THE WEB FEATURE SERVICE – TRANSACTION STANDARD

An example of prototype implementation

Jacinto Paulo Simões Estima

Project Report submitted in partial fulfilment of the requirements for the degree of *Mestre em Ciência e Sistemas de Informação Geográfica* (Master in Geographical Information Systems and Science)

Instituto Superior de Estatística e Gestão de Informação
da Universidade Nova de Lisboa

THE WEB FEATURE SERVICE – TRANSACTION STANDARD

An example of prototype implementation

Project Work supervised by

Professor Doutor Pedro da Costa Brito Cabral

July of 2010

ACKNOWLEDGMENTS

I begin by thanking to my supervisor Professor Pedro Cabral, for his patience, suggestions and guidance, and to whom I express my sincere gratitude.

Sincere thanks to my friends and colleagues António José Silva and Hugo Martins for the suggestions and support given.

A special thanks to my friend Joana Simões, for all the precious help given in programming the developed application. Thanks Joana.

Big thanks to my colleague and friend Ana Balula for the suggestions made regarding language issues.

Last but not least, to my wife Ana Cristina Estima, my son João Pedro Estima and my parents for their support and understanding during the whole process.

In general, my thanks to everyone that directly or indirectly has contributed for this work.

THE WEB FEATURE SERVICE – TRANSACTION STANDARD

An example of prototype implementation

RESUMO

Com o crescimento da Internet, a quantidade de dados geográficos disponível também aumentou. Existem, neste momento, vários servidores a nível planetário a disponibilizar informação espacial para muitos utilizadores que lhes acedem através de diferentes softwares de Sistemas de Informação Geográfica. Torna-se portanto necessário desenvolver normas por forma a resolver problemas de interoperabilidade inerentes ao facto dos dados serem distribuídos e heterogéneos. Este trabalho pretende mostrar a importância das normas particularmente o standard *Web Feature Service – Transactio* no domínio da Informação Geográfica. É também desenvolvido um protótipo que disponibiliza Informação Geográfica de acordo com esta norma, desenvolvido com recurso aos softwares: PostgreSQL/PostGIS, GeoServer, uDig, Apache e MapFish.

THE WEB FEATURE SERVICE – TRANSACTION STANDARD

An example of prototype implementation

ABSTRACT

With the Internet growth, the production and amount of available geographic data also increased. There are, at the moment, many servers around the world providing spatial information to several users, and many of them are accessing this information through different Geographic Information Systems (GIS) applications. Therefore, it becomes necessary to develop standards in order to solve the interoperability problem inherent to this heterogeneous and distributed data. This work aims at showing the significance of standards, particularly the Web Feature Service - Transaction (WFS-T) standard in the Geographic Information (GI) domain. A prototype that provides GI according with this standard is also implemented, based on PostgreSQL/PostGIS, GeoServer, uDig, Apache and MapFish.

PALAVRAS-CHAVE

Interoperabilidade

Sistemas de Informação Geográfica

Serviços Web

Standards OGC

Web Feature Service

Web Feature Service - Transaction

Web Map Service

KEYWORDS

Interoperability

Geographic Information Systems

Web Services

OGC Standards

Web Feature Service

Web Feature Service – Transaction

Web Map Service

ACRONYMS

API – Application Programming Interface

BBOX – Bounding Box

CRS – Coordinate Reference System

DB – Database

EPSG – European Petroleum Survey Group

EU – European Union

EXE – Filename extension denoting an executable file

FAQ – Frequently Asked Questions

GI – Geographic Information

GIS – Geographic Information Systems

GML – Geography Markup Language

HTML – HyperText Markup Language

HTTP – HyperText Transfer Protocol

HTTPS – Hypertext Transfer Protocol Secure

IGP – Instituto Geográfico Português (Portuguese Geographic Institute)

IT – Information Technologies

JDK – Java Development Kit

KVP – Keyword-Value Pairs

OGC – Open Geospatial Consortium

OGP – International Association of Oil & Gas Producers

OS – Operating System

SDI – Spatial Data Infrastructure

SNIG – Sistema Nacional de Informação Geográfica (National System of Geographic Information)

SOAP – Simple Object Access Protocol

SRS – Spatial Reference System

UDDI – Universal Description, Discovery and Integration

URL – Uniform Resource Locator

XML – Extensible Markup Language

XPath – XML Path

W3C – World Wide Web Consortium

WFS – Web Feature Service

WFS-T – Web Feature Service Transactional

WMS – Web Map Service

WS – Web Service

WSDL – Web Service Definition Language

WWW – World Wide Web

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1. Introduction

1.1. GIS: From Desktop to Internet

The World Wide Web (WWW) has evolved significantly in the last years, especially at the technological level. The number of users has grown exponentially and, today, almost every family of developed and developing countries has an Internet connection. Consequently, a considerable amount of information emerges from the most varied sources. Most of this information has geographic representation and the WWW could be a very important exchange and/or dissemination vehicle. This is called Internet GIS, and represents an important advancement over the traditional desktop GIS (Peng and Tsou, 2003).

1.2. Dissertation scope

1970 was one of the most important years in the history of the network communications. The Department of Defense (DoD) of the United States of America (USA) developed the progenitor of the modern internet, the ARPANET. The adoption by this project, in 1993, of the Transmission Control Protocol/Internet Protocol (TCP/IP), was a great success and brought the development of telecommunications to a new age (Peng and Tsou, 2003). After that many applications using this technology appeared, like the WWW, e-mail, Telnet, among many others. Other important protocols were the HyperText Transfer Protocol (HTTP) and he HyperText Markup Language (HTML), where the WWW is based on.

The WWW is actually one of the biggest repositories of information in the world, allowing data sharing between different people at different locations. In fact, the recent technological developments in the Internet technology, improving access speed and easiness, and in the hardware and software providing computers with better processing capacity and also less expensive, caused significant changes in the manner that people use, share and process information. Nowadays, almost everyone in developed and developing countries has a computer in work and/or at

home and uses the WWW for this purpose, changing from closed, proprietary and un-interoperable to distributed, open and interoperable applications.

In GI we have also assisted to a change in paradigm, very close to the evolution of the computer technologies, from mainframe GIS to desktop GIS to recent distributed GIS that includes mobile and Internet GIS (figure 1). All the major search engines have now GIS offerings, like Google maps geocoding capabilities (Davis, 2007).

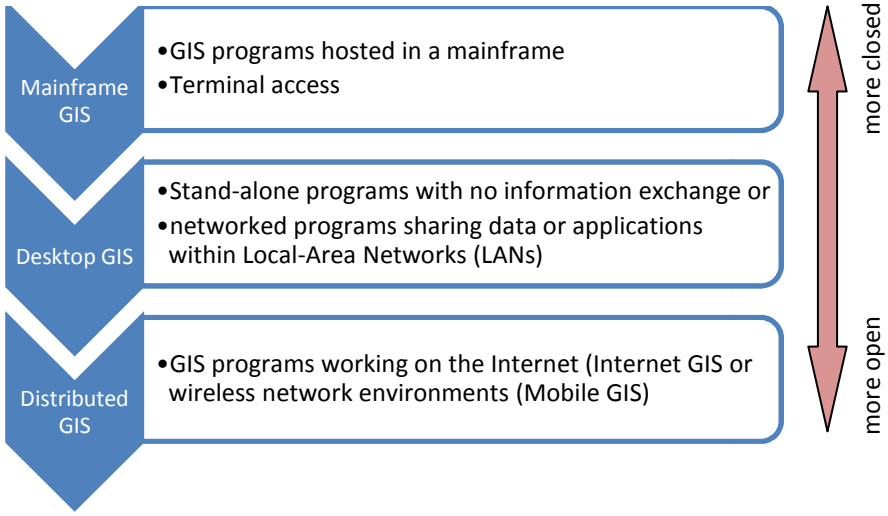

Figure 1- Evolution of GIS (adapted from Peng and Tsou (2003))

With the Internet technology, GIS concepts are more open and accessible to everyone, facilitating data access and exchange (Dragicevic, 2004).

One of the main problems in this evolution is the data, normally in proprietary format, inhibiting, for example, the use of data from different servers at different locations (Furtado, 2006). This problem is solved through the development of standards and specifications that make it possible for devices, data, and processes to operate together (Paul A. Longley, 2005).

With this problem in concern, the Open Geospatial Consortium (OGC) was created in 1994. The OGC is an international consortium of companies, government agencies and universities participating in a consensus process to develop publicly available interface specifications. OpenGIS Specifications (technical documents that

detail interfaces or encodings) support interoperable solutions that "geo-enable" the Web, wireless and location-based services, and mainstream Information Technologies (IT). The specifications empower technology developers to make complex spatial information and services accessible and useful with all kinds of applications (OGC, 2008).

The OGC has developed two important Open Standards - the Web Map Service (WMS) and the Web Feature Service (WFS), among many others. These two standards define the interfaces to remotely share and access GI. The WMS standard was first developed and offers a simple way to visualize GI, sending the "requested map" to the user as a static image while the WFS, with more functionalities, sends the features themselves with manipulation capabilities. Compliance with Open Standards is very important to warrant that different and remote systems share spatial data (Moreno-Sanchez, 2007).

Beyond these standards, it is also important to have some rules and policies to solve problems related to fragmentation of datasets and sources, gaps in availability, lack of harmonization between datasets at different geographical scales or duplication of information collection. The solution is to define Spatial Data Infrastructures (SDI). The European Union (EU), with this in mind, has created, in 2001, the initiative INSPIRE initiative[1]. *The initiative intends to trigger the creation of a European spatial information infrastructure that delivers to the users integrated spatial information services. These services should allow the users to identify and access spatial or geographical information from a wide range of sources, from the local level to the global level, in an inter-operable way for a variety of uses* (INSPIRE, 2009). To establish the general rules to achieve these objectives, the initiative has created the Directive with the same name that entered into force in May 2007.

This Directive *should be based on the infrastructures for spatial information that are created by the Member States and that are made compatible with common*

[1] INfrastructure for Spatial Information for Europe

Implementing Rules and are supplemented with measures at Community level to ensure the compatibility and usability in a Community and transboundary context (INSPIRE, 2009).

In Portugal, the organization that leads this implementation is the *Portuguese Geographic Institute* (IGP), the national authority for the geographic information, and involves the *National System of Geographic Information* (SNIG) institutions.

Henriques et al (1999) cited by Furtado (2006) states that SNIG was the first National Infrastructure of Geographic Information created in Europe, and also the first one in the world opened to the Internet, in 1995.

The IGP currently provides some GI through WMS and WFS specifications, like the "Atlas de Portugal" in WMS or the "Carta Administrativa Oficial de Portugal" both in WMS and WFS.

The Web Feature Service – Transaction (WFS-T) standard is one of the most recent specifications produced by the OGC and with enormous possibilities to be strongly used on the future, due to the benefit of editing features remotely through open standards. This is a very strong reason to study it and to make an implementation test.

1.3. Objectives and hypothesis

The main goal is to study the WFS-T standard. This includes the requirements verification to implement this specification and understand the transaction operation and how it can be implemented.

Further goals are to develop a prototype that implements the WFS with the transaction operation and to use both Open Source and proprietary software, to provide and access GI.

The principal hypothesis that this work aims to test is if it is possible to develop a FOSS prototype that implements the WFS-T standard.

1.4. Structure of the dissertation

This dissertation is organised in 5 chapters. It begins with an introductory chapter that describes the scope, objectives and structure of the thesis.

The second chapter focuses on the GI through the Internet. The interoperability problem, the OGC and its Web Services (WMS and WFS) are addressed. The Geography Markup Language (GML) and the transaction operation of the WFS are also covered.

In the third chapter, emerges the framework definition for the prototype implementation. In this chapter, the system architecture and requirements are discussed. The results of some software tests are also presented. These tests aim at verifying which software is prepared to deal with the WFS standard.

The fourth chapter describes the prototype implementation, based on the conclusions of chapter 3. It is at this stage that the application is implemented.

In chapter five, final considerations are drawn. Results are presented and some advantages and limitations of the implemented solution are highlighted. Finally, main conclusions and future work are presented and discussed.

2. GI Thru the Internet

2.1. Introduction

In this chapter, an overview of the GI thru the internet, the interoperability as well as the OGC, the WMS, the Web Feature Service and the WFS-T are presented.

2.2. Overview

Internet GIS has been the most rapidly developing field in the geospatial technology industry (Peng and Zhang, 2004).

GI uses today the WWW to disseminate its content. There are, at the moment, many servers around the world providing spatial information to several users, and these accesses this information through different GIS applications also. Consequently, a problem arises: compatibility between different applications.

This chapter introduces the interoperability problem, the OGC and some of the most important Web Services (WS) – The WMS and the WFS, the Geography Markup Language (GML) and the WFS-Transaction.

2.3. Interoperability and OGC

In the transition from isolated Desktop GIS to Distributed GIS over the network, some issues related with interoperability appeared. In fact, the first attempts to develop systems to share information were mostly based on proprietary technologies. This created a problem to users that became restricted to their own proprietary applications and could not access or share other users' data remotely that used different technologies. In the same way, the general public or small organizations with resource constraints were limited in obtaining desktop GIS software and training. Additionally, even if they used only some of the functions provided by the software, the users had to purchase the full desktop GIS (Peng and Zhang, 2004).

To deal with heterogeneous and distributed data, the applications must be able to locate, access, interpret and process the retrieved data, and the solution is not easy to achieve. There are lots of incompatibilities in data formats, software products, spatial conceptions, quality standards and models of the world (Vckovsky, 1998). This problem is known as the interoperability problem.

Interoperability refers in general to the ability of various autonomous systems to bring together parts and to operate in collaboration. In most cases this means the exchange of meaningful information (Vckovski, 1998).

According to Hecht referred by Geoffrey and Rafael (2003), there are some reasons that make interoperability desirable:

- Communication between information providers and final users without requiring that both have the same software;
- Combine accurate and up-to-date data from multiple sources opens new possibilities for improved decision making and it also makes data more valuable;
- The possibility of having multiple users, including non-GIS experts, using a particular set of data also makes the data more valuable.

For Limp referred by Peng and Zhang (2004), the OGC Web Service specifications offer a standard way for users to search for maps and geoprocessing sources over the Web from different map servers and different vendors, solving thus the interoperability problem.

2.4. Web Services

Web services are online computing processes provided remotely, using the network (Fälman, 2004), that ensure the easy exchange of data and information between different applications, using open formats (Furtado, 2006).

Beyond these, another great advantage of using Web services is related with data that is constantly changing. If a user downloads that data, it rapidly becomes out-of-date. But if he accesses data using Web services, he will have the warranty that he has the data updated and in real-time (Tyler, 2005).

The Web services framework is based on three elements: the server, the client and the catalogue, as shown in figure 2.

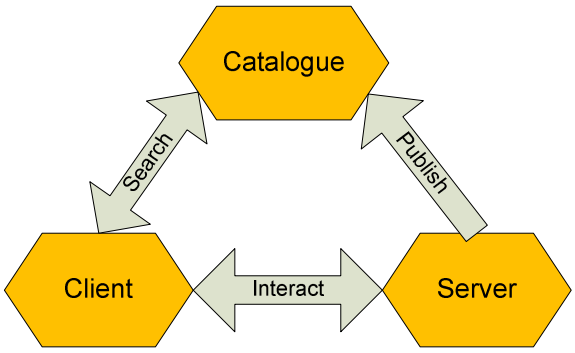

Figure 2 - Web services framework (Source: Whiteside (2005))

The server provides the service over the network. Its capabilities and localization are stored in the catalogue and the client is the user and/or application that uses the service.

In order to use this architecture, three important activities must take place: publish, search and interact. In detail, these activities are (Kreger, 2001):

a) Publish – a service description must be published in the catalogue so that can be searched by the client or requester.
b) Search – before using a service, the client must search a service description in the catalogue.
c) Interact – at this stage, the client interacts with the service at runtime, using their description obtained in the Search stage.

Moreover, these elements need protocols to communicate. HTTP is the omnipresent protocol to establish communications over the Internet, supported by

almost all internet applications, and XML (eXtensible Markup Language) which it is a standard defined by W3C[2] used to represent data. These are the basics to communicate over the Internet, but in concerning to GIS Web services, they are not sufficient. Thus, there are more three important protocols defined by the W3C: SOAP (Simple Object Access Protocol), WSDL (Web Service Definition Language) and UDDI (Universal Description, Discovery and Integration). SOAP is described below in chapter 2.4.2. WSDL is an XML-based language that provides a model for describing Web services. UDDI is a platform-independent, XML-based registry for businesses worldwide to list themselves on the Internet. UDDI is designed to be interrogated by SOAP messages and to provide access to WSDL documents describing the protocol bindings and message formats required to interact with the web services listed in its directory. The figure 3 shows the connection between all these protocols (Peng and Tsou, 2003).

[2] The World Wide Web Consortium (W3C) is an international community where Member organizations, a full-time staff, and the public work together to develop Web standards (W3C. (2010). "World Wide Web Consortium." Retrieved 14-06-2010, from http://www.w3.org/Consortium/.

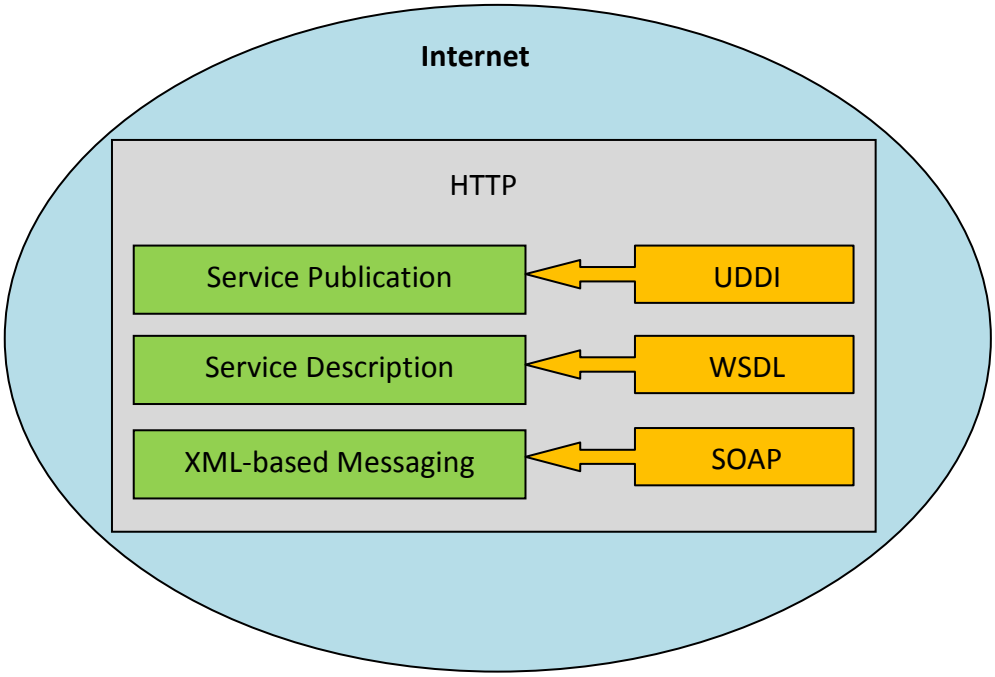

Figure 3 - Communication protocols (adapted from Kreger (2001))

In the next sections two of the most important OGC standards are described: the WMS and the WFS standards. The latter is described with higher detail because it is the one that this thesis addresses.

2.4.1. Web Map Service[3]

The WMS Implementation Specification is one of the most famous OGC specifications (Fälman, 2004). It defines the rules to request some geographic layers of an area of interest, and obtains, as result, one or more geo-registered map images (returned as JPEG, PNG, etc.) that can be displayed in a browser application (Beaujardiere, 2006).

The use of the WMS specification encompasses three operations: *GetCapabilities*, *GetMap* and *GetFeatureInfo*.

[3] Based on the WMS Implementation Specification (Beaujardiere, 2006). OpenGIS Web Map Service Implementation Specification, Version 1.3.0.

The *GetCapabilities* operation is mandatory and gives the service metadata in XML format. It is through this operation that the user, or the application, knows what kind of information is available and the supported operations. It is divided in two processes: the interrogation by the client and the response by the server. The first contains the parameters presented in table 1

Attributes	Type	Description
VERSION	optional	Specifies the version
SERVICE	mandatory	Specifies the service that is being requested (*WMS* for example)
REQUEST	mandatory	Indicates the operation to be requested, que neste (*GetCapabilities* in this case)
FORMAT	optional	Specifies the information format
UPDATESEQUENCE	optional	Keeps information between the client and the server about the service updated (new maps added)

Table 1 - Interrogation process of the GetCapabilities operation

The answer given by the server, in XML format, can be decomposed into two main elements: the service and the capabilities, i.e. the information about the supported operations and the available themes.

The service element is composed by the components shown in table 2.

Attributes	Description
NAME	Service name
TITLE	Service title
ABSTRACT	Service detailed description
KEYWORDLIST	Keywords for possible cataloguing and search
ONLINERESOURCE	Service provider site
CONTACTINFORMATION	Service responsible for contact information
FEES	Indicates whether there is any cost associated with the use of the service
ACCESSCONSTRAINTS	Points out possible restrictions in using or accessing the service

Table 2 - Service element of the *GetCapabilities* operation answer

The Capabilities element consists of the components presented in table 3.

Attributes	Description	
REQUEST	Operations list provided by the service	
EXCEPTION	Specifies the way an error is reported. XML is the default format	
LAYER	Gives information about the supported themes. This is one of the most important components of the *Capabilities* element and can be divided into the following sub-components:	
	TITLE	Title
	NAME	Name
	ABSTRACT	Summary of the layer characteristic resume
	KEYWORDLIST	Keywords list
	STYLE	Style invocation
	SRS	Map coordinate system
	LATLONBOUNDINGBOX	Describes the minimum limiting rectangle of the map, in the EPSG 4326 geographic coordinate system (WGS84)
	BOUNDINGBOX	Sets the rectangle limits of the map presented in a specific coordinate system, defined in the SRS element
	SCALEHINT	Minimum and maximum scale appropriated to represent the layer
	DIMENSION AND EXTENT	Multidimensional metadata
	METADATA URL	Detailed and normalized metadata about the layer
	ATTRIBUTION	Allows the source map identification, used in the layer or collection of layers
	IDENTIFIER AND AutorityURL	Identifiers defined by external entities. The AutorityURL parameter indicates the URL where are defined the identifiers values
	FeatureListURL	List of entities represented in the *layer*
	DataURL	Additional information about the represented data in a specific layer
	LayerAttributes	Layer attributes (parameters: *Queryable*, *Cascaded*, *Opaque*, *NoSubsets*, *FixedWidth*, *FixedHeight*)

Table 3 - Capabilities element of the *GetCapabilities* operation answer

The *GetMap* operation returns a map, in the requested image format, according to specifications provided by the applicant (user or client application). The main attributes of a *GetMap* operation are described in table 4.

Attributes	Type	Description
VERSION	mandatory	Service version
REQUEST	mandatory	Specification of the requested service, that in this case is WMS
LAYERS	mandatory	List of themes to provide for this request
STYLES	mandatory	List of styles to represent the requests themes
SRS	mandatory	Defines the coordinate system applied to the defined values in the BBOX element
BBOX	mandatory	Allows to request the view area of the map
FORMAT	mandatory	Defines the image format to be returned the map (JPEG, GIF, PNG)
WIDTH and HEIGHT	mandatory	Specifies the size of the map image to be produced, in pixels
TRANSPARENT	optional	Show the image map with some transparency
BGCOLOR	optional	Background colour of the image map
EXCEPTIONS	optional	Defines the way the errors are returned to the requester
TIME	optional	Value used when the time component is used
ELEVATION	optional	Refers to the altitude of the elements of a map

Table 4 - GetMap operation attributes

The *GetFeatureInfo* operation (optional implementation) returns more information about specific entities shown in a map than the GetMap operation, and it is only supported for those Layers for which the attribute queryable="1" (true) has been defined or inherited.

Since the protocol is static, this operation must include most of the components of the *GetMap* operation (except the VERSION and REQUEST components). Through the components of the operation *GetMap* (BBOX, SRS, WIDTH, HEIGHT) and the X and Y coordinates of the specified point, the WMS server returns information about that point.

The main components of this operation are described in table 5.

Attributes	Type	Description
VERSION	mandatory	Service version
REQUEST	mandatory	Specify the requested service, in this case - the WMS
Map_request_part	mandatory	Some of the parameters defined in the GetMap operation are repeated here
QUERY_LAYERS	mandatory	List of *layers* on which more information is wanted
INFO_FORMAT	optional	Specify the format to be used in the answer to this order
FEATURE_COUNT	optional	Refers the maximum number of entities on which more information is requested
I,J	mandatory	Specifies the point of interest in the map, in *WIDTH* and *HEIGHT* format
EXCEPTIONS	optional	Defines the way the errors are returned to the requester

Table 5 - GetFeatureInfo attributes

2.4.2. Web Feature Service[4]

The WFS specification (such as the WMS) provides the rules to request some geographic layers of an area of interest, but the response consists of the geographic features themselves as vector data. This specification defines interfaces and operations for data access and manipulation on a set of geographic features, including: to Get or Query features based on spatial and non-spatial constraints; to Create a new feature instance; to Get a description of the properties of features; to Delete a feature instance; to Update a feature instance, and to Lock a feature instance. This allows, for example, features to be edited remotely (Vretanos, 2005). These last operations (to Create, to Delete, to Update and to Lock) are part of an extended set of WFS – the WFS-T.

It is also important to note that the WFS uses a XML based language called GML to describe geographic features, which is also an OGC specification (Sælid, 2006).

[4] Based on the Web Feature Service Implementation Specification (Vretanos, P. (2005). Web Feature Service Implementation Specification, Version 1.1.0.

The WFS defines some operations as follows:

- GetCapabilities;
- DescribeFeatureType;
- GetFeature;
- GetGmlObject;
- Transaction;
- LockFeature.

There are some basic elements that are independent of particular operations or are common to several operations or interfaces, like the version numbering and negotiation, the HTTP request and response rules, the request encoding, the namespaces and the SOAP. There are also some common elements like the feature and element identifiers, the feature state, the property names, the property references, the <native element>, the filter, the exception reporting, and the common XML attributes.

Version numbering and negotiation

The version number is composed by three positive integers in the form “x.y.z”, and each OGC Web Service (OWS) Specification is numbered independently. This number appears in the capabilities XML describing a service and in the client requests to that service as a parameter. When the version number in a client request is not equal to the number declared in the capabilities XML, then occurs a negotiation in mutual agreement by both the server and the client as follows (Vretanos, 2005):

1. *If the server implements the requested version number, the server must send that version.*
2. *If the client request is for an unknown version greater than the lowest version that the server understands, the server must send the highest version less than the requested version.*

3. *If the client request is for a version lower than any of those known to the server, then the server must send the lowest version it knows.*
4. *If the client does not understand the new version number sent by the server, it may either cease communicating with the server or send a new request with a new version number that the client does understand, but which is less than that sent by the server (if the server had responded with a lower version).*
5. *If the server had responded with a higher version (because the request was for a version lower than any known to the server), and the client does not understand the proposed higher version, then the client may send a new request with a version number higher than that sent by the server.*

If the version is omitted in the request, the server sends the highest version it understands and the negotiation ends when a mutual understanding is reached or when the client determines that it cannot communicate with that server.

HTTP request and response rules

WWW is the only distributed computing platform explicitly supported by the OGC Web Services, or more specifically, Internet hosts implementing the HTTP. The online resource of each operation supported by a service instance is located through the HTTP Uniform Resource Locator (URL).

HTTP supports tow request methods: GET and POST. The first is a URL prefix to which additional parameters must be appended and construct a valid operation request like shown in table 6.

URL component	description
http://host[:port]/path?	URL prefix (mandatory)
name=value&	Additional parameters (optional)

Table 6 - General OGC Web Service request

The HTTP POST is a complete and valid URL to witch clients can transmit their requests through a POST document.

A service provider may offer Web feature services using HTTPS that is HTTP over a secure communication channel. This secure channel allows the encryption of the information transferred between machines over the WWW.

To be considered valid a request, the server should send a response with exactly it was requested by the client. Only in the case of version negotiation, described above, the server may respond with a different result.

Request encoding

The HTTP GET method uses Keyword-value pairs (KVP) to encode WFS requests and the HTTP POST method uses XML as the encoding language. A KVP is in the format "Keyword=Value", like for example "REQUEST=GetCapabilities". In table 7 is shown the WFS operation and the respective request encoding.

Operation	Request Encoding
GetCapabilities	XML and KVP
DescribeFeatureType	XML and KVP
GetGmlObject	XML and KVP
LookFeature	XML and KVP
Transaction	XML and limited KVP

Table 7 - Operation Request Encoding

Namespaces

Namespaces intend to differentiate XML vocabularies from one another. In the case of WFS there are three normative namespace definitions, the WFS interface vocabulary, the GML vocabulary and the OGC filter vocabulary.

Simple object access protocol (SOAP)

The SOAP is a protocol used for communications between applications, in this case for communications between a client and a WFS server using the HTTP POST method. SOAP is also platform and language independent and SOAP messages are encoded using XML, providing a way to communicate under different operating systems, with different technologies and programming languages.

The request and response to a WFS can occur using a body of a SOAP envelope using the XML tag <**soap:body**>.

When an exception is encountered, the WFS response is a SOAP message where the content of a <**soap:body**> element is a <**soap:Fault**> element.

Feature and Element Identifiers

These identifiers are fundamental to make possible the use of operations over features served by a WFS. Every feature instance in a WFS has its own unique identifier, that can be used to reference repeatedly. In the same way, elements have also its own unique identifier, used to refer them.

Feature state and property names

The feature state is imperative to use the transactional WFS. When the client wants to insert, update or delete a feature instance, it is important to know the state of that feature instance, which is constituted by the values of all properties of that feature instance.

Since the feature state must be expressed in GML and thus XML, they should be valid according with the XML 1.0 specification.

Property references

While GML allows geographic features to have complex or aggregate non-geometric properties, a WFS should use XML Path (XPath) expressions to referencing the

properties from the various places where they are required (e.g. query and filter expressions).

XPath Language specification is a language used, in the case of the WFS specification, for referencing XML elements and attributes within the description of a feature, requiring thus only the support of a subset and not the full XPath language.

The <Native> element

The <Native> element was created since the number of capabilities supported by an open interface is limited. This element gives the possibility to vendors create their own specific capabilities of any particular WFS or datastore. It is composed by some attributes of which is highlighted the **safeToIgnore**, that can assume tow values, True or False. A value of False indicates that this element cannot be ignored and consequently if the WFS cannot deal with it, must fail. A value of True indicates that this element can be safely ignored.

Filter

The filter element is used to define a set of feature instances to operate. This filter can be based on one or more enumerated features or defined through spatial and non-spatial constraints.

Exception reporting

When an error is encountered in a WFS request, the server sends a XML document reporting this error containing one or more WFS processing exceptions.

Common XML attributes

All XML encoded WFS request includes some mandatory attributes such as the version and service. The version specifies the version of the WFS request and used

in version negotiation. The service indicates which of the available services is invoked, at a particular service instance, which in this case is WFS.

2.4.2.1. WFS operations

WFS operations are detailed here such as the **DescribeFeatureType** operation, the **GetFeature** operation, the **GetGmlObject** operation, the **LockFeature** operation, the Transaction operation and the **GeCapabilities** operation. All these operations require a request by the user and a response by the server.

DescribeFeatureType operation

The **DescribeFeatureType** operation has the intent to generate a schema description of feature types serviced by a WFS implementation. With this schema, the client get to know how encode feature instances on input and the expected output.

The request of this operation is a XML schema with some elements like **TypeName** and **outputFormat**. The **TypeName** element encodes the names of feature types to be described. The **outputFormat** element is used to indicate the schema language to be used in the description of the features requested. The default value, in this case, indicates that a GML application schema, using XML schema, should be generated. When the **DescribeFeatureType** element is empty should be interpreted as requesting the description of all feature types serviced by the WFS.

The response should be, in the case of an output format set to GML, an XML schema document that is a valid GML application schema and defines the schema of the feature types listed in the request. As an XML schema document can only describe elements that belong to a single namespace, WFS may generate an XML schema document that imports the schemas of the features from the various namespaces in the request.

GetFeature operation

This operation allows retrieval of features from a WFS. When the value of the **outputFormat** attribute is set to **text/gml; subtype=gml/3.1.1**, a GML3 instance document is returned to the client, with the result.

The request can contain one or more query elements, an **outputFormat** attribute (optional) and a **resultType** attribute (optional). The output format is similar to that described in the **DescribeFeatureType** operation. The resultType can assume two values to control the way a WFS responds to a request, as described in table 8. The other attributes are shown in table 9.

resultType value	**Description**
Results (default value)	A complete response is generated containing all the features requested.
Hits	The response only contains the number of feature instances of the requested feature types.

Table 8 - Values for resultType attribute

Attribute	**Type**	**Description**
maxFeatures	Optional	Limit the number of requested features in a WFS response
typeName	Mandatory	Indicate the name of one or more feature type instances or class instances to be queried.
featureVersion	Optional	It is included in the **<Query>** element in order to support systems that support feature versioning
srsName	Optional	It is included in the **<Query>** element in order to specify a specific reference system to be used in returned feature geometries.
traverseXlinkDeph	Optional	Indicates the depth to which nested property XLink linking element locator attribute XLinks in all properties to the selected features are traversed and resolved if possible.
traverseXlinkExpiry	Optional	Expressed in minutes, Indicates how long a WFS should wait to receive a response to a nested **GetGmlObject** request.

Table 9 – Some attributes of the GetFeature operation

The response to a **GetFeature** operation request has its format controlled by the **outputFormat** attribute, like described in the first paragraph of this chapter. The root element of this response is the **<wfs:FeatureCollection>** element, and is dependent of the **resultType** attribute value, like described in table 8.

GetGmlObject operation

This optional operation allows a request of features served by a WFS, through an ID. In the **GetCapabilities** operation, described in chapter 0, the WFS support to the **GetGmlObject** operation has to be advertised, otherwise, this operation is not allowed.

The request is made using an object ID and contains exactly one object ID. This value is used as a unique key that identifies the element at the server and could be any identified element such as a feature, a geometry, a topology, or a complex attribute. Once more, the **outputFormat** attribute defines the format to use to generate the result set,

The response to a **GetGmlObject** request is the referenced GML element returned as an XML document fragment, and differs from a response to a **GetFeature** request that returns the complete document.

LockFeature operation

This operation is very important in transactional WFS and is related to a mechanism to guarantee that when someone is modifying a feature in the client side, no other client can update that feature at the same time in the database. Without this mechanism, the update process would not be reliable with the possibility of information loss.

The request is composed of the **<LockFeature>** element that can contains one or more **<Lock>** elements defining locking operations on feature instances of one feature type per element. The **expiry** attribute, specified in minutes, is used to

define how long a WFS should hold a lock. The **<Lock>** element contains a single **<Filter>** element that defines the set of feature instances of the specified feature type to be locked. The optional **lockAction** attribute controls how feature locks are acquired and can assume tow values, as described on table 10.

Value	Action
ALL (default)	All requested feature instances as to be locked, otherwise the operations fails.
SOME	The WFS should attempt to lock the maximum of requested feature instances

Table 10 - lookAction operation values

In figure 4 is presented a diagram to better understand the lock feature process.

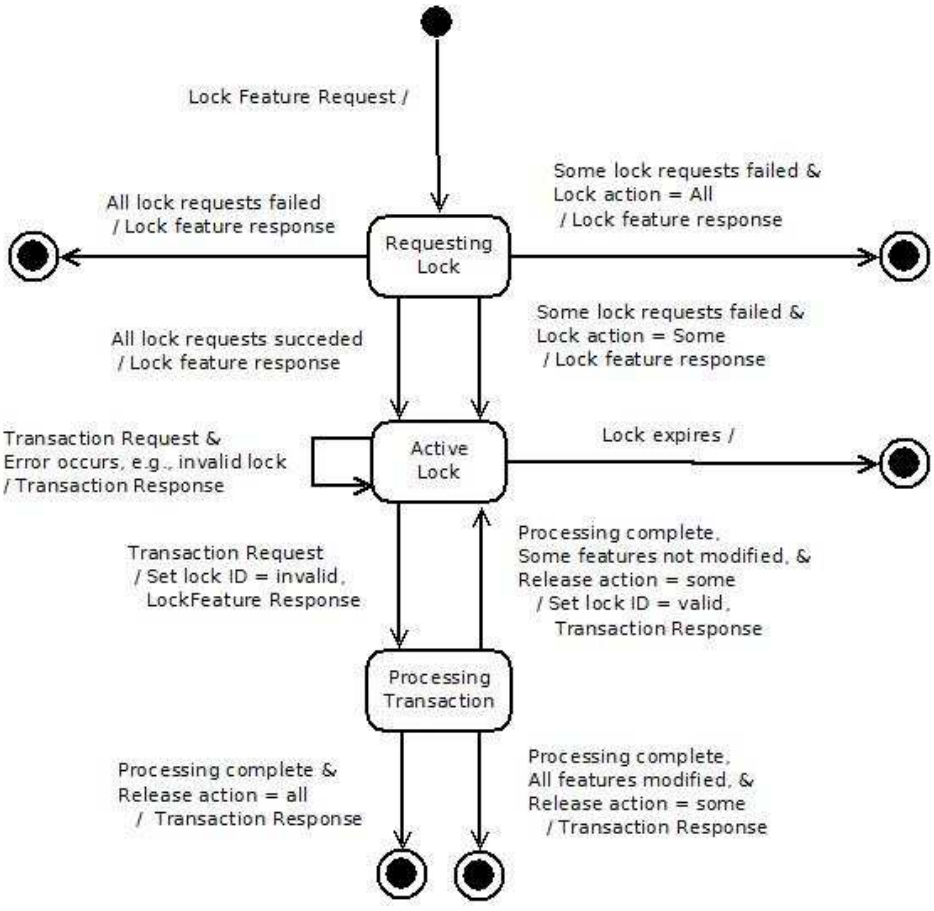

Figure 4 - State diagram for a WFS lock (Source: Vretanos (2005))

The response to a **LockFeature** operation should be an XML file, generated by the server, containing the lock identifier to use in subsequent WFS operations regarding

locked feature instances. This file could contain also the optional elements described in table 11 depending on the value of the **lockAction** attribute. For the case that a WFS do not implements the **LockFeature** operation, the server should generate an exception with the description of the error.

Element	lockAction value	Description
FeaturesLocked	ALL	If all requested features can be locked, the element should contain that information. If some requested features could not be locked, the server should respond with an exception.
	SOME	The element should list all the features locked
FeaturesNotLocked	ALL	Element not present
	SOME	The element should list all the features not locked

Table 11 - Optional elements of a LockFeature operation

Transaction operation

The **Transaction** operation is used to inform, when transformation operations applied to the feature instances are completed, about their completion status, through a XML response by the server.

The request contains various elements, between them the **<Transaction>** element that may contain zero or more **<Insert>**, **<Update>**, or **<Delete>** elements. The **<Insert>** element, that is used to create new feature instances, has an **idgen** attribute described in table 12. The **<Update>** element describes a modification that it is to be applied to a feature or a set of features of a single feature type. This element should have one or more Property elements, which are to be modified, followed by the replacement values. It can have also a **<Filter>** element to describe the scope of the **<Update>** element. The **<Delete>** element is used to delete one or more feature instances. It can contain also a **<Filter>** element to define the scope of the **<Delete>** element.

idgen value	Action
GenerateNew (default)	Unique identifiers of all newly inserted features shall be generated.
UseExisting	gml:id attribute values should be used on inserted features and elements. If any of those IDs duplicates the ID of a feature or element already stored in the WFS, the WFS shall raise an exception.
ReplaceDuplicate	When it is intended to replace existing values of feature elements that duplicate the ID of a feature or element already stored in the WFS.

Table 12 - Values of the attribute idgen

The response should be an XML document with the termination status of the transaction. For the case of the transaction request includes **<Insert>** operations, the response must report the identifiers of the entire features created. The **<TransactionResponse>** element can contain a **<TransactionSummary>** element, an optional **<TransactionResult>** element and an optional **<InsertResults>** operation as shown in table 13.

Element	Sub-element	Description
TransactionSummary	**totalInserted**	Contains a count of the number of new feature instances created.
	totalUpdated	Contains a count of the total number of features updated.
	totalDeleted	Contains a count of the number of feature instances deleted.
TransactionResult (optiona)**l**		Indicate witch actions of a transaction request failed to complete successfully. If it is not present, then all the actions of a transaction has been completed successfully.
InsertResults (optional)		It is used when a transaction contains insert actions, and indicates the feature identifiers of newly created feature instances.

Table 13 - Description of some elements of a Transaction operation response

When an error is encountered, the WFS should generate an exception.

GetCapabilities operation

Like the WMS, the WFS also must have the capacity to describe its capabilities, which is done through a **GetCapabilities** request.

The request is composed by a **<GetCapabilities>** element asking for a capabilities document from a Web feature service and the response is an XML document containing a **<WFS_Capabilities>** element divided into the follow seven sections:

1. Service Identification section – Information about the service itself.
2. Service Provider section – Information about the organization that provides the WFS server
3. Operation Metadata section – Metadata about the operations defined in the specification and implemented by the WFS server.
4. FeatureType list section – A list of feature types, and transaction and query operations on each feature type, that are available from the WFS.
5. ServesGMLObjectType list section – A list of GML object typesavailable from the WFS if it supports GetGMLObject operations.
6. SupportsGMLObjectType list section – List of GML Object Types that the WFS is capable of serving.
7. Filter capabilities section – defines the schema of the filter capabilities.

2.5. Conclusions

The future is inseparable from the Internet and, consequently, from distributed services, that are gaining territory to the isolated Desktop GIS. The distributed services are also inseparable from the interoperability concept to become more and more usable by everyone.

The standards detailed in this chapter, WMS and WFS, are a way to solve the interoperability problem, each of them in its applicability, i.e. WMS in retrieving data in image format and WFS in retrieving features in vector format.

This chapter demonstrates the importance of the OGC and its standards, particularly the WMS and WFS, and it is important to define the framework to a WFS-T implementation, presented in chapter 3.

3. Framework definition

3.1. Introduction

There is a vast number of software that handles GI - some are for desktop use, others for mobile use, and others act like servers.

For purpose of implementing the prototype, and according to the dissertation objective, it is necessary to select software that implements the WFS-T OGC Specification, both for server and client.

Thus, a server software that handles transactional operations via WFS is needed and also a client application that implements tools to deal with these operations.

In this chapter the system architecture and requirements are discussed, based on the results of some software tests and research, which aims at verifying what software is prepared to deal with the WFS-T standard.

3.2. System Architecture

Given the fact that the author of this work is more familiarized with Windows from Microsoft, all the system architecture discussed here is based on this operating system.

It is important to identify a system that implements the WFS-T OGC Specification. For this, a server and a client, at least, are needed, but it is also possible to add a database server with spatial capabilities and/or provide tools for Web access. To be able to access through a Web browser, an application that provides the tools to put the map in an HTML page or other Web language and a Web server to make available it over the WWW are necessary.

In figure 5 it is possible examine a proposal for the System Architecture, where are three main sections: the Data section, the Server section and the Client section, and the way they communicate with each other.

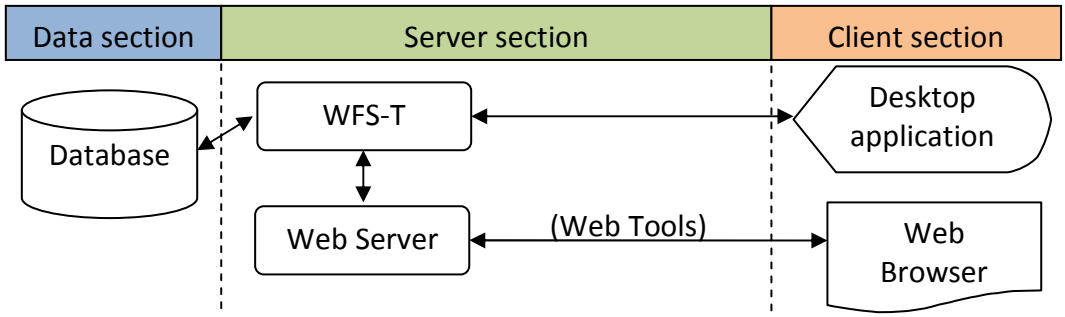

Figure 5 - System architecture

The adoption of Web Tools allows the use of simple Web browsers instead of desktop applications (Open Source, commercial, or even other type of applications), with the possibility of accessing to it anywhere - through an internet connection and an installed Web browser.

3.3. Available Software

To implement the proposed system architecture, software compatible with the OGC specifications is needed for all sections: Data, Server and Client sections. In the next sections, software possibilities are presented and analysed by section and the best set that fits the defined architecture will be selected.

3.3.1. Data section

To store the data there are a few possibilities, like the Shapefile from ESRI or some database formats. By recurring to a database, it is possible to use structured data with all the inherent advantages (Paul A. Longley, 2005):

- Redundancy reduction;
- Integrity maintenance;
- Better organization with maintenance costs and data manipulation reduction;
- Establishment and enforcement of security and standards for data and data access, among many others.

Available database software includes:

- **PostgreSQL / PostGIS** (PostgreSQL (2009) / Refractions (2009)) – PostgreSQL is an Open Source object-relational database system widely used by many organizations all over the world. Its development has started at the University of California and, at the moment, by a network of contributors. PostGIS adds support for geographic objects to the PostgreSQL and has been developed by Refractions Research as a project in Open Source spatial database technology. This is one of the most used database systems in the GI domain (Salinas and Lajara, 2009).
- **MySQL** – MySQL is Open Source database software that has a spatial extension to deal with GI. Paul Ramsey wrote, in June 2009, an article comparing this Open Source database software with PostgreSQL and have reached interesting results, i.e. "the amount of functionality in MySQL Spatial remains very very small" (Ramsey, 2009);
- **Oracle Spatial** – Commercial database software with a spatial extension to deal with GI by Oracle Corporation;
- **SQL Server 2008** - Commercial database software with a spatial extension to deal with GI by Microsoft Corporation.

3.3.2. Server section

Servers are an important piece of client-server systems. They provide data based on standards, which are a set of rules to communicate and transfer between both sides. At this section, we have the WFS-T server and the Web server.

3.3.2.1. WFS-T Server

It is one of the most important parts of the system, since it has to implement not only the WFS specification, but also allows the transaction operations. It is at this level that the Web Service is provided.

Some existent server software implements only the WFS specification, without Transaction operation capabilities.

Available software with WFS Server includes:

- **GeoServer** (Geoserver, 2009) - GeoServer is an Open Source software server written in Java in a contributor environment, with the financial support of various institutions. This software implements the WFS-T OGC Specification;
- **ArcGIS Server** (ESRI, 2009) – This software does not implement the WFS-T OGC Specification. It only implements the WFS OGC Specification with no support to transaction operations;
- **Map Server** (MapServer, 2009) – Like ArcGIS Server, this software does not implement the WFS-T OGC Specification, but only the WFS specification without transactional support.

3.3.2.2. Web Server

A Web Server is fundamental to make data or Web tools available in the WWW through Web applications. Without it, there are no possibilities to work with GI, or other kind of data, using a simple Web browser.

One of the most used Web server in the world is the Apache Web Server (Apache, 2009), with the advantage of being Open Source software. The Microsoft Web Server, called Internet Information Services (IIS), is also available. These are two of the most Web Servers used all over the world.

3.3.2.3. Web Tools

As previously mentioned, this part of the system is only necessary when the GI is handled through the Web. The access and/or manipulation of data through a Web browser has various advantages, such as the platform or application independence, but it requires some Web development to provide the necessary tools to deal with geographic data.

The available software toolkits for Web tools development are:

- **OpenLayers** (OpenLayers, 2009a) – It is an Open Source map viewing library and makes it easy to incorporate maps from a variety of sources into a Webpage;
- **ka-MAP** (ka-MAP, 2009) – It is also an Open Source project that aims at providing an API for developing highly interactive Web-mapping interfaces using features available in modern Web browsers. This software is developed to work with Map Server;
- **Mapfish client** (MapFish, 2009) – Based on Open Layers, it supports the latest Web 2.0 technology, including the OpenLayers, ExtJS and GeoExt JavaScript toolkits;
- **MapBender** (MapBender, 2009) – It is an Open Source Geospatial Foundation project for geodata management of OGC OWS architectures.

3.3.3. Client section

This section represents the client side of the prototype. The client can access the data through a Web application via WWW or a desktop application. The first option uses Web development based on Web tools presented in section 3.3.2.3 and the second uses an available desktop application from those presented below in section 3.3.3.1.

3.3.3.1. Desktop Applications

The desktop applications have a great advantage: every tool needed to manipulate the features is already in the application and further developments are not needed, but eventually some customizations can improve their use.

Some of the available software applications are:

- **gvSIG** (gvSIG, 2009) – Open Source software that implements the WFS OGC Specification. The tools to manipulate GI are only available through an extension that is, at the moment, under development;
- **uDig** (uDig, 2009) - Open Source software that implements the WFS OGC Specification with transaction operations support. The goal of this project is to bring internet mapping technologies such as WMS and WFS transparently to ordinary GIS users desktops (Ramsey, 2007);
- **Kosmo** (Kosmo, 2009) – Open Source software that does not implement the WFS OGC Specification;
- **OpenJump** (OpenJump, 2009) – Open Source software that also does not implement the WFS OGC Specification;
- **QuantumGIS** (Quantum GIS, 2009) – Open Source software that implements the WFS OGC Specification, but does not have support to transaction operations;
- **ArcGIS** (ESRI, 2009) – Proprietary software that implements the WFS OGC Specification through an extension called Interoperability. It does not have support to transaction operations;
- **Cadcorp SIS** (Cadcorp, 2009) – Proprietary software that implements the WFS OGC Specification without support to transaction operations;
- **Geomedia** (Intergraph, 2009) – Proprietary software that implements the WFS OGC Specification without support to transaction operations;
- **Supermap** (SuperMap, 2009) – Proprietary software that implements the WFS OGC Specification without support to transaction operations.

3.4. Selected software

The starting point to select the software that implements the defined system architecture is the WFS-T server section. This is the key section to have a functional implementation because if it does not support this OGC Specification, then it does

not serve the purposes of the framework. Since the GeoServer is the only software that fully implements this standard, it was the first software selected to make part of the system. The chosen version was the last one available at the moment of the writing of this thesis - the 1.7.4 version.

The Apache software was the choice for the Web server because it is Open Source and one of the most used in the world, with very good performance and stability. To facilitate its installation, it was decided to use an “easy to install” Apache distribution called XAMPP (XAMPP, 2009).

After this, the decision was to use the PostgreSQL/PostGIS for the database section, since the combination of GeoServer with PostgreSQL/PostGIS is widely used with also very keen results.

For the Client section, it was decided to use both Web tools and desktop software to improve the prototype: Mapfish client for the Web tools and uDig for the desktop. The Mapfish because it is a set of java libraries with strong graphical skills and WFS-T standard implementation, and the uDig because it is the only desktop software that implements the WFS OGC Specification, with tools and support to transactional operations.

Summarizing, the chosen software for the entire prototype is presented in the table 14 and figure 6.

Section	Type	Name
Data section	Data base	PostgreSQL (v8.3)/PostGIS (v1.4.0)
Server Section	WFS-T server	GeoServer (v1.7.4)
Server Section	Web server	XAMPP (v1.7.2) containing Apache (v2.2)
Server Section	Web tools	MapFish (v1.2) including OpenLayers (developer version)
Client section	Desktop application	UDig (v1.2-RC3)
Client section	Web browser	Mozilla Firefox

Table 14 - Chosen software for the prototype implementation

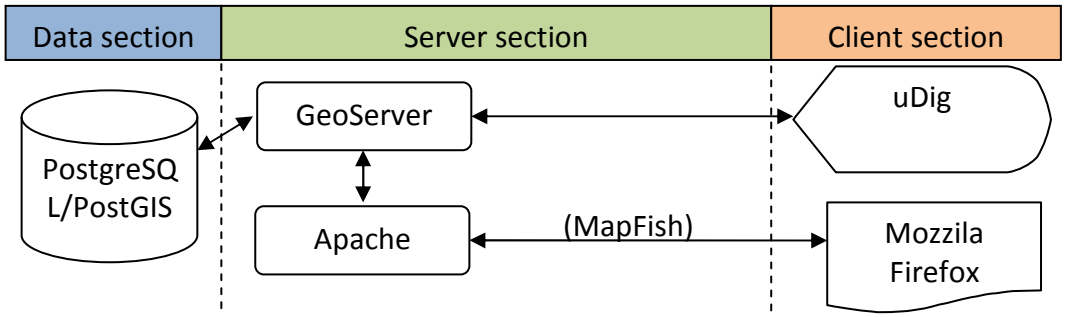

Figure 6 - System architecture with selected software

3.5. FOSS vs. Proprietary Software

Like written in the previous section, all the selected software is FOSS since no proprietary software implements the WFS-T standard, which leads us to think about that. One of the ways to try to understand this phenomenon is to take a look at the advantages and disadvantages of proprietary and FOS software, presented by Steiniger and Bocher (2009), adapted from Weis (2006), in table 15.

Proprietary software is based on a model where the source code is closed. The FOSS model is based on an opposite way, and permits allow users to modify a product's source code (Wikipedia, 2010).

FOSS implies free software that is *a matter of the users' freedom to run, copy, distribute, study, change and improve the software* (GNU, 2010).

	Free & Open source Software	Proprietary software
Advantages	• no licence fees • unrestricted use (e.g. no limits for the number of • installations) • no update enforcement • support of open standards • support usually available from several providers • customisation at API level	• warranty of developing company on product • components should work together • usually well documented software
Disadvantages	• installation know-how necessary • training costs	• software price and maintenance fees • training costs • maintenance tied to specific licensed companies • customised development can be difficult due to • available resources of vendors • support only as long as software company exists

Table 15 - Advantages and disadvantages of proprietary and FOS software, presented by Steiniger and Bocher (2009) adapted from Weis (2006).

3.6. Conclusions

As it is described in this chapter, there is a considerable amount of software that deals with GI, but only few implement the WFS-T Specification. Among these, only some implements the Transaction mechanism at the server level, as well as at the client side level.

It is also clear the importance of the development based on standards to warrant the interoperability between different applications, especially open standards like those developed by OGC.

4. Prototype implementation

In this chapter, the prototype implementation is described, based on the selected software presented in chapter 3. It begins with the software download and installation and their configuration, where the developed application is also described.

4.1. Introduction

This prototype aims at registering the existent dumps in Portugal, enabling the user to use a web or a desktop application, through the WFS-T OGC standard.

After the software definition, it is necessary to download, install and configure the system software so that it works properly. The Web application development is also required.

It is important to note that all the downloaded installation packages should be selected for windows Operating Systems (OS) because this prototype implementation is based on that OS. It is also important to refer that all these software packages are also available for Linux and others OS.

4.2. Software download and installation

The first thing to do is to download the software defined in the system architecture (chapter 3.2). This can be done through the Web pages of each software once they all are FOSS with this possibility. After that, it is necessary to install each of them, following the installation instructions that are available, normally at their home pages or in the installation package.

The installation does not have to follow any sequence, but the MapFish client should be installed after the GeoServer, since it has a folder that has to be placed inside the GeoServer installation folder.

4.2.1. XAMPP

The “easy to install” Apache distribution was downloaded from the XAMPP project Web page, choosing its windows distribution. The installation was made by using the EXE installer that was included, and according to the installation instructions.

4.2.2. PostgreSQL/PostGIS

On the PostgreSQL Web page, the windows installer package was downloaded and installed. This package installs i) a precompiled version of PostgreSQL along with pgAdmin (a graphical administrator that makes the PostgreSQL administration and management easier), ii) a selection of modules to provide additional specialised functionalities, and iii) a choice of procedural languages. With these modules came also the PostGIS that can be selected to install after the PostgreSQL installation. This module can also be downloaded and installed alone from its own Web page.

4.2.3. GeoServer

GeoServer is written in Java and requires the Java Development Kit (JDK) installed on the computer in order to run. The JDK and can be downloaded from the Sun Java downloads Web page (Java, 2009) and it is only necessary to run the installer, accepting the defaults.

Then, the GeoServer should be downloaded from its Web site and installed. The installation process it is also very easy by following the instructions provided on the user manual.

4.2.4. MapFish

This software is a Javascript API (Application Programming Interface) based on the OpenLayers.js Javascript file. It does not need installation, once it is only necessary to copy the Mapfish client folder and put it in the Web directory. This has to be done after the Web server installation.

4.2.5. UDig

For the installation of the uDig application, it is necessary to access its Web page and download the x86 installer for windows. After that, it is necessary to run it accepting the defaults to installation purpose.

4.3. Configuration

After all the software installation, it is necessary to configure the system, i.e. prepare the database and server to provide the necessary data through the WFS-T specification and, arrange the desktop and develop the Web application to access and manipulate the data.

4.3.1. Database preparation

The first thing to do is to create a PostgreSQL/PostGIS database and, inside it, a table to handle the dumps data, defining also the attributes to register, the geometry and the coordinate system. It was decided to use the polygon as geometry and the EPSG 4326 as the Coordinate Reference System (CRS). EPSG refers to the EPSG Geodetic Parameter Dataset, abbreviated to the EPSG Dataset and maintained by the OGP[5] Surveying and Positioning Committee's Geodetic Subcommittee (OGP, 2009).

The EPSG 4326 refers to the WGS84 datum, and was used because the developed web application has a base layer from Google maps that is in the same CRS. It was also determined to create four columns to store the attributes: name, type of garbage, terrain topography type and terrain access type.

Note that it is advisable for all the Database (DB) creation, for spatial purposes, to be based on a spatial template that is created when the PostGIS module is installed, called “template_postgis”. Through this, every new DB, created under these assumptions, came already with two imperative tables: the geometry_columns and

[5] OGP – International Association of Oil & Gas Producers

the spatial_ref_sys. The first one is used to register the geometry columns of the DB spatial tables and the second one has the reference systems parameters.

The geometry_columns table has some important columns to fill for every spatial table that exists in the DB: the f_table_name, the f_geometry_column and the srid column. These columns have to be filled with, respectively, the spatial table name, the column that has the geometry data and the spatial reference. In the srid column, a value that exists in the spatial_ref_sys table must be inserted.

Moreover, a spatial table is created with every defined columns and a more special one to record the geometry, called the_geom. This column has to be registered in the geometry_columns table, as described above.

4.3.2. GeoServer preparation

After the DB creation and configuration, it is important to configure the server to provide the dumps data through the WFS-T standard.

First, it is necessary to start the GeoServer, using the shortcut called “Start GeoServer”, located inside the GeoServer folder at the windows start menu. In the same location it is possible to open the Web Administration Tool, required to proceed to the next step.

The next step is to configure, through that tool, the data to provide and the WFS parameters. In the WFS configuration, accessed through the Config/WFS/Contents options at the main menu, it is important to ensure that the transactional service level is activated. Otherwise, transactions will not be allowed.

The data configuration involves three steps: the namespace, the data store and the feature type. These options are accessed through the Config/Data options at the main menu. The namespace used was the default “topp”. In the data store configuration, the DB connection parameters are provided. It is also possible to make connections to ESRI shapefiles, directories of spatial data or other Web

Feature Servers. It is in the feature type configuration that the schema base is provided. Parameters like the alias, the style, the spatial reference system, the title or the bounding box for the entire feature type, between others, are established.

4.3.3. uDig preparation

The uDig software preparation starts by adding, after opening the program, a new WFS layer through the Layer/Add... menu and selecting the Web Feature Server option, as it is shown in figure 7.

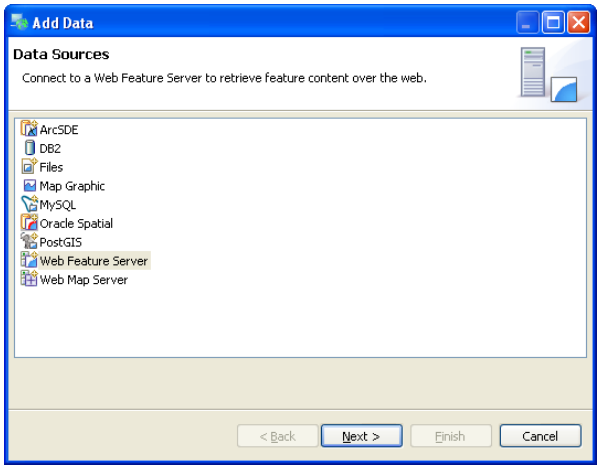

Figure 7 - Add data form

The next step is to follow the wizard, inserting the URL from the place where the WFS is served, and selecting the layer dumps from those available. In this case, once the GeoServer is installed at the localhost, the URL used was the http://localhost:8080/geoserver/wfs, as presented in figure 8.

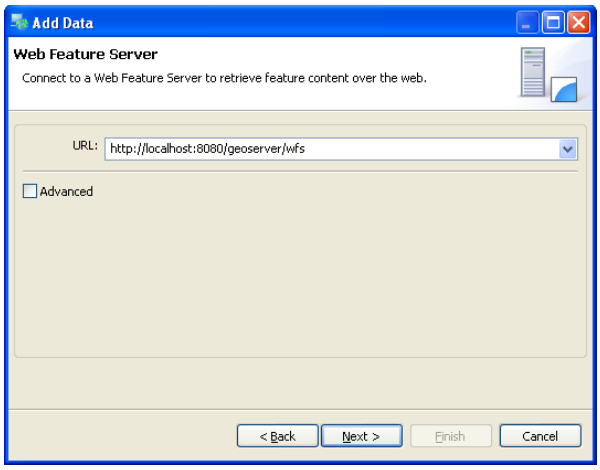

Figure 8 - URL of wfs GeoServer

After completing the wizard, the layer is shown in the uDig program interface as presented in figure 9.

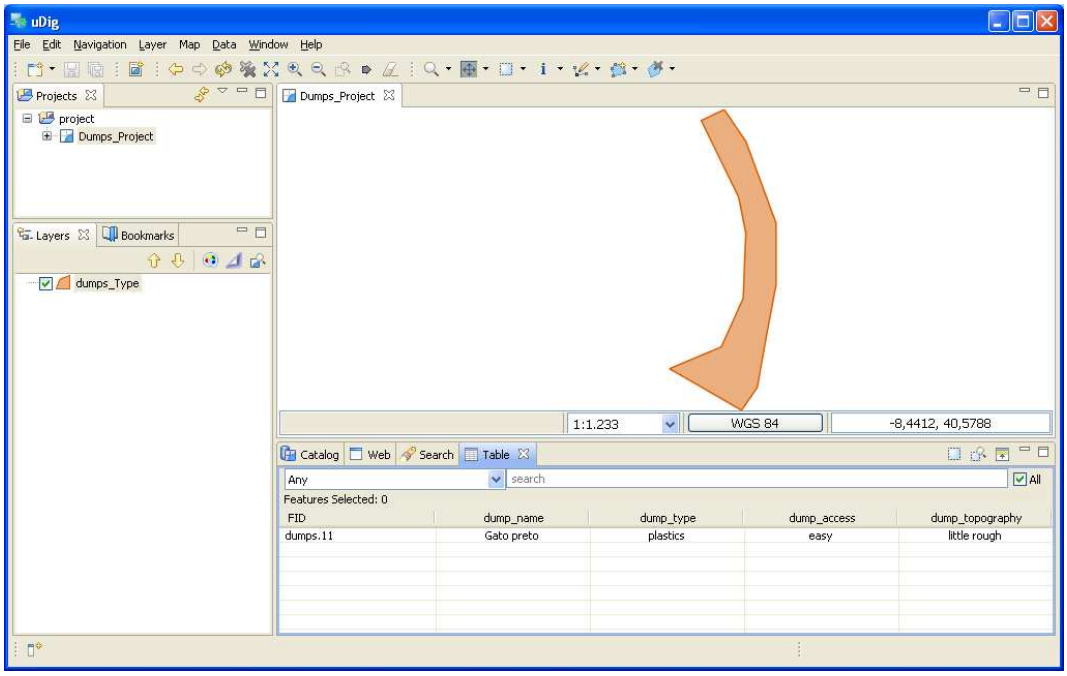

Figure 9 - WFS layer on a uDig map

It is now possible to add, edit and delete features in this WFS layer. It is important to refer that, after any change made to a feature, it is mandatory to save these changes, through the commit changes button, like showed in figure 10. If this action is not performed, the changes are not assumed in the DB and lost.

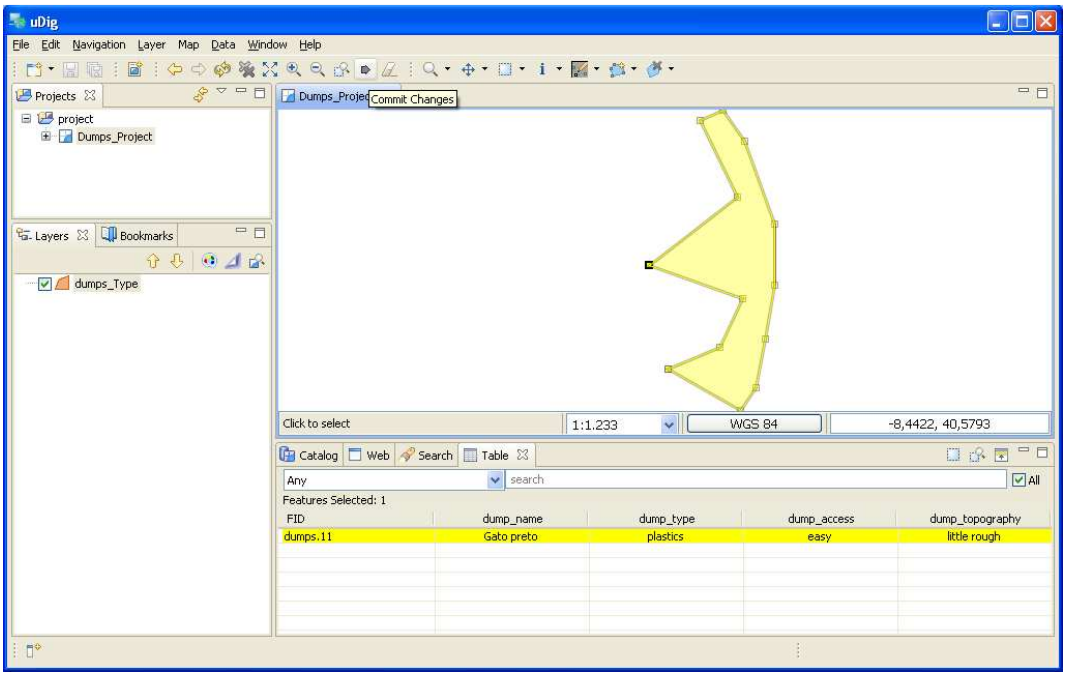

Figure 10 - Editing WFS layer on uDig map

To improve the uDig component of the prototype it would be interesting to use some imagery of the work area as context information. This would facilitate the recognition and interpretation of the geographical locations, where dumps are being registered, edited or deleted.

4.3.4. Web client application development

The starting point to the Web application development was the layout definition. It was decided to develop a layout with a map at the central region and other important information and tools in the surrounding area. The structure is shown in figure 11.

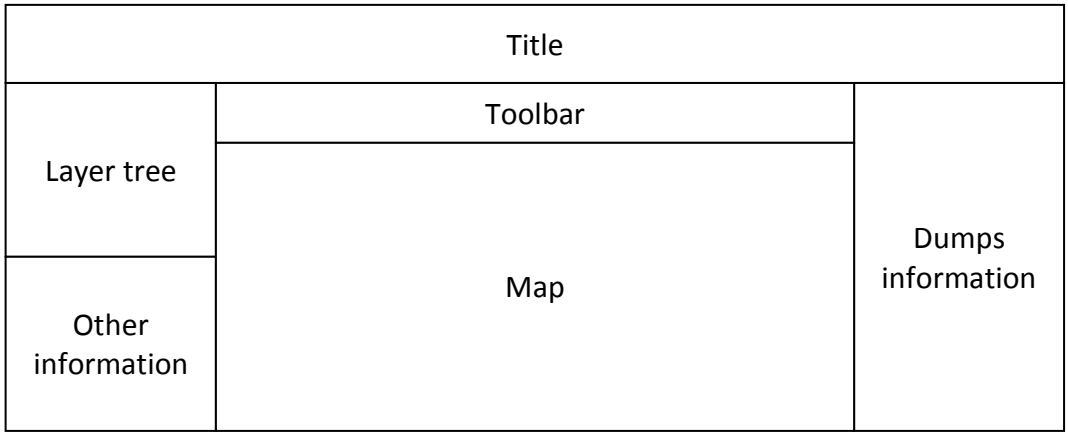

Figure 11 - Layout

As described in chapter 3.4, the Web client application was developed using Mapfish client software that is based on the Openlayers, ExtJS and GeoExt java script libraries. Therefore, all the programming is done using java script language. In the next sentences, some of the most important portions of code are detailed.

It is important to understand one of the first difficulties felt in the development – the proxy configuration. In the Frequently Asked Questions (FAQ) of the Openlayers wiki Web site (OpenLayers, 2009b), one can read: "*Due to security restrictions in Javascript, it is not possible to retrieve information from remote domains via an XMLHttpRequest. Classes like WFS and GeoRSS use XMLHTTPRequest to get their data. If they are querying a remote server (anything other than the machine hosting your page), you must install a proxy script somewhere Web accessible on that machine. See below for how to set up your own ProxyHost. If the OpenLayers.ProxyHost variable is not set to a valid proxy host, requests are sent directly to the remote servers. In most cases, the result will be a security exception, although this exception often occurs silently*". Consequently, a proxy server was installed in the local machine and the OpenLayers.ProxyHost variable was defined pointing to that proxy server location.

The next important thing to do is to include, in the developing Web application, the location of the mapfish.js, openlayers.js, ext.js and geoext.js library files, so they can

be imported when the user access it. This is presented in the next code lines. These code lines should be inserted at the <head> tag of the HTML Web page.

```
<script type="text/javascript" src="mfbase/GeoExt/lib/GeoExt.js ">
</script>
<script type="text/javascript" src="mfbase/mapfish/MapFish.js">
</script>
<script    src="http://dev.openlayers.org/sandbox/adube/openlayers
/lib/OpenLayers.js">
</script>
<script type="text/javascript" src="mfbase/mapfish/MapFish.js">
</script>
```

The OpenLayers.js file used is the development file instead of the file existent in the Web server of the prototype to guarantee always the use of the latest release.

The following code lines implements the layout defined in figure 11, using the viewport object of the ExtJS library.

```
var viewport = new Ext.Viewport({
     layout: 'border',
     items: [
           new Ext.BoxComponent ({
                 region: 'north',
                 el: 'north',
                 height: 35
           }),
           {
                 region: 'west',
                 id: 'west-panel',
                 title: '',
                 border: true,
                 width: 250,
                 split: true,
                 items: [{
                       region: 'north',
                       id: 'tree',
                       title: 'layer tree',
                       xtype: 'layertree',
                       map: map,
                       height: 150,
                       showWmsLegend: true,
                       enableDD: true,
                       model: modelo,
                       plugins: [
mapfish.widgets.LayerTree.createContextualMenuPlugin([
                             'opacitySlide',
                             'remove'
                             ])
                       ]
```

continue to the next page…

…continued from the previous page

```
                },{
                        region: 'center',
                        title: 'Other Informations',
                        contentEl: 'info',
                        collapsible: true,
                        height: 200
                }]
        }, {
                region: 'center',
                id: 'map',
                title: 'Map',
                layout: 'fit',
                split: true,
                xtype: 'mapcomponent',
                map: map,
                tbar: toolbar
        }, {
                region: 'east',
                id: 'dumps_form',
                title: 'Dumps info',
                split: true,
                width: 250,
                height: 50,
                items: [{
                        //html: szHTML
                }]
    }]
});
```

The next code lines were to create a base layer, provided by Google maps API, to make available context cartography and/or imagery. Thus, a base layer with three options was created: a Hybrid, a Satellite and a Streets map (with optional choice).

```
var ghyb = new OpenLayers.Layer.Google(
"Google Hybrid",
{MIN_ZOOM_LEVEL: 6, MAX_ZOOM_LEVEL: 20, 'type': G_HYBRID_MAP,
'sphericalMercator': true}
);

var gsat = new OpenLayers.Layer.Google(
"Google Satellite",
{MIN_ZOOM_LEVEL: 6, MAX_ZOOM_LEVEL: 20, 'type': G_SATELLITE_MAP,
'sphericalMercator': true}
);

var gstreets = new OpenLayers.Layer.Google(
"Google Streets",
{MIN_ZOOM_LEVEL: 6, MAX_ZOOM_LEVEL: 20, 'type': G_NORMAL_MAP,
'sphericalMercator': true}
);
```

To see this base layer it is necessary to include, in the <head> tag of the HTML Web page, the next code line to load the Google Maps API. Without this, the Google Map base layer is not shown.

```
<script type=text/javascript src="http://maps.google.com/maps?
file=api&amp;v=2&amp;key=ABQIAAAAjpkAC9ePGem0lIq5XcMiuhR_wWLPFku8I
x9i2SXYRVK3e45q1BQUd_beF8dtzKET_EteAjPdGDwqpQ"></script>
```

The next code block was used to create a vector layer on the WFS protocol. The “saveStrategy” is an object that deals automatically with the necessary procedures and requests to commit all the unsaved changes made to features.

```
saveStrategy = new OpenLayers.Strategy.Save();

dumps = new OpenLayers.Layer.Vector("dumps", {
    strategies: [new OpenLayers.Strategy.BBOX(), saveStrategy],
    projection: new OpenLayers.Projection("EPSG:4326"),
    protocol: new OpenLayers.Protocol.WFS({
         version: "1.1.0",
         srsName: "EPSG:4326",
         url: "http://localhost:8080/geoserver/wfs",
         featureNS :  "http://www.openplans.org/topp",
         featureType: "dumps",
         geometryName: "the_geom",
         schema:
"http://localhost:8080/geoserver/wfs/DescribeFeatureType?version=1
.1.0&typename=topp:dumps",
         extractAttributes: true
    })
});
```

The following line adds all the previous layers to the map object.

```
map.addLayers([ghyb, gsat, gstreets, dumps]);
```

The next object “tree” generates the layer tree that is located at the left side of the viewport, based on the “modelo” options. The last code line renders the tree. In that “tree” all the layers available for display are included.

```
var modelo = [{
     text: 'Base Layer',
     expanded: true,
     leaf: false,
     children:[{
           text: 'Google Hybrid',
           checked: ghyb.getVisibility(),
           layerNames: ['Google Hybrid'],
           leaf:true,
           draggable: false
     },{
           text: 'Google Satellite',
           checked: gsat.getVisibility(),
           layerNames: ['Google Satellite'],
           leaf: true,
           draggable: false
     },{
           text: 'Google Streets',
           checked: gstreets.getVisibility(),
           layerNames: ['Google Streets'],
           leaf:true,
           draggable: false
     },{
     text: 'Tematic Layer',
     expanded: true,
     leaf: false,
     children:[{
           text: 'Dumps',
           checked: true,
           layerNames: ['dumps'],
           leaf: true
     }]
     }]
}];
var tree = new mapfish.widgets.LayerTree({
     map: map,
     model: modelo,
     el: 'tree',
     enableDD: true
});
tree.render();
```

The next portion of code generates the toolbar, which is based on a MapFish widget. Inside it tools like zoom to extent, zoom in, zoom out, pan, zoom previews, zoom next, feature highlighter, draw feature, edit feature and delete feature are included. Figure 12 shows an image of the resultant toolbar.

```
var toolbar = new mapfish.widgets.toolbar.Toolbar({
     map: map,
     configurable: false
});
var tbarcontent = function() {
     toolbar.addControl(
           new OpenLayers.Control.ZoomToMaxExtent({
                 map: map,
                 title: 'Zoom to max'
           }), {
                 iconCls: 'zoomfull',
                 toggleGroup: 'map'
           }
     );
     separador();
     toolbar.addControl(
           new OpenLayers.Control.ZoomBox({
                 title: 'Zoom In'
           }), {
                 iconCls: 'zoomin',
                 toggleGroup: 'map'
           }
     );
     toolbar.addControl(
           new OpenLayers.Control.ZoomBox({
                 out: true,
                 title: 'Zoom Out'
           }), {
                 iconCls: 'zoomout',
                 toggleGroup: 'map'
           }
     );
     toolbar.addControl(
           new OpenLayers.Control.DragPan({
                 isDefault: true,
                 title: 'Pan'
           }), {
                 iconCls: 'pan',
                 toggleGroup: 'map'
           }
     );
     separador();
     var nav = new OpenLayers.Control.NavigationHistory();
     map.addControl(nav);
     nav.activate();
     var previous = new Ext.Toolbar.Button({
           iconCls: 'back',
           tooltip: 'Zoom previous',
           disabled: true,
           handler: nav.previous.trigger
     });
     var next = new Ext.Toolbar.Button({
           iconCls: 'next',
           tooltip: 'Zoom next',
           disabled: true,
           handler: nav.next.trigger
```

continue to the next page…

…continued from the previous page

```
    });
toolbar.add(previous);
toolbar.add(next);
nav.previous.events.register(
          "activate",
          previous,
          function() {
          this.setDisabled(false);
          }
    );
    nav.previous.events.register(
          "deactivate",
          previous,
          function() {
          this.setDisabled(true);
          }
    );
    nav.next.events.register(
          "activate",
          next,
          function(){
          this.setDisabled(false);
          }
    );
    nav.next.events.register(
          "deactivate",
          next,
          function() {
          this.setDisabled(true);
          }
    );
          separador();
    //GetFeatureInfo Control
    toolbar.addControl(
    FeatureHighlighter,// FeatureHighlighter call
          {
                title: 'Get Feature Info',
                iconCls: 'getinfo',
                toggleGroup: 'map'
          }
    );
    add = new OpenLayers.Control.DrawFeature(
          dumps,
          OpenLayers.Handler.Polygon,
          {
          title: 'Draw Polygon',
          handlerOptions: {multi: true}
          })
    toolbar.addControl(
          add,{
                iconCls: 'drawpolygon',
                toggleGroup: 'map'
          }
    );
    add.events.register("featureadded",'' , FeatureAdded);
```

continue to the next page…

…continued from the previous page

```
	edit = new OpenLayers.Control.ModifyFeature(lixeiras, {
		title: 'Modify Polygon'
	});
	toolbar.addControl(edit,
		{
			iconCls: 'movefeatures',
			toggleGroup: 'map',
		},
		);
	DeleteFeature = new OpenLayers.Control.DeleteFeature(
		dumps,
		{title: 'Delete dump'}
		);
	toolbar.addControl(
		DeleteFeature,
		{
			iconCls: 'removepolygon',
			toggleGroup: 'map'
		}
	);
DeleteFeature.events.register("beforefeaturesdeleted", '',
beforeDumpsDeleted);
DeleteFeature.events.register("deletefeatures", '', deleteDumps);
};
navigate();
tbarcontent();
toolbar.activate();
```

Figure 12 - Application toolbar

The next step was to develop a tool to highlight and show information about the dumps when a mouse over occurs. To achieve this goal, a listener was added to the code to listen when the mouse passes over a dump, and other listener to listen when the mouse leaves the dump, has shown in the next code lines.

```
FeatureHighlighter.events.register("featureset", '',
ObjectoSelecionadoOn);
FeatureHighlighter.events.register("featurereset", '',
ObjectoSelecionadoOff);
```

Thus, when the mouse passes over a dump, the listener “featureset” calls the “ObjectoSelecionadoOn” function, and when the mouse leaves the dump, the “ObjectoSeleccionadoOff” function is called. These tow functions are shown in the next code block.

```
function ObjectoSelecionadoOn(object) {
     var oFeature;
     if (object.geometry){
          oFeature = object;
     } else {
          oFeature = object.feature;
     }
// feature attributes parsing in html
     szHTML = "<div style='height:260px'><table border='0'>";
     if (!oFeature.cluster){
          aszAttributes = oFeature.attributes;
          for(var key in aszAttributes){
               szHTML += "<tr><td><div style='font-size:0.7em;
font-color:red;'>"
               + key + "</div></td><td><div style='font-
size:0.7em'> : "
               + aszAttributes[key] + "</div></td></tr>";
          }
     }
     szHTML +="</table></div>";
     fillViewPort();
}
function ObjectoSelecionadoOff(object) {
     clearViewPort()
}
```

The “ObjectoSelecionadoOn” function parses the feature attributes to the szHTML variable and calls the “fillVewPort” function. The “ObjectoSelecionadoOff” function calls the “clearViewPort” function.

The next code block shows the “fillVewPort” and “clearViewPort” functions. The first one puts the szHTML variable in the Dumps information region of the viewport, called “regiao”, and the second one clears that area. Thus, when the user passes the mouse over a feature, the respective information is shown in the “regiao” region and when the mouse leaves the feature, that region becomes clear.

```
function fillViewPort(){
     regiao.removeAll();
     regiao.add({
          html:szHTML
     });
     regiao.doLayout();
}
function clearViewPort(){
     regiao.removeAll();
     regiao.doLayout();
}
```

Until now, beyond the zoom tools and the feature highlighter, no functionality was added to the application. In the next code blocks, functionalities related to transactional operations, like add, edit and delete features, are shown.

When the add and edit dump buttons are selected, a form is shown in the Dumps information region of the viewport. The next code block shows the dumps form construction.

```
dumpsForm = new Ext.FormPanel({
     id: 'dumps_form',
     title: 'Dumps form',
     labelwidth: 50,
     bodyStyle:'padding:5px 5px 0',
     frame: true,
     defaults: {width: 230},
     defaulttype:'textfield',
     items: [{
           xtype: 'textfield',
           fieldLabel: 'Name',
           id: 'dump_name'
     },{
           xtype: 'textfield',
           fieldLabel: 'Type',
           id: 'dump_type'
     },{
           xtype: 'textfield',
           fieldLabel: 'Topography',
           id: 'dump_topography'
     },{
           xtype: 'textfield',
           fieldLabel: 'Access',
           id: 'dump_access'
     }],
     buttons: [{
           text: 'Save',
           tooltip: 'Save edits and geometry',
           handler: function(){
                 saveLixeiras();
           }
     },{
                 text: 'Cancel',
                 tooltip: 'Cancel edits',
                 handler: function(){
                       edit.selectControl.unselect(edit.feature)
                 }
     }]
});
```

When the add dump tool on the toolbar is selected, a handler is activated to handle a new dump draw and, when one finishes the drawing, the “FeatureAdded”

function is called to change the state of the feature to INSERT state and call the "mostrarFormulario" function. This can be seen in the next code block.

```
function FeatureAdded(object){
     var oFeature = object.feature;
     oFeature.state = OpenLayers.State.INSERT;
     mostrarFormulario(oFeature);
     edit.selectControl.select(oFeature);
}
```

The "mostrarFormulario" function, which is presented in the next code block, calls the dumpsForm, if a feature is passed to it, and places the form in the Dumps information region to fill with the inserted dump attributes.

```
function mostrarFormulario(Feature){
     if (val) {
           regiao.add(dumpsForm);
           val=!val;
     }
     dumpsForm.show();
     regiao.doLayout();
     dumpsForm.doLayout();
}
```

The next code lines are two listeners that are activated when the edit dump tool on the toolbar is selected and waits until one selects or unselects a feature to edit.

```
dumps.events.register("beforefeaturemodified", '',
iniciarModificacao);
dumps.events.register("afterfeaturemodified", '',
terminarModificacao);
```

Thus, when one selects a feature to edit, the "iniciarModificacao" function is called and when that feature is unselected, the "terminarModificacao" function is called. The code of these two functions is presented in the next code block.

```
function iniciarModificacao(object){
     var oFeature;
     if (object.geometry){
          oFeature = object;
     } else {
          oFeature = object.feature;
     }
     if(FeatureHighlighter.feature){
          FeatureHighlighter.resetFeature();
     }
          parseFeatureAttributesToForm(oFeature, dumpsForm);
     mostrarFormulario(oFeature);
}

function terminarModificacao(object) {
     var oFeature;
     if (object.geometry){
          oFeature = object;
     } else {
          oFeature = object.feature;
     }
     regiao.doLayout();
     dumpsForm.hide();
};
```

The “iniciarModificacao” function calls the “parseFeatureAttributesToForm” function, which parses the selected dump attributes and fills the dumpsForm. After that, it calls the “mostrarFormulario” function that has been described above.

The “terminarModificacao” function makes a refresh to the Dumps information region and hides the dumpsForm.

Finally, the next code lines aim at showing the deletion of a dump. In the toolbar creation code, immediately after the creation of the delete feature button, there are two listeners shown, as presented in the following code block.

```
DeleteFeature.events.register("beforefeaturesdeleted", '',
beforeDumpsDeleted);
DeleteFeature.events.register("deletefeatures", '', deleteDumps);
```

The first listener calls the “beforeDumpsDeleted” function, when a dump is selected, and it asks the user to confirm if he really wants to delete the feature. When the user answers yes, the second listener calls the “deleteDumps” function that uses the saveStrategy mechanism to make the transaction and delete the selected feature definitely.

In the next code block these two functions are presented.

```
function deleteDumps(event){
     saveStrategy.save(event.features);
}
function beforeDumpsDeleted(object){
alert("teste");
     Ext.Msg.confirm('Delete selected dumps',
     'Are you sure you want to delete the selected dumps ?',
           function(btn){
                if (btn == 'yes'){
                     DeleteFeature_copy.deleteFeatures({
                     silent: true
           });
           }
     });
     return false;
}
```

The final layout of the developed Web application can be seen, in various moments, in figures Figure 13, Figure 15, Figure 16, Figure 17 and Figure 17, respectively, normal layout, when a dump is added, when button "information" is selected and a mouse over occurs over a dump, editing a dump and deleting a dump.

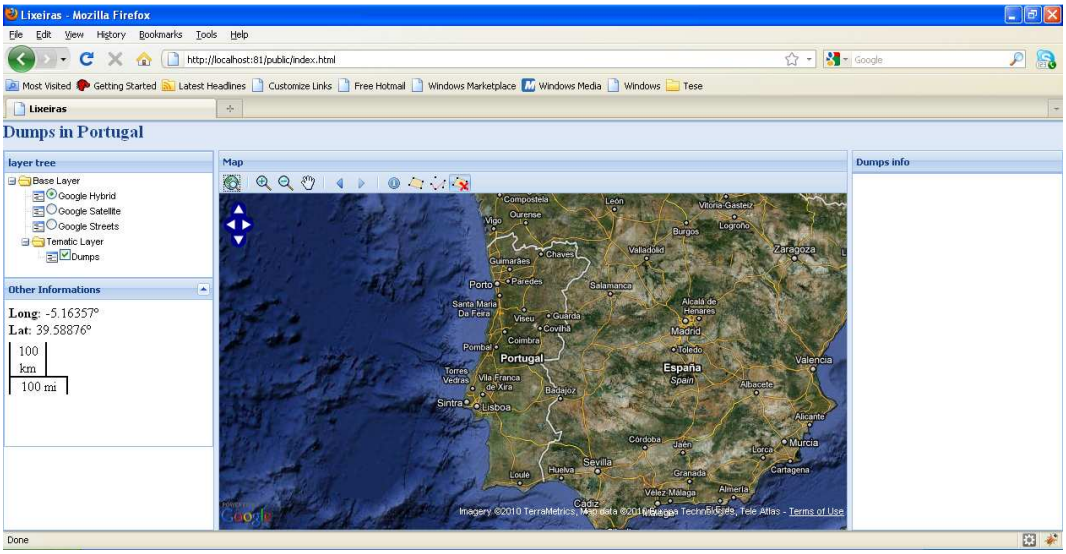

Figure 13 - Final layout of the developed Web application

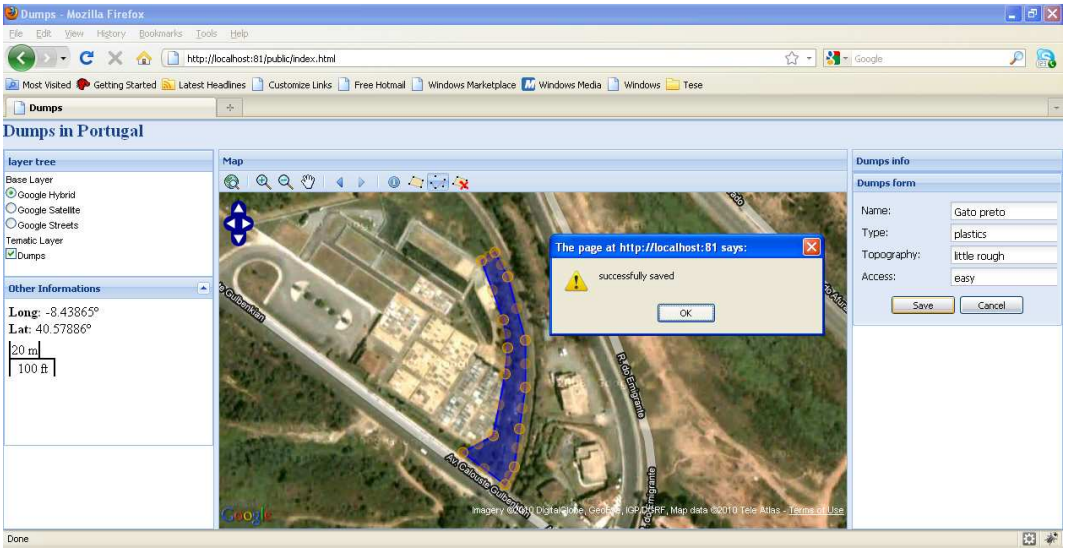

Figure 14 - Final layout when a dump is added

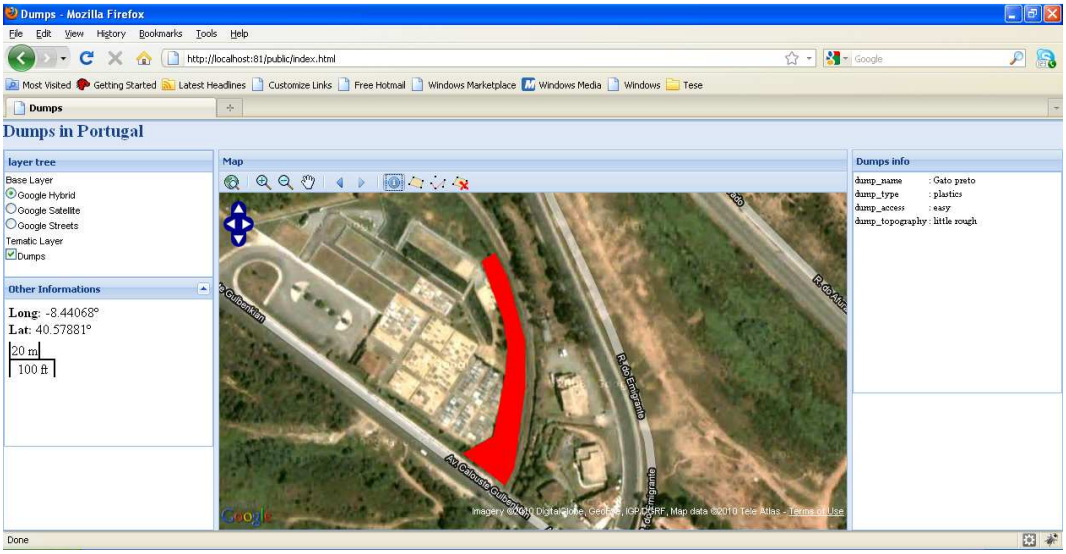

Figure 15 - Final layout when button "information" is selected and a mouse over occurs over a dump

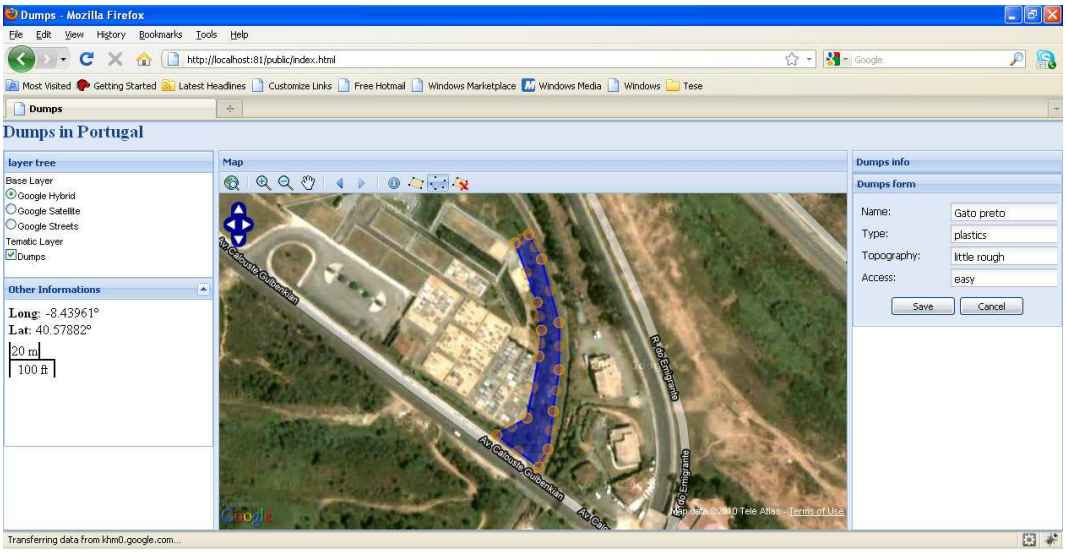

Figure 16 - Final layout when editing a dump

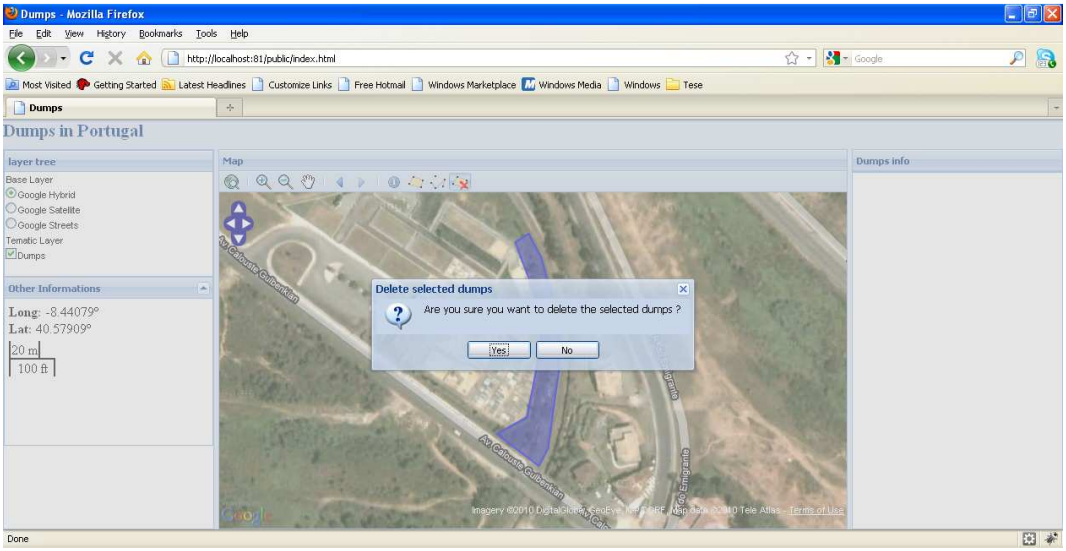

Figure 17 - Final layout when deleting a dump

4.4. Conclusions

With some development and language programming knowledge it is possible to develop a Web based GIS application based on OGC specifications, with platform independence.

Using a desktop application with WFS-T support, like uDIG, it is very simple to install and use without the need of programming knowledge.

All the software used is easily downloaded and installed on most of the Operating Systems, with all the inherent advantages.

The prototype has some limitations at the security and thrust levels. In fact, a system to control users' access or register users was not implemented. These limitations can be resolved with improvements to the prototype in future work, as it will be explained later.

5. Conclusions

5.1. The developed WFS-T-compliant FOSS prototype: advantages and limitations

The result of this work was the development of a prototype that provides data, stored in a DB through the WFS-T Standard, which can be accessed over a GIS desktop application or a developed Web application. A previous version of this prototype was already presented in the SASIG II conference (Estima and Cabral, 2009). The developed prototype has various advantages, especially concerning the choice of open formats – i.e. open standards and Web services and Open Source software used. This allows interoperability between the common desktop GIS software, and gives users the liberty to select the applications that best meet their needs.

The WFS standard has the advantage of providing the features itself, instead a representation of them in an image format, such as the WMS, for example. The transactional operations give the user the opportunity to manipulate them without needing to know the language or applications behind the system.

5.2. Main conclusions

The outlined objectives have been achieved. The WFS-T Standard was studied and explained. A prototype was developed, using only FOSS, since no commercial software implements the WFS-T standard. The prototype provides data, stored in a DB, through the WFS-T standard to be accessed by both GIS desktop software and Web developed application.

The developed Web application displays and allows that data manipulation through the standard, giving the user the possibility to create new features and edit or delete existing ones.

From the proprietary software analysed, like written above, none implements the WFS-T OGC standard. It is possible that this support is being implemented to be launched in new software versions.

The hypothesis formulated is accepted, concluding that it is possible to develop a FOSS prototype that implements the WFS-T standard.

5.3. Future work

In the future, it is necessary to continue improving the prototype, to better respond to some issues.

One of the major problems is the fact that anyone can edit or delete features created by others without their information or consent. The fact that anonymous people can create features is also a dilemma, once there is no trust in the registered data. These questions raise problems of data security and trust.

To solve these problems it is necessary to do some research regarding the user access level, especially as to control possibilities.

Another interesting development is related with the possibility to have some spatial analysis tools at the Web application level, which would allow decision support through a Web browser. This would bring the advantage of sharing not only the data, but also spatial analysis results in a distributed system, and always in open formats.

References

Apache. (2009). "The Apache Software Foundation." from http://www.apache.org.

ESRI (2009). Environmental Systems Research Institute, Inc. Web Site (URL: http://www.esri.com, retrieved in 08-09-2009).

Beaujardiere, J. (2006). OpenGIS Web Map Service Implementation Specification, Version 1.3.0.

Cadcorp (2009). Cadcorp Web Site (URL: http://www.cadcorp.com, retrieved in 08-09-2009).

Davis, S. (2007). GIS for Web Developers - Adding Where to Your Web Applications, The Pragmatic Programmers LLC.

Dragicevic, S. (2004). "The potential of Web-based GIS." Journal of Geographical Systems **6**(2): 79-81.

Estima, J., Cabral, P. (2009). O Web Feature Service – Transactional Standard – Exemplo de um protótipo de implementação. II Jornadas de Software Aberto para Sistemas de Informação Geográfica, Évora, Portugal.

Fälman, S. (2004). Using WFS to build GIS support. Department of Computing Science. Umeå, Sweden, Umeå University.

Furtado, D. N. (2006). Serviço de visualização de informação geográfica na Web – A publicação do atlas de Portugal utilizando a especificação Web Map Service. Instituto Superior de Estatística e Gestão de Informação. Lisbon, Universidade Nova de Lisboa.

Geoffrey, A. and M.-S. Rafael (2003). "Building Web-Based Spatial Information Solutions around Open Specifications and Open Source Software." Transactions in GIS **7**(4): 447-466.

Geoserver (2009). Geoserver web site (URL: http://geoserver.org, retrieved in 08-09-2009).

GNU. (2010). GNU Operating System Web Site (URL: http://www.gnu.org/, retrieved in 08-09-2009).

gvSIG (2009). gvSIG Web Site (URL: http://www.gvsig.org, retrieved in 08-09-2009).

INSPIRE. (2009). "INSPIRE - Infrastructure for Spatial Information in Europe." from http://inspire.jrc.ec.europa.eu.

Intergraph (2009). Intergraph Web Site (URL: http://www.intergraph.com, retrieved in 08-09-2009).

Java (2009). Java SE Downloads (URL: http://java.sun.com/javase/downloads/index.jsp, retrieved in 08-09-2009).

ka-MAP (2009). Ka-Map Web Site (URL: http://ka-map.maptools.org/, retrieved in 08-09-2009).

Kosmo (2009). Kosmo Web Site (URL: http://www.opengis.es, retrieved in 08-09-2009).

Kreger, H. (2001) Web Services Conceptual Architecture (WSCA 1.0).

Longley, Paul A., M. F. G., David J. Maguire, David W. Rhind (2005). Geographic Information Systems and Science, John Wiley & Sons, Ltd.

MapBender (2009). MapBender Web Site (URL: http://www.mapbender.org, retrieved in 08-09-2009).

MapFish (2009). MapFish Web Site (URL: http://mapfish.org, retrieved in 08-09-2009).

MapServer (2009). MapServer Web Site (URL: http://mapserver.org, retrieved in 08-09-2009).

Moreno-Sanchez et al. (2007). "The potential for the use of Open Source Software and Open Specifications in creating Web-based cross-border health spatial information systems." International Journal of Geographical Information Science **21**(10): 1135 - 1163.

OGC. (2008). Open GeoSpatial Consortium Web Site (URL: http://www.ogc.org, retrieved in 08-09-2009).

OGP (2009). Using the EPSG Geodetic Parameter Dataset (OGP Surveying and Positioning Guidance Note number 7, part 1).

OpenJUMP (2009). OpenJUMP Web Site (URL: http://www.openjump.org, retrieved in 08-09-2009).

OpenLayers (2009a). OpenLayers Web Site (URL: http://openlayers.org, retrieved in 08-09-2009).

OpenLayers (2009b). OpenLayers Wiki Web Site. Fromhttp://trac.openlayers.org/wiki/FrequentlyAskedQuestions#ProxyHost, retrieved in 08-04-2009).

Peng, Z.-R. and M.-H. Tsou (2003). Internet GIS: Distributed Geographic Information Services for the Internet and Wireless Networks, john Wiley & Sons, inc.

Peng, Z.-R. and C. Zhang (2004). "The roles of geography markup language (GML), scalable vector graphics (SVG), and Web feature service (WFS) specifications in the development of Internet geographic information systems (GIS)." Journal of Geographical Systems **6**(2): 95-116.

PostgreSQL. (2009). PostgreSQL Web Site (URL: http://www.postgresql.org/, retrieved in 08-09-2009).

Quantum GIS (2009). Quantum GIS Web Site (URL: http://www.qgis.org, retrieved in 08-09-2009).

Ramsey, P. (2007). The State of Open Source GIS, Refractions Research Inc.

Ramsey, P. (2009). "PostGIS versus MySQL Spatial." Retrieved 08-09-2009, from http://docs.opengeo.org/geospiel/?p=396.

Refractions. (2009). Refractions Web Site (URL: http://www.refractions.net/, retrieved in 08-09-2009).

Sælid, K. (2006). “WFS-T and Mobile Clients.” Oslo, The University of Oslo.

Salinas, J. and M. M. Lajara (2009). "Current Panorama of the FOSS4G Ecosystem." The European Journal for the Informatics Professional **X**: 43-51.

Steiniger, S. and E. Bocher (2009). "An overview on current free and open source desktop GIS developments." International Journal of Geographical Information Science **23**(10): 1345 - 1370.

SuperMap (2009). SuperMap Web Site (URL: http://www.supermap.com, retrieved in 08-09-2009).

Tyler, M. (2005). Web Mapping Illustrated, O'Reilly Media, Inc.

uDig (2009). uDig Web Site (URL: http://udig.refractions.net, retrieved in 08-09-2009).

Vckovski, A. (1998). Interoperable and Distributed Processing in GIS, CRC.

Vckovsky, A. (1998). "Guest Editorial Special Issue: Interoperability in GIS." International Journal of Geographical Information Science **12**(4): 297 - 298.

Vretanos, P. (2005). Web Feature Service Implementation Specification, Version 1.1.0.

XAMPP (2009), XAMPP Web Site (URL: http://www.apachefriends.org/en/xampp.html, retrieved in 08-09-2009).

W3C. (2010). World Wide Web Consortium Web Site (URL: from http://www.w3.org/Consortium, retrieved in 14-06-2010).

Whiteside, A. (2005) OpenGIS Web services architecture description, Version 0.1.0.

Wikipedia. (2010). "Comparison of open source and closed source." Retrieved 30.06.2010, from http://en.wikipedia.org/wiki/Comparison_of_open_source_and_closed_source.