

POLITECNICO DI TORINO

Scuola di Dottorato

Research Doctorate Course in
Environmental
Geoengineering

Dottorato di ricerca in
Ambiente e Territorio

DITAG
XXIII Cycle
A.A. 2009/2010
Final Dissertation

WebGis Architectures for Emergency Response



Simone DALMASSO

Tutor
Prof. Piero BOCCARDO

Coordinator of the Research Doctorate Course
Prof. Ambrogio Maria MANZINO

April 2011

INDEX

INTRODUCTION	9
BACKGROUND	11
TECHNOLOGY	39
CONCLUSIONS	137
BIBLIOGRAPHY/WEBOGRAPHY	139
ATTACHMENTS	141

CONTENTS

1. THE WEBGIS CONCEPT	11
1.1. Definition of Web Mapping	11
1.1.1. Advantages and issues	12
1.1.2. Web map types	12
1.2. History of web mapping	14
1.3. Open Source Vs Closed Source	15
1.4. Open Source Definition	16
1.4.1. Pros and Cons	18
1.4.2. Development philosophy	19
2. CURRENT PANORAMA IN THE WEB MAPPING FIELD	21
2.1. Public services	21
2.1.1. Google Maps	21
2.1.2. Bing and Yahoo Maps	22
2.2. Open Source ready to use products	22
2.2.1. CartoWeb	22
2.2.2. Mapbender	22
2.2.3. P.mapper	22
2.3. Open Source libraries and servers	23
2.3.1. MapServer	23
2.3.2. GeoServer	23
2.3.3. OpenLayers	24
2.3.4. GeoExt	24
2.3.5. Ext.Js	24
2.3.6. MapFish	24
2.4. Commercial tools and libraries	25
2.4.1. ArcGis Server	25
2.4.2. ArcGis client APIs	26
3. WEBGIS FIELDS OF USE	27
3.1. Public services	27
3.2. Geoportals	27
3.3. Custom WebGis for specific tasks	28
4. ITHACA	29
4.1. Area of competence	29

4.2. Projects	30
4.2.1. Spatial Data Infrastructure for the WFP	31
4.2.2. Early impact analysis	32
4.2.3. Early warning system	32
4.2.4. Floodplain modeling	32
4.3. Fields of intervention	32
5. EMERGENCY MANAGEMENT	34
5.1. Phases	35
5.1.1. Mitigation	35
5.1.2. Preparedness	36
5.1.3. Response	37
5.1.4. Recovery	37
5.2. User needs	38
6. PRINCIPLES OF WEB SERVICES	39
6.1. Internetwork	39
6.1.1. Low Level Protocols	39
6.1.2. High Level Protocols	40
6.2. Web services	42
7. PROGRAMMING LANGUAGES	44
7.1. Markup Languages	45
7.1.1. HTML	45
7.1.2. XML	46
7.1.3. XHTML	46
7.1.4. CSS	46
7.2. Scripting languages	47
7.2.1. Javascript	47
7.2.2. PHP	49
7.3. Compiled or semi compiled languages	50
7.3.1. Python	50
7.3.2. Java	52
8. JAVASCRIPT LIBRARIES	53
8.1. Browser events	53
8.2. DOM	54
8.3. Non geographic javascript libraries	55
8.3.1. Ext.js	56
8.3.2. jQuery	56

8.3.3. MooTools	58
8.4. Geographic libraries	58
8.4.1. OpenLayers	59
8.4.2. GeoExt	60
9. SPATIAL DATABASE	61
9.1. Relational model	62
10. METADATA AND GEOSPATIAL STANDARDS	64
10.1. Inspire	64
10.2. Catalogues	65
10.2.1. Geonetwork	65
10.2.2. Geoportal	66
10.3. Geospatial standards	67
10.3.1. WMS	67
10.3.2. WFS	68
10.3.3. WCS	68
10.3.4. KML	68
11. WEB MAP SERVER	69
11.1. Geoserver	69
12. LINUX	71
12.1. Low level packages	72
12.2. Security packages	72
12.3. High level tools	72
13. DJANGO AND THE MCV PATTERN	74
13.1. MVC pattern	74
13.2. Django features	76
14. HARDWARE AND SOFTWARE DEFINITION	77
14.1. Hardware	77
14.1.1. SAN	78
14.2. Server side environment	79
14.3. Client side software	80

15. HARDWARE AND SOFTWARE IMPLEMENTATION	82
15.1. Hardware link and SAN drives definition	82
15.2. Server Installation and configuration	83
15.2.1. Base configurations	83
15.2.2. Security	84
15.2.3. SAN link and database creation	85
15.3. Apache web server configuration	87
15.3.1. Handlers	87
15.3.2. Proxies	88
15.4. Database creation	89
15.5. Server side architecture design	90
15.5.1. Django based folder structure and logic	91
15.5.2. Geoserver deployment and configuration	92
15.5.3. GeoWebCache	93
15.6. Client side libraries implementation	94
15.7. Remote development environment set up	95
16. ARCHITECTURE DESIGN	96
16.1. Database design	97
16.2. Geoserver set up	98
16.3. Client side architecture design	98
16.3.1. Reference systems	100
16.4. Identification of the expected dynamic parts	100
16.5. Design of the server side architecture	101
16.5.1. URLS design	101
16.5.2. Database definition and interaction	102
16.5.3. Business logic	102
16.5.4. Template generation	102
16.5.5. Web site security	103
17. DEVELOPED SERVICES	105
17.1. Flood world wide alert system	105
17.1.1. Web application	107
17.1.2. Database	108
17.1.3. Server side logic	109
17.1.4. Client side logic	111
17.2. Drought monitoring system	113
17.2.1. Web application	113

17.2.2. Database	114
17.2.3. Server side logic	114
17.2.4. Client side logic	115
17.3. Snow cover monitoring system	115
17.3.1. Web Application	116
17.3.2. Database	116
17.3.3. Server side logic	116
17.3.4. Client side logic	117
17.4. Cyprus cultural heritage catalogue	118
17.4.1. Web application	118
17.4.2. Database	121
17.4.3. Server side logic	121
17.4.4. Client side logic	123
17.5. Haiti emergency WebGis	125
17.5.1. Web application	126
17.5.2. Database	127
17.5.3. Server side logic	128
17.5.4. Client side logic	128
17.6. Development environments	129
17.6.1. Web application	130
17.6.2. Server side logic	131
17.6.3. Client side logic	132
18. FURTHER DEVELOPMENTS AND IMPROVEMENTS	134

LIST OF FIGURES

FIGURE 1: OPEN SOURCE LOGO	18
FIGURE 2: ITHACA LOGO	29
FIGURE 3: SDI ARCHITECTURE	31
FIGURE 4: EMERGENCY MANAGEMENT CYCLE	35
FIGURE 5: HTTP MESSAGES SCHEMA	41
FIGURE 6: WEB SERVICES ARCHITECTURE	43
FIGURE 7: HTML SNIPPET	45
FIGURE 8: XML SNIPPET	46
FIGURE 9: CSS SNIPPET	47
FIGURE 10: JAVASCRIPT SNIPPET	47
FIGURE 11: PHP SNIPPET	49
FIGURE 12: PYTHON SNIPPET	50
FIGURE 13: JAVA SNIPPET	52
FIGURE 14: DOCUMENT OBJECT MODEL REPRESENTATION	55
FIGURE 15: EXT.JS HOME - HTTP://WWW.SENCHA.COM/PRODUCTS/EXTJS/	56
FIGURE 16: JQUERY HOME - HTTP://JQUERY.COM/	57
FIGURE 17: MOOTOOLS HOME - HTTP://MOOTOOLS.NET/	58
FIGURE 18: OPENLAYERS HOME - HTTP://OPENLAYERS.ORG/	59
FIGURE 19: GEOEXT HOME - HTTP://GEOEXT.ORG/	60
FIGURE 20: POSTGIS HOME - HTTP://POSTGIS.REFRACTIONS.NET/	61
FIGURE 21: UML DIAGRAM FOR RELATIONAL MODEL	62
FIGURE 22: GEONETWORK HOME - HTTP://GEONETWORK-OPENSOURCE.ORG/	65
FIGURE 23: GEONETWORK MAIN PAGE	66
FIGURE 24: GEOPORTAL HOME - HTTP://WWW.ESRI.COM/SOFTWARE/ARCGIS/GEOPORTAL/INDEX.HTML	66
FIGURE 25: SLD SNIPPET	70
FIGURE 26: LINUX LOGO	71
FIGURE 27: DJANGO LOGO	74
FIGURE 28: MVC PATTERN	75
FIGURE 29: ITHACA OPEN SOURCE ARCHITECTURE	77
FIGURE 30: APACHE LOGO	87
FIGURE 31: SQL SNIPPET	89
FIGURE 32: GEOWEBCACHE LOGO	93
FIGURE 33: ARCHITECTURE DESIGN	97
FIGURE 34: FLOOD WORLD WIDE ALERT SYSTEM	105
FIGURE 35: FLOOD APPLICATION TOOL TIP	107
FIGURE 36: FLOOD APPLICATION GEORSS TOOL TIP	108
FIGURE 37: FLOOD APPLICATION LAYER TREE AND GRID STATUS	108
FIGURE 38: FLOOD APPLICATION TOOLBAR	108
FIGURE 39: DROUGHT MONITORING SYSTEM	113
FIGURE 40: SNOW COVER MONITORING SYSTEM	115
FIGURE 41: SNOW COVER TOOLBAR	116
FIGURE 42: CYPRUS CULTURAL HERITAGE CATALOGUE	118
FIGURE 43: CYPRUS SEARCH TOOL	119
FIGURE 44: CYPRUS CONTROL WINDOW	119
FIGURE 45: CYPRUS TOOL TIP ON A FEATURE	120
FIGURE 46: CYPRUS LEGEND	120
FIGURE 47: CYPRUS ADMIN INTERFACE	121
FIGURE 48: HAITI EMERGENCY WEBGIS	125
FIGURE 49: HAITI, VIEW OF PORT AU PRINCE WITH DAMAGE ASSESSMENT	127
FIGURE 50: WEBGIS GENERATOR STARTING INTERFACE	130
FIGURE 51: GETCAPABILITIES TOOL INSIDE THE MVC JAVASCRIPT APPLICATION	130
FIGURE 52: AUTOMATICALLY GENERATED WEBGIS	131
FIGURE 53: GEONODE HOME - HTTP://GEONODE.ORG/	134
FIGURE 54: GEOEXPLORER VIEW WITH LAYER ADDING WINDOW FROM ITHACA SERVER	135
FIGURE 55: GEONODE DATA SEARCH INTERFACE	136

Introduction

In the era of information internet is the main source from which to find anything. The story of internet is recent and brief, about 20 years, during which it has evolved continuously and quickly. The information given in the first years was flat like pure text or documents but in the last 10 years the type of information shared has changed, moving to other dimensions from the flat page. Now we can experience videos, music and even maps. The concept of attaching geographic information to the information itself has spread on almost every site.

Maps are everywhere because the question “where does this come from?” has assumed the same importance as the “what”.

Georeferencing and managing spatial information is not the same as managing non spatial information because there is an increase in complexity due to the fact that geographic information is not absolute. In the world there are thousands of “spatial reference systems” (srs) and different standards on how to share spatial data. Whoever hosts spatial data must allow the gathering of data in different formats and different “srs”.

On the other hand there is the fact that with spatial data we can perform spatial analysis such as distance calculations, routing or proximity. Many services which deal with spatial data even offer some additional services such as Google Maps.

In many cases the web spatial information is gaining much importance, paper maps have always been used to plan the environment and the actions on it, but digital mapping allows more degrees of freedom granting the ability to customize what we want and how we want it presented wherever we are.

The case of humanitarian emergencies is the core example about how the management of information and the usage of it are totally displaced. Whoever manages and keeps spatial data can be in a HQ office and people in the field can use information without worrying about getting it physically. The only thing needed is an internet link, which now is granted by satellite communication even if on the ground there is lack of infrastructures.

This PhD thesis has been developed in the framework of the collaboration between Information Technology for Humanitarian Assistance, Cooperation and Action (ITHACA), a non profit association founded by the Politecnico di Torino and the Istituto Superiore per i Sistemi Territoriali e l'Innovazione (SITI) and the World Food Programme (WFP). The collaboration with the WFP characterizes the research projects in the field of humanitarian emergencies and the goal of the present thesis.

One of the core things that determine a good emergency response is data availability. This term means both immediate availability after the event and quality availability, having the most up to date and precise data.

To meet the challenge of rapid WebGis development and data sharing in case of emergency, this work has focused on the development of an architecture composed by Open Source tools integrated and tuned to allow the building of WebGis and web applications in short time but maintaining an high level of customization.

The panorama in the geographic Open Source community is populated by several tools dedicated to specific tasks like data publishing, data management or user interface; however these projects have always been independent from each other without a common design able to harmonize the efforts in one single tool.

The idea is to have a unique server architecture composed of data publishing and management systems able to output data in different ways based on the web HTTP protocol and respecting international standards.

Web applications should act as custom views on the architecture and web services should allow access to data regardless of the requesting source.

The development process has involved deep research into basic informatic tools and operating systems as well as high level programming languages. Following several web applications, an environment able to manage different technologies and data sources has been built.

The goal is to test the proposed solution during real case studies in order to exploit usability, stability as well as effectiveness during emergency.

Background

1. The WebGis concept

1.1. Definition of Web Mapping

Web mapping is the process of designing, implementing, generating and delivering maps on the World Wide Web and its product. While web mapping primarily deals with technological issues, web cartography additionally studies theoretic aspects: the use of web maps, the evaluation and optimization of techniques and workflows, the usability of web maps, social aspects, and more. Web GIS is similar to web mapping but with an emphasis on analysis, processing of project specific geodata and exploratory aspects. Often the terms WebGis and web mapping are used synonymously, even if they don't mean exactly the same. In fact, the border between web maps and WebGis is blurry. Web maps are often a presentation media in web GIS and web maps are increasingly gaining analytical capabilities. A special case of web maps are mobile maps, displayed on mobile computing devices, such as mobile phones, smart phones, PDAs, GPS and other devices. If the maps on these devices are displayed by a mobile web browser or web user agent, they can be regarded as mobile web maps. If the mobile web maps also display context and location sensitive information, such as points of interest, the term location-based services is frequently used.

The use of the web as a dissemination medium for maps can be regarded as a major advancement in cartography and opens many new opportunities, such as realtime maps, cheaper dissemination, more frequent and cheaper updates of data and software, personalized map content, distributed data sources and sharing of geographic information. It also implicates many challenges due to technical restrictions (low display resolution and limited bandwidth, in particular with mobile computing devices, many of which are physically small, and use slow wireless Internet connections), copyright and security issues, reliability issues and technical complexity. While the first web maps were primarily static, due to technical restrictions, today's web maps can be fully interactive and integrate multiple media. This means that both web mapping and web cartography also have to deal with interactivity, usability and multimedia issues.

1.1.1. Advantages and issues

The advent of web mapping can be regarded as a major new trend in cartography. Previously, cartography was restricted to a few companies, institutes and mapping agencies, requiring expensive and complex hardware and software as well as skilled cartographers and geomatics engineers. With web mapping, freely available mapping technologies and geodata potentially allow every skilled person to produce web maps, with expensive geodata and technical complexity (data harmonization, missing standards) being two of the remaining barriers preventing web mapping from fully going mainstream. The cheap and easy transfer of geodata across the internet allows the integration of distributed data sources, opening opportunities that go beyond the possibilities of disjoint data storage. Everyone with minimal knowhow and infrastructure can become a geodata provider. These facts can be regarded both as an advantage and a disadvantage. While it allows everyone to produce maps and considerably enlarges the audience, it also puts geodata in the hands of untrained people who potentially violate cartographic and geographic principles and introduce flaws during the preparation, analysis and presentation of geographic and cartographic data. Educating the general public about geographic analysis and cartographic methods and principles should therefore be a priority to the cartography community.

1.1.2. Web map types

A first classification of web maps can be done by distinguishing static and dynamic web maps and further by interactive and view only web maps. However, today in the light of an increased number of different web map types, this classification can be improved. Today, there are additional possibilities regarding distributed data sources, collaborative maps, personalized maps, etc.

Analytic web maps

These web maps offer GIS analysis, either with geodata provided, or with geodata uploaded by the map user. As already mentioned, the borderline between analytic web maps and WebGis is blurry. Often, parts of the analysis are carried out by a server side GIS and the client displays the result of the analysis. As web clients gain more and more capabilities, this task sharing may gradually shift.

Animated web maps

Animated Maps show changes in the map over time by animating one of the graphical or temporal variables. Various data and multimedia formats and technologies allow the display of animated web maps: SVG, Adobe Flash, Java, Quicktime, JavaScript etc., also with varying degrees of interaction. Examples for animated web maps are weather maps, maps displaying dynamic natural or other phenomena (such as water currents, wind patterns, traffic flow, trade flow, communication patterns, social studies projects, and for college life, etc.).

Collaborative web maps

Collaborative maps are still new, immature and complex to implement, but show a lot of potential. The method parallels the Wikipedia project where various people collaborate to create and improve maps on the web. Technically, an application allowing simultaneous editing across the web would have to ensure that geometric features being edited by one person are locked, so they can't be edited by other persons at the same time. Also, a minimal quality check would have to be made, before data goes public. Some collaborative map projects:

- OpenStreetMap
- WikiMapia

Customizable web maps

Web maps in this category are usually more complex web mapping systems that offer APIs for reuse in other people's web pages and products. Example for such a system with an API for reuse are the Open Layers Framework, Yahoo! Maps and Google Maps.

Distributed web maps

These are maps created from a distributed data source. The WMS protocol offers a standardized method to access maps on other servers. WMS servers can collect these different sources, reproject the map layers, if necessary, and send them back as a combined image containing all requested map layers. One server may offer a topographic base map, while other servers may offer thematic layers.

Dynamically created web maps

These maps are created on demand each time the user reloads the web pages, often from dynamic data sources, such as databases. The web server generates the map using a web map server or a self written software.

Personalized web maps

Personalized web maps allow the map user to apply his own data filtering, selective content and the application of personal styling and map symbolization. The OGC (Open Geospatial Consortium) provides the SLD standard (Styled Layer Description) that may be sent to a WMS server for the application of individual styles. This implies that the content and data structure of the remote WMS server is properly documented.

1.2. History of web mapping

Currently the concept of web mapping is everywhere on the net, from Google Maps to complex WebGis. But the history of the web map is relatively short, first draft are been published about ten years ago. Before some works can be found in military environments. First maps were static with no more than panning and zooming functionalities, some web sites, in the 1996, like *Mapquest* and *MultiMap* offer some basic tools for routing on a road network and some geo localization. In the end of the 1996 were released the GeoMedia WebMap 1.0, a commercial bundle to build web map web sites. The first Open Source project released was UMN MapServer from University of Minnesotha in the 1997 developed as a part of the NASA ForNet project. After the release of the HTML 4.0 at the end of 1997 and after some stable improvements on the JavaScript language several project were born using the new possibility offered by new technologies.

In 2000 the first public version of the ESRI ArcIMS has been released, the growing interests of the commercial companies in the field of web map was a boost to the development of new technologies and concepts behind the services.

In the 2005 Google has released his Google Maps, after that milestone internet has started to integrate maps in many sites and people started to understand the importance of geo localization and map to plan trips or simply to find addresses and locations. At the end of 2005 has also been released Google Earth, a more appealing tool to view geo data where create and post georeferenced informations.

One important project started in 2004, OpenStreetMap is becoming famous at present day but which was and actually is a really important concept: *Collaborative and Volunteer Mapping* released under free license both for the data and tools.

1.3. Open Source Vs Closed Source

The debate over open source vs. closed source (alternatively called proprietary software) is sometimes heated.

One source of conflict is related to economics: Making money through traditional methods, such as sale of the use of individual copies and patent royalty payment (generally called licensing), is more difficult and in many ways against the very concept of open source software.

Some closed-source advocates see open source software as damaging to the market of commercial software. This is one of the many reasons, as mentioned above, that the term free software was replaced with open source — because many company executives could not believe in a product that did not participate economically in a free-market or mixed-market economy.

The counter to this argument is the use of open source software to fuel the market for a separate product or service. For example providing support and installation services; similar to IT Security groups, Linux Distributions, and Systems companies.

Using the software as a stepping stone to sell a higher-end product or service; e.g., OpenOffice.org vs. StarOffice.

Cost avoidance / cost sharing: many developers need a product, so it makes sense to share development costs.

Since open source software is open, all of the defects and security flaws are easily found. Closed-source advocates argue that this makes it easier for a malicious person to discover security flaws. Further, that there is no incentive for an open-source product to be patched. Open-source advocates argue that this makes it easier also for a patch to be found and that the closed-source argument is security through obscurity, which this form of security will eventually fail, often without anyone knowing of the failure. Further, that just because there is not an immediate financial incentive to patch a product, does not mean there is not any incentive to patch a product. Further, if the patch is that significant to the user, having the source code, the user can technically patch the problem themselves. These arguments are hard to prove. However, research indicates that open-source software does have a higher flaw discovery, quicker flaw discovery, and quicker turn around on patches.

Open source describes practices in production and development that promote access to the end product's source materials. Some consider open source a philosophy, others consider it a pragmatic methodology. Before the term *open source* became widely

adopted, developers and producers used a variety of phrases to describe the concept; *open source* gained hold with the rise of the Internet, and the attendant need for massive retooling of the computing source code. Opening the source code enabled a self-enhancing diversity of production models, communication paths, and interactive communities. Subsequently, a new, three-word phrase "open source software" was born to describe the environment that the new copyright, licensing, domain, and consumer issues created.

The open source model includes the concept of concurrent yet different agendas and differing approaches in production, in contrast with more centralized models of development such as those typically used in commercial software companies. A main principle and practice of open source software development is peer production by bartering and collaboration, with the end-product, source-material, "blueprints" and documentation available at no cost to the public. This is increasingly being applied in other fields of endeavor, such as biotechnology.

1.4. Open Source Definition



Figure 1: Open source logo

The Open Source Definition is used by the Open Source Initiative to determine whether or not a software license can be considered open source.

The definition was based on the Debian Free Software Guidelines, written and adapted primarily by Bruce Perens. They are by no means definitive even as applied to software; note that clause 3 is the primary legal difference between free software and open source software as such, free software is stricter in interpreting 3. Note also that clauses 5 and 6 are not a condition of any major open content license regimes, which commonly do restrict types of uses and users, for instance, Creative Commons has open content licenses that explicitly forbid commercial use.

Introduction

Open source doesn't just mean access to the source code.

The distribution terms of open-source software must comply with the following criteria:

1. Free Redistribution

The license shall not restrict any party from selling or giving away the software as a component of an aggregate software distribution containing programs from several different sources. The license shall not require a royalty or other fee for such sale.

2. Source Code

The program must include source code, and must allow distribution in source code as well as compiled form. Where some form of a product is not distributed with source code, there must be a well-publicized means of obtaining the source code for no more than a reasonable reproduction cost preferably, downloading via the Internet without charge. The source code must be the preferred form in which a programmer would modify the program. Deliberately obfuscated source code is not allowed. Intermediate forms such as the output of a preprocessor or translator are not allowed.

3. Derived Works

The license must allow modifications and derived works, and must allow them to be distributed under the same terms as the license of the original software.

4. Integrity of The Author's Source Code

The license may restrict source-code from being distributed in modified form only if the license allows the distribution of "patch files" with the source code for the purpose of modifying the program at build time. The license must explicitly permit distribution of software built from modified source code. The license may require derived works to carry a different name or version number from the original software.

5. No Discrimination Against Persons or Groups

The license must not discriminate against any person or group of persons.

6. No Discrimination Against Fields of Endeavor.

The license must not restrict anyone from making use of the program in a specific field of endeavor. For example, it may not restrict the program from being used in a business, or from being used for genetic research.

7. Distribution of License

The rights attached to the program must apply to all to whom the program is redistributed without the need for execution of an additional license by those parties.

8. License Must Not Be Specific to a Product

The rights attached to the program must not depend on the program's being part of a particular software distribution. If the program is extracted from that distribution and used or distributed within the terms of the program's license, all parties to whom the program is redistributed should have the same rights as those that are granted in conjunction with the original software distribution.

9. License Must Not Restrict Other Software

The license must not place restrictions on other software that is distributed along with the licensed software. For example, the license must not insist that all other programs distributed on the same medium must be open-source software.

10. License Must Be Technology-Neutral

No provision of the license may be predicated on any individual technology or style of interface.

– *Open Source Initiative*, <http://opensource.org/docs/osd>

1.4.1. Pros and Cons

Software experts and researchers on open source software have identified several advantages and disadvantages. The main advantage for business is that open source is a good way for business to achieve greater penetration of the market. Companies that offer open source software are able to establish an industry standard and, thus, gain competitive advantage. It has also helped build developer loyalty as developers feel empowered and have a sense of ownership of the end product. Moreover less costs of marketing and logistical services are needed for OSS. It also helps companies to keep abreast of all technology developments. It is a good tool to promote a company's image, including its commercial products. The OSS development approach has helped produce reliable, high quality software quickly and inexpensively. Besides, it offers the potential for a more flexible technology and quicker innovation. It is said to be more reliable since it typically has thousands of independent programmers testing and fixing bugs of the software. It is flexible because modular systems allow programmers to build custom interfaces, or add new abilities to it and it is innovative since open source programs are the product of collaboration among a large number of different programmers. The mix of divergent perspectives, corporate objectives, and personal goals speeds up innovation. Moreover free software can be developed in accord with purely technical requirements. It does not require to think about commercial pressure that often degrades the quality of the software. Commercial pressures make traditional software developers pay more attention to customers' requirements than to security requirements, since such features are somewhat invisible to the customer.

It is sometimes said that the open source development process may not be well defined and the stages in the development process, such as system testing and documentation may be ignored. However this is only true for small (mostly single programmer) projects. Larger, successful projects do define and enforce at least some rules as they need them to make the teamwork possible. In the most complex projects

these rules may be as strict as reviewing even minor change by two independent developers.

Not all OSS initiatives have been successful, for example, SourceXchange and Eazel. Software experts and researchers who are not convinced by open source's ability to produce quality systems identify the unclear process, the late defect discovery and the lack of any empirical evidence as the most important problems (collected data concerning productivity and quality). It is also difficult to design a commercially sound business model around the open source paradigm. Consequently, only technical requirements may be satisfied and not the ones of the market. In terms of security, open source may allow hackers to know about the weaknesses or loopholes of the software more easily than closed-source software. It is depended of control mechanisms in order to create effective performance of autonomous agents who participate in virtual organizations.

1.4.2. Development philosophy

In his 1997 essay *The Cathedral and the Bazaar*, open source evangelist Eric S. Raymond suggests a model for developing OSS known as the bazaar model. Raymond likens the development of software by traditional methodologies to building a cathedral, *"carefully crafted by individual wizards or small bands of mages working in splendid isolation"*. He suggests that all software should be developed using the bazaar style, which he described as *"a great babbling bazaar of differing agendas and approaches."*

In the traditional model of development, which he called the cathedral model, development takes place in a centralized way. Roles are clearly defined. Roles include people dedicated to designing (the architects), people responsible for managing the project, and people responsible for implementation. Traditional software engineering follows the cathedral model. Fred P. Brooks in his book *The Mythical Man-Month* advocates this model. He goes further to say that in order to preserve the architectural integrity of a system, the system design should be done by as few architects as possible.

The bazaar model, however, is different. In this model, roles are not clearly defined. Gregorio Robles suggests that software developed using the bazaar model should exhibit the following patterns:

Users should be treated as co-developers

The users are treated like co-developers and so they should have access to the source code of the software. Furthermore users are encouraged to submit additions to the software, code fixes for the software, bug reports, documentation etc. Having more co-developers increases the rate at which the software evolves. Linus's law states that,

"Given enough eyeballs all bugs are shallow." This means that if many users view the source code they will eventually find all bugs and suggest how to fix them. Note that some users have advanced programming skills, and furthermore, each user's machine provides an additional testing environment. This new testing environment offers that ability to find and fix a new bug.

Early releases

The first version of the software should be released as early as possible so as to increase one's chances of finding co-developers early.

Frequent integration

Code changes should be integrated (merged into a shared code base) as often as possible so as to avoid the overhead of fixing a large number of bugs at the end of the project life cycle. Some open source projects have nightly builds where integration is done automatically on a daily basis.

Several versions

There should be at least two versions of the software. There should be a buggier version with more features and a more stable version with fewer features. The buggy version (also called the development version) is for users who want the immediate use of the latest features, and are willing to accept the risk of using code that is not yet thoroughly tested. The users can then act as co-developers, reporting bugs and providing bug fixes.

High modularization

The general structure of the software should be modular allowing for parallel development on independent components.

Dynamic decision making structure

There is a need for a decision making structure, whether formal or informal, that makes strategic decisions depending on changing user requirements and other factors.

2. Current panorama in the web mapping field

Many kind of tools usable at many different levels are available across the web. Some tools are widely known, like Google Maps, Yahoo Maps or Bing Maps, others are more specific and allow to build complex web applications with advanced and customized functionalities.

Google, Yahoo and Bing allow a user to view maps, add information as punctual markers with title and description. These products can even be used to embed a map in web sites modifying interactive tools and styles.

But there are conceptual differences between those tools that identify the proper type of usage of each of them.

We can divide the web mapping panorama in categories which are different by meaning:

- public services that share static maps with few available functionalities
- open source tools that create complete WebGis interface with standard functionalities and layout, these are easy to use but hard to customize
- open source tools that adopt the bottom up strategy to build web applications, hard to use and implement but highly customizable
- commercial tools, usually really easy to use but not customizable

The choose of a type depends of the user needs and the level of data inference that is required, public services offers a flat web service, otherwise open source or commercial allow to extract various type of informations from the data.

2.1. *Public services*

2.1.1. *Google Maps*

Google Map is mainly known as a web service that shows a world map with four different types: street, satellite, hybrid and terrain. On top of the map the user can draw point with attributes such as title, description, notes and more.

However Google Maps is more than that, Google offers the Maps API (Application Programming Interface). Through the APIs is possible to embed a map in a custom web site and add many functionalities like geocoding, positioning, measurement, directions, elevation, and places information.

New versions of the service implements the Google Earth API allowing to embed directly a light version of Google Earth in a web site.

2.1.2. Bing and Yahoo Maps

Similar to Google even Microsoft with Bing and Yahoo offers web map services and APIs to embed and implement custom web sites with maps.

2.2. Open Source ready to use products

With open source products we label a wide variety of tools, there are complete WebGis applications ready to deploy and fill with own data or low level libraries which are highly customizable but which requires deep programming skills.

Between the open source packed products some are more relevant, they are ready to use and slightly customizable. The general approach is top down, indeed they offer a variety of tools such as analysis and export that can be switched on or off but that are quite hard to modify.

2.2.1. CartoWeb

CartoWeb is a comprehensive and ready-to-use Web-GIS (Geographical Information System) as well as a convenient framework for building advanced and customized applications.

Developed by Camptocamp SA, it is based on the UMN MapServer engine and is released under the GNU General Public License (GPL).

Written using innovative language PHP5, CartoWeb is highly modular and customizable thanks to its object-oriented architecture. It runs evenly on Windows or Unix-like platforms and shows its real power when associated to PostgreSQL/PostGIS.

2.2.2. Mapbender

Mapbender is an official Open Source Geospatial Foundation project and all about managing maps. It can be used for a variety of different web mapping applications. Mapbender helps to manage, organize, protect and monitor map services, users and applications. The maps themselves come from many different Map Services and can be located anywhere in the world.

2.2.3. P.mapper

The p.mapper framework is intended to offer broad functionality and multiple configurations in order to facilitate the setup of a MapServer application based on PHP/MapScript.

2.3. Open Source libraries and servers

If the desired build strategy is the bottom up, that means build a highly customized environment which perfectly fits the user need being quick and light, the option is to choose each component.

At a low level the products must be distinguished in three main categories:

- server side engines
- client interface libraries
- middleware caching systems

On The client side, as a base javascript library, the de facto standard is OpenLayers, that is used in many other libraries or toolkits.

For a more specific description about how a typical architecture works and how the components interact, refer to chapter 6.2.

2.3.1. MapServer

MapServer is an Open Source platform for publishing spatial data and interactive mapping applications to the web. Originally developed in the mid-1990's at the University of Minnesota, MapServer is released under an MIT-style license, and runs on all major platforms (Windows, Linux, Mac OS X).

The MapServer Project Steering Committee (PSC), sanctioned by OSGeo, manages and administers the project which is maintained, improved, and supported by developers from around the world. MapServer is not a full-featured GIS system, nor does it aspire to be.

2.3.2. GeoServer

GeoServer is an open source software server written in Java that allows users to share and edit geospatial data. Designed for interoperability, it publishes data from any major spatial data source using open standards.

Being a community-driven project, GeoServer is developed, tested, and supported by a diverse group of individuals and organizations from around the world.

GeoServer is the reference implementation of the Open Geospatial Consortium (OGC) Web Feature Service (WFS) and Web Coverage Service (WCS) standards, as well as a high performance certified compliant Web Map Service (WMS). GeoServer forms a core component of the Geospatial Web.

2.3.3. OpenLayers

OpenLayers makes it easy to put a dynamic map in any web page. It can display map tiles and markers loaded from any source. OpenLayers has been developed to further the use of geographic information of all kinds. OpenLayers is completely free, Open Source JavaScript, released under a BSD-style License (also known as the Clear BSD).

OpenLayers is developed and supported by a number of organizations around the world.

2.3.4. GeoExt

GeoExt brings together the geospatial know how of OpenLayers with the user interface savvy of Ext JS to help you build powerful desktop style GIS apps on the web with JavaScript.

GeoExt is available under the BSD license and is supported by a growing community of individuals, businesses and organizations.

2.3.5. Ext.Js

Ext.Js is a cross-browser JavaScript library for building rich internet applications. Build rich, sustainable web applications faster than ever.

Ext.Js includes several features such as High performance, customizable UI widgets, Well designed and extensible Component model, An intuitive, easy to use API.

2.3.6. MapFish

MapFish is a flexible and complete framework for building rich web-mapping applications. It emphasizes high productivity, and high-quality development.

MapFish is based on the Pylons Python web framework. MapFish extends Pylons with geospatial-specific functionality. For example MapFish provides specific tools for creating web services that allows querying and editing geographic objects.

MapFish also provides a complete RIA-oriented JavaScript toolbox, a JavaScript testing environment, and tools for compressing JavaScript code. The JavaScript toolbox is composed of theExtJS, OpenLayers , GeoExt JavaScript toolkits, and specific components for interacting with MapFish web services.

MapFish is compliant with the Open Geospatial Consortium standards. This is achieved through OpenLayers or GeoExt supporting several OGC norms, like WMS, WFS, WMC, KML, GML etc..

MapFish is open source, and distributed under the GPLv3 license.

2.4. Commercial tools and libraries

On the commercial side there are many systems and applications that can cover the web mapping field.

Probably the most advanced tool chain is the ESRI suite which is composed by desktop environments like ArcGis ArcMap and servers and client.

In this section will be described just the server and client tools because the desktop environment is the huge and complex GIS field.

The ESRI tool chain is designed to have a unique workflow from the desktop environment to the client interface. This system is not only oriented to data publishing and inference but even to tools publishing.

Is possible to write an analysis tool on the desktop environment and directly publish that as a service on the web through ArcGis Server. Of course this process is used to publish data.

2.4.1. ArcGis Server

ArcGIS Server software gives you the ability to create, manage, and distribute GIS services over the Web to support desktop, mobile and Web mapping applications.

ArcGIS Server simplifies access to GIS services for GIS professionals, mobile workers, as well as knowledge workers without any GIS experience. With ArcGIS Server, you stay in control of your content through centralized management of spatial data, including imagery.

In addition, ArcGIS Server provides you with a scalable GIS server platform that can be deployed on a single machine to support small workgroups, or it can distributed across

multiple servers for supporting enterprise applications. You can also deploy ArcGIS Server on Cloud infrastructure.

2.4.2. ArcGis client APIs

On the client side ESRI has a set of APIs which are designed to fit perfectly with ArcGis server but also with other mapping services.

These APIs are written in three different languages:

- javascript
- flex with flash language
- .net for microsoft silverlight framework

Being client side libraries these are open source and free to be customized.

The flex and silverlight solution require some plugins to be installed on the client computer.

3. WebGis Fields of use

The WebGis term generally is meant to be a web map tool but actually can be more. Being the system composed by several parts each of those components can be used separately.

Database remote connections allow to retrieve data directly on a gis service.

Server connections allow to embed data in external web mapping tool.

Different components of the client can be used to build both extremely complex or simple widgets embeddable in any site.

Most of the site or services now use in some ways a couple of coordinates, obtained in many different ways like for instance the IP tracing geo positioning (the method that allow to get the approximate position of the client based on its IP address and connections).

3.1. *Public services*

The number of public services that embed in such way web map widgets is constantly increasing. The georeferencing process is mostly known as “geotagging”. Social services like flickr, google mymaps, facebook have a map somewhere.

In this cases the map is just a small widget that show a point with few information but, as a concept, add many information to the service. Because even a small map has much data and information inside, the background layers are the synthesis of ground cover data, infrastructures, administrative boundaries and many others. The higher contribution of a map is done by the background data, which are hard to gather and to keep coherent to be stylized in a readable manner.

3.2. *Geoportals*

Many public administrations such as regional bureaus or province bureaus have the need to have a public web mapping viewer to share data about environment or infrastructures. In mostly cases a WebGis for a public administration is a full page service with visualization capabilities and querying tools.

3.3. Custom WebGis for specific tasks

For who works with emergencies and deal with users spread in any part of the world, the main challenge is to keep the WebGis easy to use and first of all light and capable to work with tight broadband.

The service must be built in few time and must have specific tools for analysis. In this cases the optimal solution is offered by custom built WebGis based on low level components. The building strategy is a bottom up approach, being a complex architecture to work under emergency most of the system must be created in advance to respond in time.

A custom architecture can be really simple like a small embedded map or a complex system which makes a massive use of server side analysis tools such as GIS and statistics applications.

4. ITHACA



Figure 2: ITHACA logo

ITHACA - Information Technology for Humanitarian Assistance, Cooperation and Action - is a non-profit association, founded in November 2006 by the Politecnico of Torino and the Higher Institute on Innovation Territorial Systems (Si.T.I.). It is located in the Politecnico of Torino campus.

Through its partnership with the UN World Food Programme (WFP) - the world's largest operational humanitarian agency - ITHACA is envisioned as a centre of applied research developing IT products and services in support of humanitarian activities. ITHACA is dedicated to scientific research, delivering methodologies, analytical services and technical tools which improve the capacity of WFP and the wider International community in humanitarian early warning, early impact assessment and other related areas. The initiative was realized thanks to the support of the Politecnico di Torino, specifically with DITAG - the Land, Environment and Geo-Engineering Department - and other partners such as public and private research centers, societies operating in the aerospace and telecommunications fields and other companies supplying territorial data.

ITHACA's operational team is composed of experts from the academic, industrial and scientific environments.

4.1. Area of competence

ITHACA areas of competence relate to the acquisition, management and elaboration of geographic and cartographic data. The operational team has deep experience in several areas concerning Geomatics, including Remote Sensing, Photogrammetry and Territorial Informative Systems. As such, ITHACA is able to produce basic and thematic cartography through both consolidated and innovative procedures.

Additionally, ITHACA has web development capabilities and is able to consolidate and distribute geographic data via the internet, tailoring the outputs to the specific needs of end users.

One of the main fields in which ITHACA is engaged is early warning analysis, specifically anticipating risk in areas that are potentially vulnerable to disasters. Methodologies of elaboration, integration and analysis of data deriving from various sources are deployed to anticipate the location of a catastrophic event and its possible effects on the population.

ITHACA also is using remote sensing and satellite imagery to rapidly determine the impact of a disaster in the first hours after the event occurs.

4.2. Projects

ITHACA is involved in many projects related to spatial data. The projects are related to field of early warning systems or early impact systems or both of them.

For the major part of the projects the WebGis is the front end interface by which the results are published and shared among the community.

Here are briefly presented some of them which are directly related to the WebGis systems. The web architecture is a project itself but is not going to be explained here as is the subject of the present thesis.

4.2.1. Spatial Data Infrastructure for the WFP

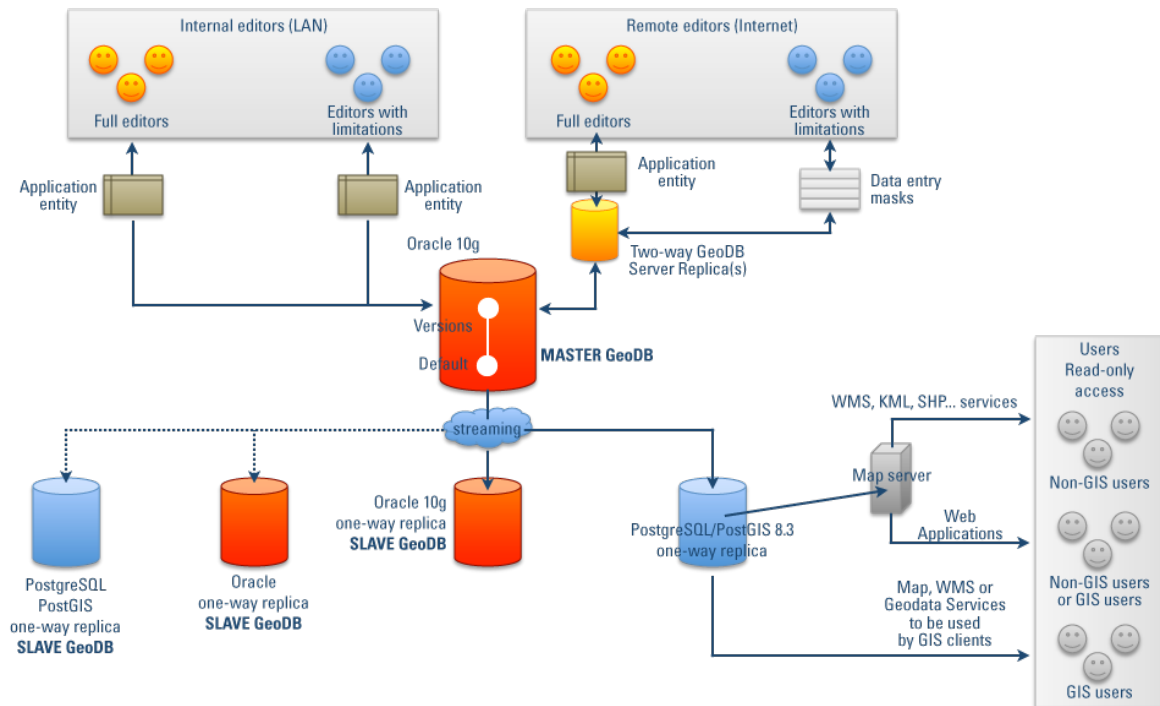


Figure 3: SDI Architecture

Accurate, easily accessible geographic information is crucial to good decision-making in humanitarian operations. The main aim of the Spatial Data Infrastructure (SDI) project is the development and implementation of a global database to store and exchange geospatial data for WFP and other actors in the humanitarian sector. The SDI, currently in development, is an efficient tool for storing, querying and manipulating large amounts of global geographic information and spatial data for analysis and visualization purposes.

The database will integrate and update data coming from multiple sources, providing each user with the latest datasets, thus avoiding duplications and mistakes. The data models will be shaped according to specific needs and demands of the user groups within the organization.

The data will be accessed through user-friendly web-based platforms (with the possibility of remote access), and it will be integrated into the automated dynamic web-based maps for operational support (Logistics Capacity Assessments, Contingency Plans, etc.)

These data will be available to GIS practitioners in-house (VAM, Emergency Preparedness and UNJLC) and, possibly, to partners. It will also provide a significant source of information for the Early Impact Analysis service, which is a parallel project undertaken by ITHACA.

The SDI acts either as base dataset for web application and high level content for thematic maps.

4.2.2. Early impact analysis

The main aim of the project is to rapidly produce geo-referenced information on the early impact of flood events, especially data on affected areas and population. This information would avoid time-consuming procedures preferring rapid mapping methodologies crucial in emergency situations. The information provided would be drawn not only from the data acquired during and after the event, but also from a database (SDI) to be used both as comparison data and as cartographic basis.

4.2.3. Early warning system

The aim of the project is the development of an early warning system based on precipitation analysis and related to historical flooded areas detection.

This target will be obtained calculating rainfall thresholds corresponding with historical flood events occurred on the field. A map based on the extent of historical affected areas will be produced if rainfall values overcome the pluviometric threshold for each specific area.

A first rough definition of the different extreme rainfall periods will be performed on the basis of the pluviometric regime derived from fifty years precipitation data with worldwide extent.

4.2.4. Floodplain modeling

The aim of the project is the development of a method to predict flood effects using hydrologic and hydraulic modeling. Several research topics are involved such as rainfall runoff modeling, river hydraulic modeling and mapping of hydraulic hazard.

4.3. Fields of intervention

Ithaca works either in preparation of an emergency and during an emergency.

The early warning system and the SDI project are the main tasks where a continuous improvement is applied, the goal is to mitigate the risk related to an emergency being prepared to face it, knowing where the event will impact and what dataset and maps are available for the specific area.

During an emergency ITHACA act as an early impact analyzer, usually the work is conducted on two main branches:

- rapid mapping production based on SDI data and any possible data gathered from the field by WFP or others
- damage assessment performed on newly acquired satellite imagery, the damage assessment can be performed visually on a single image or with stereoscopic images if available. The result of the damage assessment is time by time include on the rapid maps.

5. Emergency management

Emergency management is the discipline of dealing with and avoiding risk⁵. It is a discipline that involves preparing for disaster before it occurs, disaster response (e.g., emergency evacuation, quarantine, mass decontamination, etc.), and supporting, and rebuilding society after natural or human-made disasters have occurred. In general, any Emergency Management is the continuous process by which all individuals, groups, and communities manage hazards in an effort to avoid or ameliorate the impact of disasters resulting from the hazards. Actions taken depend in part on perception of risk of those exposed. Effective Emergency Management relies on thorough integration of emergency plans at all levels of government and non-government involvement. Activities at each level (individual, group, community) affect the other levels. It is common to place the responsibility for governmental Emergency Management with the institutions for civil defense or within the conventional structure of the emergency services. However, all emergencies are considered local, and as resources are exhausted, the next level of government is accessed for assistance. In the private sector, Emergency Management is sometimes referred to as business continuity planning.

Emergency Management is one of a number of terms which, since the end of the Cold War, have largely replaced *Civil defense*, whose original focus was protecting civilians from military attack. Modern thinking focuses on a more general intent to protect the civilian population in times of peace as well as in times of war. Another current term, *Civil Protection* is widely used within the European Union and refers to government-approved systems and resources whose task is to protect the civilian population, primarily in the event of natural and human-made disasters. Within EU countries the term Crisis Management emphasizes the political and security dimension rather than measures to satisfy the immediate needs of the civilian population. An academic trend is towards using the term disaster risk reduction, particularly for Emergency Management in a development management context. This focuses on the mitigation and preparedness aspects of the emergency cycle.

5.1. Phases

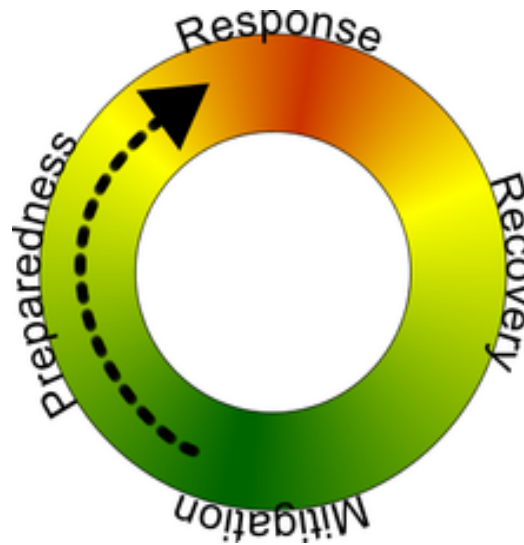


Figure 4: Emergency management cycle

The nature of management depends on local economic and social conditions. Some disaster relief experts have noted that in a sense the only real disasters are economic. Experts, have long noted that the cycle of Emergency Management must include long-term work on infrastructure, public awareness, and even human justice issues. This is not important in developing nations. The process of Emergency Management involves four phases: mitigation, preparedness, response, and recovery.

5.1.1. Mitigation

Mitigation efforts attempt to prevent hazards from developing into disasters altogether, or to reduce the effects of disasters when they occur. The mitigation phase differs from the other phases because it focuses on long-term measures for reducing or eliminating risk. The implementation of mitigation strategies can be considered a part of the recovery process if applied after a disaster occurs. Mitigative measures can be structural or non-structural. Structural measures use technological solutions, like flood levees. Non-structural measures include legislation, land-use planning (e.g. the designation of nonessential land like parks to be used as flood zones), and insurance. Mitigation is the most cost-efficient method for reducing the impact of hazards, however it is not always suitable. Mitigation does include providing regulations regarding evacuation, sanctions against those who refuse to obey the regulations (such as mandatory

evacuations), and communication of potential risks to the public. Some structural mitigation measures may have adverse effects on the ecosystem.

A precursor activity to the mitigation is the identification of risks. Physical risk assessment refers to the process of identifying and evaluating hazards. The hazard-specific risk (R_h) combines both the probability and the level of impact of a specific hazard. The equation below states that the hazard multiplied by the populations' vulnerability to that hazard produces a risk. Catastrophe modeling. The higher the risk, the more urgent that the hazard specific vulnerabilities are targeted by mitigation and preparedness efforts. However, if there is no vulnerability there will be no risk, e.g. an earthquake occurring in a desert where nobody lives.

5.1.2. Preparedness

Preparedness is a continuous cycle of planning, organizing, training, equipping, exercising, evaluation and improvement activities to ensure effective coordination and the enhancement of capabilities to prevent, protect against, respond to, recover from, and mitigate the effects of natural disasters, acts of terrorism, and other man-made disasters.

In the preparedness phase, emergency managers develop plans of action to manage and counter their risks and take action to build the necessary capabilities needed to implement such plans. Common preparedness measures include:

- communication plans with easily understandable terminology and methods.
- proper maintenance and training of emergency services, including mass human resources such as community emergency response teams.
- development and exercise of emergency population warning methods combined with emergency shelters and evacuation plans.
- stockpiling, inventory, and maintain disaster supplies and equipment[8]
- develop organizations of trained volunteers among civilian populations.

Professional emergency workers are rapidly overwhelmed in mass emergencies so trained, organized, responsible volunteers are extremely valuable. Organizations like Community Emergency Response Teams and the Red Cross are ready sources of trained volunteers. The latter's emergency management system has gotten high ratings from both California, and the Federal Emergency Management Agency (FEMA).

Another aspect of preparedness is casualty prediction, the study of how many deaths or injuries to expect for a given kind of event. This gives planners an idea of what resources need to be in place to respond to a particular kind of event.

Emergency Managers in the planning phase should be flexible, and all encompassing - carefully recognizing the risks and exposures of their respective regions and employing unconventional, and atypical means of support. Depending on the region - municipal, or private sector emergency services can rapidly be depleted and heavily taxed. Non-governmental organizations that offer desired resources, i.e., transportation of displaced homeowners to be conducted by local school district buses, evacuation of flood victims to be performed by mutual aid agreements between fire departments and rescue squads, should be identified early in planning stages, and practiced with regularity.

5.1.3. Response

The response phase includes the mobilization of the necessary emergency services and first responders in the disaster area. This is likely to include a first wave of core emergency services, such as firefighters, police and ambulance crews. When conducted as a military operation, it is termed Disaster Relief Operation (DRO) and can be a follow-up to a Non-combatant evacuation operation (NEO). They may be supported by a number of secondary emergency services, such as specialist rescue teams.

A well rehearsed emergency plan developed as part of the preparedness phase enables efficient coordination of rescue. Where required, search and rescue efforts commence at an early stage. Depending on injuries sustained by the victim, outside temperature, and victim access to air and water, the vast majority of those affected by a disaster will die within 72 hours after impact.

5.1.4. Recovery

The aim of the recovery phase is to restore the affected area to its previous state. It differs from the response phase in its focus; recovery efforts are concerned with issues and decisions that must be made after immediate needs are addressed. Recovery efforts are primarily concerned with actions that involve rebuilding destroyed property, re-employment, and the repair of other essential infrastructure. Efforts should be made to "build back better", aiming to reduce the pre-disaster risks inherent in the community and infrastructure. An important aspect of effective recovery efforts is taking advantage of a 'window of opportunity for the implementation of mitigative measures that might otherwise be unpopular. Citizens of the affected area are more likely to accept more mitigative changes when a recent disaster is in fresh memory.

5.2. User needs

On the 19 of January 2010 an earthquake has stroke the country of Haiti.

The magnitude was near 7 degrees of the Richter lather, and the effects on the population were impressive. In Port-au-Prince but even elsewhere the houses were built using the sea sand. The presence of the salt inside the walls of the houses has weakened the structures, bringing them to be highly sensitive to vibrations.

ITHACA started to work for emergency in the early morning of the 20th, producing damage assessments of buildings and monitoring the evolution of the spontaneous camps.

After few days a staff composed by three people was requested to deploy on the field in Port-au-Prince. Two my colleagues and I arrived in Port-au-Prince on 15th of February 2010.

The situation appeared immediately critic, the Toussant Louverture airport, where the UN and Minustah have their logistic base, was extremely chaotic. In an environment warm and full of smog and dust the organization is extremely difficult.

We are been engaged to retrieve and manage spatial data and share them among the community of the UN agencies located in the log base.

In the first phases of an emergency the needs of spatial data are related to basic cartography such as administrative limits, departments and street names, or airport - landing fields position. This data is expected to be accurate and up to date.

The data retrieval is the most challenging part, the earthquake destroyed the most part of administrative buildings and so the most part of the databases or archives. The knowledge of the environment is spread among many people and institutions, to get updated and reliable data, accurate interviews must be conducted.

Having an updated database is crucial to attend the user needs but this take too much time to get, check and choose the correct data. The use of a social tool to make people collaborate in data creation and maintenance can be a help even if this solution require coordination and volunteers.

In advanced phases of the emergency, when the recovery starts, more complex products are requested. In this phase analysis over the data should be performed, this implies that the database is expected to be consolidated and operative.

6. Principles of Web services

Before the definition of a web service, an overview of what is internet and which main communication systems it use, is needed.

6.1. *Internetwork*

Internet is a short form of the technical term "internetwork", the result of interconnecting computer networks with special gateways (routers).

The terms Internet and World Wide Web are often used in everyday speech without much distinction. However, the Internet and the World Wide Web the same. The Internet is a global data communications system. It is a hardware and software infrastructure that provides connectivity between computers. In contrast, the Web is one of the services communicated via the Internet. It is a collection of interconnected documents and other resources, linked by hyperlinks and URLs.

6.1.1. *Low Level Protocols*

The complex communications infrastructure of the Internet consists of its hardware components and a system of software layers that control various aspects of the architecture. While the hardware can often be used to support other software systems, it is the design and the rigorous standardization process of the software architecture that characterizes the Internet and provides the foundation for its scalability and success. The responsibility for the architectural design of the Internet software systems has been delegated to the Internet Engineering Task Force (IETF). The IETF conducts standard-setting work groups, open to any individual, about the various aspects of Internet architecture. Resulting discussions and final standards are published in a series of publications, each called a Request for Comments (RFC), freely available on the IETF web site. The principal methods of networking that enable the Internet are contained in specially designated RFCs that constitute the Internet Standards.

The Internet Standards describe a framework known as the Internet Protocol Suite. This is a model architecture that divides methods into a layered system of protocols. The layers correspond to the environment or scope in which their services operate. At the top is the Application Layer, the space for the application-specific networking methods used in

software applications, e.g., a web browser program. Below this top layer, the Transport Layer connects applications on different hosts via the network (e.g., client–server model) with appropriate data exchange methods. Underlying these layers are the core networking technologies, consisting of two layers. The Internet Layer enables computers to identify and locate each other via Internet Protocol (IP) addresses, and allows them to connect to one-another via intermediate (transit) networks. Lastly, at the bottom of the architecture, is a software layer, the Link Layer, that provides connectivity between hosts on the same local network link, such as a local area network (LAN) or a dial-up connection. The model, also known as TCP/IP, is designed to be independent of the underlying hardware which the model therefore does not concern itself with in any detail.

The most prominent component of the Internet model is the Internet Protocol (IP) which provides addressing systems (IP addresses) for computers on the Internet. IP enables internetworking and essentially establishes the Internet itself. IP Version 4 (IPv4) is the initial version used on the first generation of the today's Internet and is still in dominant use. It was designed to address up to ~4.3 billion (10^9) Internet hosts. However, the explosive growth of the Internet has led to IPv4 address exhaustion which is estimated to enter its final stage in approximately 2011. A new protocol version, IPv6, was developed in the mid 1990s which provides vastly larger addressing capabilities and more efficient routing of Internet traffic. IPv6 is currently in commercial deployment phase around the world and Internet address registries (RIRs) have begun to urge all resource managers to plan rapid adoption and conversion.

IPv6 is not interoperable with IPv4. It essentially establishes a "parallel" version of the Internet not directly accessible with IPv4 software. This means software upgrades or translator facilities are necessary for every networking device that needs to communicate on the IPv6 Internet. Most modern computer operating systems are already converted to operate with both versions of the Internet Protocol. Network infrastructures, however, are still lagging in this development.

6.1.2. High Level Protocols

Over the transport and addressing layers there is the transfer or application protocol, the protocol used depends of the type of data transmitted. The two most used probably are the HTTP and the FTP.

HTTP

The Hypertext Transfer Protocol (HTTP) is a networking protocol for distributed, collaborative, hypermedia information systems. HTTP is the foundation of data communication for the World Wide Web.

HTTP functions as a request-response protocol in the client-server computing model. In HTTP, a web browser, for example, acts as a client, while an application running on a computer hosting a web site functions as a server. The client submits an HTTP request message to the server. The server, which stores content, or provides resources, such as HTML files and images, or generates such content as required, or performs other functions on behalf of the client, returns a response message to the client. A response contains completion status information about the request and may contain any content requested by the client in its message body.

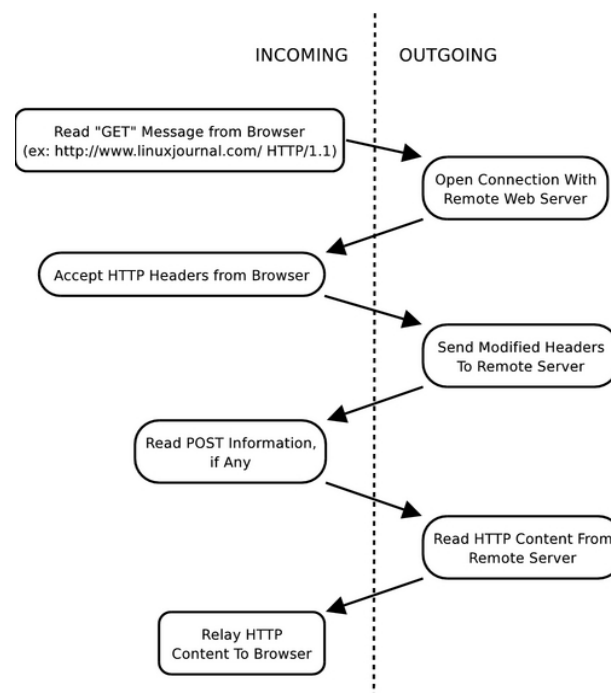


Figure 5: Http messages schema

HTTP is an Application Layer protocol designed within the framework of the Internet Protocol Suite. The protocol definitions presume a reliable Transport Layer protocol for host-to-host data transfer.

HTTP Resources are identified and located on the network by Uniform Resource Identifiers (URIs)—or, more specifically, Uniform Resource Locators (URLs)—using the `http` or `https` URI schemes. URIs and the Hypertext Markup Language (HTML), form a system of inter-linked resources, called hypertext documents, on the Internet, that led to the establishment of the World Wide Web in 1990 by English physicist Tim Berners-Lee.

Http request methods

HTTP defines nine methods (sometimes referred to as "verbs") indicating the desired action to be performed on the identified resource. What this resource represents, whether pre-existing data or data that is generated dynamically, depends on the implementation of the server. Often, the resource corresponds to a file or the output of an executable residing on the server.

Some methods are not implemented by most servers and others are used for specific tasks. The two more used for internet requests are the GET and the POST methods.

GET: Requests a representation of the specified resource. Requests using GET (and a few other HTTP methods) "SHOULD NOT have the significance of taking an action other than retrieval". Has a limitation of 255 characters. A GET request is the typical request performed while writing an address in the URL bar in a browser.

POST: Submits data to be processed (e.g., from an HTML form) to the identified resource. The data is included in the body of the request. This may result in the creation of a new resource or the updates of existing resources or both. Allow arbitrary data length.

FTP

File Transfer Protocol (FTP) is a standard network protocol used to copy a file from one host to another over a TCP/IP-based network, such as the Internet. FTP is built on a client-server architecture and utilizes separate control and data connections between the client and server. FTP users may authenticate themselves using a clear-text sign-in protocol but can connect anonymously if the server is configured to allow it.

6.2. Web services

A web service is typically an application programming interface (API) or Web API that is accessed via Hypertext Transfer Protocol (HTTP) and executed on a remote system, hosting the requested service. Web services tend to fall into one of two camps: big web services and RESTful web services.

The W3C (World Wide Web Consortium) defines a "web service" as "a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically Web Services Description Language WSDL). Other systems interact with the web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards."

The W3C also states, "We can identify two major classes of Web services, REST-compliant Web services, in which the primary purpose of the service is to manipulate XML representations of Web resources using a uniform set of "stateless" operations; and arbitrary Web services, in which the service may expose an arbitrary set of operations."

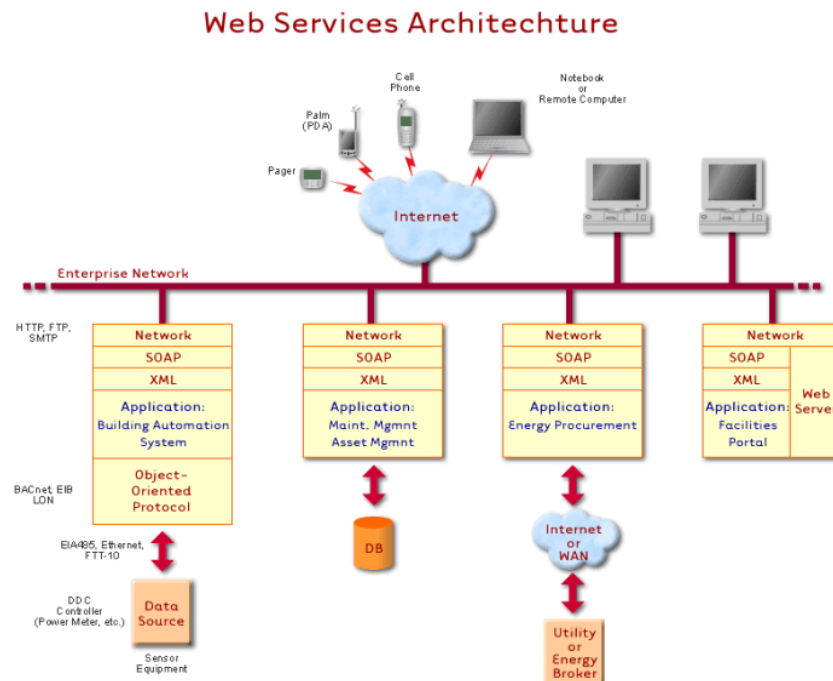


Figure 6: Web services architecture

7. Programming Languages

To build any complex software architecture the knowledge of one or more programming languages is a requisite. In the field of the web, more than one language is needed.

Actually we can divide the languages in three main categories:

- Markup languages
- Scripting languages
- Compiled or semi compiled languages

Markup languages are those that are based on the concept of tag, the tag is a marker that identifies what comes next. The HTML and the XML are markup languages.

Scripting languages are those that are read by the interpreter as they are written, without intermediate steps. An example is the Javascript which is executed by the browser without any changes on the source.

Compiled or semi compiled languages, however, get one or more steps before being executed. The process is called compiling and consists on a series of operations that translate the written code to a machine readable language. Usually this is achieved in two steps, the first is done by the language's virtual machine which translate the code in a low level language and then another interpreter reduces the code to a sequence of bits ready to be read by the operating system. Examples are Java, Python, C++, C# and more.

The advantage of a compiled language is usually due to a major speed of execution and to an increase of stability.

In web applications, server side languages are responsible for dynamic operations, such as database connection, HTML generation, proxy calls.

In the current panorama are present many different languages with different pros and cons or functionalities.

For the client side there's not many choices because the browser can read just HTML and Javascript without additional plugins.

For the server side part there are several possibilities such as Java, Python, Ruby, PHP, C#, VB, Scala and more.

The choice of a language is a process which involves personal preferences, language scalability, modernity, developer community, related products to enhance performances and productivity and portability among different platforms.

It has been chosen Python as main language for some reasons:

- Clean syntax, i.e. no need for columns or cycle ends
- Great community of developers even for related products
- Possibility to use it with most GIS like ArcGis and GRASS because they have a Python interface
- Many related projects like frameworks and libraries
- Rapidity of execution, the base libraries are written in C which is extremely quick
- Complete portability among different systems

7.1. Markup Languages

Markup languages are based on the tag concept. The main languages are:

- HTML
- XML
- XHTML
- CSS (not a proper tagging language)

7.1.1. HTML



Figure 7: HTML snippet

The HyperText Markup Language is used to build web pages. It defines the content of a web page by declaring logical block. It has been released in many versions by the W3C (World Wide Web Consortium). In the latest year the version 5.0 has been released but most of the sites are still written in the 4.01 version (released on 24th December 1999).

The tags are in the form `<tag attribute>Content</tag>`, each tag must be closed. The basic structure of a web page is divided in two main parts, the head which is not displayed in the page and which hold information about the page itself, like title, keywords, language used, the body is the content of the page which will be displayed in the page.

The attributes of a tag can describe the fonts, the colors, the id of the component, it's class and others. It must be noticed that using the style attributes like font and colors is a deprecated practice, this due to the fact that the style of a web page must be declared with the Cascading Style Sheets which decouple the style from the content.

The HTML is a really simple language which has limitations such as the absence of dynamic, with HTML is impossible to make cycles, choices, or dynamical modifications of the structure of the page.

7.1.2. XML

```
<Books>
  <Book ISBN="0553212419">
    <title>Sherlock Holmes: Complete Novels...
    <author>Sir Arthur Conan Doyle</author>
  </Book>
  <Book ISBN="0743273567">
    <title>The Great Gatsby</title>
    <author>F. Scott Fitzgerald</author>
  </Book>
  <Book ISBN="0684826976">
    <title>Undaunted Courage</title>
    <author>Stephen E. Ambrose</author>
  </Book>
  <Book ISBN="0743203178">
    <title>Nothing Like It In the World</title>
    <author>Stephen E. Ambrose</author>
  </Book>
</Books>
```

Figure 8: XML snippet

The eXtensible Markup Language is actually a meta language, it is used to build other markup languages. It is not used to build web pages and neither has predefined tags. XML consist of a series of rule or specifics on how to create another language, these rules are written by the W3C.

The XML has been created to define once a standard which could be read by any machine. Each XML document must follow rules for the correctness of the tags.

The definition of a language based on XML rules consist on the creation of a series of declarations where to define the type of the tag and the allowed content. These rules are stored in a dictionary named *dtd* (Document Type Definition).

In the first line of the new XML file is put the address of the *dtd* and so the document parser can know which tags are allowed in the structure then.

7.1.3. XHTML

The eXtensible Hypertext Markup Language is the application of the XML rules to the HTML. The tag must be lowercase, properly closed, some tag are been deprecated and the language is more oriented to style and content separation.

The current version is the 1.1 but with the HTML 5 the differences are been removed among the languages.

7.1.4. CSS



Figure 9: CSS snippet

The Cascading Style Sheet is not a proper markup language, neither a language itself, the CSS is a series of rules which apply to the components that the browser create following the HTML.

CSS define the entire stile of a component, from the font and colors to the position in the page.

The rules target the components using their id, class or their name, so it is important to define well the structure of the page with appropriate names and class of the parts.

7.2. Scripting languages

Scripting language are interpreted directly by the parser without intermediate steps.

The most important is probably the Javascript because is the only language that is interpreted by the browsers without any plugins, browser languages are commonly called client languages.

7.2.1. Javascript



Figure 10: Javascript snippet

Javascript is the most important language for the purpose of this thesis because it is responsible for all the client side functionalities like map movement, zoom, buttons, remote queries, scalebars, rulers, tooltips.

JavaScript is primarily used in the form of client-side JavaScript, implemented as part of a web browser in order to provide enhanced user interfaces and dynamic websites. However, its use in applications outside web pages for example in PDF-documents, site-specific browsers and desktop widgets (i.e. Adobe Air) is also significant.

JavaScript supports all the structured programming syntax in C (i.e., if statements, *while* loops, *switch* statements, etc.). Like C, JavaScript makes a distinction between expressions and statements. One syntactic difference from C is automatic semicolon insertion, in which the semicolons that terminate statements can be omitted (but strongly suggested).

Features of Javascript are:

- Dynamic typing (a variable can host and change dynamically the type of the data)
- Object based (Javascript is completely object oriented, however it doesn't have the class concept)
- Prototype based (it uses the prototype object to simulate classes and methods)

Client side interpretation means that is the browser responsible to read and execute the language. The client interpreter is not unique, each browser producer has developed its Javascript engine and each engine has its own differences in interpretation. Specifically if browsers like Google Chrome, Apple Safari, Mozilla Firefox have similar way to execute the language, Microsoft Internet Explorer has many differences. To face these anomalies is strongly suggested to use public libraries which wraps low level issues in high level tools. A separate chapter is further dedicate to Javascript libraries as they are crucial for mapping web apps.

Javascript is also used for other scopes: AJAX (Asynchronous Javascript And Xml) is a technique based on a Javascript object called XMLHttpRequest that is able to make remote calls to the server.

This method is used to add dynamic content to a web page without the need to refresh it. Browsers have different implementation of the XMLHttpRequest method so the use of a javascript compatibility library is always strongly suggested.

Another use of the javascript is the JSON (JavaScript Object Notation). This technique is spreading much in the web and is used to exchange information across the web. It 's based on a Javascript structure called *Object* or *Hash*. The concept of *Hash* is common to most languages even if has different names(dictionary, key-values pairs). An *Object* is similar to an array but instead of a single values has a key - value pair, the

value can host each type of data (functions, strings, numbers or even other Objects). The form of an Object is:

```
dict = {  
    key1: value1,  
    key2:[element1, element2, element3],  
    key3: function(){function code},  
    key4: {  
        subkey1: value x,  
        subkey2: number  
    }  
}
```

Each key is a sort of label and the value is the content of the dictionary. Values are accessed with square brackets such as *dict[key2]*.

The JSON notation can be used to transport configuration values for a web service or even to transport functions, in this case the hash takes the form of a library.

The server side language can read and write JSON from and to its own hash notation using its JSON modules.

7.2.2. PHP

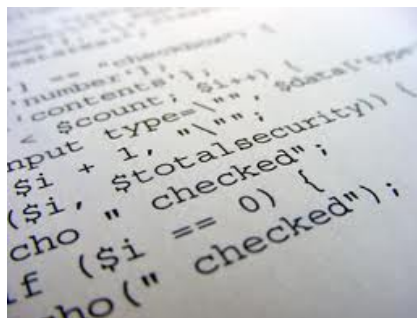


Figure 11: PHP snippet

PHP : Hypertext Preprocessor is a server side, object oriented language born in the 1994 as a series of script called Personal Home Page, then moved and extended to PHP.

This language is interpreted by the server which sends back clean HTML to the client.

A typical web architecture based on PHP is called: LAMP (Linux, Apache, MySql, PHP), where Apache is the web server and MySql the database.

PHP is an object oriented language, however is not born as object oriented but the feature was added later to face the most modern languages.

7.3. *Compiled or semi compiled languages*

The difference between compiled and semi compiled is that the first are converted in a language directly readable by the machine, the second are compiled in a language called Byte-Code that must be read by a proper virtual machine. Example of the first group are C, C++, of the second are, Java, Python, C#....

Being the compiled languages generally low level, they are used to build basic software libraries, system components or complex software. They are more complicated to use but are more quick than others.

For the purpose of the web applications compiled languages are not used because the high level languages (semi - compiled) are more simple and have many frameworks to enhance they capabilities.

Core concepts of the high level programming languages are:

- Object oriented programming, elements are actually objects instantiated from classes with attributes and methods
- Inheritance, classes and thus objects can inherit from each other avoiding code duplication
- Encapsulation, objects have their own mechanics implemented internally, the way they interact with the environment is not concern of the programmer who can create it's own objects inheriting from the base types

7.3.1. *Python*



Figure 12: Python snippet

Python is an high level programming language created by Guido Van Rossum in the beginning of 90's, takes his name from the tv series Monty Python Flying Circus.

Python is multi paradigm (objet oriented, structural or functional) and multi platform (windows, linux, mac OSX).

Has different implementations that allow to use it with the Java virtual machine or the .NET platform, but its base libraries are written in C so it is very quick comparing to other high level languages.

Is dynamically typed and strongly typed (variables can change the type of content but the control of the content is strict).

Currently is distributed in two versions, the 3.X and the 2.x.

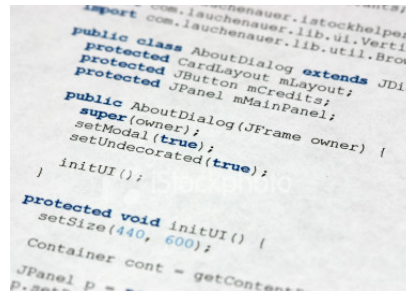
3.x is the newest branch of Python and the intended future of the language. Guido van Rossum decided to clean up Python 2.x properly, with less regard for backwards compatibility than is the case for new releases in the 2.x range. This allowed several aspects of the core language to be adjusted to be easier for newcomers to learn and to be more consistent with the rest of the language. It also allowed later language features (such as iterators) to be applied to older language features.

Linux, Mac still use the 2.x version and this is a constraint for the servers which must be as stable as possible so is not suggested to bring up non tested environments for production. The current release is the 2.7 but the most used is still the 2.6.

Python has many features of strength:

- easy and clean syntax (indentation describes the end of the blocks)
- huge amount of libraries and frameworks to develop on (i.e Django, see chapter ---)
- many GIS software both commercial and not such as ESRI ArcGis or QGIS or GRASS have a python interface, through which is possible to use the GIS functions inside its own program
- great community behind which continuously maintain, develop and release new features

7.3.2. Java



```
import com.lauchenauer.istockhelper.  
import com.lauchenauer.lib.ui.Vertical;  
public class AboutDialog extends JDialog  
protected CardLayout mLayout;  
protected JButton mCredits;  
protected JPanel mMainPanel;  
public AboutDialog(JFrame owner) {  
    super(owner);  
    setModal(true);  
    setUndecorated(true);  
    initUI();  
}  
protected void initUI() {  
    setSize(440, 600);  
    Container cont = getContentPane();  
    JPanel p = ...
```

Figure 13: Java snippet

Java is a programming language originally developed by James Gosling at Sun Microsystems and released in 1995 as a core component of Sun Microsystems' Java platform. The language derives much of its syntax from C and C++ but has a simpler object model and fewer low-level facilities. Java applications are typically compiled to bytecode (class file) that can run on any Java Virtual Machine (JVM) regardless of computer architecture. Java is a general-purpose, concurrent, class-based, object-oriented language that is specifically designed to have as few implementation dependencies as possible. It is intended to let application developers "write once, run anywhere". Java is currently one of the most popular programming languages in use, and is widely used from application software to web applications.

Java is probably the most used in enterprise environment even because many hardware vendors have JAVA certifications for their products.

The extreme strict constraints of the JAVA syntax and declarations make it perfect for big projects where the maintaining of the control in each piece is crucial. However Java suffers some performance issues comparing to python or low level languages.

8. Javascript libraries

Javascript is the client interpreted language, browsers don't need plugins or additional components to interpret it and is the core of all the dynamic of a web page. Javascript can change part of the page, modify the content, add graphical effects, perform http requests and interact with the browser.

However browsers have many different javascript engines which have different interpretation rules of objects.

To face these differences and to avoid the programmer to take care of low level repetitive tasks are borne some libraries that can be called inside the web site and used to enhance the programming experience.

To understand how the javascript works and how the libraries can interact with the page it has to be explained the concept of browser events and the *Document Object Model* (DOM).

External libraries are added to the page using an HTML tag in the *head*, like `<script src="path" />`

8.1. Browser events

To add interesting functionality to a web-page, just being able to inspect or modify the document is generally not enough. We also need to be able to detect what the user is doing, and respond to it. For this, we will use a thing called event handlers. Pressed keys are events, mouse clicks are events, even mouse motion can be seen as a series of events.

The way browser events work is, fundamentally, very simple. It is possible to register handlers for specific event types and specific DOM nodes. Whenever an event occurs, the handler for that event, if any, is called. For some events, such as key presses, knowing just that the event occurred is not good enough, you also want to know which key was pressed. To store such information, every event creates an event object, which the handler can look at.

It is important to realize that, even though events can fire at any time, no two handlers ever run at the same moment. If other JavaScript code is still running, the browser waits until it finishes before it calls the next handler. In programmer jargon, browser JavaScript is single-threaded, there are never two 'threads' running at the same time.

An event, when not handled, can 'bubble' through the DOM tree. What this means is that if you click on, for example, a link in a paragraph, any handlers associated with the link are called first. If there are no such handlers, or these handlers do not indicate that they have finished handling the event, the handlers for the paragraph, which is the parent of the link, are tried. After that, the handlers for *document.body* get a turn. Finally, if no JavaScript handlers have taken care of the event, the browser handles it. When clicking a link, this means that the link will be followed.

Javascript libraries usually simplify the task of event handling unifying methods and enhancing cross browser compatibility, even if the browsers have different implementations with different object of this part.

8.2. DOM

The DOM is a cross-platform and language-independent convention for representing and interacting with objects in HTML, XHTML and XML documents. Aspects of the DOM (such as its "Elements") may be addressed and manipulated within the syntax of the programming language in use (javascript, flash or others).

The public interface of a DOM is specified in its application programming interface (API).

All of the properties, methods, and events available to the web developer for manipulating and creating web pages are organized into objects (i.e., the document object that represents the document itself, the table object that represents a HTML table elements, and so forth). Those objects are accessible via scripting languages in most recent web browsers.

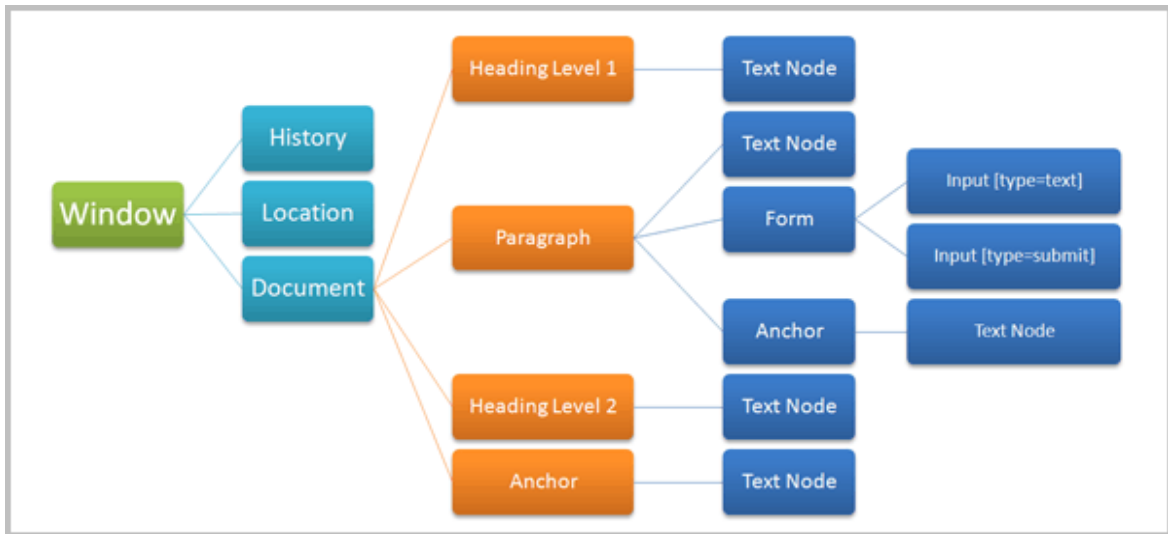
The DOM is most often used in conjunction with JavaScript. The code is written in JavaScript, but it uses the DOM to access the web page and its elements. However, the DOM was designed to be independent of any particular programming language, making the structural representation of the document available from a single, consistent API.

The World Wide Web Consortium establishes a standard for the DOM, called the W3C DOM. Now that the most important browsers should correctly implement it, enable powerful cross-browser applications.

The structure of the DOM in a browser is tree type, once the HTML file is read, the browser create nested elements starting from the main which is the window object. The head and the body part of the HTML file become two child of the window object and each nested tag inside the HTML will become a nested child with it's own names and attributes.

With javascript is possible to fetch or add / remove a single component from that structure.

Figure 14: Document object model representation



8.3. Non geographic javascript libraries

These libraries are general purpose for web development, each have it's pros and cons.

They general concept is the creation of a namespace in which wrap each function and object. They are class oriented so object are instantiated using the *new* keyword.

8.3.1. Ext.js

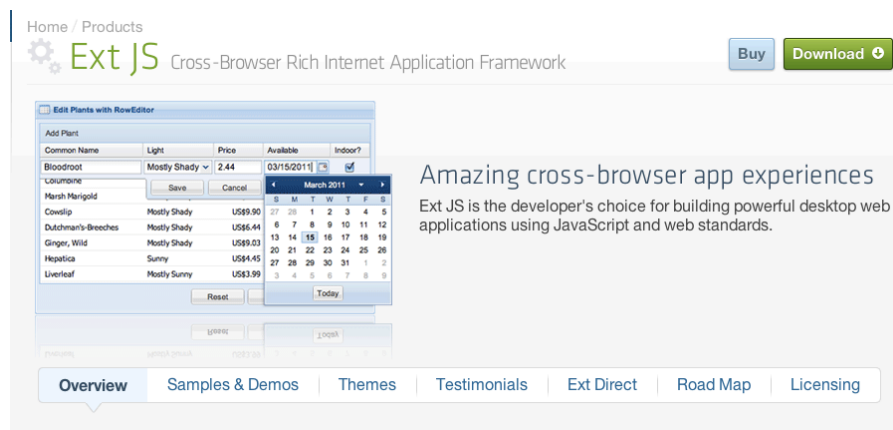


Figure 15: EXT.js home - <http://www.sencha.com/products/extjs/>

This is a complete and complex (now at version 3.3.3) library for rich web application buildings. It's APIs include:

- High performance, customizable UI widgets
- Well designed and extensible Component model
- Excellent tutorials and documentation

The UI is strongly enhanced, the creation of windows, panels, tabs or buttons is intuitive and simplified as well as DOM interaction.

However the learning curve of this library is quite steep and to be productive some initial training is mandatory.

The objects are configured using JSON notations, for example to create a window the syntax should appear like:

```
var win = new Ext.Window(  
    {  
        width: 300,  
        height: 300,  
        title: "my new window"  
        html: "the inner html to fill the window with"  
    }  
);
```

Available parameters are many such as event handling and it is even possible to create its own extending the library.

8.3.2. jQuery



Figure 16: jQuery home - <http://jquery.com/>

jQuery is a toolkit that gives you an easy to use collection of methods in a self-contained system designed to make the DOM itself more pleasant.

jQuery focus is attention of code conciseness and simplicity, in fact its learning curve is smooth and to get started with a dynamic web page few minutes and lines of code are sufficient. Current release is 1.4.4.

Particular strengths are:

- easy DOM interaction
- enhanced visual effects
- performances
- extensible with the UI component
- jQuery is mainly designed to increase the quality of the javascript written improving the code and reducing the amount of lines written.

8.3.3. MooTools

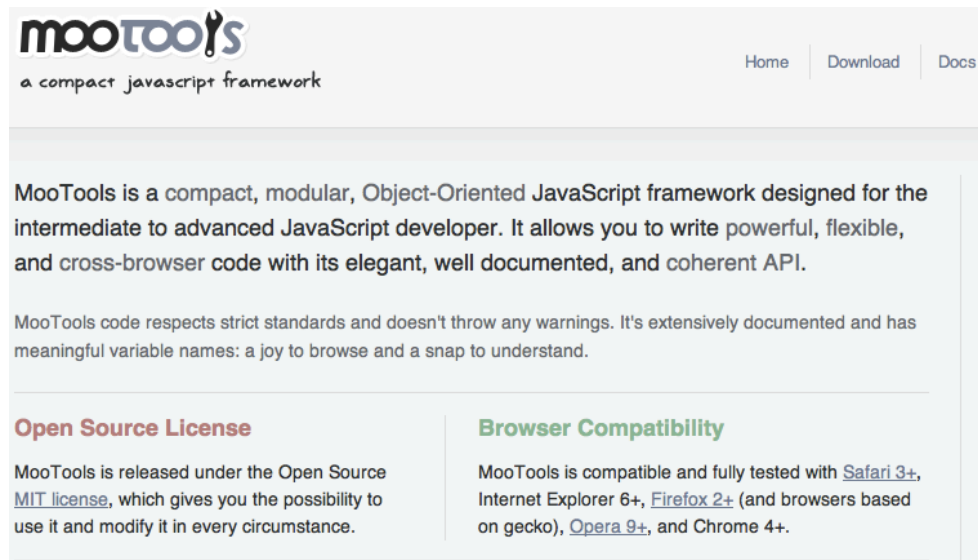


Figure 17: MooTools home - <http://mootools.net/>

MooTools is designed to write object oriented javascript and is suggested to an intermediate level to above. The aim is to implement an API that feels like javascript and enhances everything, not just the DOM.

MooTools has a quite steep learning curve compared to jQuery, this because its intent is to be a framework used to build complex javascript applications from the base and not just to enhance the code syntax.

8.4. Geographic libraries

Among the panorama of js libraries there are two in particular which are dedicated to create geographic web applications.

OpenLayers and GeoExt are different but consecutive, in fact actually GeoExt uses OpenLayers to work.

8.4.1. OpenLayers

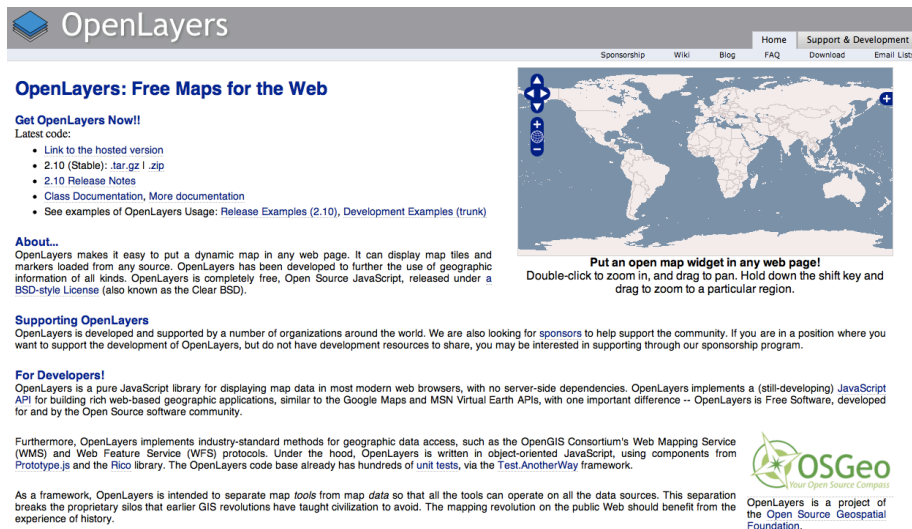


Figure 18: OpenLayers home - <http://openlayers.org/>

OpenLayers makes it easy to put a dynamic map in any web page. It can display map tiles and markers loaded from any source. OpenLayers has been developed to further the use of geographic information of all kinds. OpenLayers is completely free, Open Source JavaScript, released under a BSD-style License (also known as the Clear BSD). Current release is 2.10.

OpenLayers is developed and supported by a number of organizations around the world.

It is a framework that can be used to build widgets, like map objects, rulers, zoom tools, query tool and more. However it has functions to coordinate translations between hundred reference systems and can on the fly reproject vectors.

Strengths are:

- lightweight and customizable APIs
- re - projection
- standard OGC capabilities
- feature styling
- geographical and logical filters
- remote services decoding capabilities
- geographical events handling such as zoom, move, query

The main leak is due to the absence of UI capabilities but this is a specific choice of the development team, the purpose is to build a geographical object model which can be handled by others libraries.

8.4.2. GeoExt

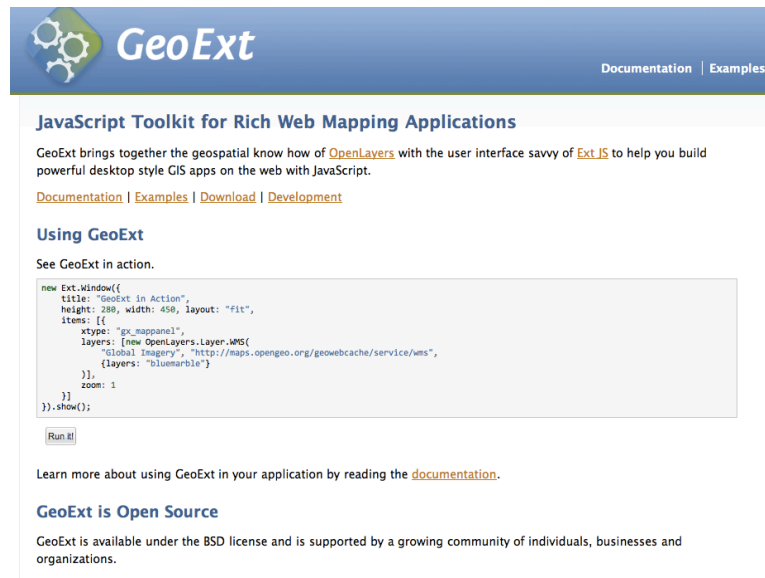


Figure 19: GeoExt home - <http://geoext.org/>

GeoExt brings together the geographical model of OpenLayers with the user interface capabilities of Ext.js. This library is not just a bridge between the two, it creates new functionalities and widgets ready to implement.

Features are:

- Legend creator
- Layer opacity management
- Features popups
- Printing capabilities
- OGC capabilities reader
- Forms and search tools
- Layers management

9. Spatial database

The database is a software specialized in data keeping and consistency. Usually a database can manage a huge amount of data. Modern database follow the relational model.

The way a user interact with the database is querying it with the SQL (Standard Query Language), either for reading or defining a table structure. However the SQL can become difficult to be managed in case of complex queries, some tools help to manage these wrapping the SQL in other easier languages (i.e. Django ORM, see chapter 13).

Probably the most famous commercial database is Oracle but the open source panorama has some valid alternatives that can be used.

MySQL or PostgreSQL are the major open source database, both of them have a spatial extension but Postgres has the most complete implementation called PostGIS.

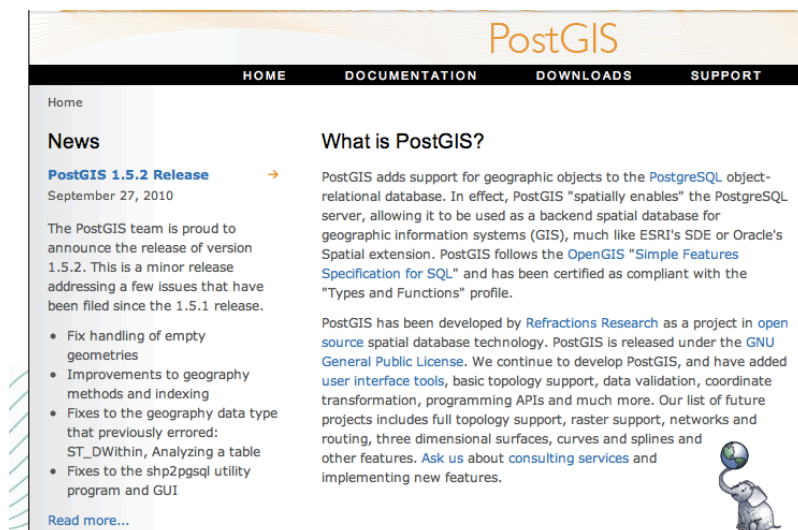


Figure 20: PostGIS home - <http://postgis.refractory.net/>

A database stores the data in a table structure, each column has its data type allowed (i.e. string, number, date), a spatial extension add data type and functions to the table schema. Spatial columns type can be:

- point and multi point
- line and multi line
- polygon and multi polygon

The db knows spatial reference systems as well and can operate transformation on data. The way the data is store in the table is to have, for each node of the element, a couple of coordinates with constraint defined by the data type (i.e. for the polygon element the first and the last point must be the same).

The built in functions allow to perform calculations directly at the database level:

- ### 9.1. Relational model



Data are operated upon by means of a relational calculus or relational algebra, these being equivalent in expressive power.

62

access plans and other implementation and operation details are handled by the DBMS engine, and are not reflected in the logical model. This contrasts with common practice for SQL DBMSs in which performance tuning often requires changes to the logical model.

10. Metadata and geospatial standards

Metadata is defined as data about the data, are commonly found in libraries and catalogues and describe three aspects of the data:

- definition
- structure
- administration

Geospatial metadata (also geographic metadata, or simply metadata when used in a geographic context) is a type of metadata that is applicable to objects that have an explicit or implicit geographic extent, in other words, are associated with some position on the surface of the Globe.

The guidelines are described in the ISO 19115 and the purpose is to create a common language readable by human and machine in order to allow easy sharing and searching for data.

The schema implementation is described in ISO 19139 provides the XML schema for ISO 19115 specifying the metadata record format and may be used to describe, validate, and exchange geospatial metadata prepared in XML.

10.1. Inspire

In 2007 has been released the INSPIRE directive establishing an infrastructure for spatial information in Europe to support Community environmental policies, and policies or activities which may have an impact on the environment.

INSPIRE is based on the infrastructures for spatial information established and operated by the 27 Member States of the European Union. The Directive addresses 34 spatial data themes needed for environmental applications, with key components specified through technical implementing rules. This makes INSPIRE a unique example of a legislative “regional” approach.

INSPIRE guidelines concern metadata and even services, defining protocols and standard to be respected.

10.2. Catalogues

Metadata catalogues are the tools used to share and describe owned data and event o browse remote sources. They act as nodes once installed and are able to harvest and share information among other nodes.

10.2.1. Geonetwork

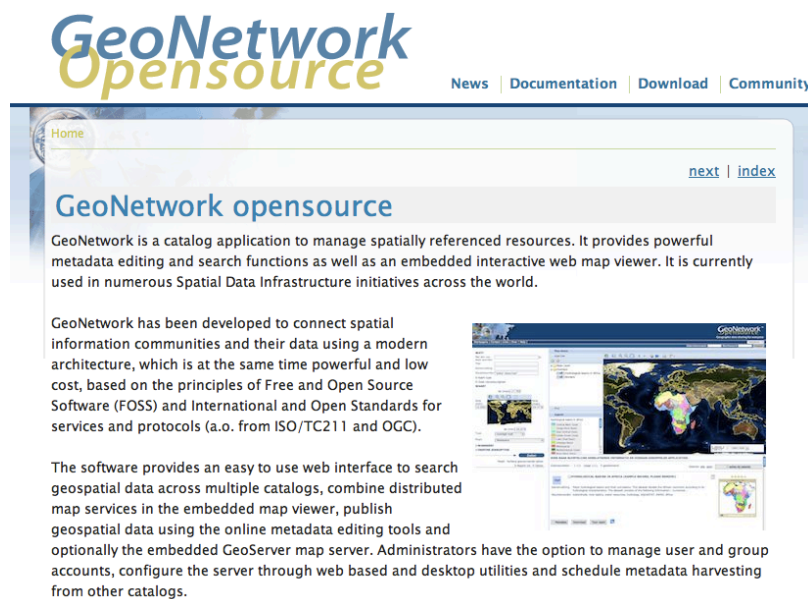


Figure 22: GeoNetwork home - <http://geonetwork-opensource.org/>

The GeoNetwork project is a free and open source cataloging application for spatially referenced resources. It is a catalog of location-oriented information.

Originally developed by the FAO (Food and Agriculture Organization) of the UN, now is an autonomous project still financed by the UN organizations.

It is a standardized and decentralized spatial information management environment designed to enable access to geo-referenced databases, cartographic products and related metadata from a variety of sources, enhancing the spatial information exchange and sharing between organizations and their audience, using the capacities of the internet. It uses standard OGC Web Catalog Service to build it's index and share information.

GeoNetwork is able to export metadata in various formats such as INSPIRE, ISO and OCHA.

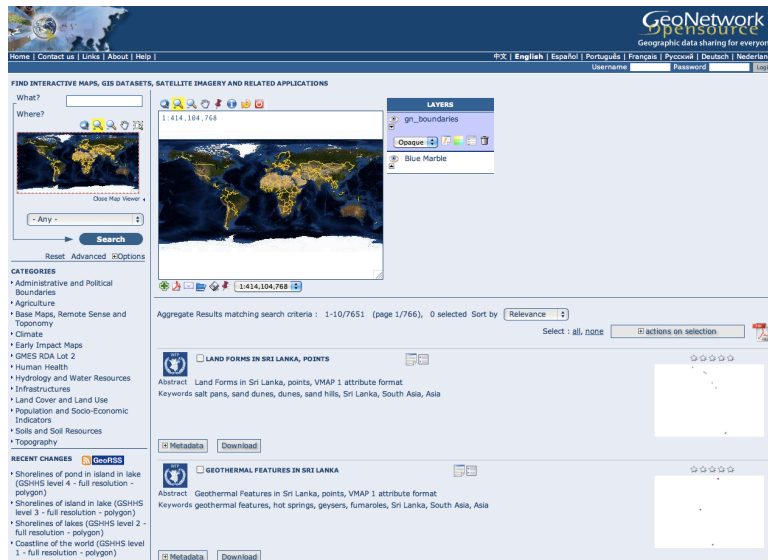


Figure 23: GeoNetwork main page

10.2.2. Geoportal



Figure 24: Geoportal home - <http://www.esri.com/software/arcgis/geoportal/index.html>

Geoportal is the commercial counterpart developed by ESRI and in the recent month released under the open source license.

The Esri Geoportal provide a web-based interface for searching and accessing the data, information, imagery, services, and applications available through the Global Earth Observation System of Systems (GEOSS). It connect users to databases, services and other portals that provide reliable, up-to-date, integrated and user-friendly information.

The Esri Geoportal offers a range of functionalities to:

- Discover Geographic Information System (GIS) data resources through the use of keyword-based searches and geographic areas drawn on a map or defined by place name or address;
- View geospatial data resources on Web-accessible map services if metadata about that mapped data is registered;
- Make maps combining GIS data from different map sources, conform them to a single map projection, and overlay them on a single base map;
- Obtain data resources from data services. Map images or composite map images can be downloaded to a user's system after the map is saved. Other information items, including any data associated with a map image can be obtained (if made available by the publisher) by using the option to link to the publisher's Web site; and
- Post user GIS data resources. If users want to post their data resources for public use, they need to review and conform to GEOSS Components and Services Registry requirements; then they can make their data available for access and external harvesting by others.

10.3. Geospatial standards

The Open Geospatial Consortium, (OGC) is a non-profit, international, voluntary consensus standards organization that is leading the development of standards for geospatial and location based services.

OGC standards are technical documents that detail interfaces or encodings. Software developers use these documents to build open interfaces and encodings into their products and services. These standards are the main "products" of the Open Geospatial Consortium and have been developed by the membership to address specific interoperability challenges. Ideally, when OGC standards are implemented in products or online services by two different software engineers working independently, the resulting components plug and play, that is, they work together without further debugging.

For the purpose of the present thesis just used standards will be briefly introduced. Those standards are at the base of client server communication and are the core of the web mapping interfaces exposed by the mapping services.

10.3.1. WMS

The Web Map Service interface allow to request geographic information using raster outputs. The map server must be able to understand a WMS request and answer with most common image formats like jpg, tiff, png, gif.

Images are georeferenced and can be either tiled or not (tiled means that the area covered by images is in fact split in smaller squares rendered by themselves), the single tile method can be very computational expensive.

It's up to the map engine to translate database information in image. The images generated in this way can be stored in a caching system in order to speed up the next data access.

10.3.2. WFS

The Web Feature Service with its extension WFS-T (Transactional) is a protocol devoted to work with vector data.

The WFS operations support INSERT, UPDATE, DELETE, LOCK, QUERY and DISCOVERY operations on geographic features using HTTP as the distributed computing platform.

The server can answer in various formats like XML, Json, Shapefile, KML, GML.

The transactional extension allow to edit and commit data to the server in order to update the database.

10.3.3. WCS

The Web Coverage Service defines a standard interface and operations that enables interoperable access to geospatial "coverages". The term "grid coverages" typically refers to content such as satellite images, digital aerial photos, digital elevation data, and other phenomena represented by values at each measurement point.

10.3.4. KML

The Keyhole Markup Language has been brought as a OGC standard in 2008. This language is an XML implementation for visualization of the data. Common map servers can respond in KML to a WFS request.

11. Web Map server

A web map server is a geographic data and service publisher.

Data sources can be of many different types such as database, raster repositories, shapefiles or others. Data is served using the OGC standard services such as WMS, WFS, WCS. Each of those services wrap different formats as described in the chapter 10.3.

One of the most important map servers is Geoserver, see the chapter 2.3.2.

11.1. Geoserver

Geoserver offers a web interface for the WMS, WFS and WCF services based mainly on two communication interfaces.

The REST interface which assume the concept that everything is a resource and The SOAP interface which uses XML file for information transfers; see chapter 6.2.

Geoserver is the OGC official implementation and thus ensures a fully interoperable system.

For the management it exposes a graphical user interface however each configuration can be changed and managed directly acting on XML files.

Customizations involve:

- Security and users
- Workspaces
- Layers and layer groups
- Stores
- Services
- Styles
- Environment tunings
- Projections
- Plugins
- Cache

The layer style is defined through the compilation of files written with an XML based implementation called SLD (Styled Layer Descriptor). These files can control the style of all the aspects of a vector data, from the colors and shapes to the labeling.

```

<?xml version="1.0" encoding="UTF-8"?>
<StyledLayerDescriptor version="1.0.0"
  xsi:schemaLocation="http://www.opengis.net/sld StyledLayerDescriptor.xsd"
  xmlns="http://www.opengis.net/sld" xmlns:ogc="http://www.opengis.net/ogc"
  xmlns:xlink="http://www.w3.org/1999/xlink"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <NamedLayer>
    <Name>Default Styler</Name>
    <UserStyle>
      <Title>Background</Title>
      <Abstract></Abstract>
      <FeatureTypeStyle>
        <FeatureTypeName>Feature</FeatureTypeName>
        <Rule>
          <Name>Background</Name>
          <MaxScaleDenominator>300000</MaxScaleDenominator>
          <PolygonSymbolizer>
            <Fill>
              <CssParameter name="fill">
                <ogc:Literal>#c9e7c1</ogc:Literal>
              </CssParameter>
              <CssParameter name="fill-opacity">
                <ogc:Literal>0.5</ogc:Literal>
              </CssParameter>
            </Fill>
          </PolygonSymbolizer>
          <LineSymbolizer>
            <Rule>
              </Rule>
            </LineSymbolizer>
          </FeatureTypeStyle>
        </UserStyle>
      </NamedLayer>
    </StyledLayerDescriptor>
  </NamedLayer>
</StyledLayerDescriptor>

```

Figure 25: SLD snippet

Geoserver has embedded a caching system called GeoWebCache, This tool is used to store WMS data in order to avoid further connections to GeoServer and thus to the database. Data are stored in a complex folder structure and the storing process can be either on demand or by custom scheduled trigger (for instance during the night). The image given by the WMS interface can be tiled or not, tiled means that the requested extent is split in several smaller images (usually 256X256 pixels) that are then managed directly by the client or can be given as a single big image. Usually is used the tiled version however for some specific cases, such as map with many labels, can be used the single tile version.

12. Linux

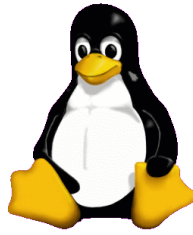


Figure 26: Linux logo

Currently there are three main operating systems commonly used:

- Windows
- Linux in his many distributions
- Apple OsX

The choice of the operating system is related to the server side needs.

Professionals usually use linux or windows on the servers, OsX is an excellent choice as well but its use is limited due to high cost of the hardware and software.

Open Source software is developed and maintained by great international community which usually are linux oriented. Linux, in fact, is the most famous Open Source operating system family.

Linux, by the way, does not mean an operating system but actually a os kernel. The kernel is the core for an operating system, is who manage the processor and peripherals as well as the file system.

The different distributions have the same type of kernel, even if is compiled with different and custom options, but wrap customized packages and applications.

During the period of this Ph.D. a custom personal Linux distribution has been built, following guidelines drawn by the Linux From Scratch (LFS) project <http://www.linuxfromscratch.org/>.

The purpose of this work has been to better understand the internal mechanics, starting from the download of a raw kernel from <http://www.kernel.org/> to the graphical interface application the operating system has been set up totally.

Linux is completely customizable, once the kernel is compiled with proper options (i.e. supported devices, processors etc..) the rest of the structure is composed by packets compiled for the machine with desired options.

The process of compiling a packet consist of choosing which features this application should have (among the available in the source or by adding it's own) and then translate it into a machine readable program.

Most of the Linux components are written in the low level language C, this is extremely fast and can be compiled by the GCC compiler.

The compilation process can take minutes to hours depending by the complexity of the application.

12.1. Low level packages

Around the kernel, which is the core component, there are some tools that are central for the starting of the machine. The things to manage when a computer starts are many: discovery of the peripherals and assigning them to system paths with appropriate drivers, GUI interface to bring up, internet protocol to start, characters set and language recognition.

Each of those tasks are lead during the start phase and governed by the kernel.

The overall startup however is responsibility of the boot loader, the only tool which is started by the BIOS before the kernel.

12.2. Security packages

Security is a core concept for operating systems. Security means virus defense as well as internet security and data consistency.

Linux is generally virus proof, this because of the relatively small diffusion compared to windows and to the differences between the various distributions which make difficult to build system wide viruses.

However the system and internet security must be lead by the administrator. Many packages are responsible for this:

- Iptables are responsible for net connections among different ports and protocols
- Ssh is responsible for remote connection to the system to transfer data and operate remotely on the console
- Sudo is responsible for internal users privileges and switch between a user and another
- Http security is lead by the Apache web server configuration
- Data security is responsibility of the database or the data container

12.3. High level tools

All other packages use the low level libraries and tools to work. Development tools, GUI, servers, databases, browsers, text editors are compiled and tuned on top of the base toolchain.

It's up to the administrator or the developer to customize any aspect of these components.

13. Django and the MCV pattern



Figure 27: Django logo

During the development process and especially for the web development process, there are many low level and repetitive tasks to deal with. For instance, the problem of a site authorization, the http requests and answers, the Content Management System, the template system or the low level database interaction.

To face the repetitiveness of those tasks are born tools called development frameworks that avoid to take care of them.

Django is one of the most famous python frameworks developed four years ago by a fast-moving online-news operation, Django was designed to handle two challenges: the intensive deadlines of a newsroom and the stringent requirements of the experienced Web developers who wrote it. It let build high-performing, elegant Web applications quickly.

Django focuses on automatizing as much as possible remaining adherent to the DRY (Don't Repeat Yourself) principle which means:

Every piece of knowledge must have a single, unambiguous, authoritative representation within a system.

The code must not be duplicated at any point, however can be duplicated some automatic products that the code produce.

Django is strictly adherent to the MCV development pattern.

13.1. MVC pattern

Model–View–Controller (MVC) is a software's architecture, currently considered an architectural pattern used in software engineering. The pattern isolates "domain logic" (the application logic for the user) from the user interface (input and presentation), permitting independent development, testing and maintenance of each.

The model manages the behavior and data of the application domain, responds to requests for information about its state (usually from the view), and responds to instructions to change state (usually from the controller). In event-driven systems, the model notifies observers (usually views) when the information changes so that they can react.

The view renders the model into a form suitable for interaction, typically a user interface element. Multiple views can exist for a single model for different purposes. A viewport typically has a one to one correspondence with a display surface and knows how to render to it.

The controller receives input and initiates a response by making calls on model objects. A controller accepts input from the user and instructs the model and viewport to perform actions based on that input.

An MVC application may be a collection of model/view/controller triads, each responsible for a different UI element. The Swing GUI system, for example, models almost all interface components as individual MVC systems.

MVC is often seen in web applications where the view is the HTML or XHTML generated by the app. The controller receives GET or POST input and decides what to do with it, handing over to domain objects (i.e. the model) which contain the business rules and know how to carry out specific tasks such as processing a new subscription, and which hand control to (X)HTML-generating components such as templating engines, XML pipelines, Ajax callbacks, etc.

The model is not a database: the 'model' in MVC is both the data and the business/domain logic needed to manipulate the data in the application. Many applications use a persistent storage mechanism such as a database to store data. MVC does not specifically mention the data access layer because it is understood to be underneath or encapsulated by the model. Models are not data access objects; however, in very simple apps that have little domain logic there is no real distinction to be made. Active Record is an accepted design pattern which merges domain logic and data access code - a model which knows how to persist itself.

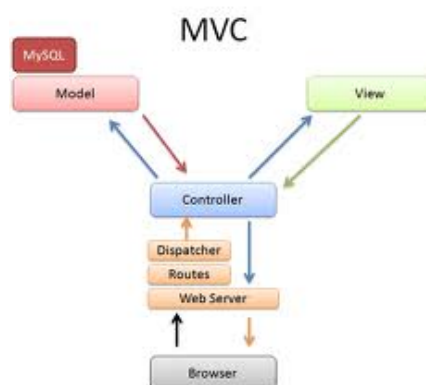


Figure 28: MVC pattern

13.2. Django features

Django is a complex python architecture which manage several aspect of the web development. It's main features are:

- ORM Object Relational Mapper, a translation engine between python ad SQL. This tool take care of the translation to the database language of the code written to define data structure as well as data interaction. Regardless of the database engine used.
- Automatic CMS Content Management System. Often is needed to have and administration interface for the data, user and security options as well as a user interface restricted to data management. Django can automatically build that interface maintaining a full customization.
- URL design, the MCV and decoupling django style allow to separate the URL definition from the physical resource. The objective is to have semantic URL for resource identification.
- Template system take care of the dynamic web page rendering. This components generates the HTML as well ad javascript based on self written templates in a data driven manner.
- Internationalization to easy manage multi language sites by translating just the words that need to be translated
- caching system to avoid regeneration of the content when is not changed
- GeoDjango spatial extension to deal with spatial databases, is a geographical extension of the ORM able to talk with db spatial extensions.

The entire Django APIs can be used either as stand alone framework or in each python program as any common python library. This give to the developer a flexible way to interact with the server and the database avoiding the write of complex SQL queries.

Implementation

14. Hardware and software definition

The work on WebGis applications started early in the 2006 with the graduation thesis entitled “Applicazioni WebGis per la gestione delle emergenze”.

Some raw but modern tools were developed using simple basic libraries. The process of building effective WebGis application has been lead by the need to improve user experience as well as functionalities. The continuous interaction with the WFP has been fundamental to define the user needs.

The actual architecture has been defined in some steps that have been followed starting to zero knowledge in programming to system development.

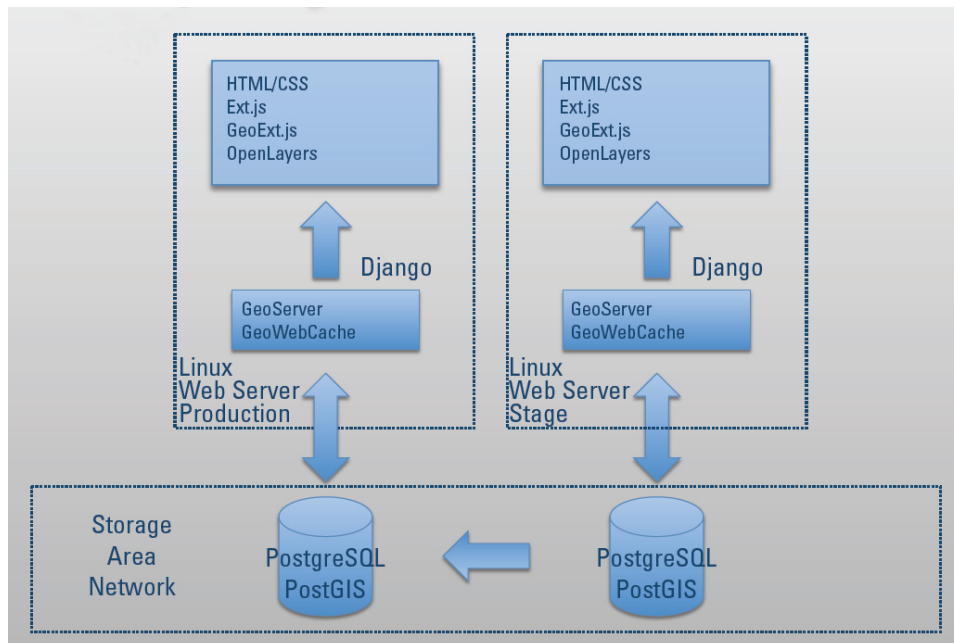


Figure 29: Ithaca Open Source Architecture

14.1. Hardware

The process of the hardware configuration involve the choice of the hardware itself, basing on the expected web traffic and the amount of operation that could be performed by the machine.

The definition of the hardware architecture has been a process that has taken two years, this because the estimation of the volume of traffic is not possible in the first periods of service release (Ithaca started its activities in 2007).

The first services was published on old servers but after few months has been bought a server with 8 Gb of RAM and 300 Gb of disk space (with the internal RAID

controller, which is composed by two twin disks mirrored) and 3.0 GHz quad core processor. This hardware configuration is enough for about 20 concurrent connections to the server with moderate processing capabilities.

First services like Flood Alert Monitoring System has been deployed for 1 year on that hardware.

As the WFP requests grown with the related projects like the SDI, Ithaca decided to make the hardware architecture evolve following the idea that the data keeper must be separate from the data processor. The fact that a single server was hosting both the service and data raise some issues related to the fact that data is the core of a service and must be kept as much secured as possible:

- if stored on the same server that is exposed to the web makes data vulnerable to intruders if they can breach the server barriers
- in case of hardware failure (i.e. hard drive break), all data are lost as the services
- data should be kept in an environment dedicated to data storage and not suitable to program running
- a server should be dedicated to “serve” a service like a filter between data and the users

To face these challenges the choice has been to purchase a Storage Area Network (SAN) as a central data server for an entire architecture avoiding single points of failure. Single point of failure mean that there must not be a single channel which transport the information in the entire process of the data.

The SAN is composed by an array of disks and two separate processing units. The disks are presented as a unique space which can be logically split and assigned to the connected host. Each host is connected with two channels to the two different logical units of the SAN in order to have failure proof double data flow.

The physical data security is assured by the internal RAID system which can be automatically recover from a single or double concurrent disk break. The system also take care of data consistency and de - duplication.

The logical security is granted by the non directly published environment and to the other step of control which is the connection between the host and the SAN.

14.1.1. SAN

A storage area network is a storage device (such as disk arrays, tape libraries, and optical jukeboxes) accessible to servers so the devices appear as locally attached to the operating system. A SAN typically has its own network of storage devices that are generally not accessible through the regular network by regular devices.

A SAN alone does not provide the "file" abstraction, only block-level operations. However, file systems built on top of SANs do provide this abstraction, and are known as SAN filesystems or shared disk file systems.

14.2. Server side environment

After trials with different linux distributions, first CentOS a branch of the Red Hat commercial distribution, then Ubuntu which is the most known and easy to use distribution, but the definitive choice has been the Debian (the parent of Ubuntu).

Main issues related to this choice are:

- CentOS was too different from Ubuntu which is often used as a development environment and either too behind the others in term of updates
- Ubuntu instead is updated every six months and new packages are not always stable and some issues related to the new generation of kernels adopted by this OS prevent it to work properly with the SAN software
- Debian is considered the professional environment in the linux distributions panorama, the version update is every two years

The definition of the server side tool chain has been driven mainly by two parameters:

- the choice of Geoserver as a map engine
- the choice of Python as a server side language

First draft web maps were static with no server side logic and based on the MapServer engine. The migration to Geoserver is due to three main reasons:

- Geoserver is more performant on vectorial data rendering and the Ithaca SDI is almost completely vectorial
- MapServer requires the compilation of a Map File to serve data, instead Geoserver uses all XML files for the configuration, this is more standard
- Geoserver has a nice web graphical interface while MapServer has not

The later choice of Python has been driven by the fact that many desktop GIS software, both open source or commercial, such as ESRI ArcGIS, Quantum GIS and GRASS GIS have a python console embedded with available API. Through the console it is possible to create procedures to interact with the data and using the python API inside self developed application is possible to use the GIS functions.

Python then is an excellent programming language with the most modern features and with a great supporting community which continuously release upgrades and extensions.

The need of more server side flexibility and interaction has conducted to the adoption of Django as a web development framework. It has been chosen Django instead of other framework or python CMS like Zope or Pylons for the following reasons:

- complete MCV and DRY compliant
- bottom up strategy (with Django its up to the developer to build features)
- high modularity

Regarding the web server software the choice is almost mandatory about the Apache project. This open source server is currently the most used web server in the world, is lightweight, highly developed and extremely modular. The web server is who cares about the internet requests, responses and redirections with appropriate filters and security constrains.

A last consideration must be done about the application server. An application server is a container used to expose some services (application on the web), in our case the application to expose is Geoserver with it interfaces (WMS, WFS, cache and others). Geoserver is written in Java a semi compiled language which need a virtual machine to work with the system. The Geoserver instance cannot be managed by Apache for this reason so a dedicate server is needed. Geoserver in fact takes care of publishing data and data security not about the connection or internet security. The role of the application server is to control access to its applications and keep safe them.

The most used Java server is Apache Tomcat, the choice is also driven by the fact that a lot of experience is spread across the web about how to make Apache and Tomcat work together.

14.3. Client side software

As the very first WebGis draft were based on primitive libraries like Ka Map or Chameleon, it has been clear that the need of flexibility and low level tools was primarily.

OpenLayer was then the optimal javascript solution to build web maps, it is modular, quite complete and is becoming a de facto standard for open source web maps. However OpenLayers as limitations concerning the graphical interface, in fact it can be assumed as a client side content manager. To face this lack has borne GeoExt which act as a bridge

between OL and Ext.js. Ext.js is a javascript library borne to enhance the GUI system. GeoExt extends OL and makes easy to insert its objects inside of the Ext.js graphical structures.

15. Hardware and software implementation

The architecture design can be split into several topics:

- Hardware link and SAN drives definition
- Server installation and configuration
- Apache web server configuration
- Database creation
- Server side architecture design
- Client side libraries implementation
- Remote development environment set up

All of these points implies specific knowledge to acquire, manuals are attached at the end of this document.

15.1. Hardware link and SAN drives definition

The concept of avoiding single points of failure must drive the entire architectural design. The Storage Area Network has two processing heads that, independently can access the virtual drives. If a head for some reasons fall, the other can supply data without interrupts.

The hard disk array in the SAN is managed by a software layer which allow to create common virtual spaces between specific disks, these spaces are called LUNs (Logical Unit Number). The space is internally protected against failures via the RAID system and the logical controller.

Then the LUNs can be assigned to the hosts connected and registered to the SAN, in fact just the host allowed by the administrator to work on the specified LUN can see it.

The ways the hosts are connected to the SAN can be two:

- fiber channel links passing through two switches
- iSCSI direct links on the ethernet protocol

The adopted solution is a mix between the two, the web servers are directly connected with two iSCSI cables for each pointing at the different SAN heads and a fiber channel solution for the high throughput servers dedicated to database elaborations.

In the ITHACA architecture the web server are two and twin, one server is dedicated to development and testing (stage) while the other is for production purpose. They are identical in each installed package except for the debugging configuration of the tools which are enabled on the development server and disabled (to increase performances) on the second.

15.2. Server Installation and configuration

The server configuration is not only a matter of Debian installation, it must be taken into consideration the security and the environment customization.

The complete manual on how to install and customize a Debian distribution with all the software needed to work with python and django as well as the Clariion Storage Area Network can be found in the attachment "Storage Area Network on Debian Lenny".

15.2.1. Base configurations

Once installed the operating system brings some default configuration that must be corrected to properly work in a server environment.

These operations are assumed to be in a disconnected environment, if not (due to a web installation) is suggested of disconnect the server until the security measures are put on.

First thing to manage is the users creation with their password and the setup of the superuser (root) password.

Considering that most of the times the server is located in a Data Center which usually is cold and noisy should be useful to work on it remotely, so the second thing to take care is the remote access manager OpenSsh associated with a raw Iptables configured to accept only LAN connections.

The Iptables are the main low level firewall that regulate internet connections to the machine.

After that is possible to connect remotely to the server and finish the configuration which will involve the standard packages and libraries installed via the package manager "apt-get".

Important geographical libraries to be installed are the GEOS and Proj.

These are components used by Postgres and GeoDjango to make reference system transformation and reprojection.

The Apache web server can be installed with a default apt-get procedure, for set up see the Apache web server configuration paragraph.

Tomcat installation is executed by default by the "apt-get" manager but the "roles" or users have to be created.

Tomcat is an application servlet for Java projects, this means that is selfish in managing the distributed java package. To deploy Geosever or even GeoNetwork the task is just to copy the “war (web archive)” file into the Tomcat applications folder. After that the application will be automatically served at the <http://host:8180/app> url. The flag 8180 means that by default, on the Debian distribution, Tomcat use the 8180 port, this can be changed however, for security reasons, it’s suggested to avoid external access to that port and use a http proxy instead (see Apache web server configuration paragraph).

A side package used for project consistency and multiple collaboration is Subversion. This is a versioning tools to share projects among a wide community and to take care of multiple editing on the same files. It has been decided to use it in ITHACA since the first definitive architecture has been implemented. The multiple developer collaboration was creating conflicts for two main reasons:

- developers need to have a local copy of the project to work on, without involve the server until a semi stable release is ready
- developers need to work on the same file in separated instances and thus to have a tool to manage the changes

Subversion creates a database on the server on which store the project, the system of remote copies allow to use perfectly working architecture on the developer machine. The concept of local copy will be even applied to the published service, in fact the web application can be considered a local copy (on the same server) of the subversion repository. The possibility to have local copies and two separate servers for development and publishing purpose makes the possibility to introduce errors in the public environment extremely small.

15.2.2. Security

The main security responsible is the Iptables firewall, it’s configuration impact access as well as performances. Complex rules are also complex matching tables to compare with.

The rule matching depends by the receiving packet origin, this sends the packet to a “chain” which is a logical sequence of processes that in some way impact on it. There are five predefined chains, though a table may not have all chains. Predefined chains have a policy, for example DROP, which is applied to the packet if it reaches the end of the chain. The system administrator can create as many other chains as desired.

These chains have no policy; if a packet reaches the end of the chain it is returned to the chain which called it. A chain may be empty.

- “PREROUTING”: Packets will enter this chain before a routing decision is made.
- “INPUT”: Packet is going to be locally delivered. (N.B.: It does not have anything to do with processes having a socket open. Local delivery is controlled by the “local-delivery” routing table: ``ip route show table local``).
- “FORWARD”: All packets that have been routed and were not for local delivery will traverse this chain.
- “OUTPUT”: Packets sent from the machine itself will be visiting this chain.
- “POSTROUTING”: Routing decision has been made. Packets enter this chain just before handing them off to the hardware.

In these configurations the administrator chose which address or address range and request type can access to the server resources.

The remote access configuration establish which users can remotely connect to the server, it is good security rule to avoid root user access, this because in case of “man in the middle attack” the password can be sniffed. Remote unprivileged users can then temporary gain superuser privileges once the ssh connection is established, using the “sudo” command. Even this command have to be configured defining which local user can gain temporary superuser privileges.

The Subversion tool uses an SSH tunnel to grant access to its repositories, the protocol, in fact, is the “svn+ssh” which means subversion protocol wrapped by secure socket connection. It is important to grant write privileges to the users allowed to modify the project repositories otherwise it will be read only and no commits will be available.

15.2.3. SAN link and database creation

To make the host (server) see the LUNs on the SAN and work with them some steps must be followed:

- installation of the SAN drivers and agents on the host
- registration of the host to the SAN
- creation, on the SAN, of a storage group and assignment in it of the host and LUNs
- setting up the auto mount of the new discovered volumes on the host and formatting of them
- creation of the database or spatial database in new volumes

The SAN agents and drivers are distributed by the vendor only for the Red Hat linux distribution. Red Hat is commercial and is the only one that usually is covered by assistance of the vendors. However it is possible to install these agents on different environment. The main difference between Red Hat and Debian is the packages distribution, the Red Hat packages are managed by a tool called RPM and thus the programs are packed to be read by this, the Debian packages are managed using the DEB format. Before the installation the administrator need to re pack these programs from RPM to DEB, this can be done using several tools one of which is “alien”.

The SAN agent is a 32 bit tool however the modern servers are 64 bit, to ensure the compatibility Debian needs 32 bit libraries that can be installed via apt-get (ia32-libs).

The connection to the SAN is a multi layer mechanism, there is the iSCSI protocol which is responsible of the discovery of the remote target and for the transport, on top of it there is the vendor agent which register the host and make sure the SAN recognize the host. We can think of the connections between the SAN and the hosts like a small private local area network, not accessible from the external and thus provided with internal fixed IP addresses, it is possible to assign IP address to the SAN heads and to the hosts ethernet ports, these address can be in the form 10.0.0.X where X is a number starting from 1 to above depending of the number of the ports. For instance the SAN addresses can be 10.0.0.1 and 10.0.0.2 while the hosts can have 10.0.0.10 to 10.0.0.13.

“discovery” is the iSCSI command to locate remote iSCSI hosts, it returns an id for each target discovered, this id is in the form (iqn....). Using the returned id it is possible to login on the target with the command “login” and the use the SAN agent to register the host.

On the SAN console will appear the host which can then be managed to be assigned to a “storage group” which is a logical container used to tie LUNs and hosts.

From that point on the host are visible the new volumes correspondent to the assigned LUNs, these volumes can be “mounted” in a custom folder and then formatted to be ready to write and read. The “fstab” file of the Linux system tells which volumes are mounted at startup, editing this will make sure to have volumes loaded correctly when the server is up.

The last step is to create a database that points to the new disks, databases have a default space in which they store the tables called “default tablespace”, however is possible to create custom tablespaces in whichever folder and tell to the db to use it to store the data.

In Postgres using the “psql” command line console can be easily created using the SQL command “create tablespace ‘name’ ‘path/to/desired/folder’”.

15.3. Apache web server configuration



Figure 30: Apache logo

The Apache web server is a modular engine designed by default to serve static web pages, however its purpose is also to manage dynamic contents and act as a proxy or redirect http calls.

15.3.1. Handlers

Apache act in different ways regarding the content type of the resource served. To tell it how to manage the resource must be defined a “handler” for the specific content type. The handlers are the inner engines that can understand a language (such as Python or php, java) and send to the web the HTML generated.

Even to serve a static file such as an image is needed an handler, the default handler that is used for standard contents.

To serve and to read the python code, Apache needs an extension, this is achieved installing a module called `mod_python` which is an add on that tells to apache which directory must be considered python and thus read by the `mod_python` handler instead of the default.

Another python handler is the WSGI module which is currently considered quicker that `mod_python` and more flexible.

For a general rule the content can be labeled in two main types:

- dynamic content
- non dynamic content

The dynamic content is the one that needs to endure some processes before being sent across the web. A page which have python or php procedures inside is a dynamic content, this page in fact require some operations to be completed before the user read it, for instance it can need of database information to fill some parts. Apache takes care to execute the language, retrieve data and complete the page.

The non dynamic content is images, pdf files, pure HTML files, Javascript files (Javascript is interpreted by the web browser so is static for the server). The default handler is designed to serve non dynamic content.

In a Django project, there are python files as well as static files like images or js, the Apache configuration must take into consideration that the python handler should be used everywhere inside the project folder except for the folders which host the static files where the default handler should be used.

15.3.2. Proxies

An http proxy is a filter between the web and the resource, the more complex proxies wrap an entire LAN behind a strict controller.

Proxy is used for two main reasons:

- it enhances security while all the connections should pass through it and through its firewall
- it act as a caching system avoiding duplicate requests to the web

In our case the proxy should be something that mask under the apache management some services that are not managed by apache and which not run under the default port (the number 80).

The module which take care of this service is the “mod_proxy” and must be configured to redirect the request from the web to the resource under the other port and that from the resource back to the web, these operations are called ProxyPass and ProxyPassReverse.

Typical resources to be wrapped under the proxy module are Geoserver and the catalogue Geonetwork which are java tools and thus managed by Tomcat (port 8180).

15.4. Database creation

```
-- Table structure for 'gotquestions'
DROP TABLE IF EXISTS `gotquestions`;
CREATE TABLE `gotquestions` (
  `RecordID` int(11) NOT NULL auto_increment,
  `Date` date NOT NULL default '0000-00-00',
  `Name` varchar(30) NOT NULL default '',
  `Email` varchar(40) NOT NULL default '',
  `Telephone` varchar(20) NOT NULL default '',
  `City` char(2) NOT NULL default '',
  `State` varchar(30) NOT NULL default '',
  `HearAbout` tinytext NOT NULL,
  `Subject` text NOT NULL,
  `Message` char(1) NOT NULL default 'N',
  `Copy` int(1) default NULL,
  `VolunteerID` int(1) default NULL,
  PRIMARY KEY (`RecordID`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1;
```

Figure 31: SQL snippet

Database structure as well as physical structure should be double, one for development purpose and one for production purpose.

The development database is suitable for many structure modifications and could have duplicated items and all temporary issues related to the development process, production database instead has a fixed structure and has all the performance tunings put on work.

The SAN can host both of them in separate LUNs and then each server (development and production) should read only their respective database.

Once the development team has defined a db structure ready for publishing the db can be moved to the production environment.

In the previous chapter it has been explained the concept of tablespace, a physical location for the data, inside the tablespace it is possible to create as many database as desired. A database is a logical container assigned to an owner and protected with password and readable by many users with different privileges. However inside the database sometimes it is necessary to have some partitions, this is the case when a global purpose database should have some subgroup or views of part of the data maybe specific for an application. These partitions are called “schema”, the default schema is the “public” other schema can be created by the admin. It’s a good rule to keep the public schema as clean as possible just having inside only the shared tables useful for other schemas, for example the Postgis spatial extension of Postgres create some tables with definitions of SRS, these tables should stay in the public schema because they identify the entire database as “spatial enabled”. Other schema will read in the public for use the spatial part.

All these operations can be done through two tools, the first one is the psql console of Postgres, this command line interface allow to send commands to the database server both in SQL form or in unix bash format. Commands like “create database”, “create schema” are used to manage the structure. When creating a new database must be remember to pass the tablespace parameter in order to build the db in the remote storage system and to assign it to an owner (a postgres user with password).

Another way to manage the Postgres db is by installing on the client the PgAdmin console, a graphical application available for Windows, Mac and Unix that quite easily allow to interact with the Postgres server.

Once a db is created it is necessary to spatially enable it and this can be done through psql by passing the location of the Postgis sql files which automatically create the spatial tables.

Postgres takes care for the access security as well, in a file called “pg_hba.conf” contains declarations of which user, from a local or remote address, can have access to a specific database. User can be free to enter in a db or can be asked for the password or even for more refined authentication methods. In this file is also written which ip address range can have access to the db.

For security reason is strongly suggested to avoid external connections other from the local machine (localhost) or at most from the local area network. Db connections in the case of web applications should only be done by the local resources like Apache and Python programs or Geoserver and Geonetwork instances. External users should read the database by passing through those tools.

15.5. Server side architecture design

The core of a web application which mixes up all the components, make connections to the database and sends data across the web is the server side architecture.

In our case is composed by Python and Django as framework, in parallel the geographic data engine Geoserver and its caching system called GeoWebCache are running.

The server side can grow a lot and thus is important to keep it straight adherent to the MCV pattern. As anticipated this pattern establish that the database connections, the business logic and the presentation layer must be separated and independent from each other.

Django helps in this task being MCV in each its component and by creating a simple folder structure in which develop projects.

15.5.1. Django based folder structure and logic

Django is a python library like many others and is a web development framework, however it brings to the system a command that can be used (but is not mandatory) to build a basic folder structure called project.

The command “`django-admin createproject name`” create a folder named with the given name in the current working directory. In this folder are present four files:

- `__init__.py` which is the mandatory file for any python module
- `manage.py` which is a simple wrapper around the “`django-admin`” command and which sets up the environment variables for the project
- `settings.py` which is the main configuration file for the entire project, it holds database information as well as languages used, template folders, enabled applications and the definition of some environment variables use by `manage.py`
- `urls.py` which is the url configuration, being django MCV it decouple the web URLs from the actual location of the resource, in this file we design the URL for the project

Inside the project django allow to create applications. Applications are the web apps with their internal logic, all apps created inside a project will have the same global settings and the same databases defined in the `setting.py`.

To create an app is possible to type the “`python manage.py startapp name`” inside the project as well as create a folder by itself. The created folder must have the `__init__.py` file inside as all python folders.

In the created app there are three files other than `__init__` :

- `views.py` which will host the business logic (is the Controller part of the MCV, the name can be changed but for simplicity is suggested to keep views)
- `models.py` which will host the database definition and structure (is the Model part of the MCV), the models will be used by the views logic to access the database and write to it.
- `tests.py` which can be used to test the logic

The presentation layer is the django template system used by the views to compile html templates.

All these folders have to be managed by the python module of apache when called but inside this structure should even be present a folder which host the static files (javascript, plain html, images and others), so apache need to be configured to use the default handler for that “static” folder.

The Ithaca architecture follows the rule that each research area should have its own project and inside the project should be present all the web applications that share something among them.

In the case that different projects need some developed tools these can be placed in a parallel folder and set as reachable by python by adding it to the PYTHONPATH environment variable of the system. This trick should be done to avoid any code duplication and to keep near the DRY principle.

15.5.2. Geoserver deployment and configuration

The deployment of geoserver is extremely easy, all it's needed is to copy the .war file into the web apps tomcat folder. From this point the geoserver instance will be reachable at the <http://localhost:8180/geoserver> address.

Geoserver need to be configured in order to access to the administration interface and set up the web services. The file users.properties located inside the data directory allow to define user names, passwords and roles, the administrator will have privileges to create layers and modify the web services parameters.

The internal logic of geoserver is based on the concept of workspaces or namespaces, this structure is a logical wrapper around data sources which is bound to a universal resource identifier URI (a permanent address to that workspace), inside the workspace are defined the stores which can be sources of various formats both vectorial and raster. From the data sources geoserver automatically harvest for data layer to be published and, for vectorial, stylized. To publish a layer it is mandatory to add some information about it like the native projection and the geographical bounds, the styling process pass through the definition of a XML file called SLD (Styled Layer Description).

Published layers are accessible through WMS and WFS protocol (if active), each resource can be obtained compiling a correct URL.

The OWS request format is a URL that specify many parameters, like any http GET request is in the form *address?parameter=value¶meter2=value2* and more.

The OWS address of a production environment is in the form <http://domain.com/geoserver/wms> or wfs. The list of available parameters is long however the most used for a WMS request are:

- format: the image format (jpeg, tiff, gif, png, png8...)
- bbox: the extent in the form lower left corner and upper right separated by a comma

- srs: the requested projection in the form EPSG:XXXX (4326 for geographic WGS84 or 900913 for Google spherical mercator)
- request: this is the type of information that are expected, a getMap returns an image, a getFeatureInfo returns information about the specified feature, getCapabilities returns information about the available data on the server
- layer: a comma separated list of layers
- width and height: the pixel dimensions of the returned image

For WFS requests the format can be shapefile, GML, KML and more.

It is possible to change the default style of the layer by passing the style parameter, this in case the feature has to be rendered in different contexts.

The SLD language is an XML implementation used to define the way Geoserver should render the features. The developer have the complete control over each part of the rendering process, it is possible to:

- assign different style at different scales
- assign more than one style to a feature in order to overlap them
- control the visibility and opacity as well as the anti alias or margins at each scale
- control the names position
- assign custom icons or images to features

The counterpart of this extreme flexibility is the number of the fields required by the SLD which make the file to grow a lot, the style of the base layer of Ithaca is about 5000 rows of code.

In the last releases of Geoserver is in beta test the module which accept the CSS language to style the layers, an inner engine take care of the translation between the two languages.

15.5.3. GeoWebCache



Figure 32: GeoWebCache logo

At the beginning of the WebGis WMS requests were always done directly to the map server; this mean that for each request the server had to call the database, get the data and apply the style to them. The result was extreme server work load an long waiting times for the user to get WMS images. To face this issue in the last five years are borne services called geographical caching systems. These systems are designed to store in

advance the images in a tiled format (a mosaic composed by several small images), if the WMS request is done to the caching system instead to the server no further call to the db will be done and the response time will be extremely short.

In the last year Geoserver has acquired in its logic a caching system called GeoWebCache. This java system is designed to make WMS request to geoserver and store the results in a complex folder structure. The caching process can be:

- on the fly, once the area is viewed for the first time the caching system call the server for the data and store them for further uses
- programmed, the caching process can be launched for a range of scales and for a determined bounding box. in this case the server will be under load for a while.

The stored images must cover all the available scales at the whole extension of the data, this implies a huge disk space consumption especially if the data, like i the Ithaca case, cover the whole world.

To get cached data instead of the on the fly generation the only thing needed is an address modification in the form: *http://domain.com/geoserver/gwc/service/wms*.

15.6. Client side libraries implementation

For the client side there is no work to do on the server except of the libraries storage itself.

As mentioned earlier the used libraries are:

- OpenLayers
- GeoExt
- Ext.js

Of course these libraries can be included in a web page directly from their distribution site, all of them have an internet page which distribute the source. However this could not be a good choice because if one of the sites drops for such reason even the web application will lack in some parts and then if the remote site changes the distributed library version there can be modification in the application out of the developer control. A good choice is to download the libraries and store them in a web accessible folder on the own server. It is possible to have multiple versions of the same library and use which want by changing the name, in the application, of the folder that host it.

For a production environment it is also suggested to limit the number of modules loaded by the libraries to which are effectively used, All those libraries in fact are included in a web page by linking a single javascript file which then take care to include all the others. By modify that file is possible to choose the files that will be included in the head for the html page, while taking into considerations the dependencies between them.

15.7. Remote development environment set up

Once the server is set up and some projects are created and managed by the Subversion repository schema it is necessary to set up a working environment on the developers machines in order to easily get the source, make it work on the own machine and the commit the changes back to the server once something new is created.

All the gathering and committing process can be hand made using a command line subversion client tool, however the availability of applications which can collaborate to set up a graphical interface can help to manage the whole structure in an easy way.

The suggested solution that work on any platform is composed by:

- Eclipse advanced text editor
- Subversion client application
- Python module for eclipse

All these components are free and open source.

In order to avoid code changes between local and test/production environment is strongly suggested to replicate locally the entire server structure, with local geoserver, database, django, python and the same folder structure. This will avoid to change parts of the code to make it work on the local machine and avoid the mistakes related to this.

16. Architecture design

First services developed in the 2007 were based on OpenLayers linked directly to Geoserver, the graphical interface was simple and the controls limited.

The developed applications were related to Ithaca projects like the Snow Cover Monitoring system and SDI publishing.

However it has been immediately clear that in an environment like Ithaca where the need of the projects to be published and the requests from the WFP to have some tools to be used when an emergency occur, the stand alone on the fly web app creation was not enough.

The research in the WebGis field has then be focused in the study and implementation of an architecture that should be:

- general purpose
- enhance code reuse
- avoid code duplication
- reduce the creation time to face the needs under emergency in the response and recovery phases
- highly extendable and customizable for the project that involve preparedness and mitigation phases
- modular

Such architecture should manage automatically the data publishing process allowing the user to choose which data to display and which controls have on the web application.

The idea is to have a unique tool chain that can be configured on the fly to get a WebGis application working in few minutes. If specific controls or features are needed they can be developed and further added as modules.

The steps necessary to achieve this are:

- database design
- geoserver set up
- client side architecture design following the MCV paradigm
- identification of the expected dynamic parts
- design of the server side architecture

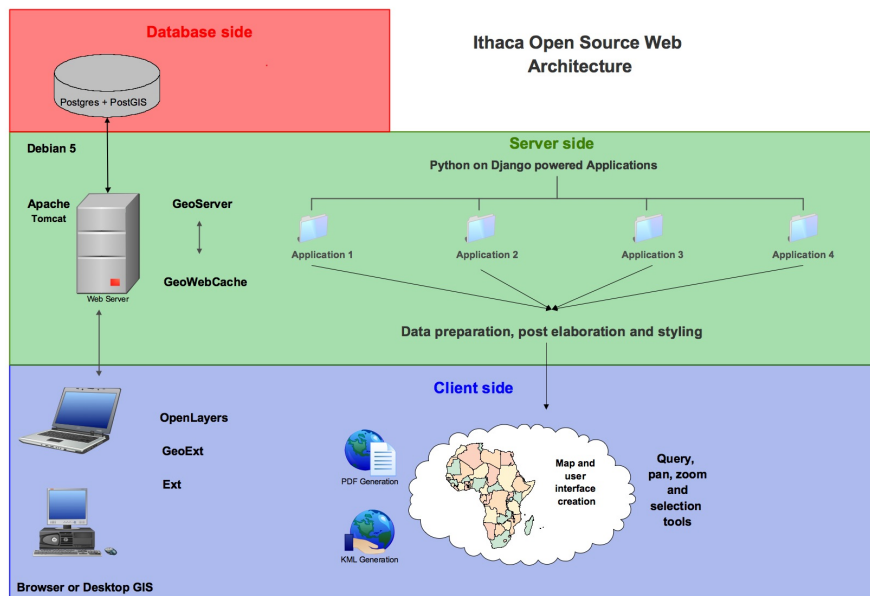


Figure 33: Architecture design

16.1. Database design

The database design should be as much simple as possible by keeping separated development data from production data and global from project specific data.

The implemented structure has:

- test database
- pre production database
- production database

In each of those databases are implemented some schemas that are:

- public for general purpose data
- geodb for global spatial data taken from the SDI project
- various schemas depending on the specific projects

The test and the pre production database are accessed only by the pre production server while the production database, which is set up with all performance tunings, is accessed just by the production server.

Some schemas like geodb are a direct export from the Oracle instance of the SDI project, others are created manually by importing shapefiles or by automatic procedures and others are managed by the django models interface.

16.2. Geoserver set up

Geoserver can access various sources from direct access to the Postgres instance to image or shapefile repository. Except for the images which usually should be stored in a structure folder, vector data should always be kept inside a database. However even for images is suggested to have them indexed inside a database and access to them by passing through the db.

Geoserver configuration should reflect the database structure in order to avoid misunderstandings. The workspace for local data should be unique being that the address from which the data are retrievable is the local geoserver instance itself. The way Geoserver makes data accessible is through a composition by their workspace and layer name in the form workspace:layer_name, this implies that the layer name must be unique for the server instance, the layer can be given an alias different from its name in the database and its alias can be a composition by the store name (which should reflect the schema name) and the actual table name in the database, the resulting published name can thus be workspace:store_layer_name.

16.3. Client side architecture design

The javascript part is the most challenging in a web application design, in non geographic web site the use of javascript is limited as much as possible, this because it increase the weigh of the page and slows the performances of the browsers or, in some and limited cases, can be inaccessible by some browsers which have the javascript interpretation disabled.

A modern approach is the rule of the “unobtrusive javascript” which means that the javascript code should be an enhancement of the web surfing experience but the page should be fully accessible and readable without the javascript code working. Even the Ajax technique should be avoided.

If this approach could make sense for common web sites like blogs newspapers or presentation sites it is completely inapplicable to geographic sites where the simple action of panning a map is javascript managed and where is not imaginable to refresh the page each time a query on the data is done.

The best the developer can do is to write as less javascript as possible and avoid complicated structure to keep code easy to be executed by the browsers. Concerning browsers must be noticed that in general the modern javascript engine are well build and no longer suffer of historical memory leaks or execution speed lacks except for the Internet Explorer engine which until the version 9.0 has still some memory issues.

Javascript is borne to be used ad a structural language, it is object oriented but doesn't have the concept of classes and methods. Modern libraries simulate the class structure by using the "prototype" attribute of js objects which allow to extend dynamically the inner functions and attributes.

The end user is expected to use this code instantiating classes and assign objects sequentially. A javascript file host working logic like cycles, comparisons and Ajax with the application parameters like map extension, controls, layers. This implies that the developer builds one or more javascript pages for each application he build and in some cases could happen that different applications have similar logic with different parameters.

In the MCV pattern the business logic should be separated from the contents and the layout so it should be avoided to change the entire javascript logic (which actually is mixed with configuration parameters) for each application built.

A new solution adopted for the most modern lthaca applications is to have business logic separated from the configuration even for the javascript code and not only for the server side logic. This solutions forces the developer to change the way the javascript has been built to work with and increases the complexity level but allows to further speed up the javascript management and application deployment.

The solution is based on three different files which take care of different tasks:

- JSON configuration file which host all the parameters of the application
- JSON library of common used functions
- business logic file which parses the configuration and build the web app using the library file

In the best solution possible the library and the logic should be unique for all the applications and just the configuration has to be customized depending on the needs. New functionalities can be added to the logic or library without impacting on the existing applications. However this panorama can be achieved for general purpose applications where the main tasks are to visualize and query data. For the applications where is requested specific data interactions it is a challenging work to keep the business logic intact. This is not the case of emergency management where, at least in the first phases, is important to have data visualized and updated quickly and thus this architecture work pretty well.

This architecture can work with or without server side logic, the javascript code is static as well as the html. To build a new application the developer is expected to manually change the configuration file and thus to know the javascript code.

16.3.1. Reference systems

Data are commonly store in the database using the WGS84 Geographic Coordinate system, however often there is the need to use external layers such as Google, Yahoo or Bing maps. Those data are shared as raster tiles using the projection called Spherical Mercator which is WGS84 Mercator projection with meters as unit system. The main reason is that the earth shows less compressed near the poles and look more comfortable to navigate even if the distortion introduced are relevant.

In order to have the own data to overlap on those layers a re projection of them is needed; Geoserver is able to serve data in many different projections and OpenLayers can even perform changes on vector data.

To get Spherical Mercator data from Geoserver each layer must be flagged with the option “SphericalMercator” set to true in each layer definition and the map should have the “InternalProjection” set to “EPSG:900913” which is the international code for the google projection.

16.4. Identification of the expected dynamic parts

The desired goal is to make dynamic the architecture described above, dynamic means that the creation of the configuration file should be automatized by a server side logic which, under user instructions, compile the parameters and get ready the file for the application.

The configuration file is a JSON file in the form of key values pairs and the business logic file expects keys to be the same or to not be present, the parts that should be dynamic are the values. A server side logic should accept input parameters from the user for instance through a web from and modify some of the values of the JSON file.

Parameters that can be changed are:

- map configuration parameters such as extension, projection, zoom levels
- application controls such as pan, zoom, query, rulers, mouse position
- layers displayed divided between base layers and overlays
- layer groups to logically split them in to categories
- application layout such as main areas and layer switcher location
- toolbar buttons

- disclaimer

Concerning the interface customization the user should be guided through two or three predefined options in order to keep the map readable, the options can be suggested by the purpose of the map. A map which have a display purpose with limited query capabilities can be a full viewport map with a floating window hosting the layer manager otherwise a map with the purpose of query a large amount of data should have a dedicated sidebar with enough space inside to visualize query results.

16.5. Design of the server side architecture

The expected tasks of the server side logic are:

- URLs desing
- database definition and interaction
- business logic
- template generation
- web site security

The entire logic is written in Python and managed by the Django development framework which takes care of the low level functionalities and help to keep separated the different logics of the MCV pattern.

16.5.1. URLS design

URLconf is the module that is in charge to manage the designed URLs, through the file `urls.py` the developer choose which addresses the application should respond to and which logic is bind to them.

Good URLs never change, a URL identifies univocally a resource across the web, for resource is meant a source of information that can be a service as well as a flat page. The step of designing URLs is the definition of the services that a web application should share.

The basic addresses are the main WebGis application page and the configuration form to it, others are the geoserver instance address with its web services (wms, wfs with their implementations) and the geonetwork metadata catalog address.

However a developer can write custom URLs for each service he want to deploy.

16.5.2. Database definition and interaction

The database interaction can be done in two ways depending if the database is or can be managed by the Django models or if the database is created by others and should just be read by the business logic. The use of the Django models allow a more strict control on the database fields by wrapping custom data types around the database native types, for instance a field labeled as a http URL will be checked by Django before being written to the db as a normal string field. Using models, then, there is no need to write SQL strings to interact with the database both for reading and writing. On top of the models is easier to build both input forms and administration pages.

However if the database is created and managed externally such as the SDI db, to interact with it is necessary to manually write the connections to the db and the SQL as well. The low level connection wrapped between python and postgres is a module called Psycopg2. Django has a tool called “inspectdb” wrapped by the manage.py program which try to create a models file once inspected an existent database but for complex structures such as an SDI is not properly working.

16.5.3. Business logic

As a core component for a dynamic web application, the business logic is an engine which receives the http request from the urlconf module, retrieve data from the db through the models module and sends back the result to the client in the form of a template, of parameters or a static file.

In Django the business logic is called “views” because they are a mask for the database. The views are written in the file views.py and some of them are directly called when a specific URL is called. Through the views is possible to get the http parameters and to answer in a different way depending which type of request is done (GET or POST).

The http request parameters retrieve is used to compile the JSON configuration file respecting user needs. A web form can send data to a specific view which is in charge for javascript completion.

16.5.4. Template generation

Concerning the desired architecture the expected templates are two:

- the main html file which is actually simple but which brings the title and external files names to be included in the application, these should be generated
- the JSON configuration file which has javascript code to be partially generated

The way the templates are managed is by matching in the file some tags written by the developer and switch these tags with the corresponding parameter get from the db or from internal calculations.

Django leave extreme flexibility in the template definition, inside the files can be present simple named tags as well as cycles or simple presentation functions but is not allowed to modify the data.

However if the cycles and presentation functions can be useful for html generation should be avoided for javascript generation, this because the javascript is an html generation language itself and javascript debug tools are extremely poor. Most of the time, when debugging javascript, the only information got when an error occurs is the line at which this has been raised, if a server side code dynamically add lines of code it becomes hard to fix the bug because the line number indicated by the client is not the same of the original file. The better solution is to use only substitution tags which change just the strict needed.

Substitution tags are written in the form `{{ tagname }}`, when the Django parser matches the tagname switches it with the passed parameter with the same name.

Cycle tags are instead written in the form `{% for element in array %}write something for each entry in the array {% endfor %}`.

16.5.5. Web site security

For a web site based on form completions the most challenging security issue is the SQL injection, this type of attack is based on the concept that if in a form field is written an SQL string and then the content of the form is sent to the database it will execute the injected SQL. This can mean the complete deletion of the database.

SQL queries always and with a semicolon, a common form field can have bind a function that executes something like `"INSERT INTO table form_content ;"`, a injection could be to insert in the form something like `"; DROP DATABASE dbname ;"` this code will do an empty insert and the a complete database drop.

Fortunately the Django functions associated with the `urlconf` module have all the security defenses. The design of of a URL in Django is done through the use of the "regular expressions" or "regex". This matching string notation was initially implemented in the Perl language and is a powerful tool to interact with strings and numbers controlling length and content. With the `regex` module Django avoid unexpected characters and data types, the control is done just after the form submission.

Another security check is due to the internal architecture design, the urlconf module is an interface to resources and files but none of them are directly published in a web accessible folder. For an external is impossible to read the source code both for the python code and for javascript generated code and neither know the actual location of these files.

17. Developed services

The developed application and services during the three years of the Ph.D. period are based on different steps of the architecture described in the chapter 16.

First services were built upon the only javascript component while after the study of the python and django libraries most of them are been enhanced with server side technology.

The services are of three categories:

- project specific for publication of the researches of my colleagues
- emergency response
- WebGis pure research

17.1. Flood world wide alert system

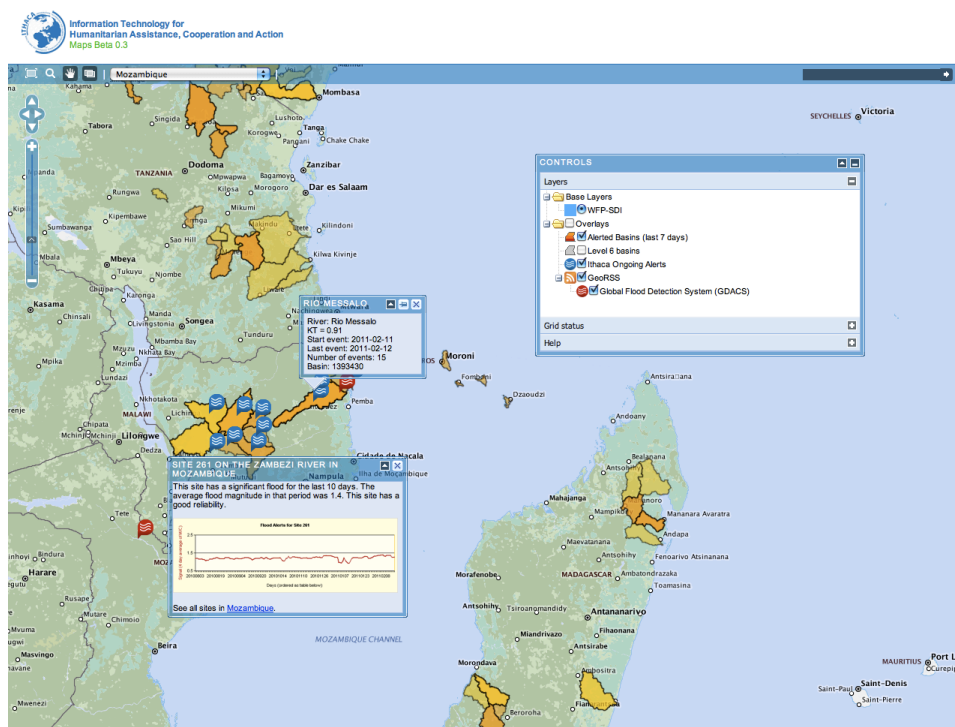


Figure 34: Flood world wide alert system

This application has been one of the first developed and then evolved with some features from the new architecture.

The full code is available in the attachements.

The purpose of this project is to share across the web the potentially flood affected areas world wide produced by the Flood Early Warning Alert System developed by Ithaca.

The application maps in near real time potentially on-going alerts on watershed basins all over the world: the application meets the need of WFP of global monitoring of potential humanitarian emergencies.

TRMM (Tropical Rainfall Monitoring Mission) data are automatically downloaded every three hours and are processed by the system.

In terms of elaboration, the Flood systems need to analyze all basins around the world in a short time, less than three hours. In consideration of this, a specific grid computing infrastructure is adopted. The grid allows resource-sharing and problem-solving in a dynamic Virtual Organization (VO). The architecture is composed of nine nodes for the data analysis and a master node for the grid management. The master node is configured as a Grid Resource Information Service (GRIS), and manages the soft-state registration of the characteristics of resources. It contains information about every resource belonging to the distributed architecture. The other nodes (Grid Index Information Service (GIIS)), can only elaborate the jobs for a specific region. For each watershed basin a specific parameter (Kt) is computed [Albanese et alii, 2008] in order to identify critical events.

The statistical method of deriving the extreme events is done by the Gumbel distribution [Chow 1954] and permits the calculation of the recurrence interval of the extreme rainfall values, measured in years and called return time (RT). Here the return time curves are calculated considering 2, 5 and 10 years and using the frequency factor (Kt) introduced by Chow [Chow 1954], function of the probability level. The formula for the Kt, expressed on the basis of Gumbel distribution is:

$$K_T = 1 - c_v \times \left[0.45 + \frac{\sqrt{6}}{\pi} \times \ln \ln \left(\frac{RT}{RT - 1} \right) \right]$$

where: Kt = the probabilistic factor of growing,

RT = the return time taken in account,

cv = variation coefficient of the averages of the annual maxima, expressed by :

$$c_v = \frac{\sigma_d}{\mu_d}$$

where:

σd = standard deviation of the maximum rainfall of duration d;

μd = average of the maximum rainfall of duration d.

When Kt values of a watershed basin is higher than a threshold which is computed for each basin starting from the historical rainfall dataset, the event is recorded in a PostgreSQL table.

17.1.1. Web application

The web application offers the fruition of this continuously updated dataset: besides mapping Kt based alerts, by coloring alerted watershed basins of the highest detail according to a color map related to Kt value (yellow in case $Kt \leq 1.5$, orange in case $1.5 < Kt \leq 1.9$, red in case $Kt > 1.9$, Fig. 3), the system makes it possible to query the dataset, by means of the map or of specific tools.



Figure 35: Flood application tooltip

A world-wide map shows alerted basins and, in order to facilitate the access to a higher detail of information in a quick way, specific tools for rapid data accessing and map navigation are available. In this way users can focus on every potentially alerted area in an independent way from the dimension of the basin on the map.

Beside normal map controls like pan, zoom and zoom to full extent (in the left part of the toolbar), the system has a drop down menu that shows a dynamic list of countries there is an ongoing event in. The selection of one of them causes the map to zoom to the country extent. Once the selected country is in the center of the map, the system generates blue markers on each alerted watershed level-6 basin. Each marker has a tooltip showing the basin, the country and Kt value for the related alert.

The map also shows alerts from a GeoRSS feed published by the JRC GDACS system (red markers); the related tooltips can show the chart of the last ten days of rainfall trend.

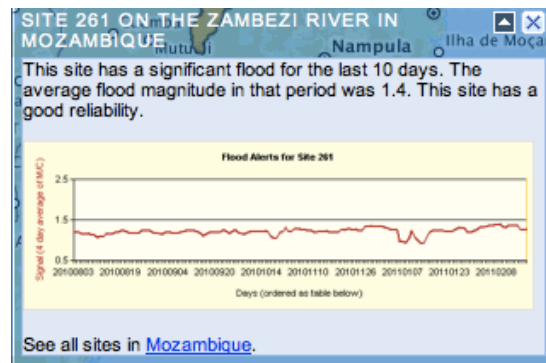


Figure 36: Flood application GeoRss tooltip

The management of the layers is entrusted to a floating window which holds the layer tree, the current status of the grid system and a help tab. The floating window can be closed to a toolbar button or collapsed.

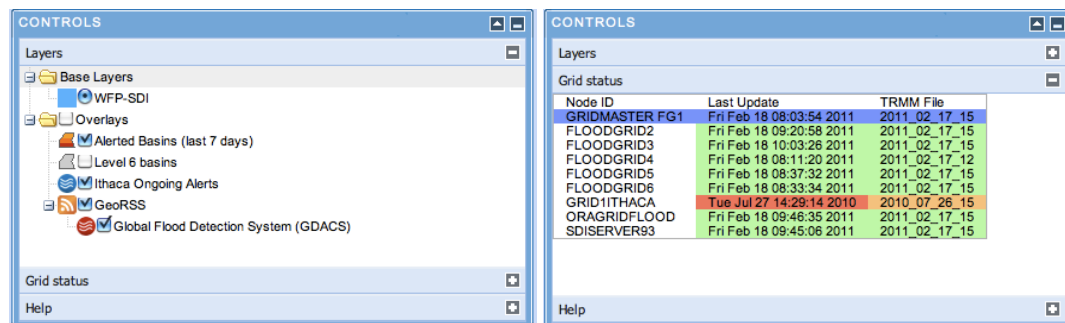


Figure 37: Flood application layer tree and grid status

Users can gather information (shown on the toolbar) about basins by using an automatic identify tool and clicking on the desired point on the map.



Figure 38: Flood application toolbar

For a rapid country focusing, the web application accepts country names directly in the URL (i.e. /floods/bangladesh). By parsing the URL the system can automatically zoom and center on the country, drawing the related markers. If specific data is available for a country, it will be added to the list shown in the layer switcher (i.e. ground station rainfall measures available for Bangladesh).

17.1.2. Database

The database is composed by SDI data present in the main geodb production schema. Data taken from this database are:

- base layer composer at geoserver layer with transportation, political boundaries, settlements, ground cover, geonames

- river basins at level 6 (the smallest river basins available)

Other data is taken from the GRID generated database which holds alerted basins as well as the GRID status.

17.1.3. *Server side logic*

Server side logic takes care of:

- render html and javascript templates base on the selected country and alerter countries
- dynamically retrieve data for alerted basins
- generate an RSS feed
- retrieve the grid status in order to display it in the window
- perform spatial topographic searches

The involved files are:

- alert.py, a class which represent the alerts
- flood_params.py, a module which host database parameters with methods to get and set them
- grid.py, a class which represent the grid element status
- rainfall.py, a class which represent the rainfall event
- station_coords.py, a dictionary which host ground station position in Bangladesh
- views.py, the main business logic core

The workflow starts from the initial login to the page which is done through the address <http://stage.ithacaweb.org/flood/countryname> the countryname parameter can be empty, for the countries with composed name the words are expected to be separated by a underscore symbol.

This http call triggers one of the two lines in the urls.py file:

```
(r'^floods/$', flood.init),
(r'^floods/(?P<country>\w+)$', flood.init),
```

These are regular expressions which assign two urls to the *flood.init* view, the second line takes the country parameter, expected to be after the “flood” name, this parameter is directly passed to the *init* view as a country parameter. With this trick we can obtain from the url the desired country to be focused by the application.

The *init* view for this GET request is simple:

```
def init(req,country='worldwide'):
    if req.method=='GET':
```

```

        return render_to_response('flood_main.html',{'country': country})
    elif req.method=='POST':
        do more...

```

It take as first argument the http request (used then to send back the results) and the country parameter, by default set to worldwide (the case of empty countryname in the url).

If the request method is GET it renders the html template by passing it the country parameter, the same coming from the url typed by the user.

In the html file the country parameter is used to name the two javascript file that will be requested to the server while loading the page:

```

<script type="text/javascript" src="flood_{{ country }}.js"></script>
<script type="text/javascript" src="flood_tools_{{ country }}.js"></script>

```

In the case of empty country parameter the files names will be, *flood_worldwide.js* and *flood_tools_worldwide.js*.

These script declaration will trigger two http requests to the *urls.py* module which will be kept by the:

```

(r'^floods/flood_tools_(?P<country>\w+).js$', flood.toolsRenderer),
(r'^floods/flood_(?P<country>\w+).js$', flood.olRenderer),

```

Like previously these lines call the *toolsRendered* and *olRenderer* functions in the *views* file by passing the country parameter.

The two functions render the javascript files with the country name and country extent retrieved from the database. With those information the application can then be zoomed and centered to the desired place.

```

def toolsRenderer(req,country='worldwide'):
    return render_to_response('flood_tools.js',{'country':country,'countries':
alertedCountries(),'elements': gridStatus()})

```

```

def olRenderer(req,country='worldwide'):
    extent = ""
    if country != 'worldwide':
        words = country.split('_')
        countrycam = ""
        for word in words:
            countrycam += word.capitalize() + ' '
        ext = countryExtent(countrycam[:-1])

```

```
extent = ext if ext is not "" else "-20037508.34 -20037508.34 20037508.34
20037508.34"
```

```
return render_to_response('flood.js',{ 'country':country, 'extent':extent})
```

Javascript files are also passed with information about alerted countries and the grid status.

To get the extent of the country the name is camelized (to match the db standards) and then passed to the *countryExtent* function which takes care of the database connection and data retrieval.

The function *alertedCountries* generates a query to the database to get currently alerted countries and returns an array of their names ready to be put in the application toolbar. And the *gridStatus* function reads information about the status of each node of the grid to be shown in the apposite tab inside floating window.

Other functions are used for specific requests from the application such as topographic search which uses the passed city name to match an entry in the database and get the coordinates of the center of the related polygon.

The *getTip* is used to retrieve information about the alerted basins inside the chosen country. These information are used by the javascript to create markers with associated tool tips in the center of the alerted basins.

17.1.4. Client side logic

As anticipated the client side logic is composed by two main file *flood_countryname.js* and *flood_tools_contryname.js*. However these files are compositions of other smaller files.

The roles of them are:

- *flood_countryname.js* takes care of the map configuration (map parameter, layers and controls)
- *flood_tools_countryname.js* takes care of all the functionalities and layout definition
- The tools file is composed by:
 - *flood_tools.js* with main functions
 - *grid.js* which takes care of the grid tab rendering
 - *layer_model.js* for the layer manager tab
 - *toolbar.js* for the toolbar rendering and functionalities

It's up to the views to populate and mix them together to prepare the end file.

The tools file is build using JSON notation where key are functions and parameter, the *init* function is called by the flood.js file while the onload event of the page is triggered calls other functions which build each component of the page.

Components are created in the following order after the onload of the page:

- map object creation with layers and controls
- toolbar creation by calling the *CreateToolbar* function (toolbar is positioned in a stand alone html container)
- page layout creation by calling the function *createViewport* (this function uses the layout manager of Ext.js)
- floating window creation by passing the layer_model to its constructor
- creation of the button to switch on/off the layer window
- activation of the toolbar and window
- registering a click event on the map with an Ajax controller associated to allow a user to get information about the name of the clicked river basin on the map
- attachment of on select events on the markers to allow the opening of tool tips while clicking on one of them
- custom additional logic if the selected country is the Bangladesh because of the availability of ground station data

Functions like *getExtentFromCountry* and *autoComboBox* are called when specific events are triggered by the user, in the first case when he choses an alerted country in the drop down menu in the toolbar, the other when he uses the topographic search tool in the toolbar. These tools have a listener attached which trigger when the user interact with the object, a listener is attached using a syntax like:

```
Ext.get('countries').on('change',this.getExtentFromCountry,this);
```

Where after get the container ('countries') with the *on* method we can attach a 'change' event which triggers the *getExtentFromCountry* function.

The 'change' event means that the listener will trigger the bound function when the selected element in a list changes.

The service also share a GeoRSS feed creator, this is a xml template managed by the *createFeed* view. Inside the <channel> tag a cycle iterate through the array of alerts to create an <item> block for each entry.

The general style of the application is managed by a CSS file which overrides the default style rules of Ext.js.

17.2. Drought monitoring system

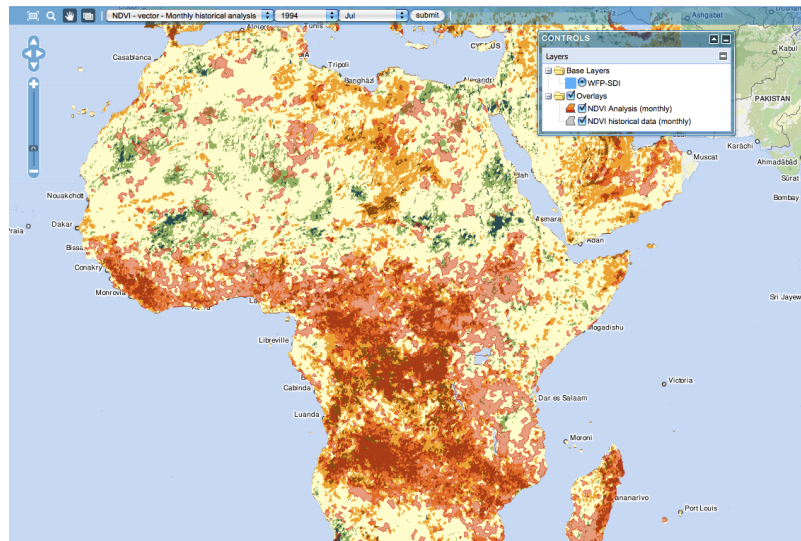


Figure 39: Drought monitoring system

This application is borne as a viewer for NDVI index trend and anomalies to help the comparison between different time period of the data and show the progress of the research team in the development of an early warning system for drought response in Africa.

Presented data are elaborations of NOAA AVHR satellite data and GIMMS NDVI historical series of about 25 years from 1981 to 2006.

Data are of two types:

- raster layer for each year and month of NOAA data NDVI situation
- vector monthly synthesis between all years of GIMMS data as anomalies aggregation

The application is still in a draft state because it has been planned to evolve as the project reach determined milestones.

17.2.1. Web application

The web application has a full view map with a floating window hosting layer management and a toolbar which, besides navigation tools, have two drop down menu to choose which data type and for which time period to display.

It allow to overlay vector data on top of the raster base and visually compare differences between the two data type, it is possible to dynamically regulate the opacity of the overlays.

It offers also a python interface to interact with the database.

17.2.2. Database

NDVI raster data are stored directly on the file system as it is not a good rule to have them in the database which is designed for tabular data, however the are stored the image references and metadata such as the reference time period. The NDVI vector synthesis is instead entirely stored in the PostGis instance using polygon data type with the geographic coordinates reference system.

The database has been created and managed through the Django models as shown in the models.py module.

Data retrieval is thus done through python statements and no SQL instructions are required.

17.2.3. Server side logic

The logic is composed by two modules:

- models.py which hold the database model
- views.py which holds the business logic

In the models it has been used the GeoDjango spatial extension by overriding the default models model with the spatial one with the import statement:

```
from django.contrib.gis.db import models
```

A spatially enabled class is then created by using the inheritance from the spatial Model class:

```
class Drought_Raster(models.Model):  
    code
```

The business logic has function, *getLayer*, to find the desired layer on the database while passing it the desired time period (year and month for rasters or just moth for vectors). Using Django models filters is quite easy to retrieve desired information from a database:

```
Drought_Raster.objects.filter(y=y).filter(m=m)
```

On the entire dataset *Drought_Raster.objects* the function filters entries by year and by month.

Another function is dedicated to the javascript rendering, in particular the dynamic part are the years drop down menu in the toolbar which is supposed to grow as more years are involved in the analysis.

17.2.4. Client side logic

Like the floods application the javascript part is composed by two files:

- drought.js as a map configurator with controls and layers
- drought_tools.js with the application layout and tools

Despite the flood application however the drought.js file is not dynamic considering the fact that the only dynamic part can be considered the years drop down menu which is defined in the tools file.

The visualization of a layer is subordinated to the submit button placed in the toolbar which triggers the *getCollectInput* function and through the urls module get data from the database.

17.3. Snow cover monitoring system

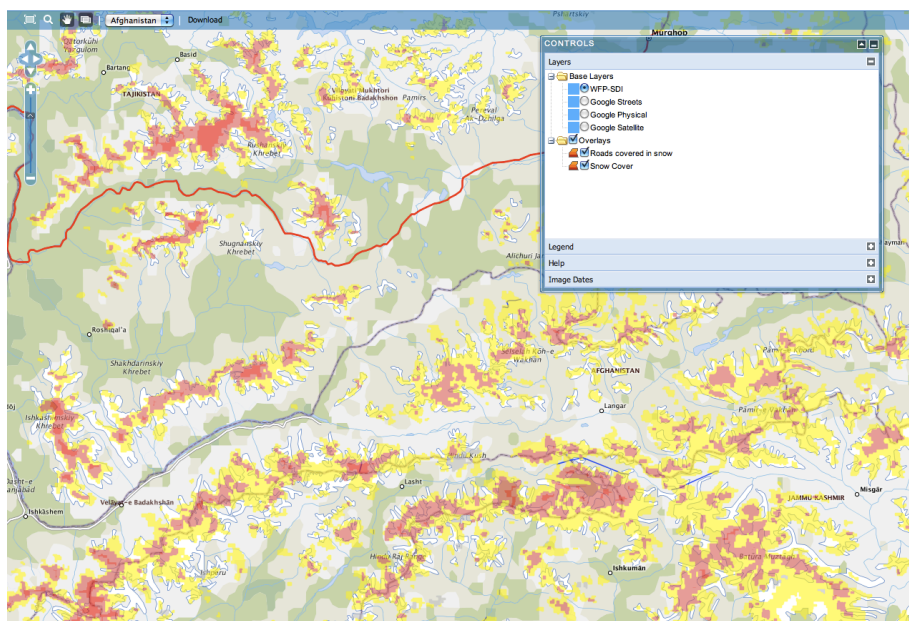


Figure 40: Snow cover monitoring system

This web application automatically daily maps snow coverages in two areas of interest for WFP (Afghanistan and Nepal), and highlights the part of the road network which has possible problems.

The extracted vector data is the results of an automated procedure which elaborates MODIS (Moderate Resolution Imaging Spectroradiometer) data, in particular the daily MODIS Snow Cover MOD10_L2 product (geometric resolution 500 m).

17.3.1. Web Application

Access to the application require to specify the Nepal or Afghanistan name in the url, the map will be centered and zoomed on the correct area.

The system present a full page interface with floating window hosting layer manager, legend, help and dates of the last images used and a toolbar with the switcher menu between Nepal and Afghanistan region and a button to download vector data in shapefile format.

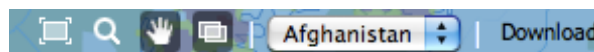


Figure 41: Snow cover toolbar

Other than the SDI background is also available the Google data in the three version of street, hybrid and physical called through the OpenLayers API.

17.3.2. Database

Data are stored by the automatic procedure in different tables:

- high snow level
- medium snow level
- low snow level
- no data
- snow covered roads

This tables are both for Nepal and Afghanistan. The data are grouped in two main WMS layers, one for Nepal and one for Afghanistan, which are directly used in the web application.

17.3.3. Server side logic

The system of detecting the country name used in the url is similar to the one used for the flood alert application, however the main challenge has been how to modify the menu selection, visualized data and downloadable data regarding the viewed country.

The views internally know the spatial extents of both the interested areas and the html template is passed with the the country name and its extent.

Specific modules are developed to get and return the extents while triggered by an user action on the drop down menu.

17.3.4. Client side logic

The javascript files are:

- snow.js as main map configurator with controls and layers
- snow_tools.js with the application logic and interface definition

To avoid duplication of the code for each of the two countries, the logic has been developed to keep track of the country viewed while accessing the page and to easily switch from a country to the other.

The line:

```
restrictedExtent: new  
OpenLayers.Bounds(country_extent.split(",")[0],country_extent.split(",")[1],country_extent.  
split(",")[2],country_extent.split(",")[3]),
```

splits the variable and uses the values to define the extent.

In order to have the layers changing when the country is chosen, a function inside the snow_tools file is triggered to get the extent from the server and to remove the layer and then create a new one with new data.

```
Ext.get('countries').on('change',this.getExtentFromCountry,this);
```

And the listener on the download button recognize which country is currently displaying and address the correct file. The url module will correctly redirect to the actual position of the tar file on the file system.

```
Ext.get('download').on('click',function(){  
    if (Ext.get('countries').getValue() == 'Nepal')  
{window.open('http://address/snowcover nepal_snow.tar')}  
    if (Ext.get('countries').getValue() == 'Afghanistan')  
{window.open('http://address/snowcover/afghanistan_snow.tar')},this);
```

17.4. Cyprus cultural heritage catalogue

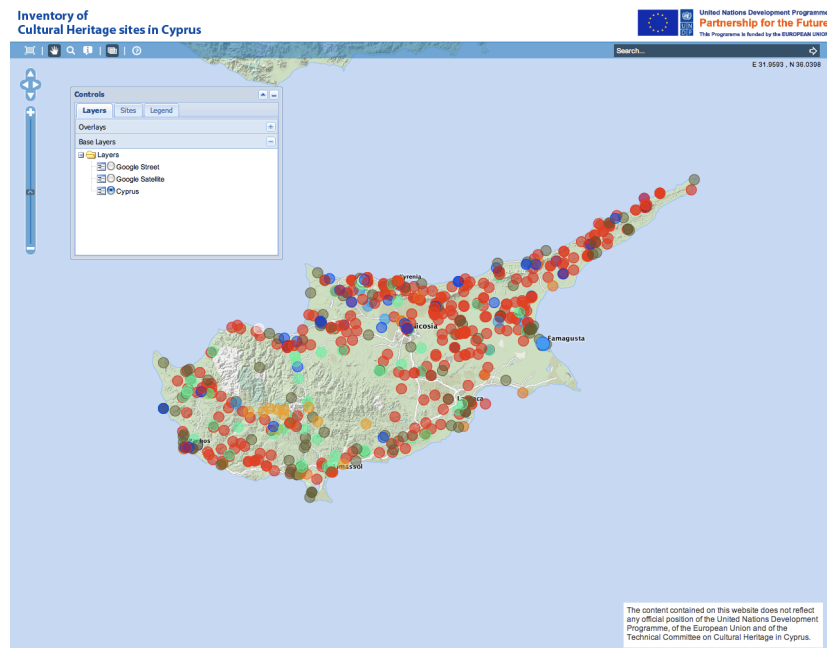


Figure 42: Cyprus cultural heritage catalogue

The Cultural Heritage catalogue has been developed for the UNDP (United Nations Development Programme) in order to collect and share information about archaeological sites, historical buildings and all significant human artifacts.

The data are been gathered by several relief missions to the Cyprus isle in order to collect photos and position for each of the 1000 sites suggested by the UNDP.

The application has been developed following the new full MVC architecture both for javascript and python; the database is managed by models modules.

17.4.1. Web application

The system offers access to about thousand points displayed on the Cyprus base map; available layers are:

- ITHACA SDI representation both with full toponyms and mute background with just six main cities
- Google layers satellite, street and physical
- Sites overlay
- Labels overlay composed by Ithaca data to be overlapped on Google satellite background

The toolbar host common controls as well as a query tool used to activate the click event on the sites layer and a search tool able to harvest between cities and sites.

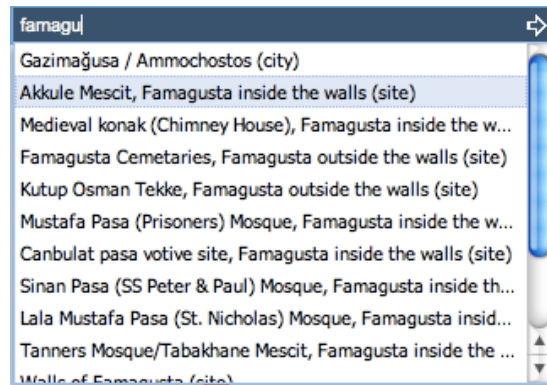


Figure 43: Cyprus search tool

In the floating window are located three tabs which host layers, currently viewed site list with category filter and the legend.

At startup are shown all points on the map except if the browser use id internet explorer, this is due to a smaller capacity of the explorer's javascript engine which cannot manage more that 150 features at the same time.

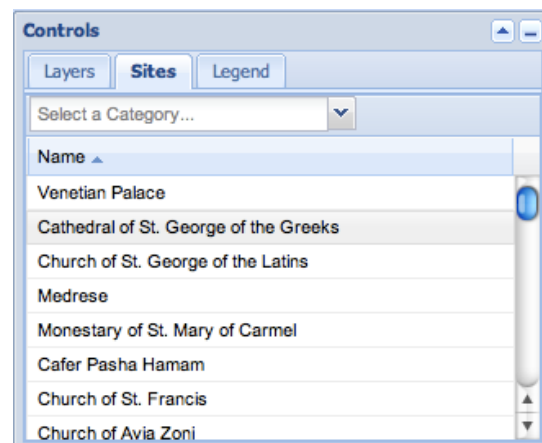


Figure 44: Cyprus control window

The features can be selected by using the proper control in the toolbar or by clicking on a name in the sites tab list. Sites list is dynamic and shows only the features currently wrapped by the map's bounding. While zooming the number of available features is reduced.

The selection of one feature opens a tool tip on it which display the photo, the district of the site and some specific pdf links if available.

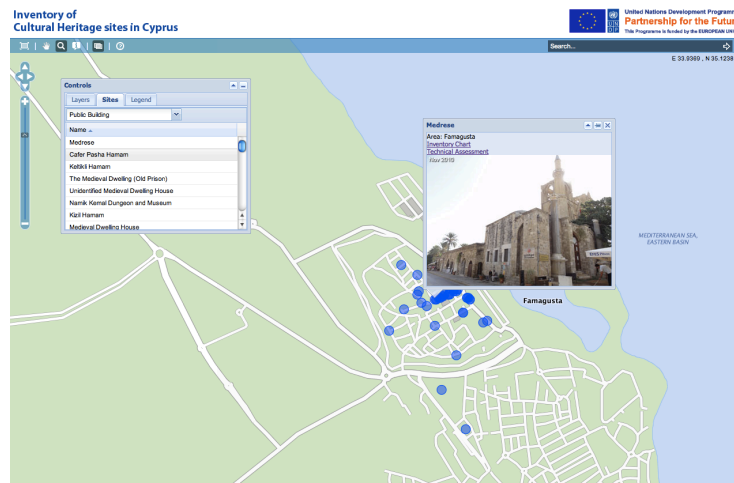


Figure 45: Cyprus tooltip on a feature

Features are categorized and displayed with different colors, available categories are:

- Archaeological sites
- Religious buildings
- Cemeteries
- Historic towns and centers
- Neighborhoods and public places
- Public buildings
- Military architectures
- Domestic architectures
- Water works
- Bridges



Figure 46: Cyprus legend

The choice of one category in the sites tab triggers a filter that displays only the features of the chosen category and reduces the sites listed in the tab.

The application exposes a database administration interface which allows privileged users to add, modify or delete sites, the background logic is configured to force each site

120

to endure an approval process once is added to the database; only approved sites will be displayed on the application.

The screenshot shows a web-based administrative interface titled "Change Cultural Heritage". It features a form with the following fields: "Site id" (5_PA_094), "Name" (Aqueduct and Millhouse), "Area" (Paphos), "City name (Turkish)" (Hirsofa), "City name (Greek)" (Chrysochou), "Lat" (34.990001), and "Lon" (32.439999). The "Category" is set to "Water Work". The "Image" field displays a thumbnail of a current image, "arch_img/APPR_004.JPG", with a "Change" button and a "Select file" option. Below the form is a map showing a geographical area with a point marker. A "Delete all Features" link is visible at the bottom left of the map area.

Figure 47: Cyprus admin interface

The privileged user can edit each single field in the database and a map helps the graphical localization of the desired point. A listener automatically fills up the lat and lon fields when the user picks up a point on the map or automatically creates the point on the map when the user fills up the lat and lon fields.

17.4.2. Database

The base layers are a custom export of the SDI with customized geo names having turkish and greek names displayed.

Sites database is instead managed by custom python models, each field is written in the models.py file. The database is configured to receive files such as photos and pdf files. The models.py file takes care of data consistency by checking string length and file size and format.

17.4.3. Server side logic

Server side logic is composed by the following files:

- views.py, the business logic
- models.py, the database model and interface
- importer.py a csv parser user for initial import of the data from excel tables
- urls.py, the urlconf module
- admin.py, the administrator interface manager

The views have three functions:

- the *init* which is called while accessing the page and which has the task of check the user identity and ask for the login if not before redirecting to the main html page
- the *wfs* function which is a workaround for local development process, this method is called only if the application is called locally on the developer machine and takes care of calling the geoserver instance running on the 8080 port. On the production server in fact, the geoserver instance respond on the classic 80 port because Apache uses a proxy for it.
- the *topoSearch* function is called when the search field is used, this tool calls the models to retrieve cities and sites from the database with their respective latitude and longitude. The method used to send result to the client is JSON encoding

In the *models.py* file is designed the Sites database, the table is described in the *ArchSites* class while writing the name of each field like an attribute. Each field has a field type with its constraints (*max_length*, *unique*, *null*, *blank*...); constraints will be translated by Django to SQL constraints.

Default data types are been extended with the “*ContentTypeRestrictedFileField*” which is module written to check the file size and type before authorizing the upload to the database; this type has been used to manage upload able files.

A particular field is used to allow the geometry management, this type is taken from the GeoDjango extension and automatically creates a geometry field in the table using Postgis.

The default save method has been overridden to manage the possible image removal when chosen in the administration interface.

The *admin.py* file host the administration interface management, it has a class called *OSMGeoAdmin* which overrides the default geo manager of GeoDjango by defining custom lat and lon of the administration interface map and by defining three fieldsets used to logically group displayed field in the interface in order to keep simple the Sites creation and edit.

17.4.4. Client side logic

Javascript files are:

- main.js, the business logic which creates the structure of the application
- lthaca_tools.js, a library which define some commonly used tools
- cyprus_config.js, the configuration file with layers, interface and controls definition.
- The configuration file is written using JSON notation and has four main parts:
- map configuration
- toolbar configuration
- layer window configuration
- viewport configuration

In the map part are defined layers as well as controls and map options in the form:

```
layers: {  
  world: {  
    address: addresses.geowebcache,  
    name: 'Cyprus',  
    params: {  
      layers: 'cyp_baselayer_m',  
      format: 'image/png8',  
      sphericalMercator: true  
    },  
    options: {  
      isBaseLayer: true  
    }  
  }  
}
```

This notation is completely different from the classic OpenLayers layer declaration:

```
new OpenLayers.Layer.WMS(layer.name,layer.address,  
layer.params,layer.options);
```

It's up to the business logic file to translate this notation to the OpenLayers one and instantiate its classes.

In the definition of toolbar buttons are defined the “handlers” these blocks of logic are triggered when the user clicks on the button, they activate controls as well as deactivate the others which must be exclusive and change the css class of the button in order to modify its appearance.

The main.js file has the logic which parses the configuration file and creates the structure. Generally the parsing of the block defined in the JSON notation are read through cycles which change their action while considering a particular parameter written in the declaration, for instance the “type” parameter triggers different layer type instantiation.

However to face presence of the high number of technological content of the client side logic, some particular portions of code are been written:

- automatic legend generation by matching categories name and feature styles
- custom feature filtering considering approval flags, spatial position and category of the sites
- custom toolbar buttons handling by grouping in different logical scopes (navigation tools, query tools)
- custom tooltip management with check of the pdf file existence
- link between displayed features on the map and listed sites in the window

The ithaca_tools.js file is a factory for common functions, is host:

- viewport, toolbar, buttons creators
- tooltip creator
- topographic search
- json reader
- getCapabilities reader

The topographic search tool has a workaround to face the possibility that a user search for a feature which is not currently displayed on the map and thus is not present in the DOM structure, in this way the map would be moved to the feature location but no feature would be displayed because the map ask for new feature each time is moved or zoomed and the AJAX mechanism is asynchronous so takes some time to get the answer and the local code doesn't wait for it. The tool creates a fake feature before the AJAX response is get and thus the feature and the tooltip are correctly displayed.

17.5. Haiti emergency WebGis

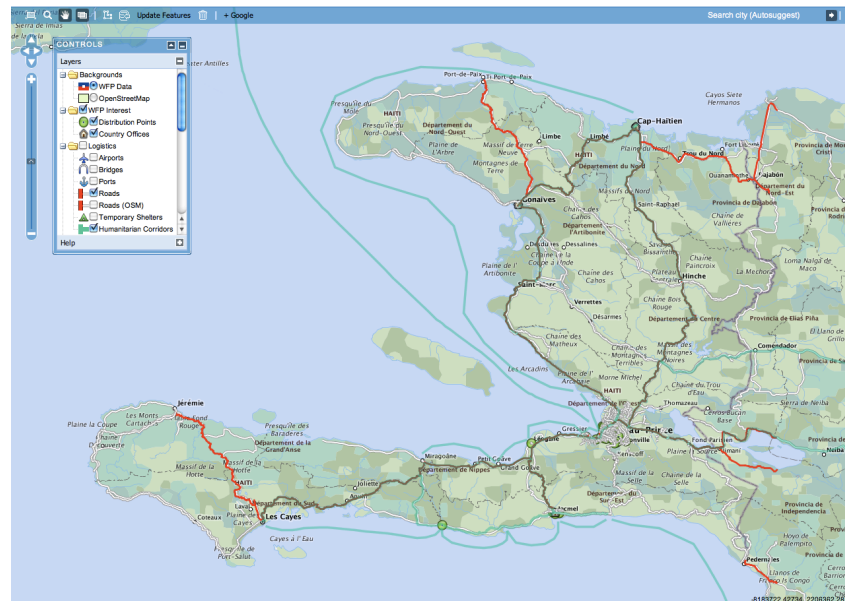


Figure 48: Haiti emergency WebGis

On the 12th of January 2010 the country of Haiti has been hit by a catastrophic earthquake which caused hundred and thousand dead.

The WFP, which has always been deployed in Port-au-Prince, has immediately started the post disaster response procedures, even many agencies spread around the world have started producing any kind of analysis the day after the earthquake.

Ithaca, in collaboration with the WFP and the spatial agencies has preformed a damage assessment on high resolution satellite and aerial images by recognizing collapsed buildings, spontaneous camps and displaced people.

From the field the WFP was collecting data about food distribution points, food stocks, country operational bases, humanitarian corridors.

Ithaca was asked to build a web application for data sharing to allow both field and head quarters users to get the data as soon they were produced or updated.

Thanks to the MCV background architecture, the application has been built in one day for the basic viewing capabilities; for the more advanced tools the entire development has taken three days.

During the period of deployment of two colleagues and I in Port-au-Prince, we have acquired georeferenced images of almost all the street of the city using a device developed by Ithaca called LCMMS (Low Cost Mobile Mapping System).

The tool is composed by four commercial web cams and a gps system able to acquire 5 frame per second for each cam and spatial position.

This survey has been used by Ithaca and other agencies to validate the damage assessments done via remote sensing technologies and to check the state of some particular building such historic houses.

The acquired images are been included in a beta version of this web application which show the road track with clickable points and street images.

17.5.1. Web application

The system is a full map interface with toolbar and floating window for layer management. Displayed data are:

- SDI or OpenStreetMap as base layers
- Distribution points
- Country offices
- Airports, ports, bridges, roads
- Temporary shelters
- humanitarian corridors
- damage assessment (collapsed buildings, damaged infrastructures, collapsed bridges, closed roads, restricted roads)
- population density grid
- custom user polygons

Application features allow to explore data through pan, zoom tools; a search field is locate on the right part this tool can be use to search a city or village and automatically zooms and re centers the map on it, has an auto suggest function integrated which displays a list of possible matching names after typing at least four words.

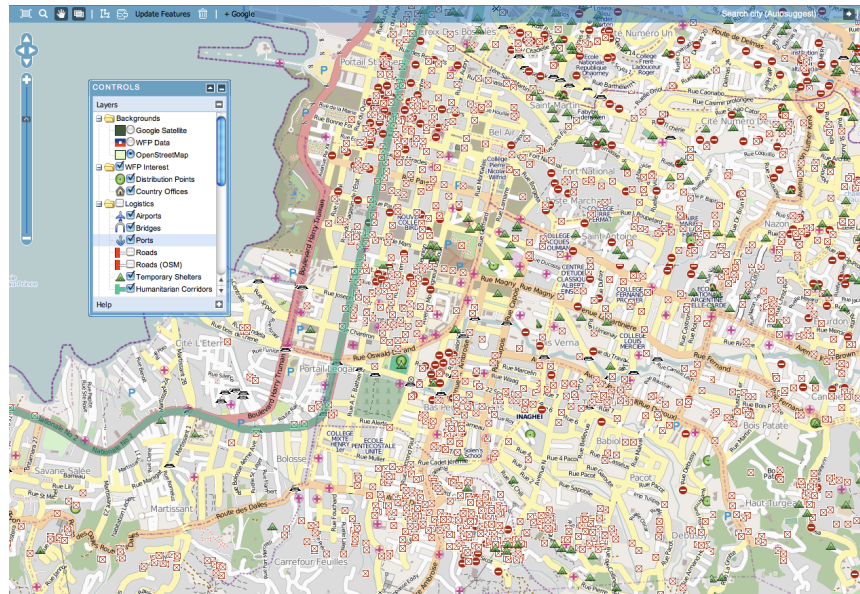


Figure 49: Haiti, view of Port au Prince with damage assessment

The layers displayed in the floating window can be right clicked to regulate the opacity as well as zoom to the layer's extent.

In the toolbar is present a “add google” button that dynamically adds google among selectable base layers, this also dynamically loads google javascript libraries. This solution has been designed to avoid a massive javascript load when the page is accessed and have more features on demand.

An experimental feature is the possibility for a user to draw a polygon on the map and add it into the database, this can be used to social mapping purposes and stored data can be used to compare damage assessment with on the field contributions. The controls for this feature are located in the middle of the toolbar and are: “draw/edit polygon”, “load existing data from the database”, “commit the feature to the database” or “delete feature”.

17.5.2. Database

Layers besides the user drawn polygons are direct export of the Ithaca SDI on a PostgreSQL instance and are published with Geoserver and GeoWebCache.

The management of the custom polygons is managed by the GeoDjango models through the declaration:

```
class Polygons(models.Model):
    geom = models.PolygonField(srid = 900913, editable = True, null = False, blank = False)
```

The database table is defined to have the Google Spherical Mercator projection and to have features editable.

The application exposes an administration interface to manage edit and check committed polygons.

17.5.3. Server side logic

Server side logic is composed by:

- views.py, the business logic
- shortcuts.py, a wrapper for custom rendering functions

The views.py has functions to:

- dynamically generate javascript files
- manage user action to draw, edit or delete features
- city search

The actions related to the polygons interactions are done via POST requests and can be “addFeatures”, “loadFeatures”, “delFeatures”.

The “addFeature” request calls the AddGeometry function by passing it the request, data are passed from the javascript in JSON format and are parsed and added as a polygon through the models.

The “delFeature” calls the "delGeometry" function by passing it the id of the feature to be removed.

The “LoadFeatures” triggers an massive load from the database; the objects are parsed by the “render_to_geojson” module (written in the shortcuts.py file) which translate in JSON notation the polygons and directly sends the result to the web with correct formats.

17.5.4. Client side logic

The client side is composed by:

- haiti.js, the map configuration with layers and controls
- haiti_tools.js, the general application layout
- layer_model.js, the definition of the layer tree
- layer_model_google.js, the layer tree with google base layer block
- toolbar.js, the configuration of the toolbar layout
- editing.js, the logic which handles the editing capabilities.

The haiti.js file has the “addGoogle” function which dynamically add the Google API and the Google layer in the WebGis; to load the API it adds a “script” tag in the Head of the web page which forces to load the javascript content from the remote Google address. Once the APIs are loaded the layer model must be updated, this can be done by passing the name of the function to be called directly as a callback parameter in the URL of the API such as “*http://maps.google.com/maps?file=api&v=2&key=&async=2&callback=createGoogleModel*”, in this case the “createGoogleModel” function is called when the APIs are finished to load, this function redraws the DOM structure of the layer tree and adds a new layer both in the map and in the window.

Edit functionalities are triggered by the toolbar button which are designed to call specific functions, this is done defining particular “handlers” launched when a click event occur on the button.

The functions defined in the editing.js file are:

- serialize, to convert to JSON notation the geometry component of the feature
- collectFeatures, collect all the features that aren't committed to the database yet, passes them to the serialize function and sends them to the server backend via POST request.
- deserialize, is called to translate back the JSON notation to the DOM structure.
- load features a function to retrieve existing features from the database
- deleteFeatures to remove a feature from the database if is committed otherwise simply removes it from the DOM structure.

On the server side the method “init” correctly manage the http POST requests triggered by these javascript functions.

17.6. Development environments

The development WebGis generator is the test environment where to build new functionalities and implement the MCV architecture. It is composed by two branches, one where is studied the dynamic javascript generation through python and the other where is developed the javascript MCV structure.

Figure 50: WebGis generator starting interface

The first environment allow a user to choose projection and layers and to get a custom web application generated on demand via the Django architecture, the second is mainly pure javascript and allow to dynamically add layers to the existing web app through a getCapabilities tool.

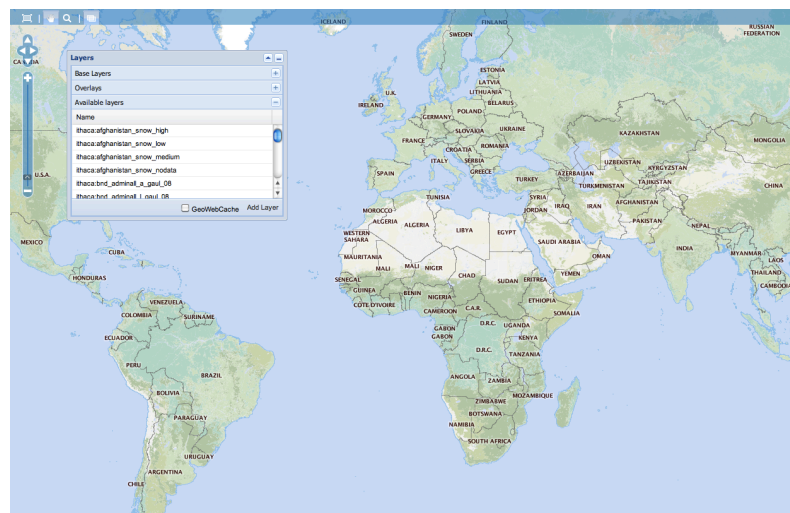


Figure 51: GetCapabilities tool inside the MVC javascript application

17.6.1. Web application

The WebGis generator present a web form as first page which allow to choose between the Spherical Mercator projection and the WGS84 projection as srs used by the application, the user is even asked to add layers by inserting the web wms address, the layer name and if the layer is a base layer or an overlay.

The submission of the form opens a new window with a new WebGis with the chosen data, the application has a side bar with the layer tree and a central area with the map.



Figure 52: Automatically generated WebGis

The MCV javascript application allow to dynamically add layers to the map by choosing them from a panel in the floating window, the layer list is get by a `getCapabilities` http request on the server. In the toolbar are located the classical map controls.

17.6.2. Server side logic

The server side logic is relevant for the WebGis generator while the other service is almost pure javascript.

The architecture has been developed to be highly modular and extensible for further developments.

Involved files are:

- `views.py` the main logic which get http request form the web and creates the map structure
- `genesis.py` the main map container
- `map`, `layer` and `wms.py` locate in the folder components which are the classes representing the the map with its parameters, a generic layer class and a wms class which inherits from the layer.

The `views.py` file is in charge to instantiate the “Viewer” class which is intended to be the WebGis instance. Each subclass is written to instantiate its own sub components, the viewer creates the container which host the map ad layers.

Parameters are picked by the “Viewer” from the POST request and dispatched to subclasses. The “map” class receives the projection parameter and optionally the *maxExtent* and *restrictedExtent* parameters, the others are calculated internally.

Once the structure is configured the viewer instance is used to fill the javascript templates to replicate on the client side the situation of the server. This approach makes

the server know the state of the client and can be useful for further developments for instance to save the state of a generated WebGis application.

17.6.3. Client side logic

For the WebGis generator the client side is composed by javascript templates rendered with the information created on the server, file are:

- viewer.js, the main configuration file which is assembled with the following three
- controls.js, the configuration file with controls declarations
- map.js, the configuration file with map declarations
- layer.js, the configuration file with layers declarations

Another file is the “main.js” which is creates the form shown at startup, this file uses Ext forms and Ajax to send encoded data to the server.

The way the javascript files are dynamic is by using Django template tags inside the code like:

```
projection: new OpenLayers.Projection('{{ map.projection }}'),  
maxResolution: {{ map.maxResolution }},  
maxExtent: new OpenLayers.Bounds{{ map.maxExtent }},  
restrictedExtent: new OpenLayers.Bounds{{ map.restrictedExtent }},  
zoom: {{ map.zoom }},
```

What is inside the double brackets is interpreted by the Django template system and modified with the appropriate value.

The javascript architecture of the MCV application is more complicated even if is not dynamic. The idea behind this application is to build a modular and MCV javascript which can work without a server side environment, however it can be transformed in dynamic by modifying just the configuration file without changing the logic at all.

The Cyprus (Chapter 17.4) application has been developed on top of that javascript architecture.

Involved files are:

- main.js, the main business logic container
- ithaca_tools.js, the library with common tools
- ithaca_config.js, the configuration file in JSON notation

The business logic can read the configuration file and, using the tools library, build a working WebGis application. Simply by changing few lines inside the configuration file is possible to add controls, layers or tools in the toolbar.

If new logic associated with a particular tool is needed it can be passed as an handler in tool declaration such as:

```
layerWindowButton: {  
    tooltip: 'Toggle layer window',  
    tooltipType: 'title',  
    enableToggle: true,  
    iconCls: 'layers',  
    id: 'layerWindowButton',  
    target: 'layerWindow',  
    icon: iconPath+'flood/img/layers.png',  
    pressed: true,  
    toggleHandler: function(){  
        var target = Ext.getCmp(this.target);  
        if(target.hidden){  
            target.show(this.id);  
        }else{  
            target.hide(this.id);  
        }  
    }  
}
```

The *toggleHandler* defines the behavior of the window button located in the toolbar.

If however new background logic should be added to improve the overall application it can be added to the main file without the risk of impact on the existing structure, this new code should just take care of being activated only if required.

18. Further developments and improvements

After a contact had with the CEO of the Open Geo, an American enterprise which develops commercial web mapping applications using open source tools, we are been asked to participate in the development of an open source system called GeoNode. Open Geo was the first initiator in the GeoNode development and after a first phase has given the project to the community to enhance and continue the work.



Figure 53: GeoNode home - <http://geonode.org/>

GeoNode is an open source platform that facilitates the creation, sharing, and collaborative use of geospatial data. The project aims to surpass existing spatial data infrastructure solutions by integrating robust social and cartographic tools.

At its core, the GeoNode has a stack based on GeoServer, Django, and GeoExt that provides a platform for sophisticated web browser spatial visualization and analysis. Atop this stack, the project has built a map composer and viewer, tools for analysis, and reporting tools.

GeoNode use the same technologies as Ithaca uses and goes in the same direction which is automate the web application to allow rapid mapping and data sharing adding tools to face emergencies.

GeoNode is a system which manage and publish spatial data using a distributed SDI in order to have data available to everyone, users can add their own data or use the available one, they can build a WebGIS with custom styles and can even share the chosen style. GeoNode integrate both Geoserver and GeoNetwork and thus can publish metadata on demand respecting different formats.

As map viewer it uses the GeoExplorer javascript application. Based on the GeoExt library this application is a powerful tool to view, export, save and print maps with

customized layers and styles. Layers can be chosen by a list of default servers or from a custom source, for vector data the style can be managed with the simple GeoExt Styler interface.

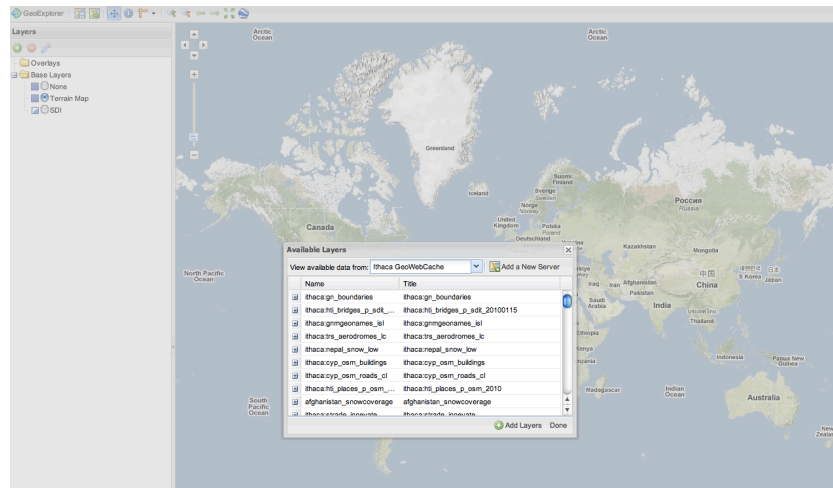


Figure 54: GeoExplorer view with layer adding window from Ithaca server

Main features are:

- Make it extremely simple to share data
- Provide user statistics
- Easily add comments, ratings, tags
- Allow collaborative filtering
- Provide rankings of best 'views' and data sets contributed—such as highest rated, most viewed, most shared
- Allow connectivity between several GeoNode instances to augment the collaborative potential of government GIS programs
- Simple installation and distribution
- Automatic metadata creation
- Versioned metadata
- Search via catalogues and search engines (Google)

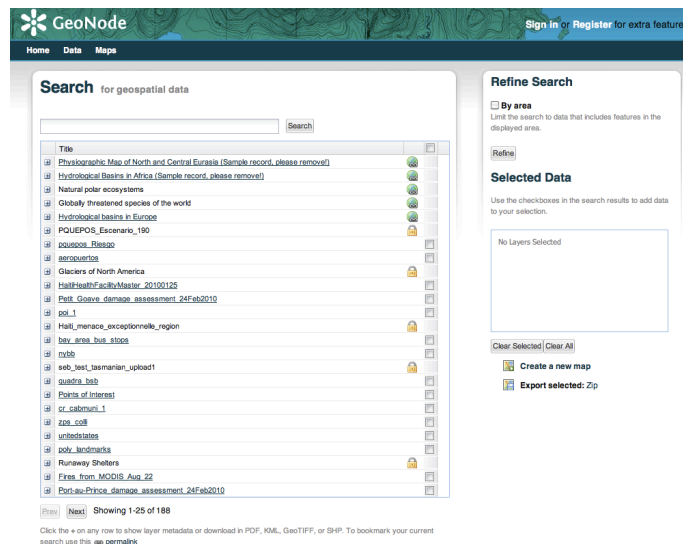


Figure 55: GeoNode data search interface

GeoNode can be the platform where to integrate the experience gained during year of cooperation with the WFP.

Major features to be added are:

- Exporting capabilities (kml, shapefile)
- Printing capabilities (in custom and professional layout)
- Layout customization
- Routing capabilities

GeoNode APIs should be extended both for the server and client side keeping the modular approach and respecting the MCV pattern which has lead the entire development of the project.

Currently a deployed version of GeoNode has been set up on ITHACA server at the address <http://geonode.ithacaweb.org>. This instance is integrated with ITHACA publishing environment (geoserver) and metadata catalogue (geonetwork).

Conclusions

The work done during the three years has followed many ways exploiting different technologies in order to find and focus on what could be the best solution both in terms of tools and development techniques to use. In order to be helpful during an emergency, a WebGis system should be quick to build, easy to use and enclose complicated functionalities in simple tools.

The developed applications have been used or are now being used by different users and services. The Haiti emergency application was used by the WFP for months after the earthquake, and their comments were that it was the perfect tool to look for the updated data and overall situation easily.

The Flood Monitoring System is currently used by users and services at all levels. The web interface is a useful tool for monitoring the overall situation and the background grid performances and the web services are integrated directly in the <http://www.hewsweb.org/hp/> Humanitarian Early Warning Service website and in the [JRC alert site](#).

The developed architecture has focused on flexibility and customization following the best available development trends and technologies. In 2010 started the development of Geonode was started (chapter 18). This is a collaborative mapping tool based on the exact technology and architecture developed in ITHACA and following the same idea of automation as much as possible. This has allowed ITHACA to start a close collaboration with the developer community in the development and improvement of Geonode, integrating its know how in the field of emergency response.

Moreover, it should be taken into consideration that an emergency response web application is subject to extremely frequent changes both in the functionalities and in data content, thus the system must be developed respecting a modular pattern avoiding repetition of code or irrational tricks to apparently solve problems.

It is not infrequent to lose control of the developed code after few changes or improvements. In order to deal with this, it is suggested to spend much time in the development phase following some steps which are crucial such as the data model, the architecture design, the tool libraries, the user interface. However the most powerful tool is the refactoring of the code. Each time new improvements are made, the developer should take some time to look afresh at the code and rationalize it in order to be definitive.

In this thesis I've given many suggestions on how to develop a better WebGis but also on how to manage an entire server structure regarding security as well as consistency. Of course there always are improvements that can be made and the technological evolution is continuous. However what is really important is to understand the technology is and will be developed and how a project should be built in the best way in order to be interoperable between platforms and standards.

The web is the best source of knowledge in which to find ideas and suggestions for problems similar to those we are facing, because working with open source is valuable but it repeatedly presents us with situations which we are often unable to solve. On the web there are many excellent developers who often write articles or entries in blogs, which are an excellent source of knowledge, tips and tricks.

As a final thought I would like to say that working with Open Source for Web development is a multidisciplinary field where a knowledge of hardware infrastructures, programming language, information security, user interface and accessibility is needed. Although this knowledge requires a lot of study, it allow us to create and invent new solutions in a wide variety of topics. These are the ingredients needed by those who love challenging and stimulating works. I love this work.

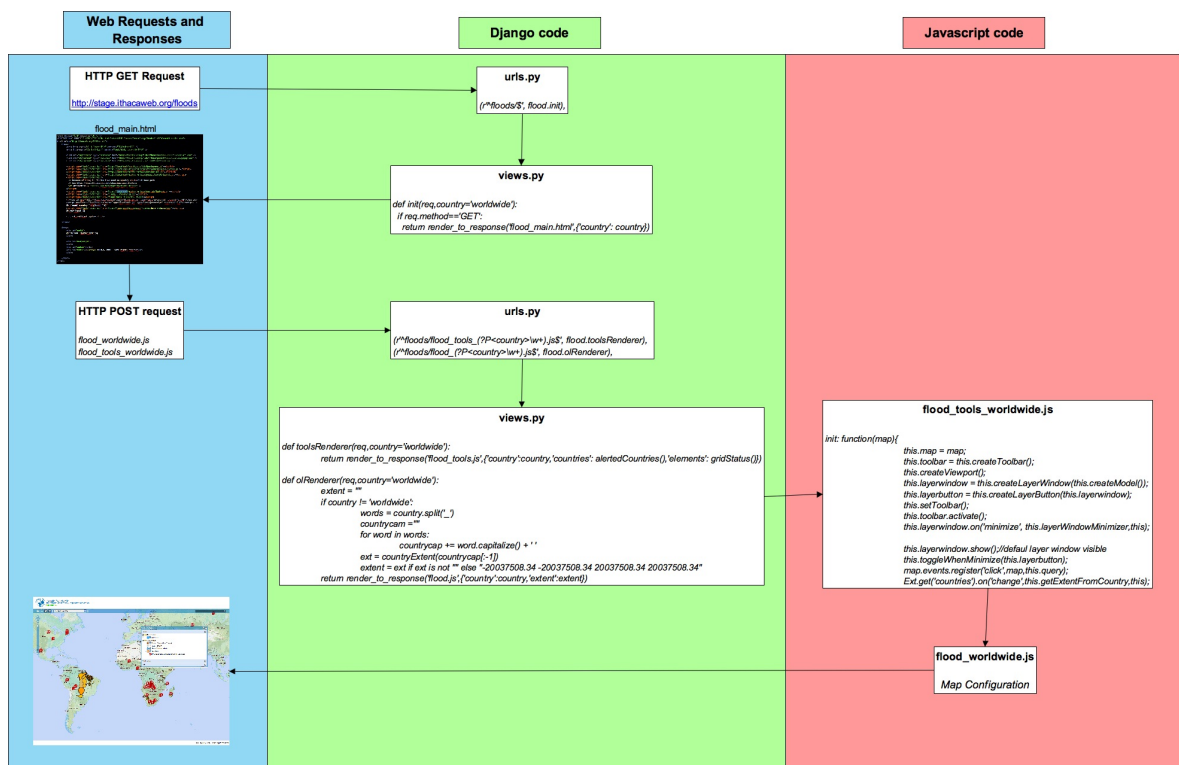
Bibliography/Webography

- Gis for web developer: Adding where to your web applications (Scott Davis)
 - Desktop Gis: Mapping the planet with opens source tools (Gary E. Sherman)
 - Internet Security (Maurizio Cinotti)
 - Learning Python (Mark Lutz)
 - Javascript: The definitive guide (David Flanagan)
 - Professional Javascript for web developers (Nicholas C. Zakas)
 - Pragmatic Version Control (Mike Mason)
 - Practical Django projects (James Bennett)
 - The Definitive guide to Django (Adrian Holovaty and Jacob Kaplan - Moss)
 - Sviluppare applicazioni web con Django (Marco Beri)
 - Tomcat: The definitive guide (Jason Brittain)
 - Learning Ext.js (Shea Frederick, Colin Ramsay, Steve Blades)
-
- <http://www.wikipedia.org/>
 - <http://www.djangoproject.com/>
 - <http://geodjango.org/>
 - <http://www.postgresql.org/>
 - <http://postgis.refractory.net/>
 - <http://www.html.it/>
 - <http://geoext.org/>
 - <http://openlayers.org/>
 - <http://www.sencha.com/products/extjs/>
 - <http://geoserver.org/>
 - <http://tomcat.apache.org/>
 - <http://www.apache.org/>
 - <http://geowebcache.org/>
 - <http://www.geonetwork.it/>
 - <http://subversion.tigris.org/>
 - <http://www.nabble.com/>
 - <http://grass.fbk.eu/>
 - <http://www.qgis.org/>
 - <http://www.kyngchaos.com/index>

- <http://www.gdal.org/>

Attachments

Flood Early Warning working schema



Web Server installation on Debian Lenny

<i>Operating system installation</i>	2
User creation	2
Internet configuration	2
Upgrades	2
<i>Synaptic installations</i>	3
<i>Independent installations</i>	3
Tomcat	3
Geoserver	3
Postgis	3
<i>Configurations</i>	4
Apache 2	4
mod_python configuration	4
Cgi directory	5
Iptables	5
<i>Tomcat</i>	6
Manager	6
Tomcat autostart	7
<i>Geoserver</i>	7
<i>Repository subversion</i>	7
<i>Postgresql</i>	7
<i>Openssh</i>	8
<i>Spatial database</i>	8
Users and tablespaces creation	8
Spatial database creation	8

Web server installation on Debian 5 Lenny amd64

Operating system installation

User creation

The main user created during the installation has the group “username”
other user can be created referring to the same group or to another group with:

- *sudo groupadd group name*
- *sudo useradd -s /bin/bash -g group name -m user*

choose a password with:

- *sudo passwd user*

insert created users in the *sudoers* file

sudo nano /etc/sudoers

modify as follows

```
root  ALL=(ALL) ALL
user  ALL=(ALL) ALL
```

Set the root password:

sudo passwd root

Internet configuration

Set the ethernet parameters by editing the file:

sudo nano /etc/network/interfaces

then:

sudo /etc/init.d/networking restart

Upgrades

When at the startup appears available updates launch the following command to install them.

sudo apt-get upgrade

if dependencies are needed to update the package the command is:

sudo apt-get dist-upgrade

Synaptic installations

Synaptic is the default package manager, these installations can be even done through apt-get interface.

```
django
apache2
postgresql-8.4-dev
sun-jdk-ejre (update-alternatives --config java)
proj
gdal
python 2.6.4
psycopg2
reportlab
openi-scsi
mod_python
subversion
openssh
```

Independent installations

These installations are not well managed by synaptic or apt-get and need to be done by hand.

Tomcat

Download *apache tomcat*.

```
tar xvf apache-tomcat-x.x.x
mv apache-tomcat-x.x.x tomcat
chown -R www-data:www-data tomcat
sudo mv tomcat /usr/local/tomcat
```

open the *profile* file:

```
sudo vi /etc/profile
```

at the bottom insert the environment variables to the java virtual machine and tomcat:

```
export CATALINA_HOME=/usr/local/tomcat
export JAVA_HOME=/usr/lib/jvm/java-6-sun-x.x.x.x
```

Geoserver

Download the *war* distribution of Geoserver from the Geoserver site.
Move the file inside the */usr/local/tomcat/webapps* folder to deploy it

Postgis

Download *postgis*

```
tar xvf postgis.x.x
sudo chown root:root postgis-x.x
cd postgis-x.x
```

```
sudo ./configure
sudo make
sudo make install
```

Configurations

Apache 2

Activate the proxy e proxy_http modules:

```
sudo a2enmod proxy
sudo a2enmod proxy_http
sudo apache2ctl restart
```

Modify the *proxy.conf* file:

```
sudo vi /etc/apache2/mods-enabled/proxy.conf
```

```
<IfModule mod_proxy.c>
    #turning ProxyRequests on and allowing proxying from all may allow
    #spammers to use your proxy to send email.

    ProxyRequests Off

    <Proxy *>
        AddDefaultCharset off
        Order deny,allow
        #Deny from all
        Allow from localhost
    </Proxy>

    # Enable/disable the handling of HTTP/1.1 "Via:" headers.
    # ("Full" adds the server version; "Block" removes all outgoing Via: headers)
    # Set to one of: Off | On | Full | Block

    ProxyVia On
</IfModule>
```

mod_python configuration

Activate mod_python:

```
sudo a2enmod mod_python
```

The configuration tells Apache which setting file to use for the specified Django folder, Apache will use the mod_python handler instead of the default to execute the python code.

```
<Directory "/home/django_ithaca/">
    SetHandler python-program
```

```

        PythonHandler django.core.handlers.modpython
        PythonPath ["'/home/django_ithaca','/home'] + sys.path"
        SetEnv DJANGO_SETTINGS_MODULE django_ithaca.settings
        #PythonOption django.root /home/django_ithaca
        PythonDebug On
        PythonInterpreter django_ithaca
    </Directory>

    <Directory "/home/django_ithaca/static">
        SetHandler default
    </Directory>

```

The last part is used to tell Apache to use the default handler for the static files.

Cgi directory

By default the CGI directory is in */usr/lib/cgi-bin*. In order to have the cgi files accessible on the web is necessary to link this directory to the */var/www*. However the best solution is to create a new one dedicated into the */var/www* folder.

```
sudo mkdir /var/www/cgi-bin
```

modify the file *000.default*:

```
sudo vi /etc/apache2/sites-enabled/000.default
```

modify the alias script and the handler as follows:

```

ScriptAlias /cgi-bin/ /var/www/cgi-bin/
    <Directory "/var/www/cgi-bin">
        AllowOverride None
        Options +ExecCGI -MultiViews +SymLinksIfOwnerMatch
        Order allow,deny
        Allow from all
    </Directory>

```

Restart apache

Iptables

Verify the default configuration of the Iptables with:

```
sudo iptables -L
```

The result should be:

```
Chain INPUT (policy ACCEPT)
```

target prot opt source destination

Chain FORWARD (policy ACCEPT)

target prot opt source destination

Chain OUTPUT (policy ACCEPT)

target prot opt source destination

With no rules inside.

All the following rules must be set as root user with the sudo command.

The first rule allows to establish connections in order to receive traffic.

iptables -A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT

Next rules are to allow traffic on specific ports:

iptables -A INPUT -p tcp --dport ssh -j ACCEPT

iptables -A INPUT -p tcp --dport 80 -j ACCEPT

At the end all the other connections are dropped:

iptables -A INPUT -j DROP

New rules should be inserted before this last rule, this can be done by inserting a number parameter after the INPUT instruction, this number specifies where to put the new rule from the beginning of the chain.

I.e.

iptables -I INPUT 1 -i lo -j ACCEPT

Save the rules in a file:

iptables-save > /etc/iptables.rules

Add the auto restore to the interfaces file:

sudo vi /etc/network/interfaces

Add it under the desired interface block:

pre-up iptables-restore < /etc/iptables.rules

Tomcat

Manager

sudo vi /usr/local/tomcat/conf/tomcat-users.xml

Insert:

```
<role rolename="manager"/>
<user username="admin" password="password" roles="manager"/>
```

start tomcat

```
sudo /etc/init.d/tomcatx.x restart
```

Tomcat autostart

```
sudo update-rc.d tomcat defaults
```

Geoserver

Modify the file users.properties:

```
sudo vi /usr/local/tomcat/webapps/geoserver/data/security/users.properties
```

modify the string like follows:

```
admin=password,ROLE_ADMINISTRATOR
```

Repository subversion

This configuration is for ssh connections to the remote repository.

Create an empty repository:

```
sudo svnadmin create /percorso/al/repository
```

import data in the repository:

```
sudo svn import -m "message" /path/to/project/folder file:///path/to/repository
```

Create remote working copy:

```
svn co svn+ssh://user@address/path/to/repository /destination/local/folder
```

To create a working copy on the server for publication use:

```
sudo svn co file:///path/to/repository /destination/local/folder
```

Postgresql

Switch on the tcp connections with:

```
sudo vi /etc/postgresql/x.x/main/postgresql.conf
```

decomment the line *listen_addresses* e change *localhost* with *

Once the db and the user are created modify the *pg_hba.conf* file:

```
sudo vi /etc/postgresql/x.x/main/pg_hba.conf
```

add the lines:

local dbname username "password"(this is the autentication method and thus must contain the string password)

for the remote hosts insert:

host all address/32 password

restart postgres:

```
sudo /etc/init.d/postgresql restart
```

Openssh

Modify the file *sshd.conf*

```
sudo vi /etc/ssh/sshd.conf
```

at the end modify as follows:

PermitRootLogin no
AllowUsers user names separated by a space

Spatial database

Users and tablespaces creation

```
createuser -S -D -R name
```

```
psql -c "ALTER USER name WITH PASSWORD 'password';"
```

```
psql -c "CREATE TABLESPACE name OWNER name LOCATION '/path/to/dir';"
```

(tablespace folder must be empty)

Spatial database creation

```
sudo su postgres
```

```
psql -c "CREATE DATABASE name OWNER name TABLESPACE name;"
```

```
createlang -d dbname plpgsql
```

```
psql -d dbname -f /usr/share/postgresql/x.x/contrib/lwpostgis.sql
```

```
psql -d dbname -f /usr/share/postgresql/x.x/contrib/spatial_ref_sys.sql
```

```
psql -d dbname -c "GRANT ALL ON spatial_ref_sys TO PUBLIC;"
```

```
psql -d dbname -c "GRANT ALL ON geometry_columns TO PUBLIC;"
```

Modify the file `/etc/postgresql/x.x/main/pg_hba.conf` and insert:

```
local dbname username password
```

To enter the database use:

```
psql -U username dbname
```