POLITECNICO DI MILANO

DEPT. OF ENVIRONMENTAL, HYDRAULIC, INFRASTRUCTURES AND SURVEYING ENGINEERING

DOCTORAL PROGRAMME IN ENVIRONMENTAL AND INFRASTRUCTURE ENGINEERING

MULTI-DIMENSIONAL AND MULTI-FRAME WEB VISUALIZATION OF HISTORICAL MAPS

Supervisor:
Prof. Maria A. BROVELLI

Tutor:
Prof. Fernando SANSÒ

Coordinator:
Prof. Fernando SANSÒ

Doctoral Dissertation of:
Luana VALENTINI
Student Id. 738792

Year 2009 - 2011 - XXIV Cycle

Acknowledgements

I would like to thank my colleagues Marco Minghini, for his precious work on historical maps georeferencing within the Web C.A.R.T.E. project, and Dr. Giorgio Zamboni, for his collaboration in writing the NASA World Wind Java code.

Abstract

In this thesis work a new approach to city evolution representation is described. In particular, a dedicated multi-frame WebGIS has been realized, in which data can be displayed not only on a 2D panel but also on a 3D virtual globe, taking into consideration the time domain too.
In recent years new GIS tools have been developed to allow a more efficient fruition of geographic information. The so-called virtual globes, like Google Earth, in fact, are becoming familiar also to non expert users for geographical data browsing, extending the perception of the interested area and giving a more realistic representation of reality. This also leads to the need of revising the way geographical data are represented in a WebGIS, which will be no longer only two-dimensional.
Furthermore, lately, many historical archives and collections have started a digitization process to enhance their cultural heritage. Early maps have a great importance for scholars and professionals because they describe with great accuracy the status of the territory represented and so they can be useful for urban planning and restoration projects.
After a georeferencing procedure, historical maps can be laid upon recent cartography or other maps in order to understand how the situation changed in time.
In the developed WebGIS it is possible to display the different map series made available by the State Archive of Como, superimposed on the current orthophoto of the city, served as a Web Map Service (WMS) by the National Cartographic Portal. City buildings are thematized according to the construction period and they are represented as extruded geometries on the virtual globe. Thanks to graphical interfaces, it is possible to dynamically display buildings according to different criteria, such as a user defined construction time period, or to display only buildings already built up in the year selected. JavaScript functions have been written in order to synchronize the two panels and allow time queries.

Contents

Introduction

In recent years, mapping technologies are undergoing a significant change since mapping went digital. As Mitchell [2005] says, one very effective way to make map information available to a non-technical end users group is to make it available through a Web page. In particular, data from different sources can be mashed-up within a single map, by using established protocols and specifications.
In addition to the traditional 2D display of geographical data, nowadays 'virtual globes' are used in order to extend the perception of the area of interest and to give a more realistic representation of the reality.
The aim of this thesis work is the realization of a first system that allows users to simultaneously analyse the same portion of the Earth both in two and three dimensions, taking into consideration the time domain. This is made possible by means of two panels: one for the usual 2D maps and one for a 3D window. The 2D map panel displays all the historical maps, the current orthophoto and the current city map. In addition to 2D maps, there is also a 3D buildings viewer based on NASA World Wind, an open source virtual globe where to place 2D and 3D objects.
In this study, 3D buildings are extruded using their mean height. Thanks to the graphical user interface, it is possible to dynamically visualize buildings on the globe according to different criteria, e.g. the construction time span, so that only the geometries fulfilling the requests are turned on.
Within the proposed application a synchronization between the two panels has been implemented, in, order to maintain a constant alignment of the two viewers, also regarding the time dimension visualization.
In the following, a brief description of each chapter is reported.

Chapter 1 describes the availability online of early maps. The different phases of digitization, georeferencing and online publication are presented. After these steps, early maps become available online and visible by a wider range of user, scholars and experts who wants to analyse temporal changes of the territory.

Chapter 2 focuses on the main geographic data modelling and standards. The main data transfer standards are presented, such as ESRI shapefile, GML and KML. In particular, different 3D modelling languages for cities are described.

Chapter 3 presents free and open source software and data in the GIS field. The main Web services are described, with a particular focus on the Web Map Service that will be used in the thesis project. With this kind of service, it is possible to handle multi-dimensional data and to take into consideration the 'time' parameter associated to a geographical data.

Chapter 4 describes the technologies adopted: MapServer at the server side, OpenLayers, Ext JS and GeoExt for the 2D frame at the client side and NASA World Wind for the 3D frame. An overview on 3D Web client visualization is reported, focusing mainly on open source technologies.

Chapter 5 illustrates the dataset: the orthophotos, the digital cartography and the early maps. In particular, map georeferencing process is presented, together with the obtained numerical results.

Chapter 6 presents the realization of the WebGIS describing the features implemented in the system, with a particular emphasis on the possibility of filtering buildings view, according to different time criteria, and the synchronization of the two frames view.

Chapter 1

Historical maps online

In recent years, the increased awareness of cultural heritage variety and worth owned by cultural institutions has revealed the need to set up projects aimed at recovery, restoration and enhancement of cartographic material.
Many institutions, both public and private, retain cartographic material of extraordinary historical, artistic and documentary value, consisting of ancient and modern maps, globes, atlases and planispheres that are still poorly known. Furthermore, historical maps are a dependable source of information regarding past city planning. To analyse maps content, the most practical method is the direct comparison with the present ones, for instance by overlaying them with the current cartography. Once early maps are digitized, in fact, they can be georeferenced to transform them into real geographic data. After this step, maps become metric representations of the territory and they can be directly comparable with current cartographic products.
For instance, it is possible to easily integrate them with Google Earth or Google Maps, taking advantage of the growing familiarity with these tools and their user interfaces. It is also possible to deliver old maps into other Web services through using open standards, as the Web Map Service protocol (see 3.1.0.0.1). In the next sections, those different passages will be briefly described and some projects of early maps archives on the Web will be presented.

1.1 Early maps digitization

In the last decade, thanks to the enlargement of the concept of cultural heritage and development of new technologies, several initiatives have been raised, some of which have integrated cataloguing activity with digitization, while others have just promoted digitization projects.

In Italy, the 'Istituto Centrale per il Catalogo Unico delle Biblioteche Italiane e per le Informazioni Bibliografiche' (ICCU - Central Institute for the Unique Catalogue of Italian Libraries and Bibliographic Information) wrote some guidelines for the digitization of cartographic material (Working group on cartographic material digitization [2006]).
According to these guidelines, a general criterion for material selection is its state of conservation: privileged documents are the most ancient, fragile, rare or large ones, that are difficult to use and more easily damageable. Before the digitization process, it is important to verify maps format and dimensions, eventually their colours and the presence of cuts and tears.
Most of the principles for map digitization are the same as for other library materials (Fleet [2009]), but the relatively larger size of maps, combined frequently with their need for details and colours, leads to more specific requirements. To guarantee a good final result, the operator has to take into account these three aspects:

- **Image capture**: to capture high-quality images there are three main types of device: studio cameras, sheet-feed scanners and flat-bed scanners. Cameras are used for capturing maps in volumes or early and fragile items. Although excellent results can be obtained, cameras often require a greater expertise to be used, more associated equipment (lights, stands, etc.) and they are vulnerable to vibration, focusing and lighting problems. Sheet-feed scanners are usually used as cheaper methods of capturing large, non-fragile, flat sheet maps. Images are captured at a consistent size and resolution, lighting and focusing are easier to standardise across the sheet. Flat-bed scanners at A3-size or smaller are common desktop accessories, but larger format devices are much more expensive. With these devices, the digitisation process is usually slower than for sheet-feed scanners.

- **Digital images**: digitization converts an image into pixels that can be black or white (binary), a shade of grey (grey scale) or colour. The term *resolution* indicates the spatial detail, i.e. the number of pixels in an image, often expressed as pixel per inch (ppi) or dots per inch (dpi). The *optical resolution*, instead, is the scanner inherent resolution, based on imaging sensor size and the magnification of the optical system. This is the most important resolution to look for when purchasing scanners. Interpolated resolution is a synthetic or calculated resolution that artificially increases the original optical captured resolution by image processing.
  Required resolution depends on the maps to be captured and the purpose of the image digitization process (i.e. whether for preservation, access

or thumbnail illustration).
Another important parameter is the *pixel bit depth*, the number of values for any pixel of the image (i.e. 1 bit - 2 values, black or white, 24 bit - 256^3 values). Most archival colour imaging uses 24-bit colour. Higher bit levels allow greater colour precision, exponentially increasing file sizes.
About file formats and compression, open and non-proprietary standards (such as TIFF and PNG) have to be preferred for preservation and interoperability over proprietary standards.

- **Metadata**: it is crucial to record metadata about images for several purposes: resource searching, management and preservation. There are different kind of metadata categories: administrative (for management and administration), descriptive (for resource searching, understanding and interpretation), preservation (for long term purposes and maintenance), technical (system functions and information behaviour) and use (level and type of use of resources) (Fleet [2009]).
In Italy, the ICCU recommends to use the MAG 2.0.1 standard for digitized maps. MAG, which stands for 'Metadati Amministrativi e Gestionali' (Administrative and Management Metadata) was published by ICCU in 2006 and it is defined as an application profile which aims to provide formal specifications for collecting, transferring and disseminating metadata and digital data in archives (Brovelli et al. [2012]).

A fundamental parameter to determine the digital image quality standards is the resolution, defined in ppi. For cartographic material, the shooting standard defined by the Library of Congress (Betz [1982, update 2000 - 2003]) is 300 ppi, the same used by the Italian State Archives and the Italian Geographic Military Institute (IGM). However, it has to be considered that tools with optics better than scanners (e.g. optical benches, professional cameras) can produce images equally satisfactory at a lower resolution (till 200 ppi). The criterion to use is definitely the full utilization of digital object: depending on circumstances, it is possible to make reasonable choices that take into account the best costs/benefit ratio.
From image capture to its accessibility in local systems or on the Internet there are a series of additional steps involving also non-neutral choices, such as image processing by controlling the brightness/contrast, colour curves, sharpness filters and finally compression algorithms. In case of digitization, the use of new tools should aim at providing an image similar as much as possible to the original one.
Finally, the format choice has a fundamental impact on the final accessibility:

the adoption of multi-resolution formats, which lets the user enlarge the image up to its maximum resolution, is nowadays widely used in digital mapping, and it is an established alternative to the traditional strategy of producing three different images: the high resolution master copy, the medium and the low resolution for other possible uses.

1.2 Maps georeferencing

Georeferencing early maps means to assign to scanned maps a 'metric reference' from the actual geospace or from its mappings (Balletti [2006]). This assignment is done by applying a geometric transformation, based on points of known coordinates, to the points of the non-metrical map. The points of known coordinates are called *ground control points* and the result of the transformation is the assignment of *predicted coordinates* to all points of the map.
The best-fitting transformation can be done by applying different algorithms, well documented in literature and operationally provided in a number of commercial software packages (Boutura and Livieratos [2006]). Among those transformation schemes, the simplest are the *similarity* (or conformal), the *affine* (widely used in satellite image processing), the *projective* (used for the rectification of central projections), the *polynomial*, usually the second order one, and the *finite elements* transformation.
In particular, a m order polynomial transformation can be described with the following formula:

$$\begin{cases} X(x,y) = & a_{0,0} + a_{1,0}x + a_{0,1}y + a_{2,0}x^2 + a_{1,1}xy + a_{0,2}y^2 + \cdots + \\ & + a_{m,0}x^m + a_{m-1,1}x^{m-1}y + a_{1,m-1}xy^{m-1} + a_{0,m}y^m \\ Y(x,y) = & b_{0,0} + b_{1,0}x + b_{0,1}y + b_{2,0}x^2 + b_{1,1}xy + b_{0,2}y^2 + \cdots + \\ & + b_{m,0}x^m + b_{m-1,1}x^{m-1}y + b_{1,m-1}xy^{m-1} + b_{0,m}y^m \end{cases} \tag{1.1}$$

where X, Y are the map coordinates, x, y the corresponding pixel coordinates and the coefficient subscripts recall the exponents of the image coordinates x, y. The particular case with $m = 1$ is known as the affine transformation. Putting $a_{1,0}$=a, $a_{0,1}$=b, $a_{0,0}$=c, $b_{1,0}$=d, $b_{0,1}$=e and $b_{0,0}$=f, the affine transformation can be written as:

$$\begin{cases} X = & ax + by + c \\ Y = & dx + ey + f \end{cases} \tag{1.2}$$

This 6 parameters planar transformation considers two translations along the axis, a rotation, two scale factors (one for each axis) and a deviation. This kind of transformation does not respect shapes and angles (a square can become a parallelogram).
A particular case of the affine transformation is the conformal transformation (or similarity), for which a=e and b=-d. It consists in a planar roto-translation with scale factor and it can be written as:

$$\begin{cases} X = & s\cos\theta \cdot x + s\sin\theta \cdot y + t_x \\ \\ Y = & -s\sin\theta \cdot x + s\cos\theta \cdot y + t_y \end{cases} \tag{1.3}$$

where c=t_x, f=t_y, a=$s\cos\theta$ and b=$s\sin\theta$.
This transformation is defined by only 4 parameters, that can model two translations along the axis, a rotation and a constant scale variation along the two directions. Using less parameters, this transformation has a lower ability with respect to the affine one to model deformations between the map and the image. Having no deviation parameter and a constant scale factor along the two directions, shapes are respected.
Since these functions correct local distortions at the ground control point location, they are very sensitive to input errors and hence GCPs have to be numerous and regularly distributed (De Leeuw et al. [1988]). A rule of thumb for GCPs location choice is that there should be a distribution of control points around the edges of the image to be corrected with a scattering of points over the body of the image. This is necessary to ensure that the mapping polynomials are well-behaved over the image (Richards and Jia [1999]).
In order to be operational, each transformation algorithm requires a different minimum number of control points with which the process can be carried out (i.e. 3 points for the similarity, 4 for the affine and 7 points for the second order polynomial transformation). The output of the transformation is distributed globally all over the plane surface and the final result depends on the number of control points involved and on their spatial distribution.
Due to the advancements of new computational technologies, old maps georeferencing is becoming easier, providing the possibility of using and exploiting them in different ways. Once a map is georeferenced, it is possible to compare it directly with other georeferenced maps, like other historical maps refering to different historical epochs or present-day maps.
Furthermore, georeferencing allows the visualization of contiguous portions of the same paper map joint together in a format easier to use.

Another advantage of the georeferencing process is the possibility of exploiting new ways of understanding the context of early maps, also integrating them with other spatial information.
On the other hand, it is important to pay attention during the georeferencing process because it can cause distortions and misrepresentations in the maps due to the image warping.

1.3 Online publication

Historical maps can provide valuable information for different disciplines and can be useful for surveyors, geographers, historians and other users interested in researching the background of an area. The historical cadastral base, for example, is a basic element for studying, researching and analysing the evolution of a city.
In recent years many historical archives, libraries and cartographic collections have started to scan at high resolution their imagery and eventually to put it on the Internet. In some cases they have just created a catalogue with the available digitized maps, such as the 'Institut Cartogràfic de Catalunya'[1], focused mainly on Spanish maps, and the 'Archivio Storico Capitolino'[2] (Rome Historical Archive). In some others, they have built up a really Web geo-catalogue service, with maps projected in an actual reference system, like for instance the *David Rumsey Map Collection*[3], with over 22000 maps and images online, focused on rare 18^{th} and 19^{th} centuries maps and other cartographic material. An example can be shown in Figure 1.1.

The National Library of Scotland geo-referenced the Town Plans using ESRI's ArcIMS software to deliver the geo-referenced mapping in a customised interface and OpenLayers open source software to create the mash-up (Henrie [2009][4]). In addition, historical maps have been overlaid upon modern Google and Bing maps, satellite and terrain layers, in order to let the final user better understand the geographical context.
Furthermore, recently they have started also the *Visualising Urban Geographies* project, to provide mapping tools for historians. In fact, it enables them to handle digitized and geo-referenced maps in addition with historical information with the aim of enriching historical understanding and analysis. The main objectives of such a kind of project are: to create a set of geo-referenced historical maps of the 19^{th} century Edinburgh and to make them accessible

[1] http://www.icc.es/
[2] http://www.archiviocapitolino.it/
[3] http://www.davidrumsey.com/
[4] http://maps.nls.uk/

Figure 1.1: Atlas map of Milan in 1832 by the Society for the Diffusion of Useful Knowledge (Great Britain). David Rumsey Historical Map Collection. (Branch, M.C. An atlas of rare city maps, p. 62-63.)

for student learning purposes; to reach a broader public and encourage local organizations and history networks to use their mapping tools; to develop, using open source software, routines and dynamic maps, graphs and diagrams to allow the historical source material to be queried, visualized and analysed with the aim of further understanding of 19^{th} and 20^{th} century cities.
A similar approach has been followed by some Italian Universities in collaboration with State Archives, such as the IUAV University and the State Archive of Venice(Contò et al. [2009]) for the realisation of an information system for the Historical Cadastre of Venice based on the Napoleonic, the Austrian and the Austro-Italian cadastres. Politecnico di Milano too, in collaboration with the Milan State Archive, built up a geo-portal[5] (Oreni et al. [2010]) designed with a double level access to the historical cadastral series ('Catasto Teresiano', 'Lombardo Veneto', 'Catasto Cessato'): besides a catalogue approach level based on research keys, an open geographic level has been implemented with ongoing functionalities.
Based on the fundamental idea that all histories take place in a well defined place and in a precise time and that they can become more meaningful when they interact with other histories, in these years the University of California, Los Angeles (UCLA), with the University of Southern California (USC) and the City University of New York (CUNY) have been developing a digital

[5]http://www.atlantestoricolombardia.it/

research and educational platform[6] for exploring, learning about and interacting with the layered cities histories, called *HyperCities*(Presner [2010]). This platform allows users to go back in time to create, narrate and explore the historical layers of city spaces and tell stories in an interactive, hypermedia environment. Each HyperCity inserted in the system is a real city overlaid with a rich array of geo-temporal information, from historical cartographies and media representations to stories of people and communities who lived and live there. An example can be seen in Figure 1.2:

Figure 1.2: An example of a digitized map of Rome superimposed on the Google imagery.
(http://hypercities.ats.ucla.edu/)

[6]http://hypercities.com/

Chapter 2

Geographical data modelling and 3D city models

The disciplines of cartography and geography, in response to technological innovations, have undergone significant changes during the past half-century (ISO/TC 211 [2009]).
From the early 1980's to the early 1990's, national and international organizations were busy developing standards for geographic data transfer between computers systems. The technical development of such standards was limited to few national and regional user communities, while there were no standards that had broad international support.
Since 1995, the Technical Committee 211 of the International Organization for Standardization (ISO/TC 211)[1] developing international standards for spatial data and the Open Geospatial Consortium (OGC)[2] developing computer interface specifications became highly visible and prominent players on the international geographic agenda. Afterwards, ISO/TC 211 and the OGC formed a joint coordination group to leverage mutual development and minimize technical overlap.
ISO/TC 211 concerns the standardization in the field of digital geographic information and aims at establishing a structured set of standards for information concerning geo-referenceable objects and phenomena.
Many bodies are actively engaged in the work of ISO/TC 211. These include national standardization bodies, the OGC, international professional bodies (such as FIG and ICA), UN agencies, and sectoral bodies (such as DGIWG and ICAO). Being composed mainly by members coming from the public sector, ISO represents the institutional point of view for standardization.

[1] http://www.isotc211.org/
[2] http://www.opengeospatial.org/

The OGC is an international industry consortium of more than 400 (in 2011) companies, government agencies and universities participating in a consensus process to develop publicly available interface standards. OGC standards support interoperable solutions that 'geo-enable' the Web, wireless and location-based services. The standards empower technology developers to make complex spatial information and services accessible and useful with all kinds of applications.
The OGC mission is to serve as a global forum for the collaboration between developers and users of spatial data products and services, and to advance international standards for geospatial interoperability development. The OGC is submitting its specifications for ISO standardization via ISO/TC 211. Such cooperation should lead to more market-relevant spatial standards, and could serve as a useful roadmap for all interested parties.
In recent years, the ISO/TC211 have developed a group of international standards called '19100 series' that support data management, acquisition, processing, analysis, access, presentation and transfer between different users, systems and locations for geographic information.
Standards that specify the infrastructure for geospatial data are the following:

- ISO 19101 Geographic information — Reference model
- ISO/TS 19103 Geographic information — Conceptual schema language
- ISO/TS 19104 Geographic information — Terminology
- ISO 19105 Geographic information — Conformance and testing
- ISO 19106 Geographic information — Profiles

In particular, the '19101:2002 Geographic Information – Reference Model' standard is a guide for structuring geographic information in a way that enables the universal usage of digital geographic information. This reference model describes the overall requirements for standardisation and the fundamental principles applied in developing and using standards for geographic information.
The focus of this standards family is to define the basic semantics and structure of geographic information for data management and data interchange purposes and specify geographic information service components and their behaviour for data processing purposes.
The two major components of the reference model are the *Domain Reference Model*, which provides a high-level representation and description of the structure and content of geographic information, and the *Architecture Reference Model*, which describes the general types of services that will be provided by

computer systems to manipulate geographic information and enumerates the service interfaces across which those services must interoperate.
In the following section, major data transfer standards are presented, both *de facto* and OGC - ISO standards.
In particular, Section 2.2.2 describes a particular standard developed with the aim of representing cities, CityGML.
Our approach, that does not make use of this standard, will be presented in Section 5.2.1. The choice of adopting the shapefile instead of CityGML is due to the fact that CityGML is not yet supported by the chosen virtual globe technology and the shapefile *de facto* standard is suitable for this first implementation.

2.1 Data transfer key standards

2.1.1 ESRI shapefile *de facto* standard

Currently, the main *de facto* standard with respect to spatial data is the shapefile format (Batty et al. [2010]). The shapefile standard has been created by ESRI[3] for the use in its GIS products. ESRI's flagship product, ArcGIS, provides an integrated approach to serve organization's geospatial data and uses shapefiles to interoperate with external data. Because of shapefile wide use, other GIS systems makers support the shapefile standard. As written in the shapefile technical description, *"a shapefile stores non-topological geometry and attribute information for the spatial features in a data set. The geometry for a feature is stored as a shape comprising a set of vector coordinates"* (ESRI [1998]).
An attempt was made to bring the shapefile specification into the OGC standardization process, but while the data specification has been published, ESRI retains control of future development and changes to specification.
Shapefiles support basic geometric elements: point, line, and area (represented as closed loop). These geometric primitives are augmented by attributes that are stored in a table of records with one-to-one shape-to-attribute mappings (Zborovskiy [2006]). This table is stored in dBASE format files.

2.1.2 Geography Markup Language standard

In year 2000, OGC released the Geography Markup Language (GML)[4]. GML is an XML grammar, written in XML Schema, developed with the aim of

[3] http://www.esri.com/
[4] http://www.opengeospatial.org/standards/gml

improving geographic information transportation and storage. GML key concepts to model the world are drawn from the ISO 19100 series of International Standards and the OpenGIS Abstract Specifications. According to ISO 19101, 'a feature is an abstraction of real world phenomena' and it becomes 'geographic' if it is associated with a location on the Earth surface.
A feature state is defined by a set of properties, according to its type definition, where each property consists in a [name, type, value] triple. In case of geographic features, their properties may be geometry-valued. A feature collection may itself be regarded as a feature, and as a consequence it has a feature type and thus may have distinct properties of its own, in addition to the features it contains. The feature types of an application or application domain is usually captured in an application schema (ISO 19109).
Following ISO 19118, GML specifies XML encodings of several of the conceptual classes defined in the ISO 19100 series of International Standards and the OpenGIS Abstract Specification (OGC [2007]). These conceptual models include those defined in:

- ISO/TS 19103 — Conceptual schema language (units of measure, basic types);
- ISO 19107 — Spatial schema (geometry and topology objects);
- ISO 19108 — Temporal schema (temporal geometry and topology objects, temporal reference systems);
- ISO 19109 — Rules for application schemas (features);
- ISO 19111 — Spatial referencing by coordinates (coordinate reference systems);
- ISO 19123 — Schema for coverage geometry and functions.

The aim is to provide a standardized encoding (i.e. a standardized implementation in XML) of types specified in the conceptual models of the International Standards listed above.
GML standard defines the XML schema syntax, mechanisms and conventions that:

- provide an open, vendor-neutral framework for the description of geospatial application schemas;
- allow profiles that support proper subsets of GML framework descriptive capabilities;

- support geospatial application schemas description for specialized domains and information communities;
- enable the creation and maintenance of linked geographic application schemas and datasets;
- support the storage and transport of application schemas and datasets;
- increase the ability of organizations to share geographic application schemas and the information they describe.

In addition, GML provides XML encodings for concepts not modelled yet in the ISO 19100, like dynamic features, simple observations or value objects. Predefined types of geographic feature in GML include coverages and simple observations. A coverage is a subtype of feature with a spatio-temporal domain and a value set range of homogeneous 1- to n-dimensional tuples. It may represent one feature or a collection of features to model and make visible spatial relationships of Earth phenomena. An observation models the act of recognizing and noting a fact or occurrence often involving measurement with instruments. It is considered to be a GML feature with a time at which the observation took place, and with a value for the observation.
In GML spatial and temporal geometries are defined. Spatial geometries are the values of spatial feature properties and they indicate the coordinate reference system in which their measurements have been made. The parent geometry element of a geometric complex or geometric aggregate makes this indication for its constituent geometries. Temporal geometries, on the other side, are the values of temporal feature properties. Like their spatial counterparts, temporal geometries indicate the temporal reference system in which their measurements have been made. Spatial or temporal topologies are used to express the different topological relationships between features.

2.1.3 Keyhole Markup Language standard

The Keyhole Markup Language (KML)[5], made popular by Google, complements GML. KML is used to encode and transport representations of geographic data for display in an Earth browser, such as a 3D virtual globe, 2D Web browser application, or 2D mobile application (OGC [2008]).
KML uses a tag-based structure with nested elements and attributes and it is based on the XML standard.
Google submitted KML to the OGC to be evolved within the OGC consensus process and KML Version 2.2 has been adopted as an OGC implementation

[5]http://www.opengeospatial.org/standards/kml/

standard. The OGC and Google agreed that there could be additional harmonization of KML with GML (e.g. to use the same geometry representation) in the future.
Whereas GML is a language to encode geographic content for any application, by describing application objects and their properties, KML is a language for the visualization of geographic information, including annotation of maps and images. KML can be used to carry GML content, and GML can be 'styled' to KML for the purposes of presentation. Currently, KML 2.2 utilizes geometry elements derived from GML 2.1.2, like point, line string, linear ring, and polygon.
KML can be used to:

- specify icons and labels to identify locations on the surface of the planet;
- create different camera positions to define unique views for KML features;
- define image overlays to attach to the ground;
- define styles to specify KML feature appearance;
- write HTML descriptions of KML features, including hyperlinks and embedded images;
- organize KML features into hierarchies;
- locate and update retrieved KML documents from local or remote network locations;
- define the location and orientation of textured 3D objects.

KML documents and their related images (if any) may be compressed using the ZIP format into KMZ archives.

2.2 3D city models

Currently, most of Web geographic data viewers are just two-dimensional (planar). Adopting the 3D visualization would give several advantages, like a more complete and realistic representation of geographic data with a greater expressive power and a more effective reading and analysis. For these reasons, nowadays it has become quite clear that 2D maps cannot precisely represent multi-dimensional and dynamic spatial phenomena like it can be done adopting different tools, such as photos, videos, 3D models and related textual

information.
An increasing number of applications, like environmental simulations, urban planning and disaster management, require additional information about city objects given in a standardised representation. This leads to the need of developing specific standards for city objects definition and schematization. The most important standard developed to satisfy this purpose is CityGML, described in Section 2.2.2.

2.2.1 X3D

X3D[6] is a XML-based ISO standard for representing 3D computer graphics. This standard is the successor of the Virtual Reality Modelling Language (VRML). It improves VRML with new features, advanced APIs, additional data encoding formats, stricter conformance and an architecture that allows to support the standard and the backward compatibility with VRML.
X3D also benefits from other open source standards like XML, DOM and XPath.
In this standard, several profiles (sets of components) are defined for various levels of capability including X3D Core, X3D Interchange, X3D Interactive, X3D CADInterchange, X3D Immersive, and X3D Full. Browser makers can define their own component extensions prior to submitting them for standardisation by the Web3D Consortium. Formal review and approval is then performed by the International Organization for Standardization (ISO).
Liaison and cooperation agreements are also in place between the Web3D Consortium and the World Wide Web Consortium (W3C), Open Geospatial Consortium (OGC), Digital Imaging and Communications in Medicine (DICOM) and the Khronos Group. The abstract specification for X3D (ISO/IEC 19775) was first approved by the ISO in 2004. X3D specifications, encodings and language bindings, defined by ISO/IEC, are the following:

- ISO/IEC 19775-1.2:2008 — X3D Architecture and base components;
- ISO/IEC 19775-2:2004 — X3D Scene access interface;
- ISO/IEC FDIS 19775-2.2:200x — X3D Scene access interface;
- ISO/IEC 19776:2005 — X3D encodings (XML, Classic VRML and Compressed Binary);
- ISO/IEC 19776-1.2 — X3D encodings — Part 1: XML encoding;

[6]http://www.web3d.org/

- ISO/IEC 19776-2.2 — X3D encodings — Part 2: Classic VRML encoding;
- ISO/IEC 19776-3 — X3D encodings — Part 3: Compressed binary encoding;
- ISO/IEC FCD 19776-3.2 — X3D encodings — Part 3: Binary encoding;
- ISO/IEC 19777:2005 — X3D language bindings (ECMAScript);
- ISO/IEC 19777:2005 — X3D language bindings (Java);
- ISO/IEC 19777:2005 — X3D language bindings (ECMAScript and Java).

In different projects X3D has been implemented for virtual cultural heritage (Guarnieri et al. [2010]; Zöllner et al. [2009]) and for 3D maps and models in mobile GIS (Nurminen [2007]; Mulloni et al. [2007]).

2.2.2 CityGML

A particular XML-based format has been developed for virtual 3D city model representation and exchange, called CityGML[7]. Based on the Geography Markup Language (GML), it has become an international standard issued by the Open Geospatial Consortium (OGC). The aim of CityGML is to reach a common definition and understanding of the basic entities, attributes and relationship within a 3D city model (Kolbe [2009]). Besides representing the shape and the graphical appearance of city models, CityGML specifically addresses the object semantics and the representation of the thematic properties, taxonomies and aggregations. In semantic models, objects are decomposed into parts using logical criteria, following given structures or those observed in the real world. The city modelling requires an appropriate qualification of 3D data, that can be done by an automated process or by manual interpretation. This last option would increase the efforts needed to create and maintain a 3D city models, so it makes sense if data can be used by different customers within multiple applications.
CityGML represents four different aspects of virtual city models: semantics, geometry, topology and appearance. All objects can be represented in five different Levels of Detail (LoD), where objects become more and more detailed with increasing LoD regarding both geometry and thematic differentiation (see Figure 2.1):

[7] http://www.citygml.org/

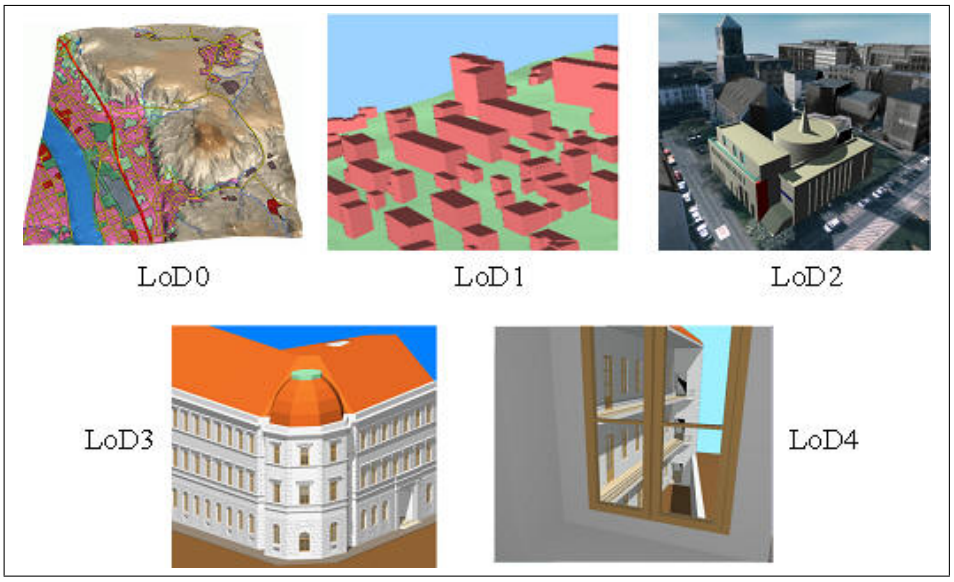


Figure 2.1: The five levels of detail (LoD) defined by CityGML. (Kolbe [2009])

- **LoD0** Regional model: the coarsest level. Essentially it is a two and half dimensional digital terrain model;
- **LoD1** City model: block model, without any roof structure;
- **LoD2** City/Site model: roof structure and larger building installations, like balconies and stairs, optional structures;
- **LoD3** Site model: architectural models with detailed wall and roof structures, doors, windows, etc.;
- **LoD4** Interior model: completes the previous model by adding interior structures, like rooms, stairs and furniture.

Often those different LoDs come from independent data collection processes and facilitate efficient visualization and analysis: in the same dataset, an object can be represented in different LoDs simultaneously, enabling the visualization with different degrees of resolution. Concerning the semantic aspects, CityGML employs the ISO 19100 standards family framework for geographic features modelling. According to these specifications, geographic features are abstractions of real world objects, modelled by classes formally specified using UML notation. CityGML provides class definitions, normative regulations and explanations of the semantics for the most important geographic feature,

including buildings, DTMs, water bodies, vegetation and city furniture.
For the geometry definition, CityGML uses the GML3 geometry model, which is an implementation of the ISO 19107 standard. Geographic features geometries are then represented as objects having an identity and further geometric substructures. GML3 provides classes for 0D to 3D geometric primitives, 1D-3D composite geometries and 0D-3D geometry aggregates. Composite geometries like *CompositeSurface* must be topologically connected and isomorphic to a primitive of the same dimension (e.g. *Surface*); aggregate geometries like *MultiSurface* or *MultiSolid* do not impose topological constraints and thus their components can also permeate each other or be disjoint. In order to ensure a broad system support, CityGML is restricted to non-curved geometries, as these often cannot be handled by GIS or spatial database managements systems, therefore curved surfaces of real world objects must be approximates by faceted surfaces of planar patches.
For many applications topological correctness of the object geometries is crucial. For example, the surfaces delimiting a building must be closed in order to be able to compute its volume. The ISO 19107 and GML3 topology model follows the line of full decomposition of n-dimensional topological primitives into (n-1)-dimensional primitives, which again are decomposed down to the level of nodes (0D). Furthermore, the topological representation in GML would require appropriate topological properties in the CityGML application schema. This means that besides geometry properties like *lod1Solid* to *lod4Solid* of class *Building* also corresponding topology properties, like *lod1TopoSolid* to *lod4TopoSolid*, would have to be added to the data model.
Information about a surface appearance is considered an integral part of virtual 3D city models in addition to semantic and spatial properties. *Appearances* capture characteristics of surfaces as they appear to specific sensors, like RGB or infrared (IR) cameras; consequently they can serve as input for both visualization and analysis tasks. Each surface is allowed to have multiple appearances assigned, one for each theme. This can be useful if one wants to visualize a 3D scene differently and the switch among them in a client viewer would mean a replacement of the surface materials and textures. By using themes, also multi-texturing can be supported by CityGML. The ISO 19109 standard for geographic features modelling implies a dual structure: thematic object classes on one side (the feature classes) and spatial properties (using the geometry and topology classes of ISO 19107) on the other side.
CityGML is complementary to 3D computer graphic standards, like X3D, VRML and COLLADA, and geovisualization standards, like KML. Although in principle it is possible to exchange semantic information using X3D or KML, both specifications do not standardise the way to represent complex geographic features and their interrelationships.

CityGML is not optimized with respect to efficient visualization, but can be considered a rich source format from which X3D or KML can be easily derived. The semantic information given by the explicit association of CityGML objects to thematic classes and thematic attributes can be exploited to filter objects and to create 3D graphical shapes, appearance properties and materials.

Chapter 3

Open source software, open data and standards

Map mash-ups and Web-based mapping are possible thanks to the existence of standard formats.
Standards support interoperable solutions and empower technology developers to make complex spatial information and services, accessible and useful with all kinds of applications.
The most used open standards are listed in Section 3.1.
Concerning GIS software, over the last few years the world of free and open source software (FOSS) experienced a considerable development. The birth of specific organization, like the OSGeo Foundation, led to the strengthening of user communities of FOSS, offering a point of contact among them.
At the moment, there are more than 350 GIS packages listed in the FreeGIS.org Website[1].
A more detailed discussion about free and open source software and data will be held in Section 3.2.

3.1 ISO/TC 211 and OGC formats

The main ISO/TC 211 and OGC service standards include:

- **Web Map Service (WMS)**[2]: it provides a simple HTTP interface for requesting geo-registered map images from one or more distributed geospatial databases. A WMS request defines the geographic layer(s) and the area of interest to be processed. The request response is one or

[1] http://freegis.org/
[2] http://www.opengeospatial.org/standards/wms

more geo-registered map images (returned as JPEG, PNG, etc) that can be displayed in a browser application. The interface also supports the ability to specify whether the returned images should be transparent, in such a way that layers from multiple servers can be combined.
The OGC's WMS Implementation Specification is available also as ISO 19128, a standard titled "IS 19128:2005 Geographic information - Web map server interface.".

- **Web Feature Service (WFS)**[3]: it represents a change in the way geographic information is created, modified and exchanged over the Internet. Rather than sharing geographic information at the file level using File Transfer Protocol (FTP), for example, the WFS offers direct fine-grained access to geographic information at the feature and the feature property level. Web feature services allow clients only to retrieve or modify the data they are seeking, rather than retrieving a file that contains possibly much more.
The OGC's WFS standard is now also an ISO/TC 211 standard (ISO 19142:2010).

- **Web Coverage Service (WCS)**[4]: defines a standard interface and operations that enable 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.

- **Styled Layer Descriptor (SLD)**[5]: defines an encoding that extends the WMS standard to allow user-defined symbolization and colouring of geographic feature and coverage data. SLD addresses the need for users and software to be able to control the visual portrayal of the geospatial data. The ability of defining styling rules requires a styling language understandable by both the client and the server.

- **Web Map Tiling Service (WMTS)**[6]: provides a standard based solution to serve digital maps using predefined image tiles. The service advertises the available tiles through a standardized declaration in the *ServiceMetadata* document, that is common to all OGC Web services.

- **Web Map Context (WMC)**[7]: Web Map Context Documents are

[3]http://www.opengeospatial.org/standards/wfs
[4]http://www.opengeospatial.org/standards/wcs
[5]http://www.opengeospatial.org/standards/sld
[6]http://www.opengeospatial.org/standards/wmts
[7]http://www.opengeospatial.org/standards/wmc

XML documents that contain all information needed to display a set of maps for a selected area or for a given size. These can come from one or more WMS and display the map composition within a given area of interest. Web Map Context Documents can be compared to 'projects' or 'workspaces' in conventional GIS desktop applications, but they are standardized and thus compatible between different software packages.

- **Filter Encoding Specification (FES)**[8]: this jointly developed OGC and ISO TC/211 standard describes an XML and Key Value Pair (KVP) encoding of a system neutral syntax for expressing projections, selection and sorting clauses collectively called query expression.

- **Sensor Observation Service (SOS)**[9]: provides an API for managing deployed sensors and retrieving sensor data (specifically "observation" data). Whether from in-situ sensors (e.g., water monitoring) or dynamic sensors (e.g., satellite imaging), measurements made from sensor systems represent in volume most of the geospatial data used in geospatial systems today.

- **Observations and Measurements (OM)**[10]: specifies an XML implementation for the OGC and ISO Observations and Measurements conceptual model (OGC Observations and Measurements v2.0 also published as ISO/DIS 19156), including a schema for Sampling Features. This encoding is an essential dependency for the OGC Sensor Observation Service (SOS) Interface Standard. More specifically, this standard defines XML schemas for observations and features involved in sampling when making observations.

- **SWE Common Data Model (SWE)**[11]: defines low level data models for exchanging sensor related data between nodes of the OGC® Sensor Web Enablement (SWE) framework. These models allow applications and/or servers to structure, encode and transmit sensor datasets in a self describing and semantically enabled way.

- **OGC Web Services Common (OWS)**[12]: specifies many of the aspects that are, or should be, common to all or multiple OGC Web Service (OWS) interface Implementation Standards. These common

[8] http://www.opengeospatial.org/standards/filter
[9] http://www.opengeospatial.org/standards/sos
[10] http://www.opengeospatial.org/standards/om
[11] http://www.opengeospatial.org/standards/swecommon
[12] http://www.opengeospatial.org/standards/common

aspects are primarily some of the parameters and data structures used in operation requests and responses.

In the next paragraph the main service that has been used in this work, the WMS, will be described in detail.

3.1.0.0.1 Web Map Service A Web Map Service (WMS) produces maps of spatially referenced data dynamically from geographic information (see OGC [2006]). This international standard defines a 'map' to be a portrayal of geographic information as a digital image file suitable to be displayed on a computer screen. A map is not the data itself: WMS-produced maps, in fact, are generally rendered in a pictorial format such as PNG, GIF or JPEG, or occasionally as vector-based graphical elements in Scalable Vector Graphics (SVG) or Web Computer Graphics Metafile (WebCGM) formats. On a WMS three operations can be defined: the first returns service-level metadata, the second returns a map with well defined geographic and dimensional parameters and the third, that is optional, returns information about particular features shown on a map.
Web Map Service operations can be invoked using a standard Web browser by submitting requests in the form of Uniform Resource Locators (URLs). The content of such URLs depends on which operation is requested. In particular, when requesting a map the URL indicates what information is to be shown on the map, what portion of the Earth is to be mapped, the desired coordinate reference system and the output image width and height.
HTTP can support two request methods: GET and POST. Both of these methods may be offered by a server, but the use of the online resource URL differs: in case of WMS, the support for the GET method is mandatory, while for the POST method it is optional.
When two or more maps are produced with the same geographic parameters and output size, the results can be accurately overlaid to produce a composite map. The use of image formats that support transparent backgrounds (e.g. GIF or PNG) allows underlying maps to be visible.
Furthermore, individual maps can be requested from different servers. The Web Map Service thus enables the creation of a network of distributed map servers from which clients can build customized maps.
A basic WMS classifies its geographic information holdings into 'Layers' and offers a finite number of predefined 'Styles' in which those layers are displayed. There are two kinds of WMS: the basic WMS and the queryable one. A **basic WMS** supports the basic service elements, the *GetCapabilities* operation and the *GetMap* operation:

- *GetCapabilities*: to obtain service metadata, that is a machine-readable

(and human-readable) description of the server information content and acceptable request parameter values (see Table 3.1). When invoked on a WMS, the response to a GetCapabilities request consists in an XML document containing service metadata formatted according to the XML Schema. The schema specifies the mandatory and optional content of the service metadata and how the content is formatted. The XML document shall contain a root element named WMS_Capabilities in the "http://www.opengis.net/wms" namespace.

Request parameter	Mandatory/optional	Description
VERSION=version	O	Request version
SERVICE=WMS	M	Service type
REQUEST=GetCapabilities	M	Request name
FORMAT=MIME_type	O	Output format of service metadata
UPDATESEQUENCE=string	O	Sequence number or string for cache control

Table 3.1: Parameters of a GetCapabilities request URL (OGC [2006])

- *GetMap*: returns a map. Upon receiving a GetMap request, a WMS either satisfies the request or issues a service exception. The possible parameters to use in the request are listed in Table 3.2. The response to a valid GetMap request is a map of the spatially referenced information layer requested, in the desired style, and with the user specified coordinate reference system, bounding box, size, format and transparency. An invalid GetMap request yields an error output in the requested exceptions format (or a network protocol error response in extreme cases).

A **queryable WMS** satisfies all the requirements for a basic WMS and supports also the *GetFeatureInfo* operation.
The GetFeatureInfo optional operation is only supported for those Layers for which the attribute queryable "1" (true) has been defined or inherited. A client shall not issue a GetFeatureInfo request for other layers. A WMS responds with a properly formatted service exception (XML) response (code = OperationNotSupported) if it receives a GetFeatureInfo request but does not support it. The GetFeatureInfo operation is designed to provide clients of a WMS with more information about features in the pictures of maps that were returned by previous Map requests. The canonical use case for GetFeatureInfo is that a user sees the response of a Map request and chooses

Request parameter	Mand./opt.	Description
VERSION=version	M	Request version
REQUEST=GetMap	M	Request name
LAYERS=layer_list	M	Comma-separated list of one or more map layers
STYLES=style_list	M	Comma-separated list of one rendering style per requested layer
CRS=namespace:identifier	M	Coordinate reference system
BBOX=minx,miny,maxx,maxy	M	Bounding box corners (lower left, upper right) in CRS units
WIDTH=output_width	M	Width in pixels of map picture
HEIGHT=output_height	M	Height in pixels of map picture
FORMAT=output_format	M	Output format of map
TRANSPARENT=TRUE—FALSE	O	Background transparency of map (default=FALSE)
BGCOLOR=color_value	O	Hexadecimal RGB colour value for the background color (default=0xFFFFFF)
EXCEPTIONS=exception_format	O	The format in which exceptions are to be reported by the WMS (default=XML)
TIME=time	O	Time value of layer desired
ELEVATION=elevation	O	Elevation of layer desired
Other sample dimension(s)	O	Value of other dimensions as appropriate

Table 3.2: Parameters of a GetMap request URL (OGC [2006])

a point (I,J) on that map for which he/she wants to obtain more information. The basic operation provides the ability for a client to specify which pixel is being asked about, which layer(s) should be investigated, and what format the information should be returned in. Because the WMS protocol is stateless, the GetFeatureInfo request indicates to the WMS what map the user is viewing by including most of the original GetMap request parameters (all but

VERSION and REQUEST). From the spatial context information (BBOX, CRS, WIDTH, HEIGHT) in that GetMap request, along with the I,J position the user chose, the WMS can (possibly) return additional information about that position. The actual semantics of how a WMS decides what to return more information about, or what exactly to return, are left up to the WMS provider.

3.1.0.0.1.1 Handling multi-dimensional data Some geographic information may be available at multiple times (for example, an hourly weather map) or at multiple elevations (for example, ozone concentrations at different heights in the atmosphere).
Regarding time, a WMS may announce available times in its service metadata, and the GetMap operation includes a parameter for requesting a particular time. The format of a time string is specified according to the ISO 8601 directive.
The basic time format uses ISO 8601:2000 'extended' format: up to 14 digits specifying century, year, month, day, hour, minute, and seconds, and optionally a decimal point followed by zero or more digits for fractional seconds, with non-numeric characters to separate each piece: ccyy-mm-ddThh:mm:ss.sssZ. The precision may be reduced by omitting least-significant digits, as in the examples below. ISO 8601:2000 prefers a decimal comma before fractional seconds but allows a decimal period as in the WMS standard. The century digits are included with the year; two-digit years should not be used. A time zone suffix is mandatory if the hours field appears in the time string. All times should be expressed in Coordinated Universal Time (UTC), indicated by the suffix Z (for "zulu"). When a local time is applied, a numeric time zone suffix as defined by ISO 8601:2004 has to be used. The absence of any suffix at all means local time in an undefined zone, which should not be used in the global network of map servers. Examples can be:

- ccyy: Year only
- ccyy-mm: Year and month
- ccyy-mm-dd: Year, month and day
- ccyy-mm-ddThhZ: Year, month, day and hour in UTC

As specified in ISO 8601, the year immediately preceding 0001 shall be denoted 0000. Years before 0000 shall be denoted by a leading minus sign (hyphen character). To denote years in the distant past, the year field shall be expanded beyond 4 digits and preceded by a minus sign.

An ISO 8601 Period is used to indicate the time resolution of the available data. The ISO 8601 format for representing a period of time is used to represent the resolution: designator P (for Period) followed by number of years Y, months M, days D; time designator T followed by number of hours H, minutes M, seconds S. For example:

- P1Y — 1 year
- P1M10D — 1 month plus 10 days
- PT2H — 2 hours
- PT1.5S — 1.5 seconds

Unneeded elements may be omitted.
When providing temporal information, a server should declare a default value in service metadata. If the client request does not include a value, a server should respond with the default value, if defined.
A WMS may announce available elevations in its service metadata too, and the GetMap operation includes an optional parameter for requesting a particular elevation. A single elevation or depth value is a number whose units, and the direction in which ordinates increment, are declared through a one-dimensional vertical CRS. Depending on the context, elevation values may appear as a single value, a list of values or an interval (see Table 3.3). A server may declare at most one vertical CRS for each layer. In the WMS standard the horizontal and vertical CRSs are treated as independent metadata elements and request parameters. A request for a map at a specific elevation includes an elevation value but does not include the vertical CRS identifier (the horizontal CRS, which is included with the horizontal bounding box in the request parameters).
When providing elevation information, a server should declare a default value in service metadata. If the client request does not include a value, a server should respond with the default value, if one has been declared.
Two types of Vertical CRS identifiers are permitted, the *label* and *URL* identifiers:

- Label: the identifier includes a namespace prefix, a colon and a numeric or string code. If the namespace prefix is "EPSG", then the vertical CRS is one of those defined in the European Petroleum Survey Group (EPSG)[13] database.

[13]http://www.epsg.org

- URL: the identifier is a fully-qualified Uniform Resource Locator that references a publicly-accessible file containing a definition of the CRS that is compliant with ISO 19111.

If the height is the vertical component of a 3-dimensional CRS, the Vertical CRS identifier has to be that of the 3-dimensional CRS.

Syntax	Description
value	Single value
value1,value2,value3,...	List of multiple values
min/max/resolution	Interval defined by its lower and upper bounds and its resolution
min1/max1/res1/,min2/max2/res2,...	List of multiple intervals

Table 3.3: Syntax for listing one or more extent values (OGC [2006])

If an object has an Elevation dimension defined, then operation requests to retrieve that object can include the parameter "ELEVATION=value". If a layer has a Time dimension defined, then requests can include the parameter "TIME=value". In either case, the value has to be written in the format described in Table 3.3 to provide a single value, a comma-separated list, or an interval of the form start/end, with or without a resolution. Values shall not contain white spaces. An interval in a request value is a request for all the data from the start value up to (and including) the end value.
The absence of Time or Elevation parameters in the request is equivalent to a request for the Layer default value (if defined) along that dimension. All parameter names are case-insensitive, so, for example, "TIME", "Time" and "time" are all equivalent.
For the TIME parameter, special keywords can be used:

- "current" may be used if the <Dimension name="time"> service metadata element includes a nonzero value for the "current" attribute;
- "TIME=current" means "send the most current data available";
- "TIME=start_time/current" means "send data from start_time up to the most current data available".

When sending a request with the "TIME" parameter, different types of time values can be specified, as:

- *single value*: e.g. ...&TIME=2011-10-12&...
- *multiple value*: e.g. ...&TIME=2011-10-12,2011-10-13&...
- *single range value*: e.g. ...&TIME=2011-10-12/2011-10-15&...
- *multiple range value*: e.g. ...&TIME=2011-10-12/2011-10-15,2011-10-18/2011-10-21&...

When the server receives a request with a "TIME" parameter, it transforms the time request into valid expressions that are assigned to the filter parameter on layers that are time-aware. For example:

- *single value*: e.g. ('[TIME_FIELD]' = '2011-10-12')
- *multiple value*: e.g. ('[TIME_FIELD]' = '2011-10-12' OR '[TIME_FIELD]' = '2011-10-13')
- *single range value*: e.g. (('[TIME_FIELD]' $\geqslant$ '2011-10-12') AND ('[TIME_FIELD]' $\leqslant$ '2011-10-15'))
- *multiple range value*: e.g. (('[TIME_FIELD]' $\geqslant$ '2011-10-12' AND '[TIME_FIELD]' $\leqslant$ '2011-10-15') OR ('[TIME_FIELD]' $\geqslant$ '2011-10-18' AND '[TIME_FIELD]' $\leqslant$ '2011-10-21'))

The WMS GetMap request may include a comma-separated list of one or more layers. In such a request, the TIME, ELEVATION (and eventually DIM_) parameters apply individually to all layers as follows:

- if the spatially referenced object has the corresponding dimension as a property, the dimension value applies as that object had been requested alone;
- if the spatially referenced object does not have that dimension, then the dimension value is ignored.

3.2 Open source software and open data

Often the results of research and development at the universities and government laboratories have been made available in the form of Public Domain software packages. This led to the idea of Open Source software.
Richard M. Stallman (Stallman [1999]) first defined the concept of *Free Software*, according to what a particular user can do with the software:

- to run the program, for any purpose;
- to modify the program to suit user needs (having access to the source code);
- to redistribute copies;
- to distribute modified versions of the program, so that the community can benefit from user improvements.

In 1984, Stallman started to work on the GNU project, and one year later the *Free Software Foundation* has been created to support the free software concept.
The GNU General Public license[14] not only grants the four freedoms described above, but it also protects them, so that nowadays the GPL is the most widely used license for free software.
The basic idea is based on the assumption that by allowing the programmers to read, redistribute and modify the source code, the software evolves (Neteler and Mitasova [2004]). Full access to the source code is particularly important in the GIS field because the underlying algorithms can be complex and can greatly influence the results of spatial analysis and modelling. The different backgrounds and expertise of developers contribute to synergethic effects leading to a faster and more cost effective software development of stable and robust products.
Over the past few years several open source GIS and GPS projects have been established with different goals. Most of them are listed at the Free GIS portal Web site[15]. Smaller projects are usually based on individual developer's initiative, when the lack of available software for a specific application is solved by its own development and the result is then made available to the public on the Internet. Depending on the level of required expertise, other programmers may join the project and contribute to develop, improve and extend its tools.

3.2.1 Open source geospatial software

Open source geospatial software tools offer new opportunities for developers to create new mash-up applications more quickly and at lower cost (Batty et al. [2010]). The developers community creates an ecosystem for rapid production of robust software applications that may even have greater reliability than some proprietary software solutions.

[14] http://www.gnu.org/licenses/gpl.html
[15] http://freegis.org/

Governments realized the benefits of open source too and are actively promoting this. An example is the UK Government Action Plan on Open Source, Open Standards and Re-Use[16].
The Open Source Geospatial Foundation[17] (OSGeo) is a not-for profit organization born in 2006 whose mission is to support and promote the collaborative development of open geospatial technologies and data. The foundation provides financial, organizational and legal support to the broader open source geospatial community. It also serves as an independent legal entity to which community members can contribute code, funding and other resources, secure in the knowledge that their contributions will be maintained for public benefit. OSGeo also serves as an outreach and advocacy organization for the open source geospatial community, which provides a common forum and shared infrastructures for improving cross-project collaboration.
From the beginning, the main benchmarks were interoperability and the choice of Open Source Initiative (OSI) certified licenses allowing different technologies not only to work together but also to integrate and to exchanges pieces of source code among the community members.

3.2.2 Open geospatial data

Increasingly, discussions over what constitutes openness moved beyond the parameters of open source software and into the meaning of openness in the context of a Web-based service, like Google (O'Reilly [2006]). For a service, in fact, is the data, rather than the software, that needs to be open. Tim Bray, an inventor of XML, argues that any online service can call itself "Open" if it makes, and lives up to, this commitment: *"Any data that you give us, we'll let you take away again, without withholding anything, or encoding it in a proprietary format, or claiming any intellectual-property rights whatsoever"* (Bray [2006]).
An example of widely used free data is OpenStreetMap (OSM)[18]. The OpenStreetMap project is a knowledge collective that provides user-generated street maps. OSM aim is to create a set of map data that's free to use, editable, and licensed under new copyright schemes. A considerable number of contributors edits the world map collaboratively using the OSM technical infrastructure, and a core group, estimated at approximately 40 volunteers, dedicates its time to creating and improving OSM's infrastructure, including maintaining the server, writing the core software that handles the transactions with the server, and creating cartographical outputs. There's also a growing

[16] http://www.netvibes.com/cabinetoffice#Open_Source
[17] http://www.osgeo.org/
[18] http://www.openstreetmap.org/

software developers community who develops software tools to make OSM data available for further use across different application domains, software platforms and hardware devices (Haklay and Weber [2008]). Recently, the generation of interactive 3D City Models[19], based on free geo-data available from the OSM project and public domain height information provided by the Shuttle Radar Topography Mission (SRTM), has been investigated (Over et al. [2010]).

[19]http://www.osm-3d.org

Chapter 4

Technologies adopted

In this chapter the technologies adopted to realize the project are presented. To build up a geo-portal several GIS software packages are needed. First of all, a *map server* to fulfil spatial queries, conduct spatial analysis, generate and deliver maps to the client on the user's requests (see Section 4.1).
While the server provides shared resources, a client is the front-end component. The client, in fact, usually has the following functions (Peng and Tsou [2003]):

- to present an interface for user interaction;
- to format requests for services or data from the server;
- to display data or query results received from the server.

In practice, when the user generates a data request, the client side accepts query instructions, formats them in the proper server language and forwards the request to the server. The request is received and processed by the server, that locates the appropriate information and sends it back to the client. The client then shows the information to the user through the interface.
The technologies adopted for the client side are described in Section 4.2.
Since the aim of the project is to realize a geo-portal that includes also a 3D city model, a brief description of projects related to this topic are presented (see 4.2.2). The focus is then put on the chosen virtual globe: NASA World Wind.
The architecture of the developed system is shown in Figure 4.1. At the server side, data are stored in the file system as vector (.shp) or raster (.tif). MapServer can manage files stored in the local file system and layers coming from other Web services. Through Apache, maps are then rendered in the HTML page. This page contains both a World Wind Java Applet and a GeoExt frame, that uses OpenLayers and Ext JS to manage maps.

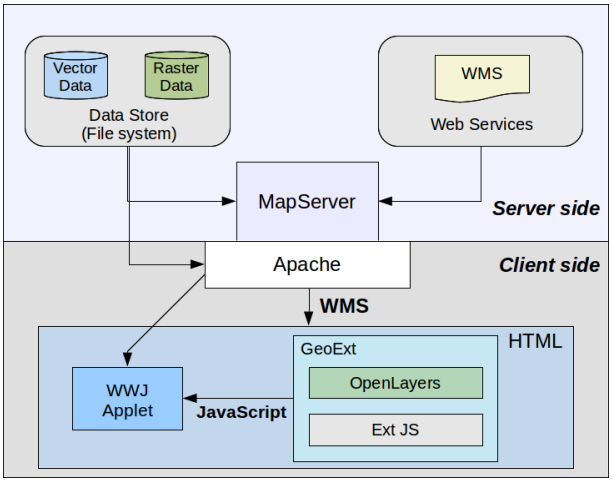


Figure 4.1: The architecture of the system

Apache is used by World Wind to access directly to the buildings shapefile and to read the height information directly from the attribute table (see Section 5.2.1), in order to build up a 3D city model on the virtual globe.

4.1 Geospatial server

A map server provides specific traditional GIS functions or services, that in principle can be located in different servers as individual components.
A map server generally has seven basic functions (Peng and Tsou [2003]):

1. to produce a symbolized map according to user's requests;
2. to answer basic queries about map content and spatial features attributes;
3. to extract data from a database based on the user's request criteria;
4. to perform spatial analysis (i.e. buffering, spatial search, feature overlay, network analysis);
5. to communicate with other programs about services and data availability, such as deliverable and queryable maps;
6. to provide an interface for associating default symbology with a dataset for use by clients with data publishing capability;
7. to invoke other map servers to perform other analysis.

The output of a map server can be in three forms: filtered feature data that are sent to the client for user manipulating (pan, zoom, query); a simple map image (i.e. GIF or JPEG); a graphic element map that is composed of discernible map elements with predefined colors, styles, legends, and so on.
The free and open source communities make available at least four products worthy of consideration: MapServer[1], GeoServer[2], MapGuide Opensource[3] and deegree[4]. The most commonly used are MapServer and GeoServer, supported by a large and active community of users and developers. For this work purposes both these servers can handle also large raster files thanks to the map tiling procedure and their performances are comparable (Aime [2010]).
Having a previous personal experience with MapServer, it has been decided to adopt it (MapServer version 5.6.5), but since the client and the server side are completely independent, the choice is up to the developer. In next section the chosen geospatial server will be described in detail.

4.1.1 MapServer

MapServer is one of the most spread open source Web mapping tool. Some of its major features include:

- support for display and querying of hundreds of raster, vector, and database formats;
- ability to run on various operating systems (Windows, Linux, Mac OS X, etc.);
- support for popular scripting languages and development environments (PHP, Python, Perl, Ruby, Java, .NET);
- on-the-fly projections;
- high quality rendering;
- fully customizable application output;
- many ready-to-use Open Source application environments.

[1]http://mapserver.org/
[2]http://geoserver.org/
[3]http://mapguide.osgeo.org/
[4]http://www.deegree.org/

MapServer is a map-rendering engine that works in a Web environment as a CGI script or as a stand-alone application via an API accessible from several programming languages (Kropla [2005]). Developed at the University of Minnesota with help from NASA and the Minnesota Department of Natural Resources, MapServer is maintained by nearly 20 developers (in 2011) from around the world and it is now a project of the Open Source Geospatial Foundation (OSGeo).
Through the use of libraries such as GDAL/OGR[5], MapServer can access various data formats without data conversion (Mitchell [2005]). It can render over 20 different vector data formats, including shapefiles, PostGIS and ArcSDE geometries, OPeNDAP, Arc/Info coverages and MapInfo TAB files. Besides vector formats, MapServer reads two raster formats natively: the GeoTIFF and the EPPL7 (Environmental Planning and Programming Language), but can read over 20 formats (including bitmaps, GIFs and JPEGs) via the GDAL package.
Furthermore, MapServer is able to access and provide data using a Web service framework. It can implement many OGC specifications, in varying degrees of completeness. In particular, MapServer can support Web Map Services (as server or client), Web Feature Services (as server or client), Web Coverage Services (as server) and Sensor Observation Services (as server). A list of the supported OGC standards follows:

- Web Map Service (OGC:WMS) 1.0.0, 1.0.7, 1.1.0, 1.1.1, 1.3.0
- Web Feature Service (OGC:WFS) 1.0.0, 1.1.0
- Web Coverage Service (OGC:WCS) 1.0.0, 1.1.0
- Geography Markup Language (OGC:GML) 2.1.2, 3.1.0 Level 0 Profile
- Web Map Context Documents (OGC:WMC) 1.0.0, 1.1.0
- Styled Layer Descriptor (OGC:SLD) 1.0.0
- Filter Encoding Specification (OGC:FES) 1.0.0
- Sensor Observation Service (OGC:SOS) 1.0.0
- Observations and Measurements (OGC:OM) 1.0.0
- SWE Common (OGC:SWE) 1.0.1
- OWS Common (OGC:OWS) 1.0.0, 1.1.0

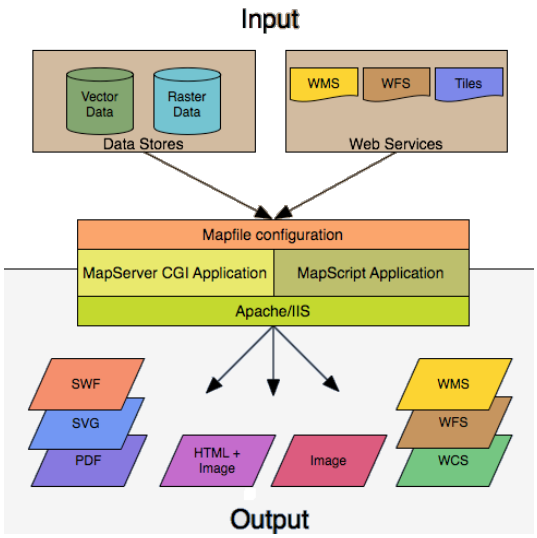


Figure 4.2: The basic architecture of MapServer applications (image taken from MapServer Website)

The basic architecture of a MapServer application, shown in Figure 4.2, consists of:

- a **Mapfile** - a structured text configuration file for your MapServer application. It defines the area of your map, specifies to MapServer where data are and where to output images. It also defines map layers, including their data source, projections, and symbology;

- **Geographic Data** - MapServer can utilize many geographic data source types;

- **HTML Pages** - the interface between the user and MapServer. CGI programs are 'stateless', every request they get is new and they don't remember anything about the last time that they were hit by the application. For this reason, every time an application sends a request to MapServer, it needs to pass context information (what layers are on, where you are on the map, application mode, etc.) in hidden form variables or URL variables.
  A simple MapServer CGI application may include two HTML pages: an initialization file, that uses a form with hidden variables to send an initial query to the Web server and MapServer, and a template file, that controls how the maps and legends output by MapServer will appear in

[5]http://www.gdal.org/

the browser. By referencing MapServer CGI variables in the template HTML, MapServer populates them with values related to the current state of the application (e.g. map image name, reference image name, map extent, etc.) as it creates the HTML page for the browser to read. The template also determines how the user can interact with the MapServer application (browse, zoom, pan, query).

- **MapServer CGI** - the binary or executable file that receives requests and returns images, data, etc. It sits in the cgi-bin or scripts directory of the Web server. The Web server user must have execute rights for the directory where it sits in, and for security reasons, it should not be in the Web root;

- **Web/HTTP Server** - serves up the HTML pages when hit by the user's browser. A working Web (HTTP) server is needed, such as Apache, on the machine on which MapServer is installed.

4.2 Client side

To be consistent with the server side, also on the client side free and open source solutions have been adopted.
There are several interactive viewers able to run in a Web browser. Depending on the technology adopted, plug-ins are sometimes requested to provide specific functionalities not available in the simple browser interaction. New generation Web map clients are completely independent from the server side: they communicate indifferently by means of OGC open protocols. With this kind of applications, that render maps from Web Map and other kind of services, users can navigate maps, zoom in/out, pan, turn layers on/off, add layers and query them. Eventually, they can even give the possibility to edit map features, if it is served with the proper service, or build other maps contacting different services on different servers.

4.2.1 2D visualization

In the open source world there are many solutions to create the client side, like Mapbender, OpenLayers, MapFish and deegree iGeoPortal.
Regarding the 2D visualization, different libraries have been adopted to serve maps and give the user all the useful tools to display and manage them, that

are OpenLayers[6] (version 2.11), GeoExt[7] (version 1.0) and Ext JS[8] (version 3.4.0).

4.2.1.1 OpenLayers, GeoExt and Ext JS

OpenLayers is a object-oriented JavaScript library, using components from Prototype.js[9] and the Rico[10] library, for displaying maps in a browser and it is free from server side dependencies. It implements a JavaScript API for building rich Web-based geographical applications. As a framework, OpenLayers is intended to separate map tools from map data so that all the tools can operate on all the data sources. Furthermore, OpenLayers implements industry-standards methods for geographic data access, such as the OGC WMS and WFS protocols.
OpenLayers is completely free and released under a BSD-style license (2-clause BSD License, also known as FreeBSD). It is a project of the Open Source Geospatial Foundation.
OpenLayers gives the possibility of overlaying multiple standards compliant map layers into a single application.
GeoExt is a JavaScript library that extends Ext JS, a rich library of Web User Interface widgets and helper classes for building Internet applications. It integrates OpenLayers as a mapping library with Ext JS framework.
GeoExt provides a suite of customizable widgets and data handling support that makes it easy to build applications for viewing, editing and styling geospatial data.
GeoExt is released under the BSD license, while Ext JS is available with an open source license compatible with the GNU GPL license v3 only for open source applications.

4.2.2 3D visualization

Once they have been created, 3D city models can be visualized on the Web through OGC Web3D Services (W3DS). This specification is currently in draft status and not yet adopted by the OGC, but it has already been implemented in some projects, like in the Spatial Data Infrastructure (SDI) for the city of Heidelberg (see Figure 4.3) in Germany[11] (Basanow et al. [2008]). A W3DS

[6] http://openlayers.org/
[7] http://www.geoext.org/
[8] http://www.sencha.com/products/extjs/
[9] http://www.prototypejs.org/
[10] http://openrico.org/
[11] http://heidelberg-3d.de

delivers 3D scenes of city or landscape models over the Web as VRML, X3D, GeoVRML or similar formats. This service is used not only for producing static scenes, but also to request data in order to stream it to the client which implements a more dynamic visualization.

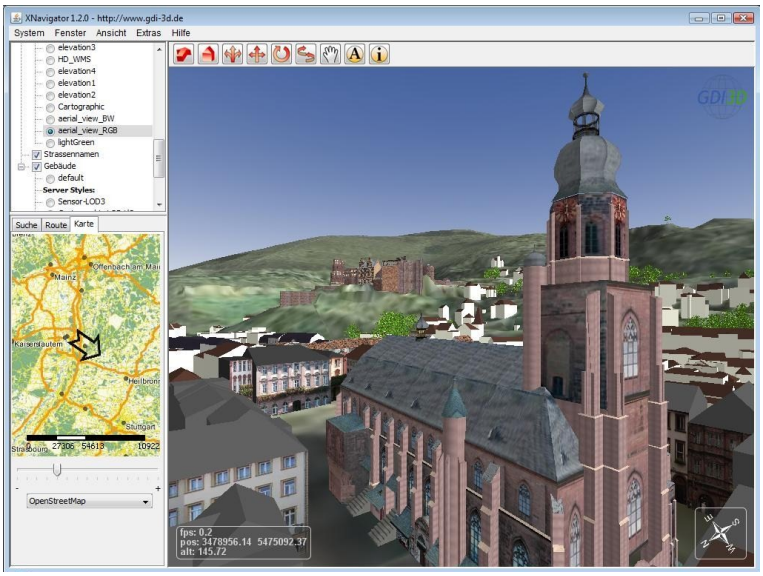


Figure 4.3: The 3D model of a church in Heidelberg. (http://heidelberg-3d.de)

A similar approach has been adopted for a project that investigate the possibility of generating a 3D city model from OpenStreetMap (OSM) free geo-data (Over et al. [2010][12]). OSM, in fact, is the most prominent example of community based mapping project and it has been proved that in some urban areas the quality of roads already surpassed that of commercial or governmental datasets (Haklay [2010]) and this leads to try to exploit OSM information in virtual city models creation. In addition to 2D vector data, height information is also needed and to avoid license problems they chose to use Shuttle Radar Mission (SRTM) height data, which is in the public domain. SRTM provides a global Digital Surface Model (DSM) with a spatial resolution of three arc-seconds (about 90m). The difference between the DSM and a Digital Terrain Model (DTM) gives vegetation and buildings elevation in forest and urban areas. The results of this project have been made available through a W3DS, which serves parts of the 3D city and landscape model as VRML models, and through an interactive Web client for navigation. An example for the city of Milan is represented in Figure 4.4. Nowadays 3D city

[12]http://www.osm-3d.org/

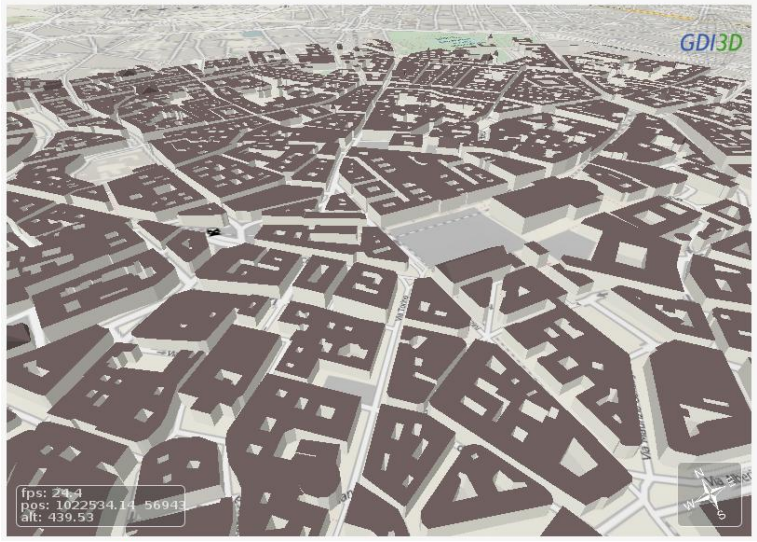


Figure 4.4: The 3D model of Milan in OSM3D.
(http://www.osm-3d.org/)

models have been adopted also to investigate past, present and future of a city, using different 3D models for the different analysed periods. An example is *Virtual Kyoto*[13], developed by the Ritsumeikan University (Takase et al. [2008]). In this case, the 3D modelling of the city began with the present Kyoto and goes back to the past eras, including the Heian era when the city was founded in the late 8^{th} century. The aim of such a project is to archive geo-referenced materials (such as current digital maps, old topographic maps, cadastral maps, archaeological sites data), create a database of all existing buildings and estimate and simulate, trough Virtual Reality (VR) models, land use and landscape changes over the studied periods. Furthermore, being Web-based, the system provides a user-friendly interface to explore comprehensive information on culture and art of Kyoto with its historical landscapes.
Another example is the *Historical Peninsula Project*, started to document and model all buildings in the area of the Greater Municipality of Istanbul, by using terrestrial laser scanning and aerial imagery of the same year (Dursun et al. [2008]). The 3D modelling using different data sources is the most time-consuming task, therefore it has been decided to generate different products with different levels of detail (LoD) and quality for the city model and the landmarks.

[13]http://www.geo.lt.ritsumei.ac.jp/webgis/ritscoe.html

One of the most significant consequences of the use of Cultural Heritage 3D models is the possibility to exploit their highly effective and intuitive means of communication. On the other hand, due to the usual complexity of the architectural and archaeological artefacts or sites, their digital models usually need to be subdivided in sub-components and organized following semantic definitions in order to facilitate data retrieval. The methodology proposed by Manferdini and Remondino [2010] is to semantically segment complex reality-based 3D models and it has been tested on different UNESCO sites and other Cultural Heritage 3D models. Once the 3D models have been segmented, manually or automatically, they can be visualized in Web based systems to allow data access to a wider range of users. The visualization methodology developed for this project is based on the O3D technology, an open-source JavaScript Web-based application which allows a completely free interactive exploration of segmented 3D models. This technology is used to built up VR models, but it does not provide a virtual globe model.
Among the most popular virtual globes, we can distinguish between closed and open source solutions. The most popular closed source technologies are Google Earth, Bing Maps 3D and Ovi Maps 3D. In the open source world, ossimPlanet, gvSIG3D, osgEarth, Norkart Virtual Globe and NASA World Wind can be recalled (Walker and Kalberer [2010]).
In this work it has been decided to adopt an open source solution because of code openness and hence the possibility of customizing it in the content and of extending its functionality. Among the possible technologies, NASA World Wind[14] has been chosen. Its characteristic of being written in Java makes it platform-independent and accessible by a simple Web browser like an applet (Brovelli and Zamboni [2011]). Furthermore, its state of development and the community support have been taken into account, too.
A NASA World Wind Java Applet has been used for the implementation of the 3D virtual globe.

4.2.2.1 NASA World Wind

NASA World Wind[15] is a free and open source API for virtual globes. Being written in Java, it is cross-platform and allows developers to create interactive visualizations of 3D globe, map and geographical information.
World Wind was released in 2004 under the NASA Open Source Agreement (NOSA) and it was ported to Java in 2007. In November 2009 was awarded NASA Software of the Year.
Differently from a 3D globe like Google Earth, it is not a completed application

[14] http://worldwind.arc.nasa.gov/
[15] http://goworldwind.org/

targeted at end users, but a Software Development Kit (SDK). World Wind Java (WWJ) SDK is a pool of components for 3D geographic information Web visualization within applications and applets. Those applications or applets use World Wind by placing one or more *WorldWindow* objects in the user interface. This component provides the 3D geographical context for the application information and behaviours. In addition to *WorldWindow*, five

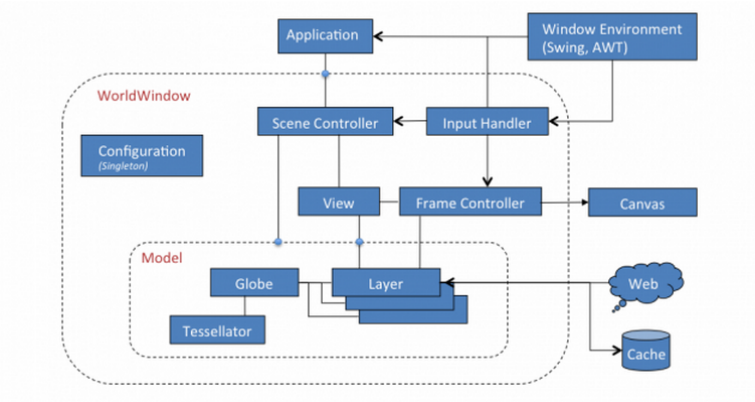


Figure 4.5: The main World Wind interfaces
(image taken from http://goworldwind.org)

main interfaces are present (see Figure 4.5):

- *Globe*, that represents the shape of the planet and the terrain;
- *Layer*, that applies imagery, shapes or other geospatial information to the *Globe*;
- *Model*, that aggregates *Globe* and *Layers*;
- *View*, that interactively controls the user's view of the *Model*, via the *InputHandler*;
- *SceneController*, that controls the rendering of the *Model* and the timing of the rendering. It associates a *Model* with a *View*.

Typically, an application associate a *Globe* and different *Layers* with a *Model*, then the *Model* is passed to a *SceneController* that displays the virtual globe with its layers in a *WorldWindow*, with an interactive *View*.
By default, the platform makes directly available a collection of pre-configured classes to project satellite images (Landsat7, Blue Marble, USGS Ortho-Topo, etc.) and Digital Elevation Models (SRTM, ASTER, USGS NED, etc.) on the

Earth as WMSs dynamically served by NASA and United States Geological Survey (USGS).
Nasa World Wind is able to accept different data formats, such as:

- GIS formats: shapefile, KML, VPF, GML, GeoRSS, GPX, NMEA;
- Imagery: JPG, PNG, GeoTIFF, JPEG2000;
- Government formats: NITF, MrSID, RPF (CADRG, CIB, etc.), DTED;
- Coordinate Systems: Lat/Lon, UTM, MGRS. Datums: WGS84, NAD27.

WWJ works with enormous quantities of data and information, all of which exist primarily on remote data servers. Retrieval and local data caching is therefore a necessary feature of World Wind. The cache has no fixed size, it can be modified programmatically by the application. All data retrievals, even from disk, are performed on background threads.
The complete freedom of customisation makes the platform suitable for scientific applications, having the possibility of controlling horizontal (with the texture) and vertical (with the DTM) components quality and accuracy. In fact, it is possible to project on the globe whichever georeferenced image taken from an OGC compliant WMS server or digital elevation model from specific WMSs, whose code is made available by NASA.
On the globe and in its surrounding space, it is possible to directly place both 2D objects (lines, polygons, markers, call-out, etc.) and 3D objects built up from geometric primitives (parallelepipeds, spheres, extruded polygons, etc.). Nowadays WMSs are the most popular way to provide geographic information, in the form of two-dimensional maps. The ability to superimpose to the terrain model implemented in Word Wind any 2D layer provided by a WMS allows the user to directly obtain a $2\frac{1}{2}$D visualization of the geographic information served. On the other hand, in the WMS-based solution there is a mismatch between the location in height of the variable represented in map form and the content represented by the map itself: in fact, although the content of the thematic map varies dynamically and coherently according to the content, the height of the map on the globe is fixed and defined by the DTM. To overcome this approximation it is necessary to switch to a true 3D modeling of the spatial domain, no longer using surfaces but volumes.
At the server side, the most appropriate format to store three-dimensional domains of irregular shape is the vector format. Such information can then be made available from a WFS server or simply from a Web server by using a standard format currently used (e.g. KML, X3D, GML or shapefile).

Chapter 5

Dataset description

In this chapter data used in this thesis work are described. Besides orthophotos and the building shapefile, early maps of Como have a significant importance for their historical content. In Section 5.3, the main features of those maps are presented, while the process for their georeferencing is shown in Section 5.3.1.

5.1 Orthophoto

As base layer, the current orthophoto provided as a WMS by the National Geoportal[1] has been used in the 2D panel, with planar coordinates in the UTM WGS84 projection (see Figure 5.1).
This orthophoto, created by the orthorectification of aerial photos taken in 2006, covers the whole national territory and gives us, with a spatial resolution of 0.5 m, a detailed view of the area in the chosen reference system.

[1] http://www.pcn.minambiente.it/gn/index.php?lan=en

Figure 5.1: WMS orthophoto of Como served by the National Geoportal

In the World Wind Java Applet the MS Virtual Earth 1998 satellite imagery has been used, with 1 m of resolution (see Figure 5.2).

Figure 5.2: MS Virtual Earth 1998 satellite imagery in the World Wind applet

5.2 Digital cartography

A recent map of buildings of the walled city of Como is present, seen in Figure 5.3. This shapefile has been created in 1994 digitizing images coming from a photogrammetric flight. The scale of this shapefile is 1:1000. In the database file ('.dbf') associated to the shapefile, different data are stored:

- construction period (date of construction and date of demolition);

Figure 5.3: Shapefile representing buildings of Como downtown (1994)

- average building height;
- ancient parishes;
- if the building is under conservation programme.

This information has been retrieved by maps of the Municipality, as the one shown in Figure 5.4. Building mean heights are instead taken by elevation points of the same photogrammetric flight used to realize the shapefile.
Using the information stored in the database, different buildings layers have been created in order to be displayed in the 2D frame, such as:

- buildings represented according to the construction period;
- buildings represented according to ancient parishes;
- buildings under conservation programme.

The average building height is used directly to build up the city building model on the virtual globe.

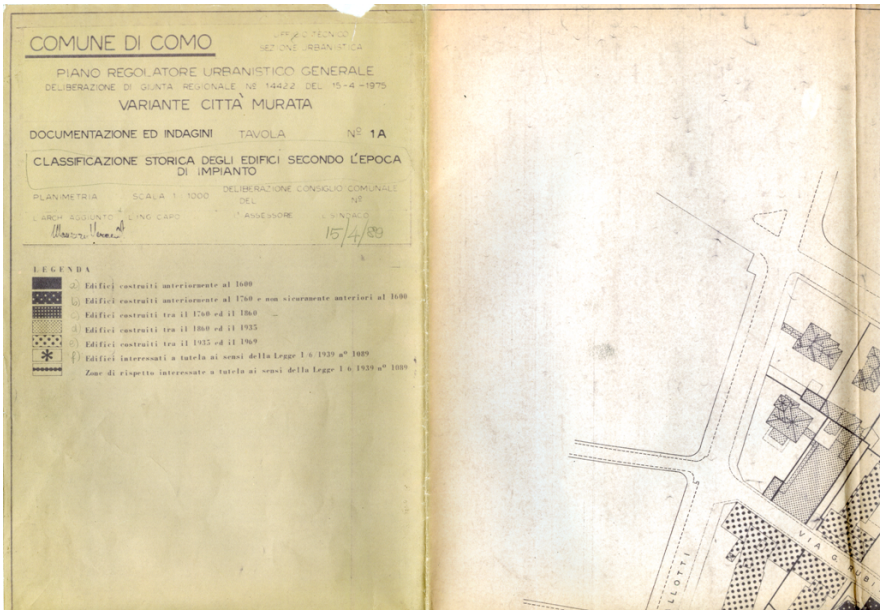


Figure 5.4: Title page of the map represented building according to the construction period

5.2.1 Data model

A specific data model has been developed for our application. An 'event-driven' approach has been adopted: if an object changes in shape or dimensions, the geometry changes too. The evolution in time is then described associating in the attribute table of the shapefile to each polygon two columns: one with the 'construction date' and the other with the 'demolition date'. This means that to every change a new geometry is associated, while the previous one dies.
This approach is similar to the one adopted in CityGML specification: in the class *AbstractBuilding* it can be specified the class of the building, the function (e.g. residential, public, or industry), the usage, the year of construction, the year of demolition, the roof type, the measured height, and the number and individual heights of the storeys above and below ground.
In particular, two parameters are interesting for this application: *yearOfConstruction* and *yearOfDemolition*. Time is specified according to the standard ISO 8601:2004 Data elements and interchange formats - Information interchange - Representation of dates and times.
Also commercial software (e.g. ArcGIS 10) has enabled time support specifying the fields in the database associated to the geometry that correspond to *yearOfConstruction* and *yearOfDemolition*.

5.3 Como early maps

The early maps of Como considered in this work have been digitized by the State Archive of Como and georeferenced within the Web C.A.R.T.E. (*Catalogo e Archivio delle Rappresentazioni del Territorio e delle sue Evoluzioni* - Catalogue and Archive of the Representations of the Territory and its Evolutions) project (Brovelli et al. [2011]). This project, supported by the '*Fondazione Provinciale della Comunità Comasca Onlus*' (Como District Community Onlus Foundation), has been started in 2010 in collaboration with the Como Campus of Politecnico di Milano. The aim of this project is the valorisation of the rich artistic and historical heritage (more than 15000 maps) preserved in the State Archive of Como. After a digitization process, maps have to be georeferenced and then inserted in a geo-catalogue, associating metadata.
Since 2001, original early maps cannot be consulted except as authorized by the person in charge at the State Archive. A digital copy of the map is made available to interested people, but in a black and white PDF format. This reproduction, realized in 1995 digitizing microfilms, is nowadays totally unsatisfactory. For this reason the State Archive has planned a new scan performed by an external company, using ad hoc cold-light planetary scanners and with acquisition parameters suggested by the ICCU (Working group on cartographic material digitization [2006]). Maps are scanned at the resolution of 300 dpi and color depth of 36 (12 x 3) bit, saving both master files in TIFF format and access files (compressed copies) in JPEG format for an easier consultation (Brovelli et al. [2012]). An example of comparison between a 1995 reproduction and the recent high resolution versions is shown in Figure 5.5.
Among the available maps, the most significant ones have been selected belonging to different time spans, so that it is possible to have different representations of the city in time. Maps belong to four distinct cadastral productions:

- the Theresian Cadastre (1718 – 1722);
- the Lombardo-Veneto Cadastre (1854 – 1858);
- updates of the Lombardo-Veneto Cadastre (1898);
- the New Lands Cadastre (1905).

In 1718 Emperor Carl VI created the Census Council and from 1721 to 1723 maps have been surveyed. The map available at the State Archive is a copy

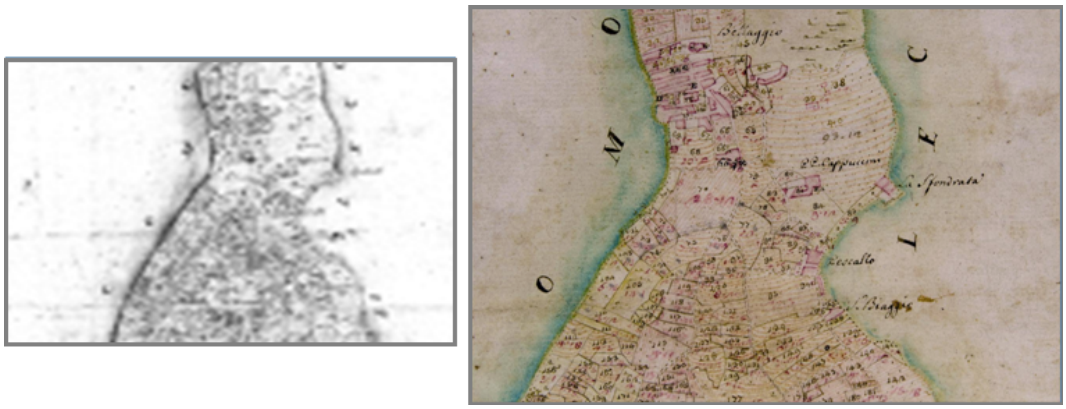


Figure 5.5: Comparison between two corresponding old and recent digitized maps

of the original one surveyed in 1722 and it is divided in several sheets. In particular, six sheets cover the area of the walled city of Como. Maps belonging to this cadastral set are not suitable for building descriptions because they were drawn just to describe and measure lands. In other words, this cadastre is not geometric but descriptive with respect to building parcels. Only in the walled city of Como it is possible to find the representation of inner porches and other details, but without a register number, assigned only to lands. This register came into force from 1760, under Maria Theresa reign.
Since the different sheets were digitized separately, to have the image in Figure 5.6 the different sheets had to be cut to remove the border and joint together using an image processing software, like Adobe Photoshop.
In Table 5.1 the main features of those images have been reported.

Size of the paper map	68.5 x 45.0 cm per sheet
Size of the digital map	16181 x 10630 pixel per sheet
Size on disk of the digital map	57MB per sheet
Geometric resolution	600 dpi
Radiometric resolution	24 bit
Format	TIFF

Table 5.1: Technical features of the Theresian map of Como

The portion of this map that refers to the walled city of Como is represented in Figure 5.6.

Figure 5.6: Portion of the Theresian map of the walled city of Como

The six sheets of the walled city have been digitized using an Epson Perfection V700 Photo scanner at the resolution of 600 dpi, and saved in TIFF format. The author of the map demonstrated to refer to a strict code of representations, using a different symbology for the different land uses, depicting in different ways woods, grasslands and different crops. The orography is represented too with shadings and lines along slopes (Minghini [2010]).
The unit of measurement is the *Milan arms* and the map scale is 1:2000.
The second cadastral map belongs to the Lombardo-Veneto series. It is divided into 6 sheets and a cover paper that reports a description of the map itself. Reading this information, it is possible to find out that this map has been surveyed in 1858, as part of the re-survey process of the Lombard provinces where the Theresian registry was in force.
The map in the State Archive of Como is a copy of the original one, realized

in 1875.
With respect to the previous one, this map appears as clearer, sharper and geometrical, a bit far from the artistic representation of the Theresian registry. In fact, there are no shadows or other graphical representations of the orography, crops and land uses. The use of colours is reduced to indicate if a building is subject to taxation (light pink) or free, like city walls, churches and monasteries (dark carmine with a capital letter), and the main water supply (light blue).
Other differences are the better state of preservation and the presence of updates, corrections and pencil annotations. Those updates account for a gradual adjustment work that continued as long as the map remained in force, until the next one surveyed in 1873.
In Table 5.2 the main features of those images have been reported.

Size of the paper map	69.7 x 53.8 cm per sheet
Size of the digital map	16465 x 12709 pixel per sheet
Size on disk of the digital map	75MB per sheet
Geometric resolution	600 dpi
Radiometric resolution	24 bit
Format	TIFF

Table 5.2: Technical features of the Lombardo - Veneto map of Como

The portion of this map that refers to the walled city of Como is represented in Figure 5.7.

Also in this case early maps have been digitized using a Epson Perfection V700 Photo scanner at the resolution of 600 dpi and saved in TIFF format. As for the previous cadastral series, it was necessary to assemble the different images in a single sheet. The junction is geometrically correct, thanks to the good quality of the different sheets, but it is possible to notice a chromatic discontinuity among the different sheets, due to different conservation and ageing processes.
The unit of measurement is this time the *meter* and the map scale is 1:1000.
In 1873 another similar map has been surveyed with the same features of the previous series and again divided in six sheets. This map came three years before the adoption of the stable census in the Province of Como, 1876. This can lead to think that the previous map of 1858 was never used for cadastral purposes.

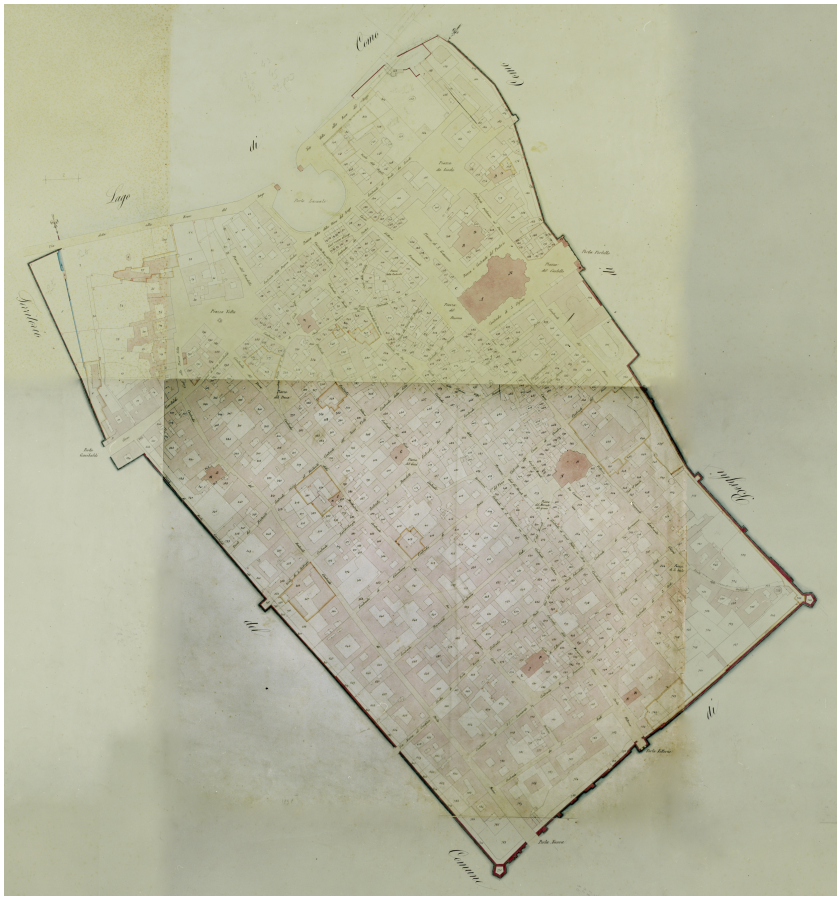


Figure 5.7: Portion of the Lombardo - Veneto registry map of the walled city of Como

In Table 5.3 the main features of those images have been reported.
As can be seen from the table, the use of the same digitization technique

Size of the paper map	69.7 x 53.8 cm per sheet
Size of the digital map	16465 x 12709 pixel per sheet
Size on disk of the digital map	75MB per sheet
Geometric resolution	600 dpi
Radiometric resolution	24 bit
Format	TIFF

Table 5.3: Technical features of the Lombardo - Veneto 1873 updated map of Como

(again with the Epson Perfection V700 Photo scanner) leads to have the same features for the digital maps for the two Lombardo - Veneto series. The joint map of the walled city can be seen in Figure 5.8.

Figure 5.8: Portion of the 1873 Lombardo - Veneto registry map of the walled city of Como

The unit of measurement is the *meter* and the map scale is 1:1000.
With respect to the previous registry map, the main difference in the urban structure is noticeable in the northern part of the city, next to the lake. In 1869, in fact, the Municipality of Como decided to close the lake port and fill it to realize a square (*Piazza Cavour*). The reasons of this decision can be due to the coming of the first steamboats, with problems related to docking and manoeuvring in the narrow space of the old port, and the attempts to contain the frequent floods of the lake.
Figure 5.9 shows a comparison between the two cadastral series in the area

of the old port.

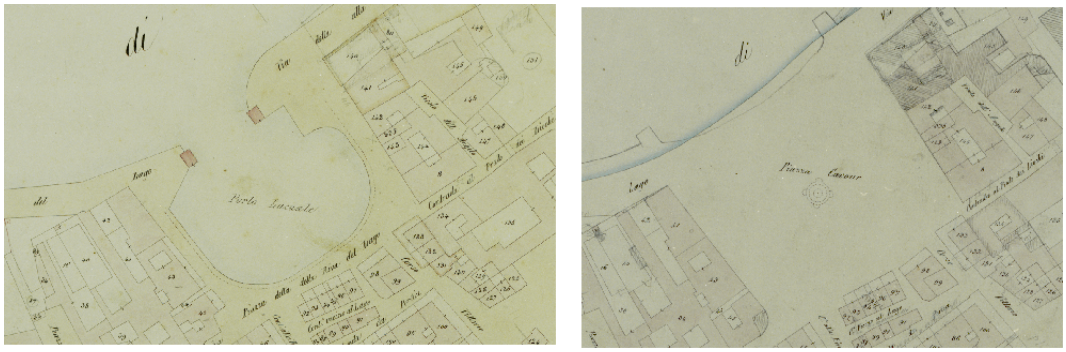


Figure 5.9: Portion of the 1858 (left) and 1873 (right) Lombardo - Veneto registry maps of the old port area

This registry remained into force until 1905, when the first common cadastre for the entire Italian territory (called New Lands Cadastre) was activated.
After the unification of Italy, in 1861, whenever the territory of a pre-unification Member State joined the new Italian State brought its particular cadastre. At the end of the process of unification, therefore, in Italy there were in effect all registers previously present in the territories acquired and, in 1886, the State was divided in 9 cadastral compartments.
As it can be easily understood, in a situation like that there were big dishomogeneity problems due to:

- different kind of registry (only 15 out of 24 were geometric);
- different methods and instruments of survey;
- different units of measurement;
- different currency;

In this context, in 1886 the so called 'Law of the land equalization' (L.3682/1886 - Legge Messedeglia) was adopted, that established the unique Italian land registry, the New Geometric Cadastre, and in particular the New Lands Cadastre (NCT), which replaced the old pre-unification registers.
The new geometric Cadastre has the following features:

- the registration of land separated from the registration of buildings;

- it is geometric particle type;
- it is based on estimations and measurements in a system of classes and rates;
- it is not probative.

In this registry the map scale is 1:2000, with attachments at larger scales (1:1000 or 1:500) for the most densely built territories. As in the previous cases, the walled city of Como map is divided in 6 sheets at scale 1:1000, came into effect in 1905.
Unlike the other early maps, those are stored at the Land Agency instead of the State Archive.
In Table 5.4 the main features of those images have been reported.

Size of the paper map	70.39 x 55.19 cm per sheet
Size of the digital map	5543 x 4346 pixel per sheet
Size on disk of the digital map	∼15MB per sheet
Geometric resolution	200 ppi
Radiometric resolution	24 bit
Format	TIFF

Table 5.4: Technical features of 1905 map of Como

The joint map of the walled city can be seen in Figure 5.10.
As it can be seen from the figure above, this map has a less relevant artistic aspect, appearing more similar to the current digital mapping.
To superimpose and compare those maps, it is necessary to georeference them, choosing a proper projection and reference system.
Besides their artistic value, these maps describe with significant accuracy the territory that they represent, turning out to be precious instruments for scholars and professionals working on this area (e.g. for urban planning or restoration plans).
A description of the georeferencing process, developed within the Web C.A.R.T.E. project by M. Minghini, is presented in the following section.

Figure 5.10: Map of the walled city (1905)

5.3.1 Early maps georeferencing

In order to transform available early map images into real geographical maps, it is necessary to perform a georeferencing step.
As described in Chapter 1, georeferencing early maps means to assign to scanned maps a 'metric reference' from the actual geospace or its mappings (Balletti [2006]).
The chosen reference system in which maps are projected is the WGS84 / UTM zone 32N. This is a CRS for large and medium scale topographic mapping and engineering survey. This reference system has been chosen since the map from which GCPs are taken is projected (see Section 5.2).
To perform this step the module OrthoEngine of the PCI Geomatics package has been used (version 10.0). Among the different algorithms provided by this software, i.e. rational functions, polynomial and thin plate spline, the polynomial one has been chosen, since spline interpolation revealed to be unsuitable for historical maps georeferencing (see Brovelli et al. [2011]).
A m order polynomial can be described with the following formula:

$$\begin{cases} X(x,y) = & a_{0,0} + a_{1,0}x + a_{0,1}y + a_{2,0}x^2 + a_{1,1}xy + a_{0,2}y^2 + \cdots + \\ & + a_{m,0}x^m + a_{m-1,1}x^{m-1}y + a_{1,m-1}xy^{m-1} + a_{0,m}y^m \\ Y(x,y) = & b_{0,0} + b_{1,0}x + b_{0,1}y + b_{2,0}x^2 + b_{1,1}xy + b_{0,2}y^2 + \cdots + \\ & + b_{m,0}x^m + b_{m-1,1}x^{m-1}y + b_{1,m-1}xy^{m-1} + b_{0,m}y^m \end{cases} \tag{5.1}$$

where X, Y are the map coordinates, x, y the corresponding pixel coordinates and the coefficient subscripts recall the exponents of the image coordinates x, y.
To evaluate the model performances, the RMS (Root Mean Square) index has been calculated, both for the Ground Control Points (GCPs), the points used to calculate the model, and the Check Points (CPs), points of known coordinates used to evaluate model performances. In particular, the most suitable polynomial order has been established as the minimum between the one that produces the lowest RMS on CPs residuals modules, and the one that results from a statistical Fisher test on GCPs residuals. This test compares the precision of each couple of consecutive-order polynomials and tells if the higher order one is able to explain the observations in a significantly better way than the other; if not, the conclusion is to stop at the lower order one (Brovelli et al. [2011]).
Statistically comparing the obtained results with the polynomial interpolations, it has been chosen to adopt the following polynomial orders listed in Table 5.5:

Theresian Cadastre	Polynomial of order 4
Lombardo-Veneto Cadastre	Polynomial of order 2
Updates of the Lombardo-Veneto Cadastre	Polynomial of order 1

Table 5.5: Polynomial order chosen in the Web C.A.R.T.E. project for the State Archive Como maps

The number of GCPs and CPs used for each map is listed in Table 5.6. The division is done taking into consideration the 'Hold Out Cross Validation' (HOCV), that divide point in two parts: 80% GCPs and 20% CPs.

Map	**GCPs**	**CPs**
Theresian Cadastre	98	26
Lombardo-Veneto Cadastre	97	25
Updates of the Lombardo-Veneto Cadastre	97	25

Table 5.6: Number of GCPs and CPs for each early map of the Web C.A.R.T.E. project (Minghini [2010])

The results are listed in Table 5.7 and 5.8.
The RMS index is calculated with the following formula, adopted by the OrthoEngine module:

$$RMS = \sqrt{XRMS^2 + YRMS^2}, \tag{5.2}$$

with

$$XRMS = \sqrt{(\mu_{RES_X})^2 + (\sigma_{RES_X})^2} \tag{5.3}$$

and

$$YRMS = \sqrt{(\mu_{RES_Y})^2 + (\sigma_{RES_Y})^2} \tag{5.4}$$

where N is the total number of GCPs or CPs, μ_{RES_X} and σ_{RES_X} are the mean and the standard deviation of of the residuals along the X direction, while μ_{RES_Y} and σ_{RES_Y} are the mean and the standard deviation of of the residuals along the Y direction:

$$\mu_{RES_X} = \frac{1}{N} \sum_{i=1}^{N} RES_{X_i} \tag{5.5}$$

$$\sigma_{RES_X} = \frac{1}{N}\sum_{i=1}^{N}(RES_{X_i} - \mu_{RES_X})^2 \tag{5.6}$$

$$\mu_{RES_Y} = \frac{1}{N}\sum_{i=1}^{N} RES_{Y_i} \tag{5.7}$$

$$\sigma_{RES_Y} = \frac{1}{N}\sum_{i=1}^{N}(RES_{Y_i} - \mu_{RES_Y})^2 \tag{5.8}$$

For each point i, RES_{X_i} and RES_{Y_i} are calculated as the difference between the estimated coordinate, $\hat{X}_i$ or $\hat{Y}_i$, and the corresponding observed one, X_i or Y_i.

Map	**Model**	**RMS**
Theresian Cadastre	Polynomial (4)	4.868
Lombardo-Veneto Cadastre	Polynomial (2)	1.751
Updates of the Lombardo-Veneto Cadastre	Polynomial (1)	1.684

Table 5.7: RMS indexes (in meters) for CPs for each early map of the Web C.A.R.T.E. project (Minghini [2010])

Map	**Model**	**RMS**
Theresian Cadastre	Polynomial (4)	0.958
Lombardo-Veneto Cadastre	Polynomial (2)	0.612
Updates of the Lombardo-Veneto Cadastre	Polynomial (1)	0.606

Table 5.8: RMS indexes (in meters) for GCPs for each early map of the Web C.A.R.T.E. project (Minghini [2010])

Georeferenced early maps are represented in the following figures (see Figure 5.11, 5.12 and 5.13).

Figure 5.11: Georeferenced Theresian map of the walled city of Como

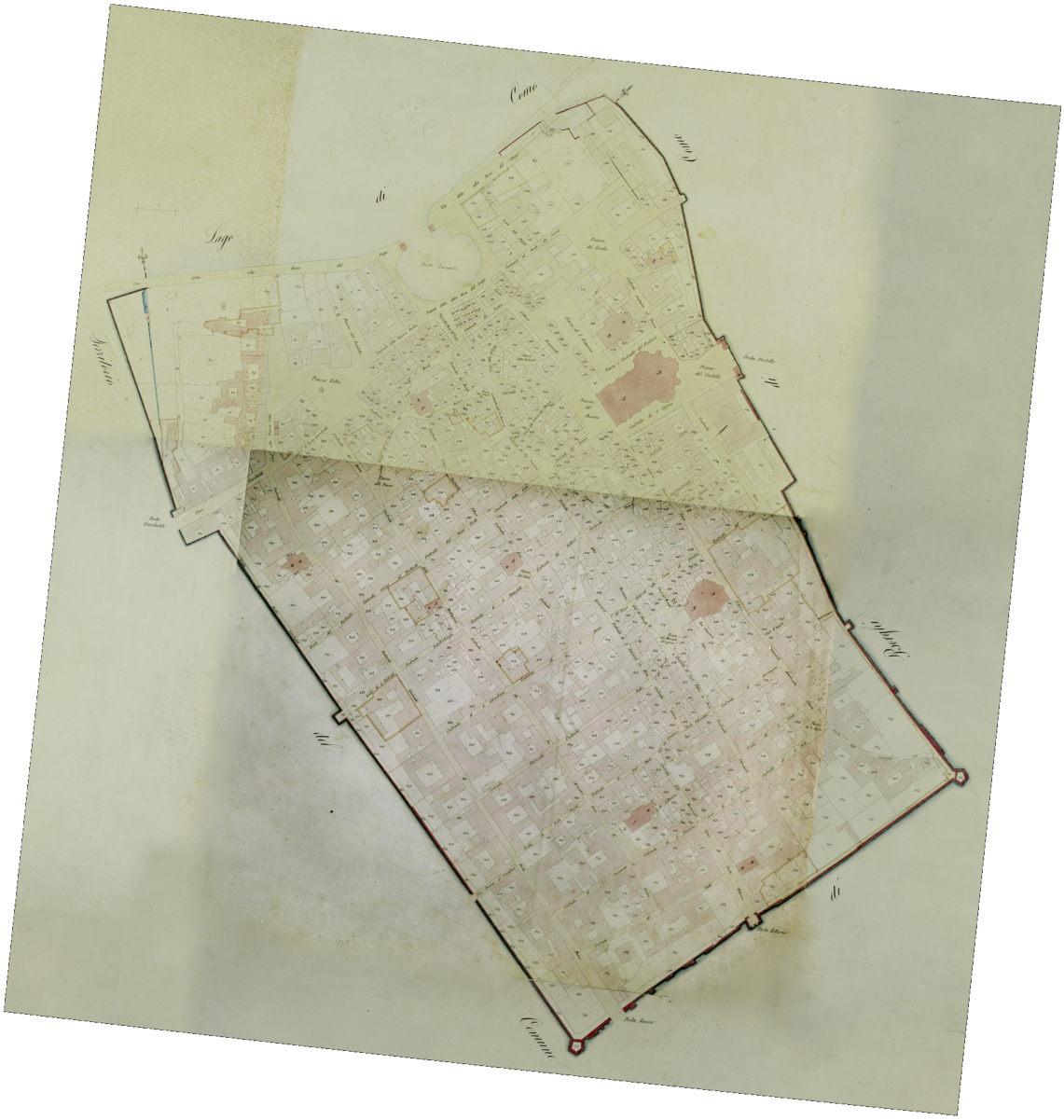


Figure 5.12: Georeferenced Lombardo - Veneto registry map of the walled city of Como

Figure 5.13: Georeferenced updated Lombardo - Veneto registry map of the walled city of Como

For the New Lands Cadastre map the georeferencing tool of the software Quantum GIS, free and open source, has been used. In this case a first order polynomial has been chosen, with 123 points used as GCPs and 24 points as CPs. For those points the RMS indexes were equal to:

- GCPs: 1.57 m;

- CPs: 1.47 m.

Georeferenced New Lands Cadastre map is reported in Figure 5.14.

Figure 5.14: Georeferenced New Lands registry of the walled city of Como

Chapter 6

WebGIS realization

Since digitized early maps are larger than 100MB, it is necessary to perform some pre-processing steps to enhance their visualization inside the WebGIS. These steps have been carried out using GDAL Python commandline utilities. In particular, TIFF images support internal tiling within files, and they will generally give better display speed for local map requests from large images. To produce a GeoTIFF file in internally tiled format using the 'TILED=YES' creation option the 'gdal_translate' utility[1] has been used. Then it is possible to build internal overviews using the 'gdaladdo' utility[2]. This command builds overviews at different decimation levels. By default it uses the nearest neighbour downsampling.
Once data have been acquired and adapted according to the scope, they have been stored in the server. In order to create the map, a mapfile need to be written to specify to MapServer layers and their appearance (see Section 4.1.1). The mapfile is listed in Appendix A.
Adopting the dataset and the technologies described in the previous chapters, a WebGIS has been developed, available at:

`http://historicalmaps.como.polimi.it`.

6.1 Implemented features

For the client side interface, an HTML page has been written (see Appendix B). In the page two panels have been created: one for the 2D map panel and one for the virtual globe.
The interface of the portal is shown in Figure 6.1:

[1] http://www.gdal.org/gdal_translate.html
[2] http://www.gdal.org/gdaladdo.html

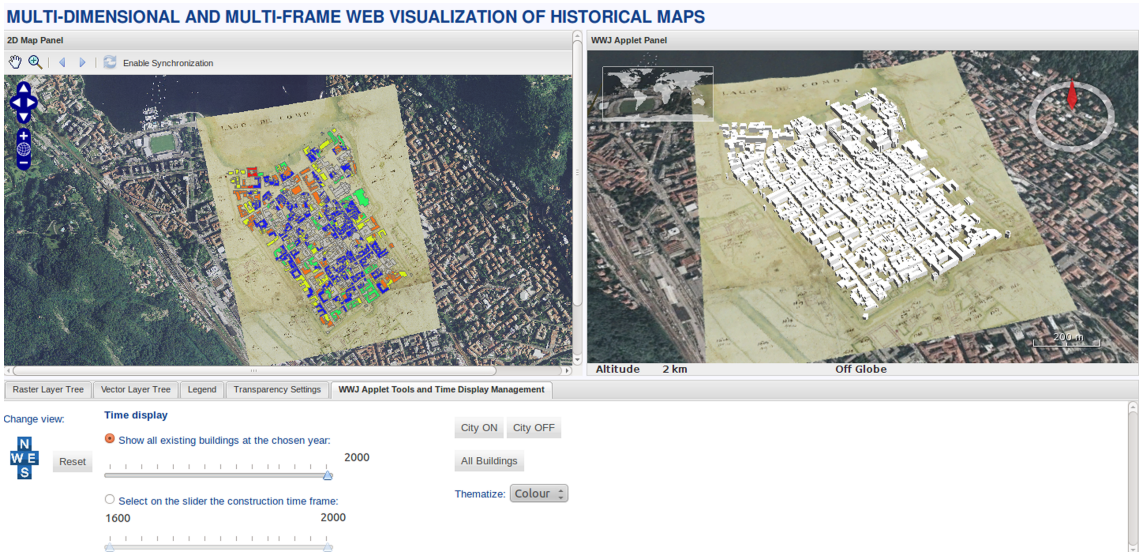


Figure 6.1: Interface of the developed WebGIS

On the left side it is possible to see a 'classical' map panel, while on the right side a virtual globe created with NASA World Wind.
The most common navigation tools have been inserted in the **2D Map Panel** toolbar, such as the *zoom in/out*, *zoom box*, *zoom to the maximum extent*, *pan* and *previous/next view* (see Figure 6.2).

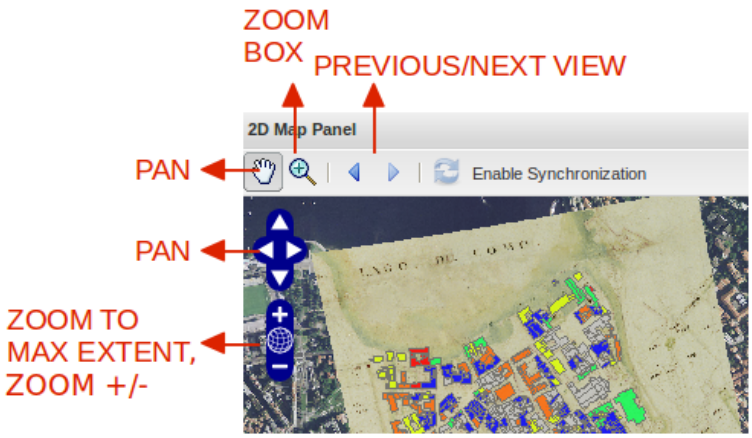


Figure 6.2: Zoom tools

The code for map panel implementation is listed in Appendix C.
The southern part of the page is divided in different tabs to manage several

functionalities.
The first tab contains the **Raster Layer Tree** (Figure 6.3):

Figure 6.3: Raster Layer Tree

The raster layers visible in the WebGIS are:

- orthophoto (2006), served as WMS from the National Geoportal. It is the base layer;
- *Carta Catasto Teresiano* - Theresian Cadastre (1722);
- *Carta Catasto Lombardo - Veneto* - Lombardo-Veneto Cadastre (1858);
- update of the Lombardo-Veneto Cadastre (1873);
- *Carta Catasto Terreni* - New Lands Cadastre (1905).

Acting on this layer tree it is possible to turn on/off the early maps overlapped to the current orthophoto of Como (set as *base layer*), simultaneously on the 2D and the 3D frame. Dragging layer names in the layer tree it is possible to change layer order in the map panel, setting top or bottom layers.
The second tab is related to the **Vector Layer Tree** (Figure 6.4):

Figure 6.4: Vector Layer Tree

The vector layers visible in the WebGIS are:

- buildings under conservation programme, taken from the 1970's municipality map;

- buildings historical classification, taken from the 1970's municipality map;

- ancient parishes of the Theresian period.

This layer tree affects the visualization of vector layers in the 2D frame and, as for the previous one, it is possible to modify the layer order. Once a vector layer is turned on, its legend appear in the **Legend** tab.
The content of the legend tab changes dynamically according to turned on layers. In particular users can obtain the legend of the 'Ancient Parishes' and 'Buildings Historical Classification' layers (Figures 6.5 and 6.6).

Figure 6.5: Ancient Parishes layer with its legend

The following tab contains the **Transparency Settings**: for each layer turned on, except for the base layer one, a transparency bar appears to let the user change the transparency settings of each map. This useful capability can help scholars and urban planners to understand city evolution in time and then how the walled city structure has been preserved or modified.
In Figure 6.7, an example is presented. Changing transparency settings it is possible to detect which buildings are still present from 1858 and which are not and the old city harbour, now Piazza Cavour.

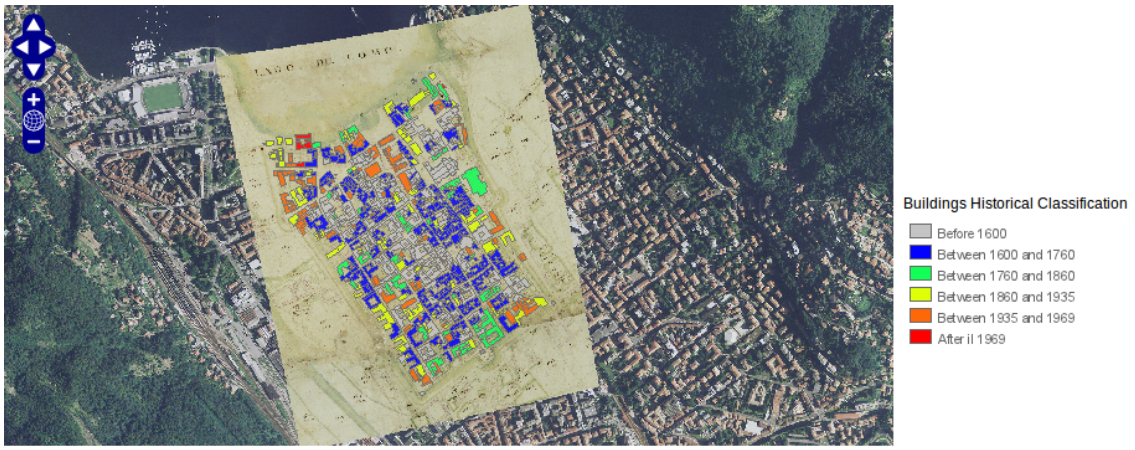


Figure 6.6: Buildings Historical Classification layer with its legend

Figure 6.7: Trasparency settings change for the 'Carta Catasto Lombardo - Veneto (1858)' map

The **WWJ Applet Panel** has all the basic features to get the information about the location on the virtual globe (latitude and longitude), the elevation (in meters) from the ground and the altitude of the view (in kilometres), the scale and the compass (see Figure 6.8).
Besides those basic tools, in the **WWJ Applet Tools and Time Display Management** tab, users have a set of functionalities to act on the applet: it is possible to turn on/off the city model, to thematize by colour or height the extruded buildings and to change the view of the virtual globe.
The Java code is listed in Appendix F.

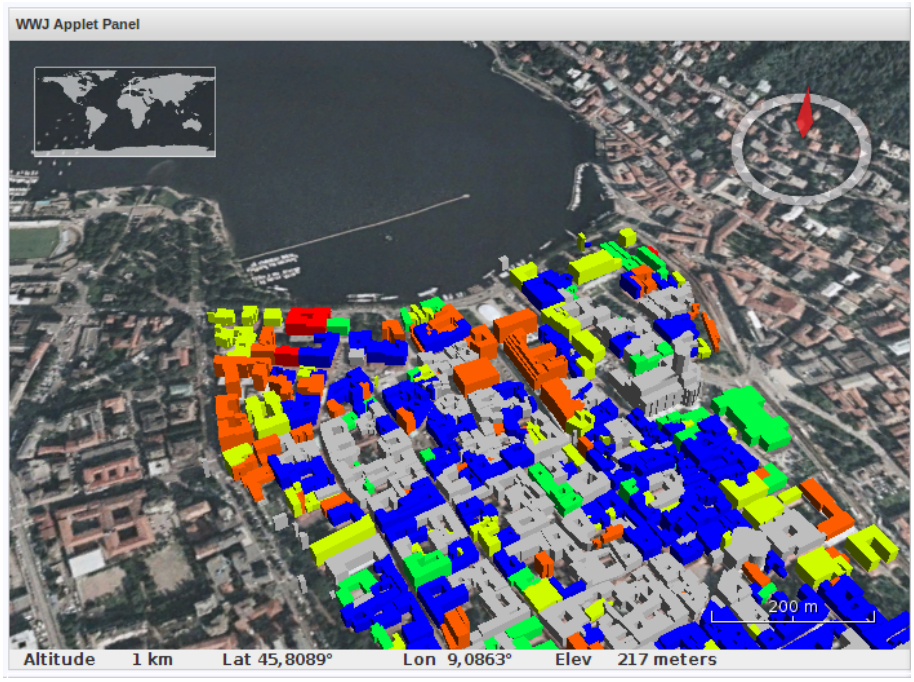


Figure 6.8: WWJ Applet Panel example

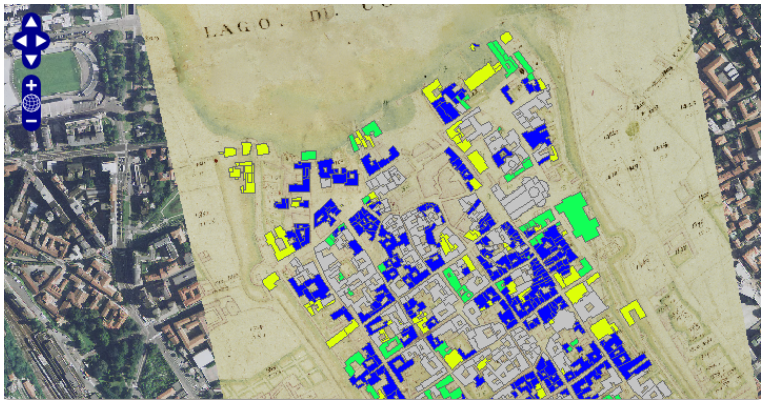
6.1.1 Temporal data handling

By means of two sliders, it is possible to filter buildings view according to different criteria. With the first slider users can visualize *all the existing buildings at a certain year*, selecting an year with the cursor from nowadays back to 1600 (see Figure 6.9).
To perform this filtering in the 2D case, it is not possible to use a WMS with time support (e.g. defining a "wms_timeitem" "DATE_OF_CONSTRUCTION" in the mapfile) but, in case of MapServer, it is necessary to use a run-time substitution of the FILTER variables.
In the mapfile a filter has to be written in the form:

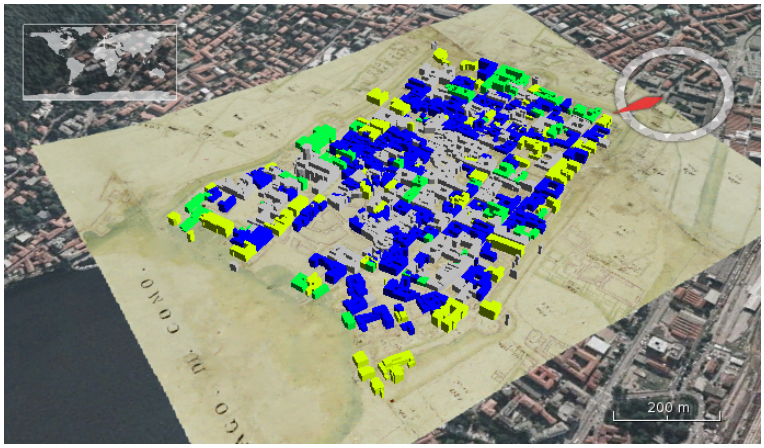
FILTER ([DATE_OF_C] <= %TC% AND [DATE_OF_D] > %TD%)

where %TC% is the 'time of construction' variable, %TD% is the 'time of demolition' one, '[DATE_OF_C]' is the attribute that refers to the construction year and '[DATE_OF_D]' is the year of demolition column in the attribute table (see Appendix A).
In the JavaScript request function it is necessary to write the name of the layer and the function that merge those parameters to the request, as:

(a) Selection of the year on the slider



(b) 2D panel



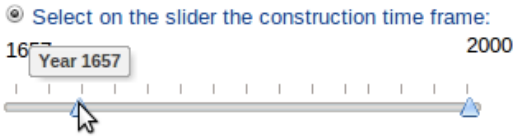
(c) 3D panel

Figure 6.9: Example of selection for the 'existing building' criterion. Only those buildings built up before or in 1865 are shown in the two frames

***layer*.mergeNewParams('TD': t_slider);**
***layer*.mergeNewParams('TC': t_slider);**

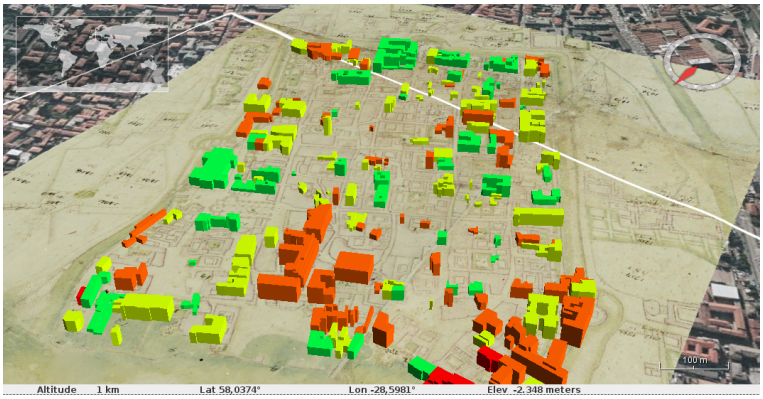
where *t_slider* is the year selected with the time slider on the bar (see Appendix D).
This is needed because the comparison is done between the year chosen by

the user and two different values. This kind of comparison is not possible using a simple WMS time request (see Paragraph 3.1.0.0.1).
The second slider enable the visualization of only those buildings *built up during a certain time span* chosen by the user from the year 2000 to the year 1600 (see Figure 6.10). In this second case, for the 2D panel it is enough to



(a) Selection of the years on the multiple slider

(b) 2D panel



(c) 3D panel

Figure 6.10: Example of selection for the construction period criterion. Only those buildings built up from 1657 to 2000 are shown in the two frames

implement a WMS time request. In the mapfile it is necessary to add a WMS metadata parameter in the layer, as:

"wms_timeitem" "DATE_OF_C"

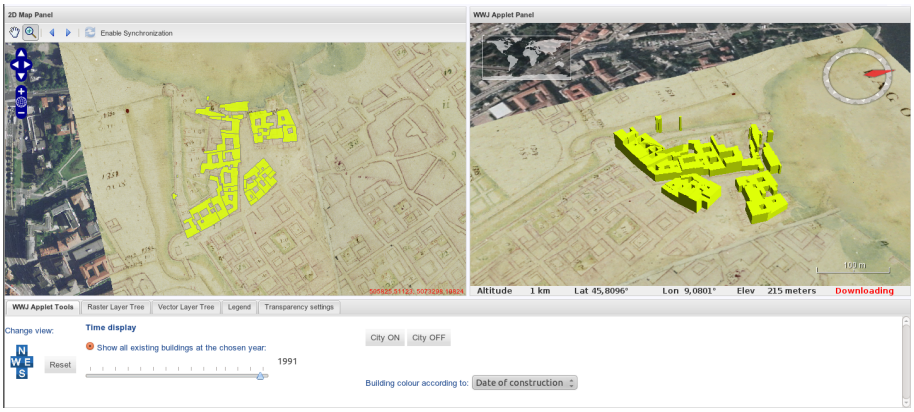
where "DATE_OF_C" is the name of the attribute that correspond to the date of construction in the shapefile table (see Appendix A).
In the JavaScript function the two years select by the user on the slider are caught to define the time interval variable:

t = t_min + "/" + t_max;

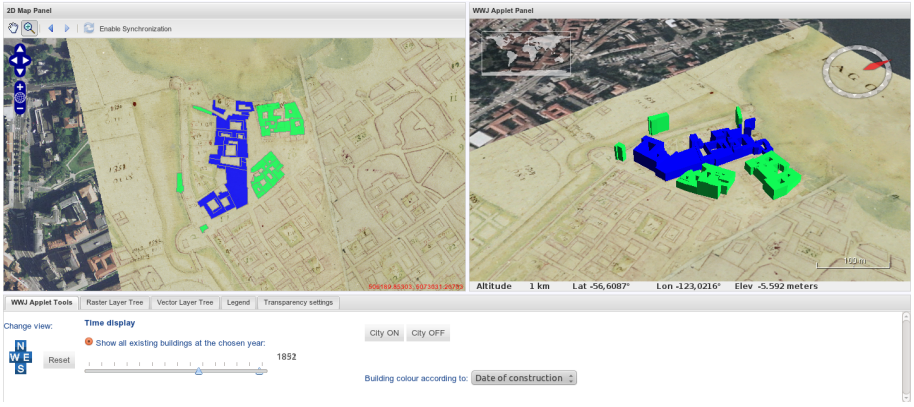
and then the request:

***layer*.mergeNewParams('time': t);**

where t is the time interval previously defined (see Appendix A and D). This function extracts from the selected layer only those buildings with a time of construction greater or equal to *t_min* and lower or equal to *t_max*.
Both the sliders act on the two panels in different ways. Regarding the 3D frame, a Java function is activated to select only the requested buildings according to the different criteria.
To reset the time requests it is possible to click on the 'All Buildings' button, that shows all the buildings of the layer.
Unfortunately, with the available data it is not possible to describe completely the buildings evolution in time, since we do not have enough information about buildings heights change.
With this technology a further improvement is achievable. To verify the possibility of using this system also in case of more detailed data, a test dataset has been carried out. It is hypothesized to have the shape and height of all buildings along time. To each geometry change a *year of construction* and *year of demolition* is assigned. In this way every modification of a building is dynamically represented: acting on a time bar users can select the year of visualization and see the transformation in time, not only demolitions but also changes in shape, dimensions and height.
This leads to have a fully 4D system, in which also the *'time'* dimension is easily managed and presented to the end-user of the service in an intuitive way. In Figure 6.11 it is possible to see how the same portion of the city changed in time: to different years correspond buildings shape and height changes.



(a) Year 1991



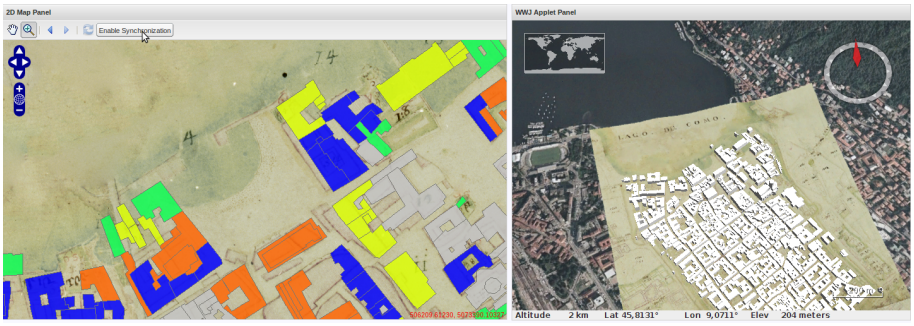
(b) Year 1852

(c) Year 1722

Figure 6.11: Example of visualization of the same portion of the city in different years

6.1.2 Panels synchronization

Since the two panels are built up with completely independent technologies, it is necessary to develop a set of JavaScript and Java functions to let the two panels communicate.
Fist of all, a JavaScript function has been inserted in the raster layer tree code to register the node status: if a map layer is turned on/off for the OpenLayers panel, then it is turned on/off also on the virtual globe (see line 236-238 in Appendix C).
The biggest effort has been done for the synchronization of the location: if the synchronization is enabled we want to let the user see the same portion of the Earth both in the planar view and on the virtual globe. Of course, if the view is not perfectly nadiral on the globe we cannot have exactly the same location on the two panels. The synchronization registers every change applied with a navigation tool (pan, zoom in, zoom out) and pass the bounding box coordinates to another function. While the 2D panel makes use of planar coordinates, in fact, the virtual globe works with geographic ones, so it is necessary to transform the coordinates extent from planar to latitude and longitude. This transformation is done with a JavaScript function that makes use of the traditional Gauss formulas to pass from UTM WGS84 32N zone to WGS84 geographic coordinates (see Appendix E). Having the extent in the WGS84 reference system, it is possible to set the virtual globe view on the desired location. An example is visible in Figure 6.12.



(a) Before the synchronization

(b) After the synchronization

Figure 6.12: Example of synchronization of the two frames

Conclusions

In the last years, Internet GIS tools have been developing and becoming widespread, to permit a more efficient distribution and use of geographic information. An example, which is investigated in this thesis, is the visualization in a WebGIS of different early maps digitized by historical archives, to make this information more user friendly and directly compare historical maps with the actual situation.
Besides the artistic aspect, maps describe with great accuracy the status of the territory and therefore they are a valuable tool for scholars and professionals working on the area (e.g. for urban planning and restoration projects).
Regarding 2D maps, the most rigorous georeferencing procedures have been applied, in order to obtain an accurate result and a good correspondence between the different cadastral maps.
In addition to the traditional 2D display of geographical data, nowadays the so-called *virtual globes*, like Google Earth, are increasingly used in order to extend the perception of the area of interest and give a more realistic representation of reality. These tools are becoming familiar also for non expert users for geographical data browsing. The potential of a 3D system is manifold: complete and more realistic representation of geographic data, greater expressive power, better contextualisation and intuitive navigation. This leads to the need of revising the way data are presented into WebGISs, which will be no longer two-dimensional only.
In this thesis work a new solution for managing historical and current information, as well as 2D and 3D representation, is presented. This is done exploiting the potentiality of free and open source software, that gives the possibility of extending and modifying the software code according to the particular needs of the problem under study. A multi-frame WebGIS has been realized, in which a 3D panel, synchronized with the 2D one, helps users in data visualization and understanding, considering also the time domain.
This is a first example of implementation of such a system, that presents a 3D city model and takes into consideration the time domain too. In fact, assigning to the geometry two dates (i.e. 'date of construction' and 'date of

demolition') it makes possible with a slider to dynamically view how buildings changed in time, both in shape and height.
This new approach can help in the study of city development and in the analysis of different aspects: which are the area that experienced most changes in time, which are the most 'historical' one and in which time period most of building changes took place.
In the WebGIS it is possible to display the various map series made available by the State Archive of Como superimposed on the current orthophoto of the city served as a Web Map Service (WMS) by the National Cartographic Portal. In this way users can appreciate the walled city of Como time evolution.
As mentioned before, the solution can manage the time dimension too. In fact, assigning to building geometries two dates ('date of construction' and 'date of demolition'), it is possible to dynamically view how buildings changed in time, both in shape and height, simply acting on a geographical interface.
At the server-side MapServer has been used. It is an open source platform for publishing spatial data and interactive mapping applications on the Web and serves maps as WMSs.
At the client side, the 2D map panel has been realized using the OpenLayers Web-client and the support of GeoExt and Ext JS libraries. It displays all the historical maps, a recent orthophoto and the current city cartography, representing buildings according to the construction period, the parishes of XVIII century and buildings under conservation programme. A widget allows users to dynamically change map transparency, so that it is possible to visually detect differences among the different epochs.
The 3D frame is obtained through NASA World Wind, a particular virtual globe which is customizable in content area (texture), elevation (digital elevation model) and in which 2D/3D objects can be placed and themed using colours and/or variable geometry, according to specific attributes associated with the objects themselves.
Having the possibility of improving software code, *ad hoc* functionalities have been developed. A specific data model has been designed to enable the time support at the server side. In particular, 3D buildings are displayed using their mean height stored in the attribute table of the shapefile, directly read by the NASA World Wind Java Applet. Buildings can be thematized by colour or height according to the construction period and turned on/off by the users.
JavaScript graphical user interfaces make data visualization intuitive also for non-expert users. Thanks to sliders, in fact, it is possible to dynamically display buildings on the globe according to different criteria. A synchronization between OpenLayers and World Wind has been implemented in order to maintain a constant alignment of the content represented by the two viewers.

Once map layers are turned on for the 2D panel, they are automatically visible also on the globe. The location can be synchronized, too: enabling the synchronization, bounding box extent coordinates are read, transformed from projected to geographic and sent to the NASA World Wind frame in order to visualize the same portion of the globe.
Possible developments of this project could concern the possibility of inserting in World Wind not only an extruded buildings model, but also other models with an higher level of details (e.i. 'COLLADA' models), to provide a more realistic and complete city model.
This project has been presented to Alta Scuola Politecnica (ASP, a school for young talents of Politecnico di Milano e Politecnico di Torino) students of the *CitySpaces* project as an example of platform for exploring and learning about city histories.
In June 2011 this work won the best presentation award at the Open Source GIS Conference 2011 (Nottingham, UK).

Bibliography

A. Aime. FOSS4G 2010 WMS Benchmarking, September 2010. URL `http://2010.foss4g.org/wms_benchmarking.php`.

C. Balletti. Georeference in the analysis of the geometric content of early maps. *e-Perimetron*, 1(1):32–39, 2006.

J. Basanow, P. Neis, S. Neubauer, A. Schilling, and A. Zipf. Towards 3D Spatial Data Infrastructures (3D-SDI) based on open standards — experiences, results and future issues. In Peter Oosterom, Sisi Zlatanova, Friso Penninga, and Elfriede M. Fendel, editors, *Advances in 3D Geoinformation Systems*, Lecture Notes in Geoinformation and Cartography, pages 65–86. Springer Berlin Heidelberg, 2008. ISBN 978-3-540-72135-2.

M. Batty, A. Crooks, A. Hudson-Smith, R. Milton, S. Anand, M. Jackson, and J. Morley. Data mash-ups and the future of mapping. *JISC*, 2010.

E. W. Betz. *Graphic materials: Rules for describing original items and historical collections*. 1982, update 2000 - 2003.

C. Boutura and E. Livieratos. Some fundamentals for the study of the geometry of early maps by comparative methods. *e-Perimetron*, 1(1):60–70, 2006.

T. Bray. Open Data. weblog, July 2006. Available on-line at: http://www.tbray.org/ongoing/When/200x/2006/07/28/Open-Data.

M. A. Brovelli and G. Zamboni. EST-WA: a doxel-based tool for web visualization of environmental variables. In *Geospatial World Forum Dimensions and Directions of Geospatial Industry Conference Proceedings, Hyderabad, India*, 2011.

M. A. Brovelli, M. Minghini, and L. Valentini. Web Services and Historical Cadastral Maps: the first Step in the Implementation of the Web C.A.R.T.E. System. In Anne Ruas, editor, *Advances in Cartography and GIScience.*

Volume 2, volume 6 of *Lecture Notes in Geoinformation and Cartography*, pages 147–161. Springer Berlin Heidelberg, 2011. ISBN 978-3-642-19214-2.

M. A. Brovelli, M. Minghini, G. Giori, and M. Beretta. Web geoservices and ancient cadastral maps: the Web C.A.R.T.E. project. *Transactions in GIS*, 2012. In press.

F. Contò, G. Fanello, and M. Pillon. An information system for historical Cadastre of Venice. *e-Perimetron*, 4(4):240–246, 2009.

A. J. De Leeuw, L. M. M. Veugen, and H. T. C. Van Stokkom. Geometric correction of remotely-sensed imagery using ground control points and orthogonal polynomials. *International Journal of Remote Sensing*, 9(10-11): 1751–1759, 1988.

S. Dursun, D. Sagir, G. Büyüksalih, S. Buhur, T. P. Kersten, and K. Jacobsen. 3D City Modelling of Istanbul Historic Peninsula by Combination of Aerial Images and Terrestrial Laser Scanning Data. In *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, volume XXXVII/B7 of *Proceedings of the ISPRS Commission VII*, pages 1239–1246, 2008.

ESRI. ESRI Shapefile Technical Description. Technical report, ESRI Inc., 1998. Available on-line at: http://www.esri.com.

C. Fleet. The ABC of map digitization. Published on-line, 2009. URL `http://help.oldmapsonline.org/scan/07LocScot_Map_digitisation.doc?attredirects=0`.

A. Guarnieri, F. Pirotti, and A. Vettore. Cultural heritage interactive 3d models on the web: An approach using open source and free software. *Journal of Cultural Heritage*, 11(3):350–353, 2010.

M. Haklay. How good is volunteered geographical information? A comparative study of OpenStreetMap and Ordnance Survey datasets. *Environment and planning C, Government policy*, 37(4):682–703, 2010.

M. Haklay and P. Weber. Openstreetmap: User-generated street maps. *Pervasive Computing, IEEE*, 7(4):12 –18, oct.-dec. 2008. ISSN 1536-1268. doi: 10.1109/MPRV.2008.80.

D. Henrie. Ordnance Survey historic town plans of Scotland (1847-1895): Georeferencing and web delivery with ArcIMS and OpenLayers. *e-Perimetron*, 4(2):73–85, 2009.

ISO/TC 211. ISO/TC 211 Advisory Group on Outreach – Standards Guide. Published on-line, 2009. URL http://www.isotc211.org/Outreach/ISO_TC_211_Standards_Guide.pdf.

T. H. Kolbe. Representing and Exchanging 3D City Models with CityGML. In Jiyeong Lee and Sisi Zlatanova, editors, *3D Geo-Information Sciences*, Lecture Notes in Geoinformation and Cartography, pages 15–31. Springer Berlin Heidelberg, 2009. ISBN 978-3-540-87395-2. URL http://dx.doi.org/10.1007/978-3-540-87395-2_2.

B. Kropla. *Beginning MapServer*. 2005. ISBN 1-59059-490-8.

A. M. Manferdini and F. Remondino. Reality-based 3D modeling, segmentation and web-based visualization. In *Proceedings of the Third international conference on Digital heritage*, EuroMed'10, pages 110–124, Berlin, Heidelberg, 2010. Springer-Verlag. ISBN 3-642-16872-8, 978-3-642-16872-7. URL http://portal.acm.org/citation.cfm?id=1939603.1939614.

M. Minghini. Trasformazioni cartografiche, geocatalogo e servizio Web di visualizzazione dei catasti storici di Como. Master's thesis, Politecnico di Milano, 2010.

T. Mitchell. *Web Mapping Illustrated*. O'Reilly, 2005. ISBN 0-596-00865-1.

A. Mulloni, D. Nadalutti, and L. Chittaro. Interactive walkthrough of large 3D models of buildings on mobile devices. In *Proceedings of the twelfth international conference on 3D web technology*, Web3D '07, pages 17–25, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-652-3. doi: http://doi.acm.org/10.1145/1229390.1229393. URL http://doi.acm.org/10.1145/1229390.1229393.

M. Neteler and H. Mitasova. *Open Source GIS: a GRASS GIS Approach - Second Edition*. Kluwer Academic Publishers, 2004. ISBN 1-4020-8064-6.

A. Nurminen. Mobile, hardware-accelerated urban 3d maps in 3g networks. In *Proceedings of the twelfth international conference on 3D web technology*, pages 7–16. ACM, 2007.

OGC. *OpenGIS Web Map Server Implementation Specification*. Open Geospatial Consortium Inc., 2006. Available on-line at: http://www.opengeospatial.org/standards/wms.

OGC. *OpenGIS Geography Markup Language (GML) Encoding Standard*. Open Geospatial Consortium Inc., 2007. Available on-line at: http://www.opengeospatial.org/standards/gml.

OGC. *OGC KML*. Open Geospatial Consortium Inc., 2008. Available on-line at: http://www.opengeospatial.org/standards/kml.

T. O'Reilly. Open Source Licenses are Obsolete. weblog, August 2006. Available online at: http://radar.oreilly.com/2006/08/open-source-licenses-are-obsol.html.

D. Oreni, R. Brumana, M. Scaioni, and F. Prandi. Navigating on the past, as a bird flight, in the territorial scale of historical topographic maps. WMS on the "Corografie delle Province del Regno Lombardo-Veneto", for accessing cadastral map catalogue. *e-Perimetron*, 5(4):194–211, 2010.

M. Over, A. Schilling, S. Neubauer, and A. Zipf. Generating web-based 3D City Models from OpenStreetMap: the current situation in Germany. *Computers, Environment and Urban Systems*, 34(6):496–507, 11 2010.

Z. R. Peng and M. H. Tsou. *Internet GIS: distributed geographic information services for the internet and wireless networks*. John Wiley & Sons, 2003. ISBN 0-471-35923-8.

T. Presner. HyperCities: A Case Study for the Future of Scholarly Publishing. In Jerome McGann, editor, *The Shape of Things to Come*, pages 251–71. Rice University Press, 2010.

J. A. Richards and X. Jia. *Remote Sensing Digital Image Analysis*. Springer-Verlag, 1999. ISBN 3-540-25128-6.

R. Stallman. *Open Sources : Voices from the Open Source Revolution*, chapter The GNU Operating System and the Free Software Movement. O'Reilly, 1999.

Y. Takase, K. Yano, T. Nakaya, Y. Isoda, T. Kawasumi, K. Matsuoka, T. Seto, D. Kawahara, A. Tsukamoto, M. Inoue, and T. Kirimura. Virtual Kyoto: visualization of historical city with 4D-GIS, virtual reality and web technologies. In *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, volume XXXVII of *Proceedings of the ISPRS Commission V*, pages 223–234, 2008.

M. Walker and P. Kalberer. Comparison of open source virtual globes. In *FOSS4G 2010 Presentations*, Barcelona, 2010.

Working group on cartographic material digitization. *Linee guida per la digitalizzazione del materiale cartografico*. ICCU - Istituto Centrale per il Catalogo Unico, Roma, 2006.

M. Zborovskiy. *Representing and manipulating spatial data in interoperable systems and its industrial applications.* PhD thesis, Massachusetts Institute of Technology, System Design and Management Program, 2006.

M. Zöllner, J. Keil, H. Wuest, and D. Pletinckx. An augmented reality presentation system for remote cultural heritage sites. VAST, 2009.

Appendix A

The Mapfile

The following code refers to the mapfile written for MapServer to generate Web Map Services.
In the first part of the mapfile (see Figure A.1), after opening the 'MAP' object, map characteristics are set: map extent (in the chosen reference system), the projection, the unit of measure, the background colour of the map image, the fontset and the path to the folder containing shapefiles.

```
 1 MAP
 2                 NAME "como"
 3                 EXTENT 505399 5072464 507373 5073500
 4                 UNITS meters
 5                 SHAPEPATH "./ComoStorica/data"
 6                 FONTSET "./ComoStorica/css/fonts/msfontset_orig.txt"
 7                 IMAGECOLOR -1 -1 -1
 8                 TRANSPARENT ON
 9                 IMAGETYPE png
10                 PROJECTION
11                 "+proj=utm +zone=32 +ellps=WGS84 +datum=WGS84 +units=m +no_defs  no_defs"
12                 END
13 #--------------------------------------------------------------------------------------------------
```

Figure A.1: Beginning of the mapfile

A mapfile can declare one or more OUTPUTFORMAT objects, defining available output formats supported (see Figure A.2).

```
15   OUTPUTFORMAT
16   NAME "png"
17   DRIVER "GD/PNG"
18   MIMETYPE "image/png"
19   IMAGEMODE RGB
20   FORMATOPTION  INTERLACE=OFF
21   TRANSPARENT OFF
22   EXTENSION "png"
23   END
24
25   OUTPUTFORMAT
26   NAME "png8"
27   DRIVER "GD/PNG"
28   MIMETYPE "image/png"
29   IMAGEMODE PC256
30   FORMATOPTION  INTERLACE=OFF
31   TRANSPARENT OFF
32   EXTENSION "png"
33   END
34
35   OUTPUTFORMAT
36   NAME "jpeg"
37   DRIVER "GD/JPEG"
38   MIMETYPE "image/jpeg"
39   IMAGEMODE RGB
40   FORMATOPTION "QUALITY=70"
41   EXTENSION "jpg"
42   END
43
44   OUTPUTFORMAT
45   NAME GTiff
46   DRIVER "GDAL/GTiff"
47   MIMETYPE "image/tiff"
48   IMAGEMODE RGB
49   EXTENSION "tif"
50   END
51
52   OUTPUTFORMAT
53   NAME imagemap
54   MIMETYPE "text/html"
55   FORMATOPTION SKIPENDTAG=OFF
56   DRIVER imagemap
57   END
58 #--------------------------------------------------------------------------------------
```

Figure A.2: Output format declaration

To setup a WMS server some parameters and some metadata entries are mandatory. Most of the metadata is required in order to produce a valid GetCapabilites output. For example, it is necessary to add:

- wms_title
- wms_onlineresource
- wms_srs (unless PROJECTION object is defined using “init=epsg:...”)
- wms_enable_request

in the METADATA object (see Figure A.3). Inside the WEB object it is possible to specify IMAGEPATH, the path to the temporary directory fro writing temporary files and images, and IMAGEURL, the base URL for IMAGEPATH.

```
59          WEB
60                  IMAGEPATH "/var/www/tmp/"
61                  IMAGEURL "/tmp/"
62
63                  METADATA
64                          "wms_title"           "WMS Demo Server"
65                          "wms_onlineresource"  "http://myhost/mywms?"
66                          "wms_srs"             "EPSG:32632"
67                          "wms_feature_info_mime_type" "text/html"
68                          "wms_enable_request" "*"
69                  END
70          END
```

Figure A.3: The WEB object

Then it is possible to define the legend style (see Figure A.4). The KEYSIZE parameter sets the size of symbol key boxes in pixels (default is 20 by 10). In the LABEL object the style of labels is defined: the font, the type of font to use, the font size and colour.

```
73          LEGEND
74                  STATUS ON
75                  KEYSIZE 18 12
76                  LABEL
77                          FONT "Arial"
78                          TYPE truetype
79                          SIZE 8
80                          COLOR 102 102 102
81                  END
82
83          END
84 #--------------------------------------------------------------------------------------------------
```

Figure A.4: The LEGEND object

Since the mapfile is read from top to bottom by MapServer: this means that LAYERs near the top of the mapfile will be drawn before those near the bottom. Therefore, raster layers are inserted before vector ones (see Figure A.5). The OFFSITE parameter is equal to 'black', that means that the black background frame has to be set transparent.

```
85          LAYER
86                  NAME "teresiano"
87                  STATUS ON
88                  DATA "/var/www/ComoStorica/data/raster/Catasto_teresiano_1722_polinomio4_GCP.tif"
89                  TYPE raster
90                  OFFSITE 0 0 0
91                  METADATA
92                          DESCRIPTION "Mappa del Catasto Teresiano (1722)"
93                          wms_title "teresiano"
94                  END
95          END
96  #-------------------------------------------------------------------------------------------------
97          LAYER
98                  NAME "lv_1858"
99                  STATUS ON
100                 DATA "/var/www/ComoStorica/data/raster/Catasto_lv_ute_1858-1900_polinomio2_GCP.tif"
101                 TYPE raster
102                 OFFSITE 0 0 0
103                 METADATA
104                         DESCRIPTION "Mappa del Catasto Lombardo - Veneto (1858)"
105                         wms_title "lv_1858"
106                 END
107         END
108 #-------------------------------------------------------------------------------------------------
109
110         LAYER
111                 NAME "lv_1873"
112                 STATUS ON
113                 DATA "/var/www/ComoStorica/data/raster/Catasto_ute_1873_polinomio1_GCP.tif"
114                 TYPE raster
115                 OFFSITE 0 0 0
116                 METADATA
117                         DESCRIPTION "Aggiornamento del Catasto Lombardo - Veneto (1873)"
118                         wms_title "lv_1873"
119                 END
120         END
121 #-------------------------------------------------------------------------------------------------
122         LAYER
123                 NAME "map_1905"
124                 STATUS ON
125                 TYPE RASTER
126                 DATA "/var/www/ComoStorica/data/raster/photoshop/1905_merged_geo_p2_cub_t_t.tif"
127                 METADATA
128                         wms_title "map_1905"
129                 END
130         END
131 #-------------------------------------------------------------------------------------------------
```

Figure A.5: Early maps raster layers

After the raster maps vector layers are inserted. First the "time_layer" (see Figure A.6), in which buildings are represented with different colours according to different construction time periods. This means that it is necessary to specify the different classes styles and definitions. The CLASSITEM parameter defines the item name in the attribute table to use for class lookups. Then the CLASS objects are defined: inside a layer, only a single class will be used for the rendering of a feature. Each feature is tested against each class in the order in which they are defined in the mapfile. The first class that matches the its min/max scale constraints and its EXPRESSION check

for the current feature will be used for rendering. Expression are used to match attribute values with certain logical checks. There are three different types of expressions:

- string comparisons: a single attribute is compared with a string value;
- regular expressions: a single attribute is matched with a regular expression;
- logical "MapServer expressions": one or more attributes are compared using logical expressions.

In this case, the "DATE_O_C" contains strings (the different classes defined in the Municipality map, see Figure 5.4), so attribute values are checked if they are equal to some value. String comparisons are the simplest form of MapServer expressions and the fastest option.
To perform the query on time information (see Chapter 6) a FILTER has been implemented: in this case two columns are read, 'DATE_OF_C' (year of construction) and 'DATE_OF_D' (year of demolition). Then runtime substitutions can be used to change values within a FILTER. In this case, the two variables are 'TD' and 'TC', with a default value set to 2011.
In the METADATA object are inserted the parameters for the WMS time support, like "wms_timeextent" and "wms_timeitem", to indicate the attribute to refer to in case of time queries.
Next layer is the one representing buildings according to their parishes affiliation in the XVIII century. Also in this case different classes have to be defined.
Last layer highlights buildings under conservation programme, as identified in the map depicted in Figure A.6. At the end, the 'END' tag closes the 'MAP' object.

```
133         LAYER
134                 NAME "time_layer"
135                 STATUS ON
136                 DATA "/var/www/WWJ_Applet/WorldWind_Library/Mydata/comoSt2_nn"
137                 TYPE polygon
138                 PROJECTION
139                         "proj=latlong"
140                         "ellps=WGS84"
141                         "datum=WGS84"
142                 END
143                 CLASSITEM "DATE_O_C"
144                 FILTER ([DATE_OF_D]>%TD% AND [DATE_OF_C]<=%TC%)
145                 CLASS
146                         NAME "Before 1600"
147                         EXPRESSION "Before 1600"
148                         STYLE
149                                 COLOR 196 196 196
150                                 OUTLINECOLOR 105 105 105
151                                 SIZE 0.26
152                         END
153                 END
154                 CLASS
155                         NAME "Between 1600 and 1760"
156                         EXPRESSION "Between 1600 and 1760"
157                         STYLE
158                                 COLOR 2 2 255
159                                 OUTLINECOLOR 105 105 105
160                                 SIZE 0.26
161                         END
162                 END
163                 CLASS
164                         NAME "Between 1760 and 1860"
165                         EXPRESSION "Between 1760 and 1860"
166                         STYLE
167                                 COLOR 18 255 87
168                                 OUTLINECOLOR 105 105 105
169                                 SIZE 0.26
170                         END
171                 END
172                 CLASS
173                         NAME "Between 1860 and 1935"
174                         EXPRESSION "Between 1860 and 1935"
175                         STYLE
176                                 COLOR 221 255 8
177                                 OUTLINECOLOR 105 105 105
178                                 SIZE 0.26
179                         END
180                 END
181                 CLASS
182                         NAME "Between 1935 and 1969"
183                         EXPRESSION "Between 1935 and 1969"
184                         STYLE
185                                 COLOR 255 104 8
186                                 OUTLINECOLOR 105 105 105
187                                 SIZE 0.26
188                         END
189                 END
190                 CLASS
191                         NAME "After il 1969"
192                         EXPRESSION "After 1969"
193                         STYLE
194                                 COLOR 255 0 0
195                                 OUTLINECOLOR 105 105 105
196                                 SIZE 0.26
197                         END
198                 END
199                 TEMPLATE "void"
200                 TOLERANCE 6
201                 METADATA
202                         "wms title" "time_layer"
203                         "wms_timeextent" "1000/9999"
204                         "wms_timeitem" "DATE_OF_C"
205                         DESCRIPTION "Date of Construction"
206                         RESULT_FIELDS "DATE_O_C,SAFEGUARD,PARISH1722,LINK,LINKIT"
207                         RESULT_HEADERS "Date of Construction,Safeguarded,Zone,MORE INFORMATION,ALTRE
                            INFORMAZIONI"
208                         RESULT_HYPERLINK "LINK"
209                         'default_TD' '2011'
210                         'default_TC' '2011'
211                 END
212         END
213
214 #-----------------------------------------------------------------------------------------------
```

Figure A.6: The "time_layer" definition

```
215         LAYER
216                 NAME "parrocchie"
217                 STATUS on
218                 DATA "comoSt"
219                 TYPE polygon
220                 CLASSITEM "PARISH1722"
221                 CLASS
222                         NAME "S. Eusebio"
223                         EXPRESSION "S. Eusebio"
224                         STYLE
225                                 COLOR 251 128 114
226                                 OUTLINECOLOR 105 105 105
227                                 SIZE 0.26
228                         END
229                 END
230                 CLASS
231                         NAME "S. Nazaro"
232                         EXPRESSION "S. Nazaro"
233                         STYLE
234                                 COLOR 217 217 217
235                                 OUTLINECOLOR 105 105 105
236                                 SIZE 0.26
237                         END
238                 END
239                 CLASS
240                         NAME "S. Giacomo"
241                         EXPRESSION "S. Giacomo"
242                         STYLE
243                                 COLOR 179 222 105
244                                 OUTLINECOLOR 105 105 105
245                                 SIZE 0.26
246                         END
247                 END
248                 CLASS
249                         NAME "S. Provino"
250                         EXPRESSION "S. Provino"
251                         STYLE
252                                 COLOR 188 128 189
253                                 OUTLINECOLOR 105 105 105
254                                 SIZE 0.26
255                         END
256                 END
257                 CLASS
258                         NAME "S. Maria"
259                         EXPRESSION "S. Maria"
260                         STYLE
261                                 COLOR 252 205 229
262                                 OUTLINECOLOR 105 105 105
263                                 SIZE 0.26
264                         END
265                 END
266                 CLASS
267                         NAME "S. Fedele"
268                         EXPRESSION "S. Fedele"
269                         STYLE
270                                 COLOR 253 180 98
271                                 OUTLINECOLOR 105 105 105
272                                 SIZE 0.26
273                         END
274                 END
275                 CLASS
276                         NAME "S. Benedetto"
277                         EXPRESSION "S. Benedetto"
278                         STYLE
279                                 COLOR 190 186 218
280                                 OUTLINECOLOR 105 105 105
281                                 SIZE 0.26
282                         END
283                 END
284                 CLASS
285                         NAME "S. Donnino"
286                         EXPRESSION "S. Donnino"
287                         STYLE
288                                 COLOR 255 0 0
289                                 OUTLINECOLOR 105 105 105
290                                 SIZE 0.26
291                         END
292                 END
293                 CLASS
294                         NAME "S. Sisto"
295                         EXPRESSION "S. Sisto"
296                         STYLE
297                                 COLOR 204 235 197
298                                 OUTLINECOLOR 105 105 105
299                                 SIZE 0.26
300                         END
301                 END
302                 TEMPLATE "void"
303                 TOLERANCE 6
304                 METADATA
305                         wms_title "parrocchie"
306                         DESCRIPTION "Parishes in XVIII cent."
307                 END
308         END
309
310 #--------------------------------------------------------------------------------------------
```

Figure A.7: The “parrocchie” layer definition

```
311         LAYER
312                 NAME "tutela"
313                 STATUS ON
314                 DATA "como_merge_SAF_Yes_14_12"
315                 TYPE polygon
316                 CLASS
317                         STYLE
318                                 COLOR 255 102 102
319                                 OUTLINECOLOR 196 0 38
320                         END
321                 END
322
323
324                 METADATA
325                     "wms_title"  "tutela"
326                     "wms_srs"    "EPSG:3003 EPSG:4326"
327                 END
328
329         END
330
331 #--------------------------------------------------------------------------------------------------
332 END              # end of the Map Object
```

Figure A.8: Buildings under conservation programme layer

Appendix B

The HTML page code

In the following, the HTML code of the WebGIS page has been listed:

```
<!-- <!DOCTYPE html> -->
<html xmlns="http://www.w3.org/1999/xhtml">
<head> <title>Como</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<!--Link to css stylesheets-->
<link rel="stylesheet" href="./OpenLayers-2.11/theme/default/style.css"
    type="text/css" />
<link rel="stylesheet" type="text/css"
    href="./ext-3.4.0/resources/css/ext-all-notheme.css" />
<link rel="stylesheet" type="text/css"
    href="./ext-3.4.0/resources/css/xtheme-gray.css" />
<link rel="stylesheet" href="./css/style.css" type="text/css" />
<link rel="stylesheet" href="./css/ol_como.css" type="text/css" />
<!--JavaScripts-->
<script type="text/javascript" src="./OpenLayers-2.11/OpenLayers.js"></
    script>
<script type="text/javascript" src="./ext-3.4.0/adapter/ext/ext-base.js"></
    script>
<script type="text/javascript" src="./ext-3.4.0/ext-all.js"></script>
<script type="text/javascript" src="./GeoExt/script/GeoExt.js"></script>
<script type="text/javascript" src="./js/proj4js-combined.js"></script>
<script type="text/javascript" language="JavaScript" src="./js/ol_como.js"><
    /script>
<script type="text/javascript" language="javascript" src="./js/wwj.js"></
    script>
<script type="text/javascript" language="javascript" src="./js/utm2geo.js"><
    /script>
<script type="text/javascript" language="javascript" src="./js/multiSlider.
    js"></script>
<script type="text/javascript" language="javascript" src="./js/singleSlider.
    js"></script>
<script type="text/javascript" language="javascript" src="./js/syncro.js"></
    script>
</head>
<body>
<div id="north">
MULTI-DIMENSIONAL AND MULTI-FRAME WEB VISUALIZATION OF HISTORICAL MAPS
</div>
<!--Map panel-->
<div id="mappanel"></div>
```

```
<div id="description">
<!--WWJ Applet-->
<applet id="wwjApplet" code="org.jdesktop.applet.util.JNLPAppletLauncher"
    archive="Test_Applet_WWJ_files/gluegen-rt.htm" height="100%" width="100%
    " >
    <param name="jnlp_href" value="http://myhost/WWJ_Applet/
        WorldWind_Library/WWJApplet.jnlp" />
    <param name="codebase_lookup" value="false" />
    <param name="subapplet.classname" value="gov.nasa.worldwind.examples.
        applet.WWJApplet" />
    <param name="subapplet.displayname" value="WWJ Applet" />
    <param name="noddraw.check" value="true" />
    <param name="progressbar" value="true" />
    <param name="jnlpNumExtensions" value="1" />
    <param name="jnlpExtension1" value="http://myhost/WWJ_Applet/
        Jogl_Library/jogl.jnlp" />
</applet>
</div>
<div id="str_wwj" class="x-hide-display">
<form name="controlPanel" id="controlPanel" action="">
<div id="navigation3"><br />
Change view:<br/><br/>
<input type="button" class="button_north"
    onclick="getWWJApplet().setHeadingAndPitch(0, 60);" /><br />
<input type="button" class="button_west"
    onclick="getWWJApplet().setHeadingAndPitch(90, 60);" /><br />
<input type="button" class="button_east"
    onclick="getWWJApplet().setHeadingAndPitch(-90, 60);" /><br />
<input type="button" class="button_south"
    onclick="getWWJApplet().setHeadingAndPitch(180, 60);" /><br />
<input type="button" class="reset" value="Reset"
    onclick="getWWJApplet().setHeadingAndPitch(0, 0);" /><br />
</div>
<div id="my_slider"><b>Time display</b><br />
<br /><form name="timeForm" action="">
<input type="radio" name="tempo" value="time2" id="time2" checked onclick="
    barra.setValue(1,2000,true);barra.setValue(0,999,true);barra.disable();
    barra2.enable();" /> Show all existing buildings at the chosen year:<br
    />
<div id="labelSlider" class="label_slider"><input type="text" class="text"
    value="2000" id="textSlider" /></div>
<div id="singleSlider" style="float: left"></div>
<br />
<br />
<br />
<br />
<input type="radio" name="tempo" value="time1" id="time1" onclick="barra2.
    setValue(2000);barra2.disable();barra.enable();" /> Select on the slider
     the construction time frame:<br />
<div id="labelMin" class="label_min"><input type="text"
    class="text" value="1600" name="minimo" id="textMin" /></div>
<div id="labelMax" class="label_max"><input type="text"
    class="text" value="2000" name="massimo" id="textMax" /></div>
<div id="multiSlider"></div>
</form>
</div>
<div id="navigation4"><input value="City ON "
    onclick="getWWJApplet().setVisibleLayer('City',true);" type="button"
    class="button" /> <input value="City OFF"
    onclick="getWWJApplet().setVisibleLayer('City',false);" type="button"
    class="button" /><br />
<br />
```

```
84  <input value="All Buildings"
85      onclick="getWWJApplet().attivaPerCostruzione(900,9999);barra.disable();
            time1.checked='false';time2.checked='false';barra2.enable();barra2.
            setValue(2000);"
86      type="button" class="button" /><br />
87  <br />
88  Thematize: <select>
89      <option onclick="getWWJApplet().tematizza('DATE_OF_C',true,false);">
            Colour</option>
90      <option onclick="getWWJApplet().tematizza('DATE_OF_C',false,true);">
            Height</option>
91  </select>
92  </div>
93  </form>
94  </div>
95  <div id="lt" class="x-hide-display"></div>
96  <div id="vlt" class="x-hide-display"></div>
97  <div id="transp" class="x-hide-display">
98      <div id="o_slider">
99      </div>
100 </div>
101 </body>
102 </html>
```

Listing B.1: The 'index.html' HTML code

Appendix C

The OpenLayers JavaScript file

The following code refers to the main JavaScript file that creates and manage the map and the functionalities of the 2D panel.

```
1  OpenLayers.Control.Click = OpenLayers.Class(OpenLayers.
       Control, {//click on map control
2    defaultHandlerOptions : {
3      'single' : true,
4      'double' : false,
5      'pixelTolerance' : 0,
6      'stopSingle' : false,
7      'stopDouble' : false
8    },
9
10   initialize : function(options) {
11     this.handlerOptions = OpenLayers.Util.extend({}, this.
         defaultHandlerOptions);
12     OpenLayers.Control.prototype.initialize.apply(this,
         arguments);
13     this.handler = new OpenLayers.Handler.Click(this, {
14       'click' : this.trigger
15     }, this.handlerOptions);
16   }
17 });
18
19 //  ------------------------------------------------
20 // map
21 //  ------------------------------------------------
22
23 var map, ortofoto, teresiano, map_1905, lv_1858, lv_1873,
       tutela, time_layer;
24 var mapPanel;
25
26 Ext.onReady(function() {
```

```
27
28  var options = {
29    maxExtent : new OpenLayers.Bounds(505399, 5072200,
          507373, 5073500),
30    maxResolution : 'auto',
31    projection : "EPSG:32632",
32    units : "m",
33    numZoomLevels : 20,
34    zoom : 20
35
36  };
37  map = new OpenLayers.Map('map', options);
38
39  var extent = new OpenLayers.Bounds(505424.06, 5072301.56,
      507349.86, 5073398.44);
40  ortofoto = new OpenLayers.Layer.WMS.Untiled(// ortophoto
      from National Geoportal
41  "Orthophoto 2006", "http://wms.pcn.minambiente.it/cgi-bin/
      mapserv.exe?map=/ms_ogc/service/ortofoto_colore_06_f32.
      map&", {
42    layers : "ortofoto_colore_06_f32",
43    "format" : "image/png",
44    "version" : "1.1.1"
45  }, {
46    isBaseLayer : true
47  });
48  map.addLayer(ortofoto);
49  teresiano = new OpenLayers.Layer.WMS('Carta Catasto
      Teresiano (1722)', 'http://myhost/mywms?', {
50    layers : 'teresiano',
51    "format" : "image/png",
52    "version" : "1.1.1",
53    "transparent" : "TRUE"
54  }, {
55    isBaseLayer : false
56  });
57  map_1905 = new OpenLayers.Layer.WMS('Carta Catasto Terreni
      (1905)', 'http://myhost/mywms?', {
58    layers : 'map_1905',
59    "format" : "image/png",
60    "version" : "1.1.1",
61    "transparent" : "TRUE"
62  }, {
63    isBaseLayer : false
64  });
65  lv_1858 = new OpenLayers.Layer.WMS('Carta Catasto Lombardo
      - Veneto (1858)', 'http://myhost/mywms?', {
66    layers : 'lv_1858',
67    "format" : "image/png",
```

```
68      "version" : "1.1.1",
69      "transparent" : "TRUE"
70    }, {
71      isBaseLayer : false
72    });
73    lv_1873 = new OpenLayers.Layer.WMS('Update Carta Catasto
          Lombardo - Veneto (1873)', 'http://myhost/mywms?', {
74      layers : 'lv_1873',
75      "format" : "image/png",
76      "version" : "1.1.1",
77      "transparent" : "TRUE"
78    }, {
79      isBaseLayer : false
80    });
81    tutela = new OpenLayers.Layer.WMS("Buildings under
          conservation programme", "http://myhost/mywms?", {
82      layers : 'tutela',
83      format : "image/png",
84      version : "1.1.1",
85      transparent : "TRUE"
86    }, {
87      isBaseLayer : false
88    });
89    parrocchie = new OpenLayers.Layer.WMS("Ancient Parishes", "
          http://myhost/mywms?", {
90      layers : 'parrocchie',
91      "format" : "image/png",
92      "version" : "1.1.1",
93      "transparent" : "TRUE"
94    }, {
95      isBaseLayer : false
96    });
97    time_layer = new OpenLayers.Layer.WMS("Buildings Historical
          Classification", "http://myhost/mywms?", {
98      layers : 'time_layer',
99      "format" : "image/png",
100     "version" : "1.1.1",
101     "transparent" : "TRUE"
102   }, {
103     isBaseLayer : false
104   });
105
106   // add layers and set opacity / visibility
107   map.addLayer(teresiano);
108   teresiano.setOpacity(1);
109   map.addLayer(lv_1858);
110   lv_1858.setVisibility(false);
111   map.addLayer(lv_1873);
112   lv_1873.setVisibility(false);
```

```
113  map.addLayer(map_1905);
114  map_1905.setVisibility(false);
115  map.addLayer(parrocchie);
116  parrocchie.setVisibility(false);
117  parrocchie.setOpacity(0.85);
118  map.addLayer(time_layer);
119  time_layer_layer.setOpacity(0.85);
120  map.addLayer(tutela);
121  tutela.setVisibility(false);
122  tutela.setOpacity(0.75);
123
124  map.addControl(new OpenLayers.Control.MousePosition());
125  // add mouse position control
126
127  //  ------------------------------------------------
128  // toolbar
129  //  ------------------------------------------------
130
131  var navigationHistoryControl = new OpenLayers.Control.
         NavigationHistory();
132  map.addControl(navigationHistoryControl);
133  // add navigation history control
134
135  var zoomBoxAction = new GeoExt.Action({// GeoExt zoom box
         control
136    control : new OpenLayers.Control.ZoomBox(),
137    map : map,
138    tooltip : "Draw a rectangle of the area you wish to
           enlarge",
139    iconCls : 'zoomin',
140    toggleGroup : 'map'
141  });
142
143  var panAction = new GeoExt.Action({// GeoExt pan control
144    iconCls : "icon-pan",
145    control : new OpenLayers.Control.Navigation(),
146    map : map,
147    tooltip : "Pan",
148    toggleGroup : 'map'
149  });
150
151  var previousAction = new GeoExt.Action({// GeoExt back
         action
152    iconCls : 'back',
153    control : navigationHistoryControl.previous,
154    disabled : true,
155    tooltip : "Previous View"
156  });
157
```

```
158   var nextAction = new GeoExt.Action({// GeoExt next action
159     iconCls : 'next',
160     control : navigationHistoryControl.next,
161     disabled : true,
162     tooltip : "Next View"
163   });
164
165   var synchro_but = new Ext.Action({// Ext action to
          synchronize view (see synchro.js function)
166     iconCls : 'icon-synchro',
167     handler : function() { synchro;
168       map.events.register('moveend', map, synchro);
169     },
170     disabled : true
171   });
172
173   var attiva_disat = new Ext.Action({// Function to enable/
          disable synchronization
174     text : 'Enable Synchronization',
175     handler : function() {
176       synchro_but.setDisabled(!synchro_but.isDisabled());
177       synchro();
178       this.setText(synchro_but.isDisabled() ? 'Enable
            Synchronization' : 'Disable Sync');
179       if(synchro_but.isDisabled() == true) {
180         map.events.unregister('moveend', map, synchro);
181       } else {
182         map.events.register('moveend', map, synchro);
183       }
184     }
185   });
186
187   //  ---------------------------------------------
188
189   var toolbarItems = [panAction, zoomBoxAction, "-",
          previousAction, nextAction, "-", synchro_but,
          attiva_disat, "->"];
190
191   //  ---------------------------------------------
192
193   topToolbar = new Ext.Toolbar(toolbarItems);
194   var mousePositionElement = Ext.get('mouseposition');
195
196   //  ---------------------------------------------
197   // map panel
198   //  ---------------------------------------------
199
200   mapPanel = new GeoExt.MapPanel({
201     title : "2D Map Panel",
```

```
202     renderTo : "mappanel",
203     stateId : "mappanel",
204     height : 600,
205     width : 900,
206     map : map,
207     tbar : topToolbar
208   });
209
210   //  ------------------------------------------------
211   // layer trees
212   //  ------------------------------------------------
213
214   var rasterList = new GeoExt.tree.LayerContainer({
215     text : 'Raster Layers',
216     layerStore : mapPanel.layers,
217     leaf : false,
218     expanded : true,
219     loader : {
220       filter : function(record) {
221         return record.get("layer").name.indexOf('Carta') !==
              -1 || record.get("layer").name.indexOf('Orthophoto
              ') !== -1 || record.get("layer").name.indexOf('osm
              ') !== -1
222       }
223     }
224   });
225
226   var rasterTree = new Ext.tree.TreePanel({
227     renderTo : 'lt',
228     map : map,
229     root : rasterList,
230     autoScroll : true,
231     enableDD : true,
232     border : false,
233     listeners : {
234       'checkchange' : function(node, checked) {
235         if(checked) {
236           getWWJApplet().setVisibleLayer(node.text, true);
237         } else {
238           getWWJApplet().setVisibleLayer(node.text, false);
239         }
240
241         var d = document.getElementById('o_slider');
242         d.innerHTML = '';
243         for(var i = 0; i < map.layers.length; i++) {
244
245           if(map.layers[i].visibility == true && map.layers[i
                ].isBaseLayer == false) {
246
```

```
247             var txt = document.createTextNode(map.layers[i].
                    name);
248
249             document.getElementById('o_slider').appendChild(
                    txt);
250
251             var o_slider = new GeoExt.LayerOpacitySlider({
252               renderTo : 'o_slider',
253               aggressive : true,
254               width : 200,
255               inverse : true,
256               layer : map.layers[i],
257               dynamic : true,
258               plugins : new GeoExt.LayerOpacitySliderTip({
259                 template : "Transparency of " + map.layers[i
                        ].name + ": {opacity}%"
260               })
261             });
262
263           }
264         };
265       }
266     }
267   });
268
269   var vectorList = new GeoExt.tree.LayerContainer({
270     text : 'Vector Layers',
271     layerStore : mapPanel.layers,
272     leaf : false,
273     expanded : true,
274     loader : {
275       filter : function(record) {
276         return record.get("layer").name.indexOf('Buildings')
                    !== -1 || record.get("layer").name.indexOf('
                    Parishes') !== -1
277       }
278     }
279   });
280
281   var vectorTree = new Ext.tree.TreePanel({
282     renderTo : 'vlt',
283     map : map,
284     root : vectorList,
285     autoScroll : true,
286     enableDD : true,
287     border : false,
288     listeners : {
289       'checkchange' : function(node, checked) {
290         if(checked) {
```

```
291           getWWJApplet().setVisibleLayer(node.text, true);
292         } else {
293           getWWJApplet().setVisibleLayer(node.text, false);
294         }
295 
296         var d = document.getElementById('o_slider');
297         d.innerHTML = '';
298         for(var i = 0; i < map.layers.length; i++) {
299 
300           if(map.layers[i].visibility == true && map.layers[i
                ].isBaseLayer == false) {
301 
302             var txt = document.createTextNode(map.layers[i].
                   name);
303 
304             document.getElementById('o_slider').appendChild(
                   txt);
305 
306             var o_slider = new GeoExt.LayerOpacitySlider({
307               renderTo : 'o_slider',
308               aggressive : true,
309               width : 200,
310               inverse : true,
311               layer : map.layers[i],
312               dynamic : true,
313               plugins : new GeoExt.LayerOpacitySliderTip({
314                 template : "Transparency of " + map.layers[i
                        ].name + ": {opacity}%"
315               })
316             });
317           }
318         };
319       }
320     }
321   });
322 
323   //  ------------------------------------------------
324   // tabs
325   //  ------------------------------------------------
326 
327   var tabs = new Ext.TabPanel({
328     region : "south",
329     activeTab : 4,
330     split : true,
331     items : [{
332       contentEl : 'lt',
333       title : 'Raster Layer Tree'
334     }, {
335       contentEl : 'vlt',
```

```
336         title : 'Vector Layer Tree'
337       }, {
338         title : "Legend",
339         border : true,
340         filter : function(record) {
341           return record.get("layer").name.indexOf('Buildings')
                  !== -1 || record.get("layer").name.indexOf('
                  Parishes') !== -1
342         },
343         dynamic : true,
344         autoScroll : true,
345         xtype : "gx_legendpanel"
346       }, {
347         contentEl : 'transp',
348         title : 'Transparency Settings',
349         autoscroll : true,
350         dynamic : true
351       }, {
352         contentEl : 'str_wwj',
353         title : 'WWJ Applet Tools and Time Display Management',
354         autoscroll : true,
355         dynamic : true
356       }]
357     });
358
359     //  ------------------------------------------------
360     // viewport components
361     //  ------------------------------------------------
362
363     var viewport = new Ext.Viewport({
364       layout : "border",
365       items : [new Ext.BoxComponent({
366         region : 'north',
367         el : 'north',
368         height : 30,
369         margins : {
370           left : 5,
371           top : 5
372         }
373       }), new Ext.BoxComponent({
374         region : 'center',
375         el : 'mappanel',
376         split : true,
377         autoScroll : true
378       }), {
379         region : "east",
380         id : 'wwj_applet',
381         title : 'WWJ Applet Panel',
382         contentEl : "description",
```

```
383         width : 700,
384         split : true,
385         autoScroll : true
386       }, tabs]
387     });
388
389     //  ---------------------------------------------
390     // viewport
391     //  ---------------------------------------------
392
393     viewport.doLayout();
394     var click = new OpenLayers.Control.Click();
395     map.addControl(click);
396     click.activate();
397
398     //  ---------------------------------------------
399     // transparency settings
400     //  ---------------------------------------------
401
402     for(var i = 0; i < map.layers.length; i++) {
403
404       if(map.layers[i].visibility == true && map.layers[i].
              isBaseLayer == false) {
405
406         var txt = document.createTextNode(map.layers[i].name);
407
408         document.getElementById('o_slider').appendChild(txt);
409
410         var o_slider = new GeoExt.LayerOpacitySlider({
411           renderTo : 'o_slider',
412           aggressive : true,
413           width : 200,
414           inverse : true,
415           layer : map.layers[i],
416           dynamic : true,
417           plugins : new GeoExt.LayerOpacitySliderTip({
418             template : "Transparency of " + map.layers[i].name
                    + ": {opacity}%"
419           })
420         });
421       }
422     };
423
424 });
```

Listing C.1: The OpenLayers JavaScript code

Appendix D

The JavaScript slider functions

In the following, the JavaScript code of the function for managing the single time slider.

```
1  var t_slider; //Time variable
2  var barra2; //Single time slider variable
3
4  Ext.onReady(function() {
5
6    var tip = new Ext.slider.Tip({//Slider tip function
7      getText : function(thumb) {
8        return String.format('<b>Year {0}</b>', thumb.value);
9      }
10   });
11   barra2 = new Ext.slider.SingleSlider({//Single slider
       function
12     renderTo : 'singleSlider',
13     width : 290,
14     minValue : 1599,
15     maxValue : 2000,
16     value : 2000,
17     plugins : tip,
18     listeners : {
19       change : function() {
20
21         //Set the time variable equal to the year selected by
               the user
22         t_slider = barra2.getValue(0);
23
24         //Set the text in the page equal to the time variable
25         Ext.get('textSlider').set({
26           value : t_slider
27         });
```

```
28
29          //Activate the selection in the WWJ Applet
30          getWWJApplet().attivaPerCostruzione(t_slider);
31
32          //Set the time variables in the layer filter
33          t_d = t_slider;
34          time.mergeNewParams({
35            'TD' : t_d
36          });
37          time.mergeNewParams({
38            'TC' : t_d
39          });
40        }
41      }
42    });
43 });
```

Listing D.1: The 'singleSlider' JavaScript code

Code of the JavaScript function for managing the multiple time slider:

```
1  var t_min; //Minimum time variable
2  var t_max; //Maximum time variable
3  var barra; //Multiple time slider variable
4
5  Ext.onReady( function() {
6    var tip = new Ext.slider.Tip( {  //Slider tip function
7      getText : function(thumb) {
8        return String.format('<b>Year {0}</b>', thumb.value);
9      }
10   });
11
12   barra = new Ext.slider.MultiSlider( { //Multi slider
        function
13     renderTo : 'multiSlider',
14     width : 290,
15     minValue : 1600,
16     maxValue : 2000,
17     values : [1600, 2000],
18     disabled: true,
19     plugins : tip,
20     listeners : {
21       change : function() {
22
23         //Set the min and max time variables equal to the
              years selected by the user
24         t_min = barra.getValue(0);
25         t_max = barra.getValue(1);
26
```

```
27          //Set the text in the page equal to the t_min
                variable
28          Ext.get('textMin').set( {
29            value : t_min
30          });
31
32          //Set the text in the page equal to the t_max
                variable
33          Ext.get('textMax').set( {
34            value : t_max
35          });
36
37          //Activate the selection in the WWJ Applet
38          getWWJApplet().attivaPerCostruzione(t_min, t_max);
39
40          //Merge the time parameter to the WMS request
41          t = t_min + '/' + t_max;
42          time.mergeNewParams( {
43            'time' : t
44          });
45        }
46      }
47    });
48
49 });
```

Listing D.2: The 'multiSlider' JavaScript code

Appendix E

The synchronization function

This function read the map extent from the UTM WGS84 32N panel, transforms the coordinates in geographical and goes to the same location in the WWJ Applet Panel:

```
1  function syncro() {
2    //Get the minimum horizontal coordinate of the map extent
3    left_ex = map.getExtent().left;
4
5    //Get the maximum horizontal coordinate of the map extent
6    right_ex = map.getExtent().right;
7
8    //Get the minimum vertical coordinate of the map extent
9    bottom_ex = map.getExtent().bottom;
10
11   //Get the maximum vertical coordinate of the map extent
12   top_ex = map.getExtent().top;
13
14   //Lower left corner coordinates transformation from
        geographical to UTM WGS84 32N}
15   l = utm2geo(left_ex, bottom_ex).latitude;
16   b = utm2geo(left_ex, bottom_ex).longitude;
17
18   //Upper right corner coordinates transformation from
        geographical to UTM WGS84 32N
19   r = utm2geo(right_ex, top_ex).latitude;
20   t = utm2geo(right_ex, top_ex).longitude;
21
22   //Activation of the WWJ Applet function that synchronize
        the view on the same extent
23   getWWJApplet().zoomToSector_Ver1(l, b, r, t);
```

```
};
```

Listing E.1: The 'synchro' JavaScript code

The 'utm2geo.js' transformation function is listed in the following:

```
//Transform coordinates from UTM WGS84 32N to latitude and
    longitude with Gauss equations

function utm2geo(E, N) {

  var sa, sb, e2, e2quadro, c, x, y, zone, S, lat, v, a, a1,
     a2, j2, j4, j6, alfa, beta, gamma, Bm, b, Epsi, Eps, nab
     , senoheps, Delt, TaO;
  //Semi-major axis
  sa = 6378137.000000;
  //Semi-minor axis
  sb = 6356752.314245;
  e2 = Math.pow((Math.pow(sa, 2) - Math.pow(sb, 2)), 0.5) /
     sb;
  e2quadro = Math.pow(e2, 2);
  c = (sa * sa) / sb;
  X = E - 500000;
  //Relative to central meridian
  Y = N;
  zone = 32;
  S = ((zone * 6) - 183);
  lat = Y / (6366197.724 * 0.9996);
  //0.9996 - scale along the central meridian of zone
  v = (c / Math.pow((1 + (e2quadro * Math.pow(Math.cos(lat),
     2))), 0.5)) * 0.9996;
  a = X / v;
  a1 = Math.sin(2 * lat);
  a2 = a1 * Math.pow(Math.cos(lat), 2);
  j2 = lat + (a1 / 2);
  j4 = ((3 * j2) + a2) / 4;
  j6 = ((5 * j4) + a2 * Math.pow(Math.cos(lat), 2)) / 3;
  alfa = (3 / 4) * e2quadro;
  beta = (5 / 3) * Math.pow(alfa, 2);
  gamma = (35 / 27) * Math.pow(alfa, 3);
  Bm = 0.9996 * c * (lat - alfa * j2 + beta * j4 - gamma * j6
     );
  b = (Y - Bm) / v;
  Epsi = ((e2quadro * Math.pow(alfa, 2)) / 2) * Math.pow(Math
     .cos(lat), 2);
  Eps = a * (1 - (Epsi / 3));
  nab = (b * (1 - Epsi)) + lat;
  senoheps = (Math.exp(Eps) - Math.exp(-Eps)) / 2;
  Delt = Math.atan(senoheps / Math.cos(nab));
```

```
37    TaO = Math.atan(Math.cos(Delt) * Math.tan(nab));
38    longitude = (Delt * (180 / Math.PI)) + S;
39    latitude = (lat + (1 + e2quadro * (Math.pow(Math.cos(lat),
          2)) - (3 / 2) * e2quadro * Math.sin(lat) * Math.cos(lat)
           * (TaO - lat)) * (TaO - lat)) * (180 / Math.PI);
40
41    function coord(latitude, longitude) {
42      this.latitude = latitude;
43      this.longitude = longitude;
44    }
45
46    coord = new coord(latitude, longitude);
47    return coord;
48  }
```

Listing E.2: The 'utm2geo.js' JavaScript code

Appendix F

The JAVA World Wind code

In the following, the code of the World Wind Java Applet is reported.
In particular, in this implementation of the WWJ Applet class it is possible to render a 'cityLayer' (described in the following), a particular layer specifically developed to represent cities assigning a mean height to the different layers. Among the different methods, some of them have been specifically implemented (or adapted) for this work, such as:

- *activateForConstruction*: activate a building according to a construction time span or a construction year;

- *thematize*: change city layer thematization according to the construction period (colours or heights);

- *zoomToSector* (two versions): to synchronize the visualization on the two panels;

- *setVisibleLayer*: with this function it is possible to turn on/off the same layers on the two frames.

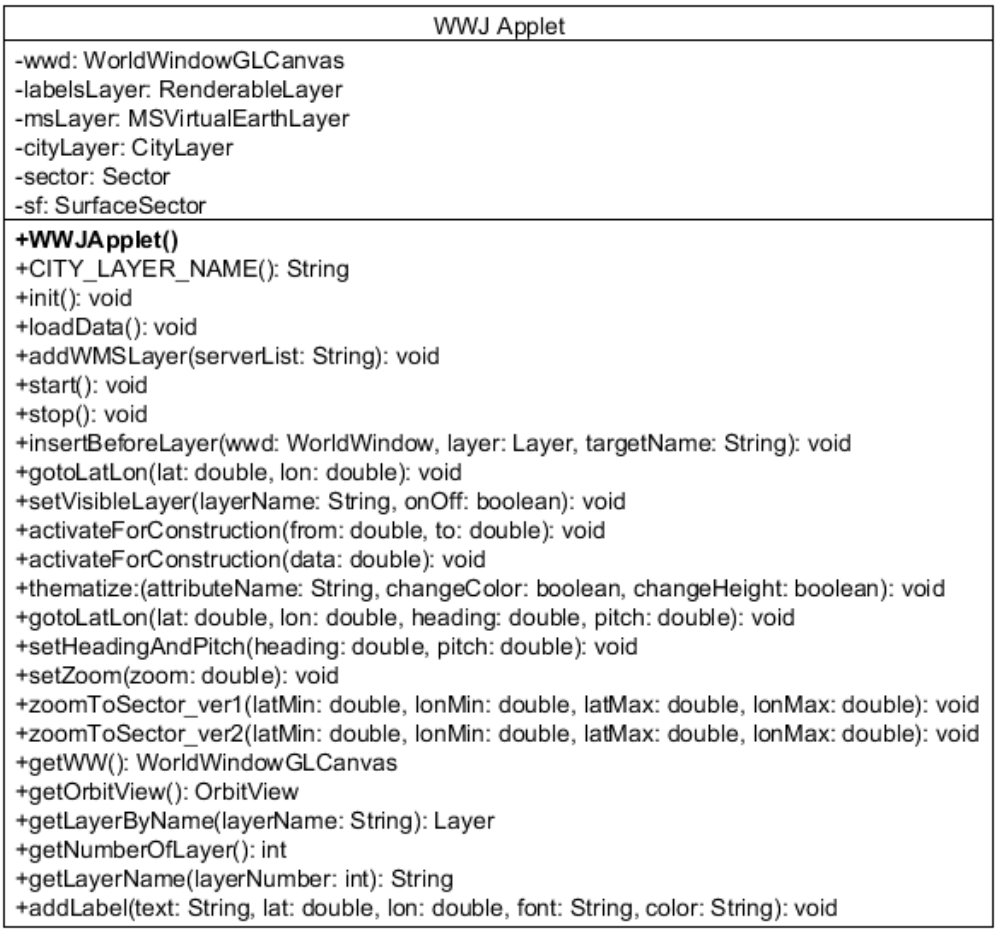


Figure F.1: World Wind Java applet UML diagram

```
1  package gov.nasa.worldwind.examples.applet;
2
3  import edu.PoliCo.CityViewer.*;
4
5  import gov.nasa.worldwind.layers.Earth.*;
6  import gov.nasa.worldwind.util.StatusBar;
7  import gov.nasa.worldwind.view.orbit.*;
8  import gov.nasa.worldwind.geom.Sector;
9  import gov.nasa.worldwind.geom.LatLon;
10 import gov.nasa.worldwind.geom.Extent;
11 import gov.nasa.worldwind.globes.Globe;
12 import netscape.javascript.JSObject;
13
```

```
14 import gov.nasa.worldwind.examples.*;
15 import gov.nasa.worldwind.awt.WorldWindowGLCanvas;
16 import gov.nasa.worldwind.geom.Angle;
17 import gov.nasa.worldwind.geom.Position;
18 import gov.nasa.worldwind.render.*;
19 import gov.nasa.worldwind.view.orbit.BasicOrbitView;
20 import gov.nasa.worldwind.*;
21 import gov.nasa.worldwind.avlist.*;
22 import gov.nasa.worldwind.layers.*;
23 import javax.swing.*;
24 import java.awt.*;
25 import java.net.*;
26
27 public class WWJApplet extends JApplet
28 {
29     private WorldWindowGLCanvas wwd;
30     private RenderableLayer labelsLayer;
31     private MSVirtualEarthLayer msLayer;
32     private CityLayer cityLayer;
33     private Sector sector;
34     private SurfaceSector sf;
35     public static final String CITY_LAYER_NAME = "City";
36
37     public WWJApplet()
38     {
39     }
40
41     @Override
42     public void init()
43     {
44         try
45         {
46             // Check for initial configuration values
47             String value = getParameter("InitialLatitude");
48             if (value != null)
49                     Configuration.setValue(AVKey.
                        INITIAL_LATITUDE, Double.parseDouble(
                        value));
50             value = getParameter("InitialLongitude");
51             if (value != null)
52                     Configuration.setValue(AVKey.
                        INITIAL_LONGITUDE, Double.parseDouble(
                        value));
53             value = getParameter("InitialAltitude");
54             if (value != null)
55                     Configuration.setValue(AVKey.
                        INITIAL_ALTITUDE, Double.parseDouble(
                        value));
56             value = getParameter("InitialHeading");
```

```
57              if (value != null)
58                      Configuration.setValue(AVKey.
                           INITIAL_HEADING, Double.parseDouble(
                           value));
59              value = getParameter("InitialPitch");
60              if (value != null)
61                      Configuration.setValue(AVKey.
                           INITIAL_PITCH, Double.parseDouble(
                           value));
62
63              // Create World Window GL Canvas
64              this.wwd = new WorldWindowGLCanvas();
65              this.getContentPane().add(this.wwd, BorderLayout.
                   CENTER);
66
67              // Create the default model as described in the
                   current worldwind properties.
68              Model m = (Model) WorldWind.
                   createConfigurationComponent(AVKey.
                   MODEL_CLASS_NAME);
69              this.wwd.setModel(m);
70
71              // Add a renderable layer for application labels
72              this.labelsLayer = new RenderableLayer();
73              this.labelsLayer.setName("Labels");
74              insertBeforeLayerName(this.wwd, this.labelsLayer,
                    "Compass");
75
76              this.msLayer = new MSVirtualEarthLayer();
77              this.msLayer.setName("MSLayer");
78              insertBeforeLayerName(this.wwd, this.msLayer, "
                   Compass");
79
80              this.cityLayer = new CityLayer();
81              this.cityLayer.setName(CITY_LAYER_NAME);
82              insertBeforeLayerName(this.wwd, this.cityLayer, "
                   Compass");
83              this.loadData();
84
85              sf = new SurfaceSector();
86              sector = Sector.fromDegrees(45,46,9,10);
87              sector = Sector.fromDegrees
                   (45.8064,45.8114,9.0801,9.0879);
88              sf.setSector(sector);
89              ShapeAttributes attributes = new
                   BasicShapeAttributes();
90              attributes.setDrawInterior(true);
91              attributes.setDrawOutline(true);
```

```
92              attributes.setInteriorMaterial(new Material(Color
                    .RED));
93              attributes.setOutlineMaterial(new Material(Color.
                    WHITE));
94              attributes.setInteriorOpacity(0.1);
95              attributes.setOutlineOpacity(1);
96              attributes.setOutlineWidth(3);
97              sf.setAttributes(attributes);
98              RenderableLayer lay = new gov.nasa.worldwind.
                    layers.RenderableLayer();
99              lay.addRenderable(sf);
100             lay.setName("Settore");
101             lay.setPickEnabled(false);
102             insertBeforeLayerName(this.wwd, lay, "Settore");
103
104             // Add the status bar
105             StatusBar statusBar = new StatusBar();
106             this.getContentPane().add(statusBar, BorderLayout
                    .PAGE_END);
107
108             // Forward events to the status bar to provide
                    the cursor position info.
109             statusBar.setEventSource(this.wwd);
110
111             // Setup a select listener for the worldmap click
                    -and-go feature
112             this.wwd.addSelectListener(new
                    ClickAndGoSelectListener(this.wwd,
                    WorldMapLayer.class));
113
114             // Call javascript appletInit()
115             try
116             {
117                 JSObject win = JSObject.getWindow(this);
118                 win.call("appletInit", null);
119             }
120             catch(Exception ignore) {}
121         }
122         catch (Throwable e)
123         {
124             e.printStackTrace();
125         }
126     }
127
128     public void loadData()
129     {
130         LoadGeomShpDBT loaderSHP = new LoadGeomShpDBT();
131         City city =loaderSHP.creaCityFromShape(getCodeBase().
                toString() + "/Mydata/config.dat");
```

```
132         this.cityLayer.setCity(city);
133         this.addWMSLayer(getCodeBase().toString() + "/Mydata/
                Server_List.dat");
134     }
135 
136         public void addWMSLayer(String serverList)
137         {
138             URL urlWMS=null;
139             LoadSaveAscii loader = new LoadSaveAscii();
140 
141             try
142             {
143                 urlWMS = new java.net.URL(serverList);
144             }
145             catch (java.net.MalformedURLException ex)
146             {}
147 
148             java.util.ArrayList<String[]> WMSLayerList =
                    loader.loadFromURL(urlWMS);
149             for (int c=0; c<WMSLayerList.size(); c++)
150             {
151                 String[] infoLayerWMS = WMSLayerList.get(c);
152                 Layer WMSLevel = WMSUtility.creaLayerWMS(
                        infoLayerWMS);
153                 insertBeforeLayerName(this.wwd, WMSLevel, "
                        Compass");
154             }
155         }
156 
157     @Override
158     public void start()
159     {
160         // Call javascript appletStart()
161         try
162         {
163             JSObject win = JSObject.getWindow(this);
164             win.call("appletStart", null);
165         }
166         catch(Exception ignore) {}
167     }
168 
169     @Override
170     public void stop()
171     {
172         // Call javascript appletSop()
173         try
174         {
175             JSObject win = JSObject.getWindow(this);
176             win.call("appletStop", null);
```

```
177         }
178         catch(Exception ignore) {}
179 
180         // Shut down World Wind
181         WorldWind.shutDown();
182     }
183 
184     /**
185      * Adds a layer to WW current layerlist, before a named
            layer.
186      * Target name can be a part of the layer name
187      * @param wwd the <code>WorldWindow</code> reference.
188      * @param layer the layer to be added.
189      * @param targetName the partial layer name to be matched
               - case sensitive.
190      */
191     public static void insertBeforeLayerName(WorldWindow wwd,
            Layer layer, String targetName)
192     {
193         // Insert the layer into the layer list just before
               the target layer.
194         LayerList layers = wwd.getModel().getLayers();
195         int targetPosition = layers.size() - 1;
196         for (Layer l : layers)
197         {
198             if (l.getName().indexOf(targetName) != -1)
199             {
200                 targetPosition = layers.indexOf(l);
201                 break;
202             }
203         }
204         layers.add(targetPosition, layer);
205     }
206 
207     // ============== Public API - Javascript
            ======================= //
208 
209     /**
210      * Move the current view position
211      * @param lat the target latitude in decimal degrees
212      * @param lon the target longitude in decimal degrees
213      */
214     public void gotoLatLon(double lat, double lon)
215     {
216         this.gotoLatLon(lat, lon, Double.NaN, 0, 0);
217     }
218 
219     public void setVisibleLayer(String layerName, boolean
            onOff)
```

```
220     {
221         Layer level = this.getLayerByName(layerName);
222         if (level!=null)
223         {
224             if (onOff)
225                 level.setEnabled(true);
226             else
227                 level.setEnabled(false);
228         }
229     }
230 
231     public void activateForConstruction(double from, double
            to)
232     {
233         this.cityLayer.getCity().
                activateBuildingForConstruction(from, to);
234         this.wwd.redraw();
235     }
236 
237     public void activateForConstruction(double data)
238     {
239         this.cityLayer.getCity().activateBuildingForExistence
                (data);
240         this.wwd.redraw();
241     }
242 
243     public void thematize(String attributeName, boolean
            changeColor, boolean changeHeight)
244     {
245         this.cityLayer.getCity().computeThematicValues(
                attributeName, changeColor, changeHeight);
246         this.wwd.redraw();
247     }
248 
249     /**
250      * Move the current view position, zoom, heading and
             pitch
251      * @param lat the target latitude in decimal degrees
252      * @param lon the target longitude in decimal degrees
253      * @param zoom the target eye distance in meters
254      * @param heading the target heading in decimal degrees
255      * @param pitch the target pitch in decimal degrees
256      */
257     public void gotoLatLon(double lat, double lon, double
            zoom, double heading, double pitch)
258     {
259         BasicOrbitView view = (BasicOrbitView)this.wwd.
                getView();
```

```
260         if(!Double.isNaN(lat) || !Double.isNaN(lon) || !
                Double.isNaN(zoom))
261         {
262             lat = Double.isNaN(lat) ? view.getCenterPosition
                    ().getLatitude().degrees : lat;
263             lon = Double.isNaN(lon) ? view.getCenterPosition
                    ().getLongitude().degrees : lon;
264             zoom = Double.isNaN(zoom) ? view.getZoom() : zoom
                    ;
265             heading = Double.isNaN(heading) ? view.getHeading
                    ().degrees : heading;
266             pitch = Double.isNaN(pitch) ? view.getPitch().
                    degrees : pitch;
267             view.addPanToAnimator(Position.fromDegrees(lat,
                    lon, 0),
268                     Angle.fromDegrees(heading), Angle.
                        fromDegrees(pitch), zoom, true);
269         }
270     }
271
272     /**
273      * Set the current view heading and pitch
274      * @param heading the traget heading in decimal degrees
275      * @param pitch the target pitch in decimal degrees
276      */
277     public void setHeadingAndPitch(double heading, double
            pitch)
278     {
279         BasicOrbitView view = (BasicOrbitView)this.wwd.
                getView();
280         if(!Double.isNaN(heading) || !Double.isNaN(pitch))
281         {
282             heading = Double.isNaN(heading) ? view.getHeading
                    ().degrees : heading;
283             pitch = Double.isNaN(pitch) ? view.getPitch().
                    degrees : pitch;
284
285             view.addHeadingPitchAnimator(
286                 view.getHeading(), Angle.fromDegrees(heading)
                        , view.getPitch(), Angle.fromDegrees(pitch
                        ));
287         }
288     }
289
290     /**
291      * Set the current view zoom
292      * @param zoom the target eye distance in meters
293      */
294     public void setZoom(double zoom)
```

```
295     {
296         BasicOrbitView view = (BasicOrbitView)this.wwd.
                getView();
297         if(!Double.isNaN(zoom))
298         {
299             view.addZoomAnimator(view.getZoom(), zoom);
300         }
301     }
302 
303     public void zoomToSector_Ver1(double latMin, double
            lonMin, double latMax, double lonMax)
304     {
305         sector = Sector.fromDegrees(latMin,latMax,lonMin,
                lonMax);
306         sf.setSector(sector);
307         BasicOrbitView view = (BasicOrbitView) this.wwd.
                getView();
308         SceneController sc=this.wwd.getSceneController();
309         LatLon center = sector.getCentroid();
310         Position position = new Position(center,0);
311         Globe globe = view.getGlobe();
312         double ve = sc.getVerticalExaggeration();
313         double[] minAndMaxElevations = globe.
                getMinAndMaxElevations(sector);
314         double minElevation = minAndMaxElevations[0];
315         double maxElevation = minAndMaxElevations[1];
316         Extent extent = Sector.computeBoundingCylinder(globe,
                ve, sector, minElevation, minElevation);
317         Angle fov = view.getFieldOfView();
318         double zoom = extent.getRadius() / (fov.tanHalfAngle
                () * fov.cosHalfAngle());
319         zoom = extent.getRadius() / (fov.tanHalfAngle());
320         view.goTo(position, zoom);
321     }
322 
323     public void zoomToSector_Ver2(double latMin, double
            lonMin, double latMax, double lonMax)
324     {
325         sector = Sector.fromDegrees(latMin,latMax,lonMin,
                lonMax);
326         sf.setSector(sector);
327         BasicOrbitView view = (BasicOrbitView) this.wwd.
                getView();
328         double delta_x = sector.getDeltaLonRadians();
329         double delta_y = sector.getDeltaLatRadians();
330         double earthRadius = this.wwd.getModel().getGlobe().
                getRadius();
331         double horizDistance = earthRadius * delta_x;
332         double vertDistance = earthRadius * delta_y;
```

```
333         double distance = Math.max(horizDistance,
                vertDistance) / 2;
334         double altitude = distance / Math.tan(view.
                getFieldOfView().radians / 2);
335         LatLon latlon = sector.getCentroid();
336         Position pos = new Position(latlon, altitude);
337         view.goTo(pos, altitude);
338     }
339
340     /**
341      * Get the WorldWindowGLCanvas
342      * @return the current WorldWindowGLCanvas
343      */
344     public WorldWindowGLCanvas getWW()
345     {
346         return this.wwd;
347     }
348
349     /**
350      * Get the current OrbitView
351      * @return the current OrbitView
352      */
353     public OrbitView getOrbitView()
354     {
355         if(this.wwd.getView() instanceof OrbitView)
356             return (OrbitView)this.wwd.getView();
357         return null;
358     }
359
360     /**
361      * Get a reference to a layer with part of its name
362      * @param layerName part of the layer name to match.
363      * @return the corresponding layer or null if not found.
364      */
365     public Layer getLayerByName(String layerName)
366     {
367         for (Layer layer : wwd.getModel().getLayers())
368             if (layer.getName().indexOf(layerName) != -1)
369                 return layer;
370         return null;
371     }
372
373     public int getNumberOfLayer()
374     {
375         return wwd.getModel().getLayers().size();
376     }
377
378     public String getLayerName(int layerNumber)
379     {
```

```
380         String levelName="";
381         if (layerNumber<this.getNumberOfLayer())
382         {
383             Layer level = this.wwd.getModel().getLayers().get
                (layerNumber);
384             levelName = level.getName();
385         }
386         return levelName;
387     }
388 
389     /**
390      * Add a text label at a position on the globe.
391      * @param text the text to be displayed.
392      * @param lat the latitude in decimal degrees.
393      * @param lon the longitude in decimal degrees.
394      * @param font a string describing the font to be used.
395      * @param color the color to be used as an hexadecimal
            coded string.
396      */
397     public void addLabel(String text, double lat, double lon,
            String font, String color)
398     {
399         GlobeAnnotation ga = new GlobeAnnotation(text,
                Position.fromDegrees(lat, lon, 0),
400              Font.decode(font), Color.decode(color));
401         ga.getAttributes().setBackgroundColor(Color.BLACK);
402         ga.getAttributes().setDrawOffset(new Point(0, 0));
403         ga.getAttributes().setFrameShape(FrameFactory.
                SHAPE_NONE);
404         ga.getAttributes().setEffect(AVKey.
                TEXT_EFFECT_OUTLINE);
405         ga.getAttributes().setTextAlign(AVKey.CENTER);
406         this.labelsLayer.addRenderable(ga);
407     }
408 }
```

Listing F.1: The 'WWJApplet.java' code

The City layer is created from the shapefile called at line 131 using a 'config.dat' file, hereafter listed:

```
 # directory server http:
http://myhost/WWJ_Applet/WorldWind_Library/Mydata/

# shapefile name:
comoSt2_nn.shp

# orthometric height of building bases:
```

```
215

# building height attribute:
ALTEZZA

# construction date of the building attribute:
DATE_OF_C

# demolition date of the building attribute:
DATE_OF_D

# min value NODATA (to indicate a non-definite construction date):
1000

# max value NODATA (to indicate a non-definite demolition date -
no demolition):
9999

# number of classes N for colour thematization (min <= value < max):
6

# definition of interval:  min; max; red; green; blue;
0;1600;196; 196; 196
1600;1760;2; 2; 255
1760;1860;18; 255; 87
1860;1935; 221; 255; 8
1935; 1969;255; 104; 8
1969;9999;255; 0; 0
```

To add a WMS layer on the globe, as early maps, it is necessary to add their URL in the 'Server_List.dat' file (at line 133):

```
#;name;;;;;WMS_url;;;;;;layer_name;;;;;;style_name(optional)

Carta Catasto Terreni (1905);http://myhost/mywms?;map_1905
Carta Catasto Teresiano (1722);http://myhost/mywms?;teresiano
Carta Catasto Lombardo - Veneto (1858);http://myhost/mywms?;lv_1858
Update Carta Catasto Lombardo - Veneto (1873);http://myhost/mywms?;lv_1873
```

To define a city layer, two Java classes have been implemented: 'City.java' and 'CityLayer.java'. City is the container of the city buildings ('Builds'),

associated to the geometries and attributes of the shapefile. CityLayer, that extends 'AirspaceLayer', permits to build geometries as layers on the globe.

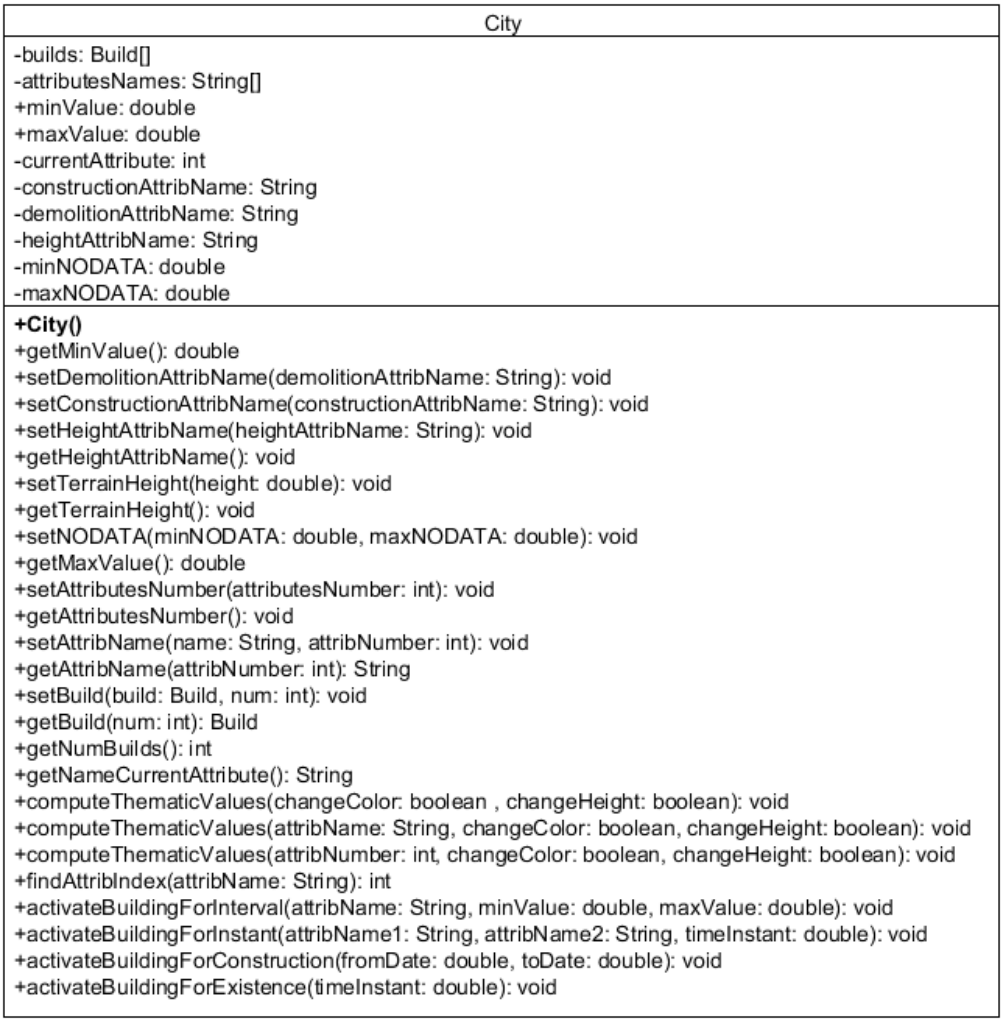


Figure F.2: City class UML diagram

In particular, with the City class it is possible to build a 'city' with the desired number of 'builds' and assigning attributes ('attribNames'). The most relevant methods for our purpose are:

- *set/getTerrainHeight*: to assign or get the terrain mean height beneath buildings (on the sea level);

- *setConstruction/Demolition/HeightAttribName*: to set attributes of date of construction, date of demolition and building relative height;
- *computeThematicValues*: to calculate values to assign to different thematism;
- *activateBuildingForInterval* and *activateBuildingForConstruction*: to activate buildings according to the construction time span chosen by the user;
- *activateBuildingForInstant* and *activateBuildingForExistence*: to activate buildings according to the existence in the year chosen by the user.

```
1  package edu.PoliCo.CityViewer;
2  import java.awt.Color;
3
4  public class City
5  {
6      private Build[] builds;
7      // attribute names
8      private String[] attribNames;
9      public double minValue=Double.POSITIVE_INFINITY;
10     public double maxValue=Double.NEGATIVE_INFINITY;
11     private int currentAttribute=0;
12     // height terrain
13     private double terreinHeight=200;
14     // construction date attibute
15     private String constructionAttribName;
16     // demolition date attibute
17     private String demolitionAttribName;
18     // height attibute
19     private String heightAttribName;
20     private double minNODATA;
21     private double maxNODATA;
22
23     public City(int numBuilds)
24     {
25         this.builds=new Build[numBuilds];
26         this.attribNames=new String[1];
27     }
28
29     public double getMinValue()
30     {
31         return this.minValue;
32     }
33     // get terrain height
```

```
34      public double getTerrainHeight()
35      {
36          return this.terreinHeight;
37      }
38      // set terrain height
39      public void setTerrainHeight(double height)
40      {
41          terreinHeight=height;
42      }
43
44      public void setConstructionAttribName(String constrAttrib
            )
45      {
46          this.constructionAttribName=constrAttrib;
47      }
48
49      public void setDemolitionAttribName(String demolAttrib)
50      {
51          this.demolitionAttribName=demolAttrib;
52      }
53
54      public void setHeightAttribName(String heightAttrib)
55      {
56          this.heightAttribName=heightAttrib;
57      }
58
59      public void setNODATA(double minNODATA, double maxNODATA)
60      {
61          this.minNODATA=minNODATA;
62          this.maxNODATA=maxNODATA;
63      }
64
65      public double getMaxValue()
66      {
67          return this.maxValue;
68      }
69
70      public void setAttributesNumber(int attributesNumber)
71      {
72          this.attribNames=new String[attributesNumber];
73      }
74
75      public int getAttributesNumber()
76      {
77          return this.attribNames.length;
78      }
79
80      public void setAttribName(String name, int attribNumber)
81      {
```

```
82          this.attribNames[attribNumber]=name;
83      }
84
85      public String getAttribName(int attribNumber)
86      {
87          return this.attribNames[attribNumber];
88      }
89
90      public void setBuild(Build build, int num)
91      {
92          this.builds[num]=build;
93      }
94
95      public Build getBuild(int num)
96      {
97          return this.builds[num];
98      }
99
100     public int getNumBuilds()
101     {
102         return this.builds.length;
103     }
104
105     public String getNameCurrentAttribute()
106     {
107         return this.attribNames[this.currentAttribute];
108     }
109
110     public void computeThematicValues(boolean changeColor,
            boolean changeHeight)
111     {
112         this.computeThematicValues(this.currentAttribute,
                changeColor, changeHeight);
113     }
114
115     public void computeThematicValues(String attribName,
            boolean changeColor, boolean changeHeight)
116     {
117         int attribIndex = this.findAttribIndex(attribName);
118         if (attribIndex!=-1)
119             this.computeThematicValues(attribIndex,
                    changeColor, changeHeight);
120     }
121     // compute thematic values
122     public void computeThematicValues(int attribNumber,
            boolean changeColor, boolean changeHeight)
123     {
124         Build building;
125         Double attribute;
```

```
126         this.currentAttribute=attribNumber;
127         this.minValue=Double.POSITIVE_INFINITY;
128         this.maxValue=Double.NEGATIVE_INFINITY;
129         int heightAttributeNumber = this.findAttribIndex(this
                .heightAttribName);
130         for (int c=0; c<this.builds.length; c++)
131         {
132             building = (Build)this.builds[c];
133             attribute = building.getAttribValueAsNumber(
                    attribNumber);
134             if (attribute !=null)
135             {
136                 if (attribute!=minNODATA && attribute !=
                        maxNODATA)
137                 {
138                     if (attribute<minValue)
139                         minValue=attribute;
140                     if (attribute>maxValue)
141                         maxValue=attribute;
142                 }
143             }
144         }
145         if (minValue==Double.POSITIVE_INFINITY && maxValue==
                Double.NEGATIVE_INFINITY)
146         {
147             minValue=maxNODATA;
148             maxValue=maxNODATA;
149         }
150         double factor= 1d/(maxValue-minValue);
151         BuildColor bc= new BuildColor(minValue,maxValue);
152         double percentage=0;
153         double height=0;
154         for (int c=0; c<this.builds.length; c++)
155         {
156             building = (Build)this.builds[c];
157             attribute =  building.getAttribValueAsNumber(
                    attribNumber);
158             building.setCurrentValue(attribNumber);
159             if (heightAttributeNumber!=-1)
160                 height = building.getAttribValueAsNumber(
                        heightAttributeNumber);
161             else
162                 height = 100;
163             if (attribute !=null)
164             {
165                 if (attribute!=minNODATA && attribute !=
                        maxNODATA)
166                 {
```

```
167                     percentage = (attribute-minValue)*factor
                            * 100;
168                     if (changeHeight && minValue!=maxValue)
169                         building.setAltitudes(terreinHeight,
                                terreinHeight+20+percentage);
170                     else
171                         building.setAltitudes(terreinHeight,
                                terreinHeight+height);
172
173                     if (changeColor)
174                         building.setColor(bc.getColore(
                                attribute));
175                     else
176                         building.setColor(java.awt.Color.
                                WHITE);
177                 }
178                 else
179                 {
180                     Color noDataColor=null;
181                     if (attribute==minNODATA)
182                     {
183                         percentage=0;
184                         noDataColor=Color.lightGray;
185                     }
186                     else
187                     if (attribute==maxNODATA)
188                     {
189                         percentage=100;
190                         noDataColor=Color.darkGray;
191                     }
192                     if (changeHeight && minValue!=maxValue)
193                         building.setAltitudes(terreinHeight,
                                terreinHeight+20+percentage);
194                     else
195                         building.setAltitudes(terreinHeight,
                                terreinHeight+height);
196
197                     if (changeColor)
198                         building.setColor(noDataColor);
199                     else
200                         building.setColor(java.awt.Color.
                                WHITE);
201                 }
202             }
203             else
204             {
205               if (changeHeight)
206                     building.setAltitudes(terreinHeight,
                            terreinHeight+100);
```

```
207                 else
208                       building.setAltitudes(terreinHeight,
                             terreinHeight+height);
209                 building.setColor(java.awt.Color.WHITE);
210             }
211         }
212     }
213     // find attibute index
214     public int findAttribIndex(String attribName)
215     {
216         int attribNumber=-1;
217         for (int c=0; c<this.attribNames.length; c++)
218         {
219             if (this.attribNames[c].equals(attribName))
220             {
221                 attribNumber=c;
222             }
223         }
224         return attribNumber;
225     }
226     // activate buildings according to the chosen interval
227     public void activateBuildingForInterval(String attribName
        , double minValue, double maxValue)
228     {
229         Build building;
230         double attribute;
231         int attribNumber = this.findAttribIndex(attribName);
232         if (attribNumber!=-1)
233         {
234             for (int c=0; c<this.builds.length; c++)
235             {
236                 building = (Build)this.builds[c];
237                 attribute = building.getAttribValueAsNumber(
                        attribNumber);
238                 if ((attribute>=minValue) && (attribute<=
                        maxValue))
239                     building.setVisible(true);
240                 else
241                     building.setVisible(false);
242             }
243         }
244     }
245     // activate buildings according to the chosen year
246     public void activateBuildingForInstant(String attribName1
        , String attribName2, double timeInstant)
247     {
248         Build building;
249         double attribValue1;
250         double attribValue2;
```

```
251         int attribNumber1 = this.findAttribIndex(attribName1)
                ;
252         int attribNumber2 = this.findAttribIndex(attribName2)
                ;
253         if (attribNumber1!=-1 && attribNumber2!=-1)
254         {
255             for (int c=0; c<this.builds.length; c++)
256             {
257                 building = (Build)this.builds[c];
258                 attribValue1 = building.
                        getAttribValueAsNumber(attribNumber1);
259                 attribValue2 = building.
                        getAttribValueAsNumber(attribNumber2);
260                 if ( (attribValue1 <= timeInstant) && (
                        attribValue2>= timeInstant) )
261                     building.setVisible(true);
262                 else
263                     building.setVisible(false);
264             }
265         }
266     }
267
268     public void activateBuildingForConstruction(double
            fromDate, double toDate)
269     {
270         this.activateBuildingForInterval(this.
                constructionAttribName, fromDate, toDate);
271     }
272
273     public void activateBuildingForExistence(double
            timeInstant)
274     {
275         this.activateBuildingForInstant(
                constructionAttribName, demolitionAttribName,
                timeInstant);
276     }
277 }
```

Listing F.2: The ‘City.java’ code

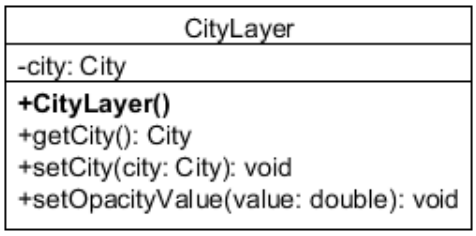


Figure F.3: CityLayer class UML diagram

Finally, the 'CityLayer' class is used to render the 'City' and eventually change layer opacity ('setOpacityValue' method).

```
1  package edu.PoliCo.CityViewer;
2  import gov.nasa.worldwind.layers.AirspaceLayer;
3
4  public class CityLayer extends AirspaceLayer
5  {
6      private City city;
7
8      public CityLayer()
9    {
10         super();
11     }
12     // get city layer
13     public City getCity()
14     {
15         return this.city;
16     }
17     // set city layer
18     public void setCity(City city)
19     {
20         Build build;
21         this.city=city;
22
23         for (int cont=0; cont<city.getNumBuilds(); cont++)
24         {
25             build = city.getBuild(cont);
26             if (build!=null)
27                 this.addAirspace(build);
28         }
29     }
30     // set opacity
31     public void setOpacityValue(double value)
32     {
```

```
33          this.setOpacity(value);
34          Build build;
35          for (int cont=0; cont<city.getNumBuilds(); cont++)
36          {
37              build = city.getBuild(cont);
38              if (build!=null)
39                  build.setOpacityValue(value/10);
40          }
41      }
42  }
```

Listing F.3: The 'CityLayer.java' code