            }
        }
    }],
    responseForm:{
        rawDataOutput:{
            mimeType:"application/json",
            identifier:"statistics"
        }
    }
});

```

```

OpenLayers.Request.POST({
  url:"http://localhost:8080/geoserver/ows",
  data:doc,
  headers: {
    "Content-Type":"application/xml;charset=utf-8"
  },
  success: function(response){
    var response=JSON.parse(response.responseText);

    statisticsMessage="<p>The statistics for the selected area are:
    "+"<br>"+<b>Minimum value:</b>
    "+JSON.parse(response.features[0].properties["min"])+<br>"+
    "<b>Maximum value:</b>
    "+JSON.parse(response.features[0].properties["max"])+<br>"+
    "<b>Average value:</b>
    "+roundNumber((JSON.parse(response.features[0].properties["avg"])
    ),2)+<p>";
    resultWindow= new Ext.Panel({
      renderTo: "ResultsPanel",
      html: statisticsMessage
    });

    resultWindow.show();
  }
});
}

```

/*PROCESS 2. DESCRIBE ROADS. Return some roads statistics within a polygon defined by the user*/

```

function DescribeRoadsProcess(){
  var format= new OpenLayers.Format.WPSExecute();
  doc=format.write({
    identifier:"JTS:length",
    dataInputs:{
      identifier:"geom",
      reference:{
        mimeType:"text/xml;subtype=gml/3.1.1",
        href:"http://geoserver/wps",
        method:"POST",
        body:{
          identifier:"gs:CollectGeometries",
          dataInputs:{
            identifier:"features",
            reference:{
              mimeType:"text/xml",
              href:"http://geoserver/wps",
              method:"POST",
              body:{
                identifier:"gs:IntersectionFeatureCollection",
                dataInputs:{
                  identifier:"first feature collection",
                  reference:{
                    mimeType:"text/xml",
                    href:"http://geoserver/wfs",
                    method:"POST",
                    body:{
                      wfs:{
                        version: "1.0.0",
                        outputFormat:"GML2",
                        featureNS:"http://localhost:8080/thesis/forestry",
                        featureType:"forestry:RoadsCanalNavarres3857_3"
                      }
                    }
                  }
                }
              }
            }
          }
        }
      }
    }
  });
  identifier:"second feature collection",
  reference:{
    mimeType:"text/xml",
    href:"http://geoserver/wfs",
    method:"POST",
    body:{
      wfs:{
        version: "1.0.0",
        outputFormat:"GML2",

```

```

        featureNS:"http://localhost:8080/thesis/forestry",
        featureType:"forestry:computestatisticspolygon"
    }
}
}
}],
responseForm:{
  rawDataOutput:{
    mimeType:"text/xml; subtype=wfs-collection/1.0",
    identifier:"result"
  }
}
}],
responseForm:{
  rawDataOutput:{
    mimeType:"text/xml; subtype=gml/3.1.1",
    identifier:"result"
  }
}
}],
responseForm:{
  rawDataOutput:{
    identifier:"result"
  }
}
});

OpenLayers.Request.POST({
  url:"http://localhost:8080/geoserver/ows",
  data:doc,
  headers:{
    "Content-Type":"application/xml;charset=utf-8"
  },
  success: function(response){
    totalLength=response.responseText;
  }
});

}

function calculatePolygonSurface(){
  var format=new OpenLayers.Format.WPSExecute();
  doc=format.write({
    identifier: "JTS:area",
    dataInputs: [{
      identifier:"geom",
      reference:{
        mimeType:"text/xml;subtype=gml/3.1.1",
        href:"http://geoserver/wps",
        method:"POST",
        body:{
          identifier:"gs:CollectGeometries",
          dataInputs:[{
            identifier:"features",
            reference:{
              mimeType:"text/xml",
              href:"http://geoserver/wfs",
              method:"POST",
              body:{
                wfs:{
                  version:"1.0.0",
                  outputFormat:"GML2",
                  featureNS:"http://localhost:8080/thesis/forestry",
                  featureType:"forestry:computestatisticspolygon"
                }
              }
            }
          ]
        }
      }
    }
  ]
  }
  });
  responseForm:{
    rawDataOutput:{
      mimeType:"text/xml; subtype=gml/3.1.1",

```



```

        identifier:"result"
    }
}
}
}
}
});
OpenLayers.Request.POST({
    url:"http://localhost:8080/geoserver/ows",
    data:doc,
    headers:{
        "Content-Type":"application/xml;charset=utf-8"
    },
    success: function(response){
        totalArea=response.responseText;

        var roadDensity=roundNumber((totalLength/(totalArea/10000)),2);
        var densityClassification;

        if (roadDensity<=30){
            densityClassification="Road density is very low";
        }
        else if (roadDensity<=100){
            densityClassification="Road density is adequate";
        }
        else {
            densityClassification="Road density too high";
        }

        RoadsMessage="<p>The characteristics of the roads in the selected area
are: "+<br>"+<br>"+<b>Total road length:</b>
"+roundNumber(totalLength,2).toString()+" <b>m</b>"+<br>"+
"<b>Total road density</b>
"+roadDensity.toString()+"<b>m/ha"<br>"+densityClassification

        resultWindow= new Ext.Panel({
            renderTo: "ResultsPanel",
            html: RoadsMessage
        });

        resultWindow.show();
    }
});
}

/*PROCESS 3. BUFFER CALCULATION. Classical buffer calculation from polygon geometry*/
function calculateBuffers(){
    var format= new OpenLayers.Format.WPSExecute();

    var doc= format.write({
        identifier: "gs:BufferFeatureCollection",
        dataInputs:[{
            identifier:'features',
            reference:{
                mimeType:"text/xml; subtype=wfs-collection/1.0",
                href: "http://geoserver/wfs",
                method: "POST",
                body: {
                    wfs: {
                        version:"1.0.0",
                        outputFormat: "GML2",
                        featureNS:"http://localhost:8080/thesis/forestry",
                        featureType:"forestry:computestatisticspolygon"
                    }
                }
            }
        }
    ],
    {
        identifier:'distance',

```

```

        data:{
            literalData:{
                value: distanceValue
            }
        }
    },
    responseForm:{
        rawDataOutput:{
            mimeType:"application/json",
            identifier:"result"
        }
    }
});

OpenLayers.Request.POST({
    url:"http://localhost:8080/geoserver/ows",
    data:doc,
    headers: {
        "Content-Type":"application/xml;charset=utf-8"
    },
    success: function(response){
        result=response.responseText;
        var geojson_format = new OpenLayers.Format.GeoJSON();
        vector_layer = new OpenLayers.Layer.Vector();
        map.addLayer(vector_layer);
        vector_layer.addFeatures(geojson_format.read(result));
    }
});
}

```

/*PROCESS 4. VEGETATION STATISTICS. Extraction of attributes of National Forest Map*/

```

function describeVegetation(){
    var format= new OpenLayers.Format.WPSExecute();

    var doc= format.write({
        identifier:"gs:IntersectionFeatureCollection",
        dataInputs:[{
            identifier:"first feature collection",
            reference:{
                mimeType:"text/xml",
                href:"http://geoserver/wfs",
                method:"POST",
                body:{
                    wfs:{
                        version: "1.1.0",
                        outputFormat:"GML2",
                        featureNS:"http://localhost:8080/thesis/forestry",
                        featureType:"forestry:forestryMap"
                    }
                }
            }
        }],{
        identifier:"second feature collection",
        reference:{
            mimeType:"text/xml",
            href:"http://geoserver/wfs",
            method:"POST",
            body:{
                wfs:{
                    version: "1.1.0",
                    outputFormat:"GML2",
                    featureNS:"http://localhost:8080/thesis/forestry",
                    featureType:"forestry:computestatisticspolygon"
                }
            }
        }
    }
    });
    OpenLayers.Request.POST({
        url:"http://localhost:8080/geoserver/ows",
        data:doc,
        headers: {
            "Content-Type":"application/xml;charset=utf-8"
        },
    },

```

```

success: function(response){
    var content=response.responseText;
    var response=JSON.parse(content);
    var vegetationTypes=[];
    for (i=0; i<(response.features).length;i++){
        vegetationTypes.push(response.features[i].properties["forestryMap_
        NOM_FORARB"]);
    }

    function removeDuplicatesInPlace(arr) {
        var i, j, cur, found;
        for (i = arr.length - 1; i >= 0; i--) {
            cur = arr[i];
            found = false;
            for (j = i - 1; !found && j >= 0; j--) {
                if (cur === arr[j]) {
                    if (i !== j) {
                        arr.splice(i, 1);
                    }
                    found = true;
                }
            }
        }
    }

    return arr;
};

var resultVegetation=removeDuplicatesInPlace(vegetationTypes);

vegetationMessage="<p>In this polygon the vegetation we can find is:
"+resultVegetation+"</p>";

resultWindow= new Ext.Panel({
    renderTo: "ResultsPanel",
    html: vegetationMessage
});

resultWindow.show();
}
});
</script>
</head>
<body>
</body>

```





Masters
Program
in **Geospatial
Technologies**
