Deusto

**Facultad de Ingeniería**
Universidad de Deusto

**Ingeniaritza Fakultatea**
Deustuko Unibertsitatea

# Ingeniero en Informática
# Informatikako Ingeniaria

## Proyecto fin de carrera
## Karrera amaierako proiektua

# Asbtract

The goal of this project is the combination of semantic, web and geospatial technologies for the development of and on-line GPS community.

A web application will be developed using NodeJS, a server side JavaScript programming platform, and AngularJS a single page application building framework based on the model-view-controller pattern. This system will allow users to share trails and points of interest which will be structured according to a ontology developed for this system and stored on a triplestore.

The semantic nature of the data will allow the users for a more intelligent and precise search, beyond the classic key word search. Besides, with the use of GeoSPARQL, a standard for the representation and querying of geospatial linked data the implicit relations among this data, such as which points of interest surround a trail, can be obtained.

The web application will allow the users to share their trails and enrich their information. Besides, a mobile application will give support to the system allowing the users to follow the trails of others, discovering information near them in real time and allowing them to post location aware notes on their actual location.

# Descriptores

Semantic Web, GeoSPARQL, Linked Data, NodeJS, AngularJS.

# Contents

**1 Introduction** **1**

1.1 Motivation . . . . . 1
1.2 State of the art . . . . . 2
1.2.1 The Semantic Web . . . . . 2
1.2.2 Linked Open Data . . . . . 4
1.2.3 Introduction to GIS systems . . . . . 5

**2 Project Goals** **7**

2.1 Overview . . . . . 7
2.2 Project definition . . . . . 7
2.2.1 Goals . . . . . 7
2.2.2 Scope of the project . . . . . 8
2.3 Final product . . . . . 8
2.3.1 Server . . . . . 9
2.3.2 Web application . . . . . 10
2.3.3 Mobile application . . . . . 11
2.4 Project realization . . . . . 11
2.4.1 Realization methodology . . . . . 11
2.4.2 Intermediate products . . . . . 15
2.5 Organization and team . . . . . 15
2.5.1 Organizational structure . . . . . 15
2.5.2 Human resources plan . . . . . 16
2.6 Execution conditions . . . . . 17
2.6.1 Hardware . . . . . 18
2.6.2 Software . . . . . 18
2.6.3 Change control . . . . . 19
2.6.4 Product reception . . . . . 19
2.7 Planning . . . . . 20
2.8 Project budget . . . . . 25

**3 Technological Research** **27**

3.1 Overview . . . 27
3.2 RDF . . . 27
3.3 OWL . . . 32
3.4 SPARQL . . . 35
3.5 GeoSPARQL . . . 36
3.6 NodeJS . . . 38
3.7 Leaflet . . . 43
3.8 Phonegap . . . 45
3.9 GPX . . . 46
3.10 GeoJSON . . . 47

**4 Requirements Specification** **51**

4.1 Overview . . . 51
4.2 Requirements for the ontology . . . 51
4.3 Requirements specification for the server . . . 53
4.4 Requirements of the web application . . . 54
4.5 Requirements of the mobile application . . . 56
4.6 Non-functional requirements . . . 57

**5 Design specification** **59**

5.1 Overview . . . 59
5.2 System architecture . . . 59
5.2.1 Central server . . . 59
5.2.2 Semantic spatial storage system . . . 61
5.2.3 Web client . . . 61
5.2.4 Mobile application . . . 61
5.3 Design of the ontology . . . 61
5.3.1 Class hierarchy . . . 61
5.3.2 Properties . . . 63
5.3.3 Advantages of ontology based design . . . 67
5.4 Development environment . . . 69
5.4.1 Protégé-OWL . . . 70
5.4.2 UML . . . 70
5.4.3 SASS . . . 71
5.4.4 Twitter Bootstrap . . . 71

**6 Implementation details** **73**

**7 Testing** **75**

**8 User guide** **77**

**9 Conclusions and future lines of work** **79**

**Bibliography** **81**

# List of Figures

**Capítulo 1**


1.1 The semantic web architecture . . . . . 3
1.2 The Linked Data cloud . . . . . 5


**Capítulo 2**


2.1 Iterative development phases . . . . . 11
2.2 Work breakdown structure . . . . . 14
2.3 Organizational structure . . . . . 16
2.4 Gantt diagram . . . . . 23
2.5 Network diagram . . . . . 24


**Capítulo 3**


3.1 An RDF graph describing a person . . . . . 28
3.2 The OWL levels of expressivity . . . . . 32
3.3 View-controller binding through the scope . . . . . 41
3.4 Leaflet with OSM mapnik tile layer . . . . . 43
3.5 Phonegap Build process . . . . . 46


**Capítulo 5**


5.1 Architecture design of the system . . . . . 60
5.2 The SORELCOM ontology . . . . . 62
5.3 Inference of the husband relation as inverse of the wife relation . . . . . 68
5.4 Inference of all the classes of a Trail . . . . . 68
5.5 Inference of the husband relation as inverse of the wife relation . . . . . 69
5.6 The Protégé OWL ontology editor . . . . . 70
5.7 SASS to CSS transformation . . . . . 71

# List of Tables

### Capítulo 2


2.1 Intermediate products of the project. . . . . 15
2.2 Workload estimation for the project manager. . . . . 20
2.3 Workload estimation for the system architect. . . . . 20
2.4 Workload estimation for the researcher. . . . . 20
2.5 Workload estimation for the back-end developer. . . . . 21
2.6 Workload estimation for the front-end web developer. . . . . 21
2.7 Workload estimation for the mobile application developer. . . . . 22
2.8 Workload estimation for the graphics designer. . . . . 22
2.9 Real work plan. . . . . 22
2.10 Hardware related budget. . . . . 25
2.11 Human resources budget. . . . . 25
2.12 Summary of budget. . . . . 25


### Capítulo 5


5.1 Trails on the SORELCOM data model. . . . . 64
5.2 Points of interest on the SORELCOM data model. . . . . 64
5.3 Geolocated notes on the SORELCOM data model. . . . . 65
5.4 Persons on the SORELCOM data model. . . . . 65
5.5 Media on the SORELCOM data model. . . . . 66
5.6 Inferences used on the SORELCOM ontology classes. . . . . 66
5.7 Inferences used on the SORELCOM ontology classes. . . . . 67

# Listings index

### Capítulo 1

1.1 Set of triples in pseudocode representing Bob. . . . . 4

### Capítulo 3

3.1 Different types of triple statements. . . . . 29
3.2 Triple statements with a blank node. . . . . 29
3.3 RDFS class and property hierarchy. . . . . 31
3.4 OWL property characteristic example. . . . . 33
3.5 OWL property restriction example. . . . . 34
3.6 OWL property restriction example. . . . . 35
3.7 SPARQL query for retrieving all triples on the dataset. . . . . 36
3.8 SPARQL query for retrieving names on the dataset. . . . . 36
3.9 A feature in the GeoSPARQL vocabulary. . . . . 37
3.10 Spatial query in SPARQL. . . . . 38
3.11 NodeJS ”hello world” web server. . . . . 39
3.12 Express ”hello world” program. . . . . 40
3.13 AngularJS example. . . . . 41
3.14 AngularJS example. . . . . 43
3.15 Leaflet layers. . . . . 44
3.16 Leaflet layers. . . . . 44
3.17 Reading battery status with Phonegap. . . . . 45
3.18 GPX way point representation. . . . . 47
3.19 GPX track representation. . . . . 47
3.20 GPX route representation. . . . . 47
3.21 A GeoJSON `GeometryCollection` object. . . . . 48
3.22 A GeoJSON `Feature` object. . . . . 49

# 1. INTRODUCTION

The goal of this report is to describe the process of design and implementation of the Final Undergraduate Project *"SORELCOM: Design and Development of a semantic geospatial web and mobile application"*.

This project has been carried by Aimar Rodríguez Soto, 4th year student of an undergraduate degree in computer science engineering.

The different chapters that form this document are the following:

1. **Introduction:** Brief descriptions of the contents of the report, as well as a small introduction to the concepts of Semantic Web and GIS System.
2. **Project goals:** The goals, the limits, the tasks to perform, working methodologies and the working plan are presented.
3. **Technological research:** Detailed description of the different technologies used throughout the project, thus presenting the results of the technological research.
4. **Requirement Specification:** Requirements of the project and the final product are listed and analyzed.
5. **Design specification:** Design considerations of the project are described, both from the technological and the user experience point of views.
6. **Development:** Usage of the described technologies, solutions to the common problems and details about the implementation are given.
7. **Testing:** Testing plan, tools, methodologies and special test cases are described.
8. **User guide:** Document explaining how to use the developed tools and platforms at a user level.
9. **Conclusions and future lines of work:** Analysis of the results of the project and further considerations, such as improvements and future work to be done.

## 1.1. MOTIVATION

Geospatial data is becoming more and more relevant in our world. In roughly 30 years, we have evolved from a region based spatial information, to human and device centric information. The amount of location-aware devices has multiplied and with this, location-based services are thriving. Users can retrieve directions and information based on their current location, the number of social media users which set their devices to include location information on their phones is growing. There is no doubt that the amount of spatial data is increasing and that it is making an impact in our societies.

As a consequence, the amount of Linked Open Data with spatial context made available on the web has also increased. The principles of Linked Open Data[6] encourage the usage of RDF and

SPARQL to query and model data. This is useful for querying for the explicit relations on the datasets, however, the implicit spatial relationships cannot be easily queried.

To fight this issue, the Open Geospatial Consortium(OGC) has defined a standard, named GeoSPARQL to address the issues of spatial information representation and querying on the semantic web. Thanks to this, it is possible to include a spatial factor in the reasoning done over RDF datasets.

The platform SORELCOM (Social Routes Empowered by Linked Contents and Context Mining) aims to use GeoSPARQL and data from over the web to create a user-centred GPS community which is able to offer location aware service to users and to publish the data generated in a spatially aware linked dataset.

The platform aims to combine semantic and spatial technologies to create a tool to capture routes and enrich their information from sources around the web, providing in addition a powerful location and preference based recommendation service.

## 1.2. STATE OF THE ART

This section introduces the main fields related to the projects and gives a small insight on the current state and work done on this fields.

### 1.2.1 The Semantic Web

The semantic web is a extension of the current web in which information is given well-defined meaning, better enabling computers and people to work in cooperation. [5]

The idea behind the semantic web is to bring meaning and structure to the contents of web pages, creating an environment where software agents can roam the web carrying out sophisticated tasks for end users. Thanks to the semantic web, this agents will be able to understand the relationships among the information on a web page, beyond simply identifying certain keywords.

This machine-readable web of data is based standard languages which allow a uniform representation of the information throughout the different semantic data sources. Through this common infrastructure it is possible to share and process information in a relatively simple way, enabling better solution to the usual problem of searching through the web of information. In short, the Semantic Web attempts to give meaning and structure to the contents on the web, allowing computers to better understand this information to do more useful work.

#### The fundamentals

The Semantic Web is based on standard formats used for a universal representation and manipulation of data. This standards are the following:

**RDF:** The Resource Description Framework (RDF) is a standard language for representing information about resources in the World Wide Web. Resources can be anything, including documents, people, physical object and abstract concepts. [34]

**OWL:** The Web Ontology Language (OWL) is a semantic markup language for creating and sharing ontologies [3]. It is used to define ontologies when the content needs to be processed by applications or agents, instead of just presented to people [35].

SPARQL: the SPARQL Protocol and RDF Query Language is a language designed to make queries over resources in RDF format [43].

The idea of the semantic web is based on a stack architecture, illustrated on figure 1.1. This stack includes more layers than the mentiones, such as RDF Schema (RDFS) and the eXtensible Markup Language (XML). For further detail into these technologies and their role in the semantic web refer to chapter 3.

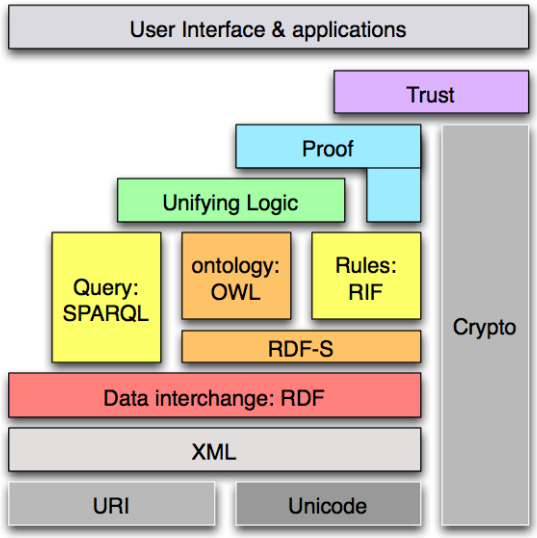


Figure 1.1: The semantic web architecture

*source: http://www.w3.org/2006/Talks/1023-sb-W3CTechSemWeb/#(19)*

### Introduction to RDF

RDF provides a framework for representing information which can be exchanged between application without loss of meaning and it is intended for situation in which this information needs to be processed by machines instead of displayed to humans, thus making it fitting for the goals of The Semantic Web [54].

This framework has several uses, such as:

- Adding machine readable information to web pages, enabling them to be displayed in enhanced formats or to be processed by other agents.
- Enriching a database by linking it to third party datasets.
- Interlinking API feeds, so that clients can easily discover how to access more information.
- Using datasets labeled as Linked Data to build aggregations of data around specific topics or concepts.

RDF information is stored in triples which follow a simple subject-predicate-object pattern. One example of RDF triples which represent knowledge about a person named Bob can be found in

listing 1.1. Note that in this set of triples, most attributes appear between brackets, these are International Resource Identifiers (IRIs) linking to other resources, while the other attributes are just literal values.

```
1 Bob> <is_named> "Bob"
2 Bob> <is_a> <Person>
3 Bob> <is_a> <Student>
4 Bob> <knows> <Alice>
5 Bob> <is_interested_in> <Computer_Science>
```

Listing 1.1: Set of triples in pseudocode representing Bob.

These IRIs are identifiers for resources on the Semantic Web. Up to now the concepts of URI and IRI have been mixed, to make it clear, an IRI is a generalization of an URI. The form of a IRI is similar to that of the Uniform Resource Locators (URL) used in the World Wide Web, in fact these last are just a form of IRI, thus they look like the following: `http://dbpedia.org/resource/Leonardo_da_Vinci`

#### Introduction to Ontologies

In the Semantic Web, vocabularies or ontologies define the concepts and relationships used to describe and represent a certain area of knowledge, they define the terms that can be used on a application, the possible relationships and their constraints. They are used to help data integration when ambiguity may exist between terms in different datasets, or simply to organize knowledge [53].

According to Tom Grubber [28] An ontology is a formal specification of a shared conceptualization. The concept of Ontology comes from the field of philosophy, however they have been adopted into Artificial Intelligence and Knowledge Representation [15]. Since ontologies are used to create conceptual structure on a domain, they can be used by software for problem solving and reasoning among other tasks.

We can find three types of ontologies:

**Top-level ontologies** : Describe very general notions which are independant of a particular domain and are applicable to different areas, for example; time, space, events, etc.

**Domain ontologies** : The knowledge represented in these ontologies is particular to a certain field of knowledge, such as computers, persons, forestry, etc. They provide information about concepts in the domain and about the theories governing the domain.

**Application ontologies** : This is the least general type of ontology. It describes knowledge specific to a certain a application or task, due to this, they are useful for problem solving.

### 1.2.2 Linked Open Data

Linked Data refers to a pattern for interlinking machine-readable data sets to each other, specially via the use of RDF and URIs. In summary, Linked Data is about using the web to create typed links between data from different sources [7].

Linked data relies on technologies that are fundamental to the web, mainly Uniform Resource Identifiers (URIs) and the Hypertext Transfer Protocol (HTTP). URIs are used as a unique identifier across the web to the resources on the Linked Data sets and are looked up using the HTTP protocol.

These technologies are supported by a technlogy that is crucial for the web of data - RDF. RDF and the ontology definition language OWL can be used to create vocabularies used to describe concepts and entities and how they are related.

This idea's more visible adoption is the Linking Open Data project [50] whose aim is to bootstrap the Web of Data by identifying existing data sets that are available under open licenses, converting these to RDF according to the Linked Data principles, and publishing them on the Web. An cloud representing the already linked data sets can be found in figure 1.2.

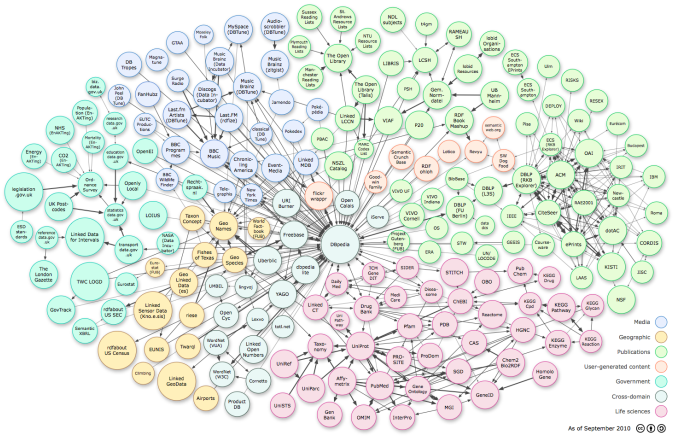


Figure 1.2: The Linked Data cloud

***source: http://linkeddatabook.com/editions/1.0/***

### 1.2.3 Introduction to GIS systems

A Geographical Information System (GIS) integrates hardware, software and data for capturing, managing and displaying all forms of geographically referenced information [22] A GIS provides a framework for gathering and organizing spatial data and related information so it can be displayed and analyzed.

Data on such a system is a digital representation of real world objects. Since this kind of data can be divided into two abstractions, discrete objects (a house, a park) or continuous fields (elevation), two methods are used to store data of this abstractions: Raster and Vector. [26]

#### Raster

Raster data types consist of rows and columns of cells each of them storing a single value, in the say way that a image is composed by pixels. In fact, a Raster data type is, in essence, a type of digital image represented in grids. On a Raster Image each pixels stores a color value which represents certain data, for example, the amount of rain in that region.

This kind of data may be stored in different ways, for example, as a regular .JPEG image or directly in a relational database.

### Vector

Vector data is used to express geographical features, which are represented as geometrical shapes of different types.

- *Points:* They are used to describe features that can be represented with a single point, that is, the simplest of features, for example, the location of a monument, the peak of a mountain, etc.
- *Lines or polylines:* They are used to represent linear features, such as rivers, roads, trails, etc. It is possible to measure distance on line features.
- *Polygons:* They are used for the representation of features that cover a particular area of the earth's surface, for example, a country. It is possible to measure area and perimeter of polygons.

# 2. PROJECT GOALS

## 2.1. OVERVIEW

An overview of the project goal definition document is presented below. The different sections which compose it are the following:

- **Project definition:** Statement of the goals of the project, specification of its functional aspects and scope.
- **Final product:** Description of the products that are to be developed on the project.
- **Project realization:** Definition of the different activities to be carried out in order to fulfill the goals of the project.
- **Organization and team:** Description of the Work Team that will carry out the project, as well as its organizational structure and human resources plan.
- **Execution conditions:** Project related work conditions, definition of the criteria over which product reception as well as the Change Control.
- **Planning:** Establishing of the phases and tasks needed to carry the project, together with orientative dates for their realization.
- **Project budget:** Determination of the budget of the project and the associate costs.

## 2.2. PROJECT DEFINITION

### 2.2.1 Goals

The main goal of this project is the combination of geospatial and semantic technologies to build a location aware service based on linked data principles.

The main purpose of the service is to provide the users the capacity to share their knowledge in form of routes and points of interest and to enrich the information obtained using data from the web. Then the generated knowledge can be used to recommend the users new routes or points of interests based on their location and preferences.

The knowledge obtained on the system will be published following the Linked Open Data best practices, so that it can be used by other developers or interested groups.

In addition, it will be possible to obtain an objective difficulty rating for the routes or trails that the users share in the system, relieving from the users the problem of rating their own routes and eliminating possible subjective interpretations.

Together with this, several other services will be provided, allowing users to follow trails from others, to post notes at certain location and to obtain in any moment information regarding their

surroundings; all in order to build a GPS community around this service.

So, in short, the goals of the project can be stated in the following way:

- To build a semantic location-aware service
- To publish the data of the system following the pattern of Linked Open Data
- To create a model for the semantic representation of trails and points of interest
- To build a service that allows the correction and evaluation of a route
- To build a GPS community based on the location-aware service

### 2.2.2 Scope of the project

In order to fulfill the stated goals, a GIS system based on semantic data will be built. The scope of the project is limited to the following:

- Research over the areas, technologies and standards that the project touches.
- Design of a model for the semantic representation of trails and points of interest and a infrastructure that uses and supports this model.
- Development of a service that makes use of the researched technologies and standards and that follows the previously created design.

#### Scope of the research

Research will be carried over several topics, listed below in order of relevance:

- The Semantic Web
- Linked Open Data
- GIS
- GPS devices
- Geographic information standards
- Development tools and platforms
- Similar systems

#### Scope of the development

The design and development will cover the following issues:

- Design and creation of a semantic ontology
- Development of a location aware web service
- Creation of a web application to access the service
- Development of a mobile application to complement the web
- Extraction of data from third party sources on the web

## 2.3. FINAL PRODUCT

The project will result in a GIS system which relies on a semantic back end, that is, it stores the data on RDF format and is able to reason over it. The system will be composed by a server, a web client and a mobile application.

This section describes the characteristics of the products that will compose the system.

### 2.3.1 Server

The server will be formed by two main components, a semantic data store and a web server. The web server will expose the a endpoint that allows direct read-only access to the database, so that other developers are able to query it. In addition, the server will have a NoSQL database used to store sensible information about the users of the system, with the purpose of providing secure authentication.

#### Semantic data store

The main function of this component will be the storage of the information existing on the system. Since semantic databases - called triplestore because they store the data using a data structure called triples - use ontology oriented data modeling (see section 3.3), the knowledge kept will not belong solely to domain related data, but will also include a ontology that represents the schema of the database and the relations among the resources.

This data store will be accessed through the HTTP protocol, even locally. In fact, it will be possible to query the data base through a read-only endpoint, accessible using HTTP. This way, any developer will be able to use the data in their application directly, without having to follow any predetermined access functions.

Besides, the component will be complemented by a reasoner. This reasoner will be able to infer new knowledge from the existing one. In this context, this means that it will be able to analyze the statements in the database and using the ontology as a basis it will be able to produce new statements.

In addition to regular reasoning, it will also be possible to reason spatially over the data. This way, it will be possible to use the spatial representations of resources in the system to make spatial queries, based on distance and geometric relations.

#### Web server

The web server is the component that will contain most of the functionality of the server. First, it will implement most of the functions encountered in typical web systems, including user authentication, media upload and read, create and modify operations. All these functions will be exposed through a API, following the REST style.

In addition to these basic functionality, the server will also be in charge of receiving the information about trails and points of interest and analyzing it to obtain additional data. This includes applying a mathematical method to calculate a difficulty score for trails.

The API will also expose function allowing spatial queries without any need of previous knowledge or expertise on this area. For example, a function will allow asking for all the points of interest at a certain distance from another feature. This queries will be limited to retrieval of routes, points of interest and geolocated notes.

The spatial information that the API provides will be exposed following the GeoJSON format.

The server will communicate with the data store using an HTTP interface and sending SPARQL queries for managing the data. In addition to this, the server will carry the task of analyzing the profile of each user and recommending routes that can interest them.

In addition to the mentioned functionality, the server will also expose a HTML5 web socket based API that the mobile application will use to provide real time services to the users.

### 2.3.2 Web application

The web application will be the main interface through which the users will be able to access the service. The web application will be developed taking into account that it will have to be visualized correctly in all kind of devices and that minimum loading times are desired.

The client will allow users to upload routes and points of interest from GPX files, a standard format for the interchange of GPS information (see section 3.9). In addition, user will also be allowed to draw the routes on a map or mark the points of interest.

It will be possible to search information about the different features and users of the system, taking advantage of the semantic data, which allows to easily build search engines which go beyond the classic keyword search. The application will offer a detailed view of the data and the possibility to comment on the different user generated features. All this information is to be presented in a easy to read format, that is, using interactive maps to show geographical information.

In addition to all of these, the web will provide a interactive map of the world. Moving and zooming this map will show the different routes and points of interest that the current view of the map covers, this way it will be possible to explore the data on the system without the need of a text search.

A exclusive functionality that the client will offer is a interactive editor. This editor will allow drawing trails and points of interest, providing a way to share knowledge without the need of a file recorded from a GPS. In addition, it will be possible to edit existing routes or trails imported from a file, providing the users with a way of correcting possible GPS generated errors.

The editor will allow the following functions:

**Draw a route on the map:** By repeatedly clicking on the map, specifying the coordinates which form the trail, the users will be able to generate data without a need of GPX files, however, it will be impossible to analyze this trails due to the lack of altitude data.

**Mark a point of interest in the map:** With a single click on the map, it will be possible to mark a point of interest in the specified point. The user will be prompted with a form, and by filling it, the point of interest will be uploaded to the server.

**Edit a existing route:** Whether it is downloaded from the platform or imported from a file, it will be possible to edit the coordinates of a route by dragging the points that form it.

**Fusing two routes:** It will be possible to join the end of a route with the start of another to create a new route.

### 2.3.3 Mobile application

The mobile application will provide the users with a mean of viewing and browsing though the data on the system, in a similar manner to the web client. However, many of the functions present on the browser base client, for example the editor, will no be present on the mobile client, for the web application will be designed to be usable in mobile browsers.

In addition to the basic functionality, this component will allow the users to record their own trail as they traverse it using the GPS on the mobile device. It will be possible to upload this record to the server or to export it as a GPX or GeoJSON.

The application will also provide a function that allows user to receive real time notifications about the nearby points of interest and geolocated notes. These features will be filtered according to the user preferences, before notifying the user.

It will also be possible to follow the trails that another user has shared, with the mobile application sending a notification to the person using it every time that he/she deviates to much from the established route and indicating how to correct its trajectory.

Finally, the application will allow the creation of geolocated notes. To do this, the user will simply need to write a text, take a photo or record a video and after specifying some privacy settings, the note will be recorded in the system on the current coordinates of the user.

## 2.4. PROJECT REALIZATION

### 2.4.1 Realization methodology

The development method to be followed through the project consists on a prototype based iterative development methodology. This procedure allow to develop a project splitting it into different phases and developing a functional part of the project at each phase. Figure 2.1 shows a graphical conceptualization of this development methodology.

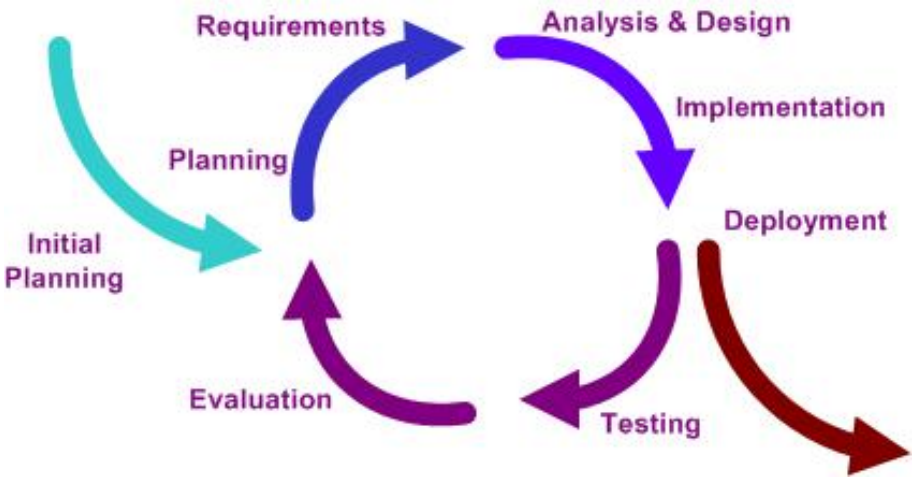


Figure 2.1: Iterative development phases

*source: http://www.wikipedia.org/*

Thanks to this working methodology, it should be possible to develop independently every components that forms the system and it will be possible to realize effective unit testing over these components.

The development of the project has been split into six phases, whose tasks are listed below. A Work breakdown structure containing the tasks of the project can be found in figure 2.2.

### Project opening

This phase consists on the start, organization and planning of the project. The tasks on this phase are not directly related to the development of the final product, yet they are necessary for the development of the project. The tasks in this phase are the following:

**T1.1** Planning and organization

The resources necessary for the realization of the project will be determined and the tasks needed to carry it out will be organized. The development of the project through time will be established.

**T1.2** Monitoring

This task will be carried through the whole project. The current status of the project will be monitored to determine the possible existing issues and to ensure that the goals are being reached.

### Initial research

The research related with the areas that the project touches will be carried out. The goal of the phase is to obtain the sufficient information to develop a efficient and valid system.

**T2.1** Research on the Semantic Web and Linked Open Data

Research will be done on topics and technologies related to the Semantic Web and Linked Open Data. Other topics will also be investigated, for instance, the representation of spatial data using RDF.

**T2.2** Research on technologies

GIS systems, and web and application development tools will be investigated with the goal of establishing an adequate development environment for the project.

**T2.3** Analysis of standards and similar system

Existing standards for trail and point of interest representation will be analyzed, as well as the use of this on other platforms and the functionality that similar services offer.

### Base system development

The design of the architecture of the system and the development of the software that will be present on the server will be done in this phase. The tasks consist mostly on design, coding and testing.

**T3.1** System design

The technological architecture of the system and of its components will be designed.

**T3.2** Test design

The tests to be carried on every component will be determined.

**T3.3** Database installation and testing

The semantic database will be installed and tested to guarantee that the desired functionality is granted.

**T3.4** Web server implementation and testing

The basic functionality of the web server and related software will be developed and tested. An example of these additional software pieces is the data store connector.

**T3.5** API implementation and testing

The functions that the API of the web server exposed will be developed and tested.

#### Web application development

The web application will be developed using the server produced on the previous phase. Tasks involve mainly interface design, web development and testing.

**T4.1** Interface design

The design of the web interface will be done and the needed resources, such as images and fonts, will be obtained.

**T4.2** Implementation and testing

The logic on the web application will be coded and test will be carried out over the developed functions.

**T4.3** Editor implementation and testing

The functionality regarding the map editor will be developed and tested.

#### Mobile application development

The mobile application will be developed using the API of the produced server . Tasks involve mainly interface design, HTML5 development and testing.

**T5.1** Interface design

The design of the mobile interface will be done and the needed resources, such as images and fonts, will be obtained.

**T5.2** Implementation and testing

The logic on the mobile application will be coded and test will be carried out over the developed functions.

### Project finalization

The last phase will take care of the tasks related to project closure and deployment.

**T6.1** System deployment

The web server will be deployed to a accessible URL and the mobile application will be published on the corresponding markets.

**T6.2** Project closure

A final check of the completed goals will be done, future lines of work and possible upgrades will be established and the project will be closed.

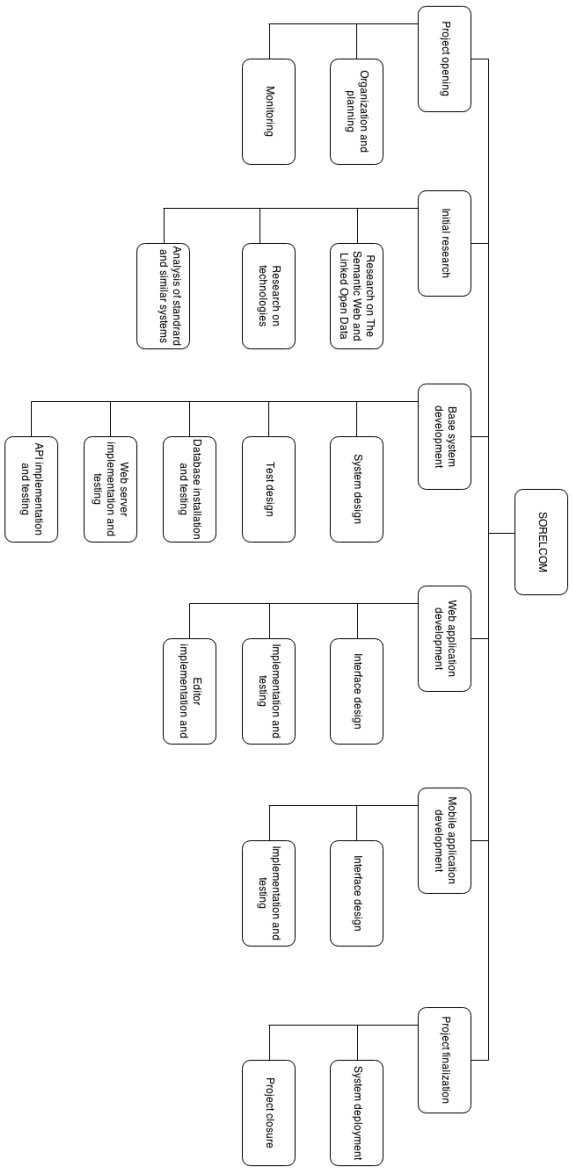


Figure 2.2: Work breakdown structure

### 2.4.2 Intermediate products

The project is divided on several phases. An intermediate product is obtained from each of the phases or tasks. Table 2.1 offers a reference of the intermediate products of the project and the tasks or phases they relate to.

Table 2.1: Intermediate products of the project.

| **Name** | *phase* | *task code* |
|---|---|---|
| Technological research | 2. Initial research | T1.1, T1.2 |
| Ontology specification | 3. Base system development | T3.1 |
| System architecture | 3. Base system development | T3.1 |
| Design of the web server | 3. Base system development | T3.1 |
| Design of the web application | 3. Base system development | T3.1 |
| Design of the mobile application | 3. Base system development | T3.1 |
| Test suit | 3. Base system development | T3.2 |
| Design of the web application interface | 4. Web application development | T4.1 |
| Design of the mobile application interface | 5. Mobile application development | T5.1 |

These intermediate products are all documents, their contents are listed below:

**Technological research:** Contains a list and of the main technologies that will be involved in the project and a description for each of them.

**Ontology specification:** Describes the classes, properties and relations that will appear on the ontology and how they are used to model the data on the system.

**System architecture:** Defines which are the components that form the system and how they are related among them.

**Design of the web server:** Describes the logical architecture of the web server.

**Design of the web application:** Describes the logical architecture of the web application.

**Design of the mobile application:** Describes the logical architecture of the mobile application.

**Test suite:** Describes the tests to be carried on each of the components of the system.

**Design of the web application interface:** Describes the appearance of the web interface.

**Design of the mobile application interface:** Describes the appearance of the mobile interface.

## 2.5. ORGANIZATION AND TEAM

### 2.5.1 Organizational structure

The organizational structure is relatively simple. The organization involves two main groups, the project manager and the work team.

The project manager is the one in charge of the organization of the project and his function is the direction of the work to do. It takes care of the monitoring and planning of the work as well as most of the validations.

The work group carries out the tasks related to the development of the products of the project. This tasks require research, designs, testing and implementation. The work group will be formed by a single student performing all the required roles.

The organizational schema including all the roles on the project is presented in figure 2.3.

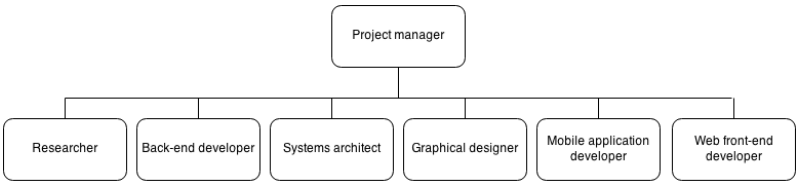


Figure 2.3: Organizational structure

### 2.5.2 Human resources plan

The organization will be formed by the following profiles, each related to the different areas of competence that this project comprises.

#### Project manager

The project manager is the person in charge of the opening, organization, monitoring and closure of the project. In addition, it is the person in charge of communicating with the management team.

The skills of the project manager should include at least the following:

- Planning
- Organizing
- Communication

In addition, it is convenient for it to have knowledge on the areas that the project touches.

#### Researcher

Due to the innovative nature of the project and the amount of work related to the semantic web, the profile of a researcher is required. The researches takes care of investigating the current state in the areas that the project touches and elaborating state of the art documents needed for the development of the project.

The researcher needs to have knowledge on the areas related to the project, that is, The Semantic Web and Linked Open Data. Learning and writing capabilities are also convenient.

#### Back-end developer

This profile takes care of developing the back-end of the application, the web server. Due to the nature of its work and the technologies to be used in the project, the following knowledge is required.

- JavaScript programming skills
- Knowledge of the NodeJS programming environment
- Knowledge of the SPARQL query language

#### Front-end web developer

The web developer takes care of creating the web application. Its work involves development of web based graphical user interfaces, due to this and the technologies chosen for the project, the following skills are required.

- HTML5, CSS3 and JavaScript
- The AngularJS web development framework
- The CSS framework Twitter Bootstrap
- The JavaScript map development library Leaflet

#### Mobile application developer

This profile takes care of developing the mobile application. Proficiency on mobile application programming is needed, however, it is also necessary to know how to use HTML5 native development tools. The required skills can be summarized in the following:

- HTML5, CSS3 and JavaScript
- The HTML5 web sockets protocol
- The HTML5 native mobile development library Phonegap
- The JavaScript map development library Leaflet

#### Graphical designer

The graphics designer takes care of designing the interface of the application. In order to present this design it is required that the designer knows how to use some graphic design tool such as GIMP or Photoshop.

#### Systems architect

The systems architect takes care of designing the system architecture and the logical design of each component. It should have the following skills:

- Software design
- The modeling standard UML

## 2.6. EXECUTION CONDITIONS

The usual workplace will be the DeustoTech office located in the 4th floor of the ESIDE building in the university of Deusto. In this environment ethernet network connection is provided as well as a power source for computers. Still, a personal laptop is to be used for the development instead of office computer. Mobile devices used for testing will also be properties of the student.

The weekly schedule is of 18 hours, divided in three hours per day.

### 2.6.1 Hardware

Since web and mobile applications are to be developed, there is a need to carry out tests and preview results in various kind of devices, such as mobile phones and tablets. The hardware that is available is the following:

- Laptop computer, Inter i7 processor, 15 inch screen
- Android Smartphone, Nexus 5 model
- iPad tablet

### 2.6.2 Software

The software tools to be used in the project will be open source in most cases, and just free in the rest. In this section the software that is available is listed:

#### Programming tools

- NodeJS 0.10.25 - JavaScript runtime environment
- NPM 1.3.10 - NodeJS package manager
- Bower 1.3.3 - JavaScript library management tool
- Yeoman 1.1.2 - JavaScript project scaffolding tool
- Express 3.4.3 - NodeJS web development library
- AngularJs 1.2.11 - JavaScript web application development framework
- Leaflet 0.7.3 - JavaScript map developing library
- Twitter Bootstrap 3 - CSS responsive web development framework
- SASS 3.3.8 - CSS development framework
- Phonegap 2.9.1 - Mobile HTML5 programming platform

#### Web browsers

- Google Chrome 27
- Mozilla Firefox 22
- Safari 6
- Opera 12
- Internet Explorer 10

#### Others

- Ubuntu 14.04 - Open source desktop operating system
- Atom 0.105 - JavaScript based code editor
- Parliament 2.6 - Semantic spatial database, reasoner and endpoint
- GIT 2.0.0 - Version control system
- GIMP 2.8 - Image manipulation program

### 2.6.3 Change control

The changing requirements or petitions will follow the proceeding described below:

1. The change will be communicated to the work group, in case it comes from outside stakeholders.
2. The work group will study the request and evaluate the impact of the potential change in the project.
3. The work group will produce a report that will be sent to the project manager.
4. The project manager will take the final decision on whether to accept the changes or not.

### 2.6.4 Product reception

During this project two types of product will be created, documents and software. Both of them will be accepted following different procedures.

The documents will follow a previously determined structure. This structure requires each document to have a index, a figure and table index and a list of references used. There is only one requirement regarding the content, documents must start with a overview of the contents of the paper. The author and title of the document must appear at the beginning.

These documents will be sent to the project manager, who will have to accept it in a period of 5 working days. In case of a lack of response, the work team will assume that the document has been accepted so that the development of the project is not stopped.

The software on the other side will follow more rigorous criteria. On the task **SYS2**, a document specifying the test suite for each component will be created. Software components will only be accepted when they have passes all the specified tests.

## 2.7. PLANNING

This section presents several aspects related to the project planning. The planning considers a work schedule of 3 hours each day. The project starts Monday 3 of march 2014 and will end the 10th of July of the same year.

### Workload estimation per profile

The estimated work to be done by each profile is shown on this section.

Table 2.2 contains the work to be done by the project manager.

Table 2.2: Workload estimation for the project manager.

| **Task** | *hours* |
|---|---|
| T1.1 | 6 |
| T1.2 | 16 |
| T6.2 | 3 |
| **Total** | 25 |

Table 2.3 contains the work to be done by the system architect.

Table 2.3: Workload estimation for the system architect.

| **Task** | *hours* |
|---|---|
| T3.1 | 45 |
| T3.2 | 6 |
| **Total** | 51 |

Table 2.4 contains the work to be done by the researcher.

Table 2.4: Workload estimation for the researcher.

| **Task** | *hours* |
|---|---|
| T2.1 | 15 |
| T2.2 | 12 |
| T2.3 | 15 |
| **Total** | 42 |

Table 2.5 contains the work to be done by the back-end developer.

Table 2.6 contains the work to be done by the front-end web developer.

Table 2.7 contains the work to be done by the mobile application designer.

Table 2.8 contains the work to be done by the graphics designer.

Table 2.5: Workload estimation for the back-end developer.

| **Task** | *hours* |
|---|---|
| T3.3 | 3 |
| T3.4 | 30 |
| T3.5 | 24 |
| T6.2 | 3 |
| **Total** | 60 |

Table 2.6: Workload estimation for the front-end web developer.

| **Task** | *hours* |
|---|---|
| T4.2 | 27 |
| T4.3 | 21 |
| **Total** | 48 |

## Network and Gantt diagram

Figure 2.4 shows the gantt diagram for the project, which establishes the temporal planning. Figure 2.5 shows the network diagram of the tasks on the project, in order to identify the dependencies among them.

## Work plan

The work plan obtained after establishing the schedule and the interdependencies among tasks is presented in table 2.9.

Table 2.7: Workload estimation for the mobile application developer.

| **Task** | *hours* |
|---|---|
| T5.2 | 21 |
| **Total** | 21 |

Table 2.8: Workload estimation for the graphics designer.

| **Task** | *hours* |
|---|---|
| T4.1 | 12 |
| T5.1 | 9 |
| **Total** | 21 |

Table 2.9: Real work plan.

| **Identifier** | *start* | *end* | *days* | *profile* | *work (h)* |
|---|---|---|---|---|---|
| **T1 - Project initiation** | | | | | |
| T1.1 | 03/03 | 05/03 | 2 | Project manager | 6 |
| T1.2 | 10/03 | 13/06 | 8 | Project manager | 16 |
| **T2 - Initial research** | | | | | |
| T2.1 | 05/03 | 12/03 | 5 | Researcher | 15 |
| T2.2 | 19/03 | 25/03 | 4 | Researcher | 12 |
| T2.3 | 12/03 | 26/03 | 5 | Researcher | 15 |
| **T3 - Base system development** | | | | | |
| T3.1 | 26/03 | 18/04 | 15 | System architect | 45 |
| T3.2 | 18/03 | 22/04 | 2 | System architect | 6 |
| T3.3 | 22/04 | 23/04 | 1 | Back-end developer | 3 |
| T3.4 | 23/04 | 08/05 | 10 | Back-end developer | 30 |
| T3.5 | 08/05 | 21/05 | 8 | Back-end developer | 24 |
| **T4 - Web app. development** | | | | | |
| T4.1 | 21/05 | 28/05 | 4 | Graphics designer | 12 |
| T4.2 | 02/06 | 16/06 | 9 | Front-end developer | 27 |
| T4.3 | 26/06 | 08/07 | 7 | Front-end developer | 21 |
| **T5 - Mobile app. development** | | | | | |
| T5.1 | 28/05 | 02/06 | 3 | Graphics designer | 9 |
| T5.2 | 16/06 | 26/06 | 7 | Mobile app developer | 21 |
| **T6 - Project finalization** | | | | | |
| T6.1 | 08/07 | 09/07 | 1 | Back-end developer | 3 |
| T6.2 | 09/07 | 10/07 | 1 | Project manager | 3 |

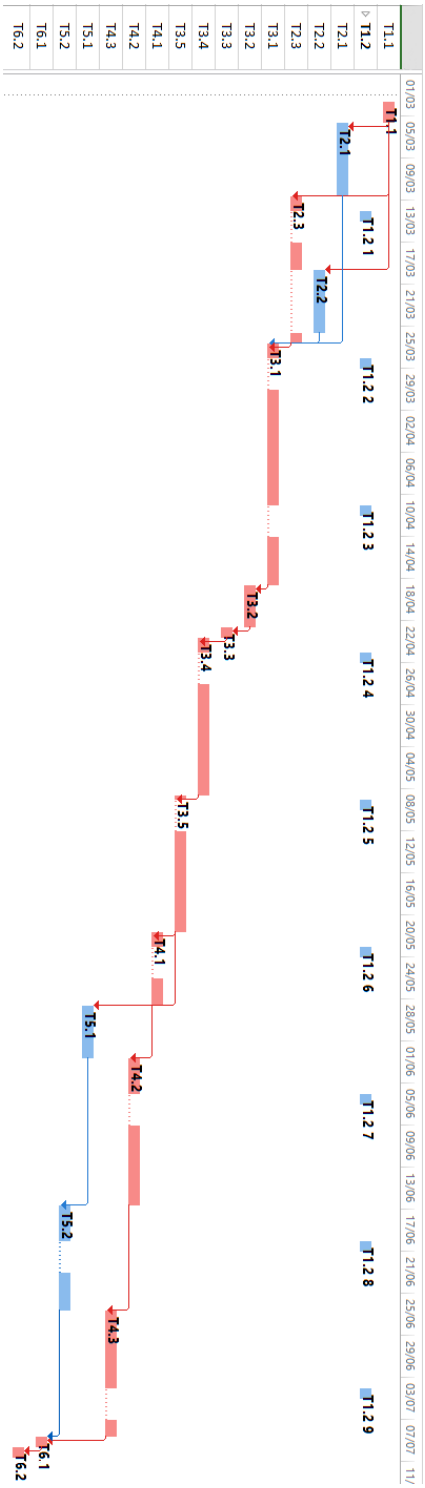


Figure 2.4: Gantt diagram

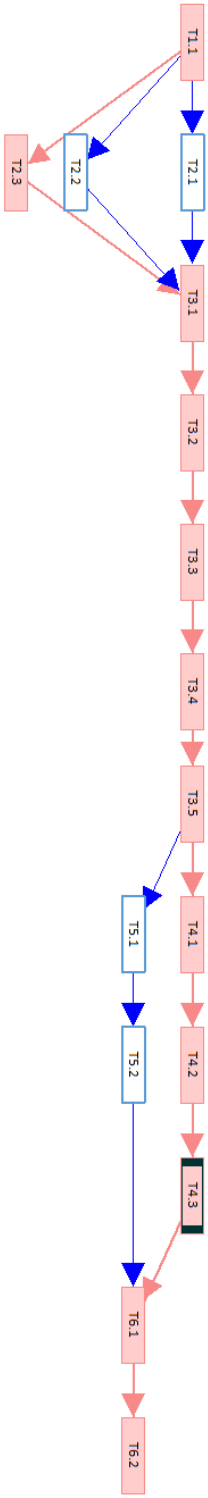


Figure 2.5: Network diagram

## 2.8. PROJECT BUDGET

The costs associated with the project are divided into two main sources. The first group is the hardware related costs, which comprises the budget required for the physical technological resources used in the realization of the project. Table 2.10 reflects these costs.

The second cost it the one destined to the human resources team. An overview of the costs generated by each profile may be found in table 2.11.

No budget will be destined to cover software cost, for the tools used on the project will all be either open source or free-ware. A summary of the project budget can be found in table 2.12.

Table 2.10: Hardware related budget.

| **Name** | *units* | *unit price (€)* | *cost (€)* |
|---|---|---|---|
| Laptop computer | 1 | 1000.00 | 1000.00 |
| Android smart phone | 1 | 450.00 | 450.00 |
| iOS tablet | 1 | 500.00 | 500.00 |
| **Total** | | | 1950.00 |

Table 2.11: Human resources budget.

| **Name** | *work (h)* | *cost (€)* |
|---|---|---|
| Front-end web developer | 48 | 240.00 |
| Back-end developer | 60 | 300.00 |
| Mobile application programmer | 21 | 105.00 |
| Graphics designer | 21 | 105.00 |
| Researcher | 42 | 210.00 |
| System architect | 51 | 255.00 |
| Project manager | 25 | 125.50 |
| **Total** | | 1340.00 |

Table 2.12: Summary of budget.

| **Name** | *cost (€)* |
|---|---|
| Hardware | 1950.00 |
| Human resources | 1340.00 |
| **Total** | 3290.00 |

# 3. TECHNOLOGICAL RESEARCH

## 3.1. OVERVIEW

This chapter presents the results of the initial technological research done to carry out the project. In here, the main technologies used in the project are explained in detail, as well as the benefits of using them and the reasons for doing so.

The most relevant tools and frameworks used on the project are the following:

- RDF
- OWL
- SPARQL
- GeoSPARQL
- NodeJS
- AngularJS
- Leaflet
- Phonegap
- GPX
- GeoJSON

There are many other software tools and libraries are used through the project, however, only the most relevant of them, and the ones on which the initial investigation has been carried are shown in this chapter. The rest of the tools are briefly described in chapters 5, 6 and 7.

## 3.2. RDF

RDF stands for Resource Description Framework, and it is a framework for expressing information about resources. These resources can be anything, documents, people, physical objects, even abstract concepts.

RDF is intended for situation in which data is to be processed by machines and not simply displayed to humans, due to this, it is very useful for the goals of the Semantic Web. It can be used to publish and interlink data on the web. For example, retrieving a resource which represents a person, say Bob, could provide us with the fact that Bob knows another person, say, Alice, who is represented by her URI. Retrieving the resource representing Alice may then yield links to datasets about other persons and so on. A computer can automate this process and follow these links, aggregating data from different resources. This kind of uses are often known as Linked Data [54].

There are many reasons to choose using RDF, the following are some of them:

- Adding machine readable information to Webpages. This allows them to be displayed in enhanced format or to be processed by roaming agents.
- Enriching datasets by linking them to other sources of data.
- Interlinking APIs so that clients are able to discover new sources of information.
- Using the datasets currently published as Linked Data
- Providing a standards compliant way of exchanging data between applications.
- Interlinking various datasets within an organization and allowing cross-dataset queries by using SPARQL.

## RDF data model

Data in RDF is formed by statements. This statements are called *triples* and follow a subject-predicate-object structure. A statement on RDF represents a relationship between two resources, the **subject** and **object** represent the resources being related while the **predicate** represents the nature of the relationship. These relationships are phrased in a unidirectional way, from subject to object, and are called properties.

The same subject is often referenced in multiple triples and can of course be the object of other triples. These ability to have the same resource as subject and object in different triples makes it possible to find connections between RDF resources and is one of the most powerful features of the framework. A set of triples can be viewed as a connected **graph**, in figure 3.1 an example of the visual representation of an RDF graph is given.


Figure 3.1: An RDF graph describing a person

*source: http://www.w3.org/TR/2004/REC-rdf-primer-20040210/*

There are three types of RDF data that can occur in triples: IRIs, Literals or Blank nodes [54].

### IRIs

IRI is an abbreviation for International Resource Identifier, which is used to identify a resource. On RDF, IRIs may appear in any component of triples, be it subject, object or predicate. The URLs (Uniform Resource Locators) which identify web pages are a form of IRI, however other forms of IRI can identify a resource without specifying its location. An IRI is a generalization

of URIs allowing the use of non-ASCII characters unlike the latter. Thus, an IRI is defined as a set of characters unreserved characters which can include characters of the UCS [4, 21](Universal Character Set, [ISO10646]).

An example of a IRI from dbpedia can be the following: `http://dbpedia.org/resource/Leonardo_da_Vinci`. Even if RDF does not explicitly specify what a certain IRI represents, they can take meanings from vocabularies or conventions.

In SPARQL (see 3.4) and in some RDF serialization models prefixes can be defined. A dataset, for example http://example.org, could be given the name *my*. By giving this name we can reference http://example.org/Bob, a resource on the dataset as my:Bob. This is useful for it helps to avoid clutter on the query and helps human understanding, thus, from now on, prefix notation will be used to express datasets and vocabularies on the document.

### Literals

Literals are the basic values that are not IRIs. Literals are associated with data types, for example, the *Integer* and *Float* data types are equivalent to the traditional types from the programming languages. The string typed literals may optionally have a language associated allowing internationalization. Most of the regularly used data types are defined by XML Schema [37].

One particularity of RDF data types is that they can be defined on ontologies. Due to this, it is possible to find custom types, such as a string representing the serialization in WKT [17] format of a geometry.

An example of different types of triples, including literals can be found in listing 3.1.

```
<my:Bob> <foaf:name> "Bob"
<!--The object is a literal of type String -->
<my:Bob> <rdf:type> <foaf:Person>
<my:Bob> <foaf:age> "19"^^xsd:Integer
<!-- The literal is of type Integer -->
```

Listing 3.1: Different types of triple statements.

### Blank nodes

Blank nodes are used to refer to resources who don't have or need a global identifier (an IRI). For example, we may want to refer to the right arm of a person, however, we may think that it is too verbose or unnecessary to actually give a identifier to this arm, so we could use a blank node to represent it, as illustrated in listing 3.2. In this example it makes sense to have a blank node, for the right arm is directly dependent on the person, an arm (usually) makes no sense detached from a person.

```
<my:Bob> <rdf:type> <foaf:Person>
<my:Bob> <my:hasRightArm> :_b1
:_b1 <rdf:type> <my:Arm>
```

Listing 3.2: Triple statements with a blank node.

## RDF vocabularies

The data model of RDF does not make any assumptions about the meaning of the IRIs representing the resources. However, in the reality, RDF is usually used in conjunction with a series of vocabularies or other conventions that give semantic information about these resources.

Many vocabularies are currently considered standard due to their wide use among Linked Open Data systems. Some of these ontologies are the following.

Friend of a Friend (FOAF): One of the first vocabularies used worldwide, it is used to represent persons, the properties that identify them and the relations among them [11].

Dublin Core: A metadata element set for describing a wide range of resources. Contains properties such as the creation date of the resource [29].

schema.org: A vocabulary for marking up web pages so that search engines can more easily find them [45].

## RDF schema

RDF Schema (RDFS) is a extension of RDF. It provides a data-modeling vocabulary for RDF data, in this way, RDFS extends RDF by giving externally specified semantics to specific resources, semantics which cannot be captured in plain RDF [12]. RDF Schema provides information about interpretation of the statements in a data model, however, it does not constrain the syntactical appearance of RDF descriptions.

The extension provides mechanisms to describe groups of related resources and the relationships between these resources, as well as classification hierarchies. The class and property system that can be expressed is similar to the type systems of object oriented programming, however, instead of defining classes in terms of the properties their instance have, they are defined in terms of classes of resource to which they apply [10].

RDF schema is written in plain RDF, thus it takes form of a vocabulary. The classes and properties provided are listed below:

### Classes

`rdfs:Resource` All things described by RDF are called resources and are instances of this class. It is the class of everything and all other classes are *subclasses* of it.

`rdfs:Class` It is the class of all the resources that are classes, thus, `rdfs:Class` is an instance of `rdfs:Class`.

`rdfs:Literal` It is the class of literal values such as strings and integers. It is an instance of `rdfs:Class`.

`rdfs:Datatype` It is the class of datatypes, such as `integer`.

`rdf:langString` It is the class of language tagged string values, an instance of `rdfs:Datatype` and subclass of `rdfs:Literal`.

`rdf:HTML` It is the class of HTML string values, an instance of `rdfs:Datatype` and subclass of `rdfs:Literal`.

`rdf:HTML` It is the class of HTML string values, an instance of `rdfs:Datatype` and subclass of `rdfs:Literal`.

`rdf:XMLString` It is the class of XML literal values, an instance of `rdfs:Datatype` and subclass of `rdfs:Literal`.

### Properties

`rdfs:range` It is used to indicate that the values of a property are instances of a certain class.

`rdfs:domain` It is used to indicate that any resource that has the given property is a instance of a certain class.

`rdfs:type` It is used to state that a resource is a instance of a certain class.

`rdfs:subClassOf` It is used to state that all instances of one class are also instances of another.

`rdfs:subPropertyOf` It is used to state that all resources related by a class are also related by another.

`rdfs:label` It is used to provide a human readable version of the name of the resource.

`rdfs:comment` It is used to provide a human readable description of a resource.

### Use of RDFS

RDFS provides with a vocabulary to express relations among RDF resources, which allow for reasoning and inference over datasets. However, this vocabulary is very limited, it does not allow much more that the definition of classification hierarchies. These hierarchies are defined in regular RDF triples, as shown in listing 3.3.

```
1 <foaf:Person> <rdf:type> <rdfs:Class>
2 <my:knows> <rdf:type> <rdf:Property>
3 <my:knows> <rdfs:domain> <foaf:Person>
4 <my:knows> <rdfs:range> <foaf:Person>
5 <my:friendOf> <rdfs:subPropertyOf> <my:knows>
```

Listing 3.3: RDFS class and property hierarchy.

In addition to ability to express relations, RDFS provides additional constructs for the representation of resources. The vocabulary defines containers, such as bags and sequences, in addition to properties to indicate membership to these. Besides collections, the ontology defines a more sophisticated mechanism for data structuring; RDF lists, which are similar to the Linked List data structure. However, the usage collections bring an additional layer of complexity to the dataset, thus they are scarcely used.

### Why RDF?

Using RDF can greatly benefit this project. It can give a representation of the relationships on the system which can be easily analyses by machines. This can help building important functions, such as the recommendation system.

However, the use of RDF on this project is almost mandatory, for one of the goals is achieving data interoperability and publishing the data as Linked Open Data on the web. Anyway, the expressive level provided by RDF schema is not enough for the needs of the project, so another vocabulary must be used.

## 3.3. OWL

The Web Ontology Language (OWL) is a semantic markup language for publishing and sharing ontologies in the World Wide Web [3]. The language is designed for applications that need to process the content of the information instead of just presenting it to humans. OWL provides greater machine interpretability than RDF or RDF schema by providing additional vocabularies along formal semantics.

OWL is used to explicitly represent the meaning of terms in vocabularies and the relationships among these terms. These representations are called ontologies [35]. The standard is part of the stack of W3C recommendations for the semantic web, together with XML, XML Schema, RDF and RDF Schema.

OWL provides three sub-languages with increasing expressive power. Each subset of OWL includes all the previous ones, in addition to RDFS and RDF, so that there is no loss on expressive power as illustrated in figure 3.2.

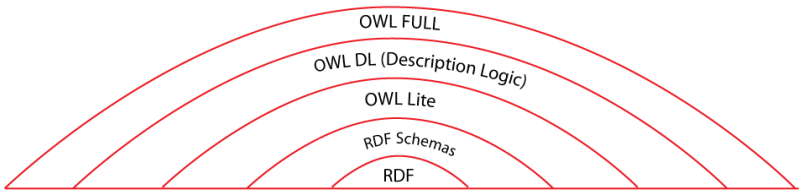


Figure 3.2: The OWL levels of expressivity

*source: http://www.w3.org/*

### OWL Lite

OWL Lite is designed to support those users who need only a classification hierarchy and simple constraints. For example, while supporting cardinality constraints, it only allows cardinality 0 or 1, for the sake of simplicity. Due to this it should be simpler to provide tool support for OWL lite than to its other, more expressive, relatives. It also has a lower level of formal complexity than its more expressive siblings.

This sub-language contains all the features of RDF schema, such as category definitions through classes and hierarchical classification through subclasses, plus additional restrictions and expressions.

OWL Lite provides tools for defining equality and inequality relations, property characteristics, property restrictions, restricted cardinalities and class intersections.

### Equality and Inequality

They are used to indicate equality and inequality relations among classes. One example

**`equivalentClass`** Two classes may be stated as equivalents. This equality is used to create synonyms.

**`equivalentProperty`** It is used to define synonymous properties, the same way as with classes.

**`sameAs`** It states that two individuals are the same.

**`differentFrom`** The opposite to **`sameAs`**, used to indicate that two individuals are different.

**`allDifferent`** It can ve used to state that a number of individuals are different from each other.

### Property characteristics

Property characteristics are special identifiers in OWL that provide information concerning the properties and their values. These characteristics are placed on properties, with relation to no class, as in listing 3.4.

```
1 <owlx:ObjectProperty owlx:name="adjacentRegion" owlx:symmetric="true">
2   <owlx:domain owlx:class="#Region" />
3   <owlx:range owlx:class="#Region" />
4 </owlx:ObjectProperty>
```

Listing 3.4: OWL property characteristic example.

**`inverseOf`** Describes that a property is inverse to another. If A is related to B by the property P1 and P2 is the inverse of P1, then B will relate to A by property P2.

**`TransitiveProperty`** States that a property is transitive. For example, if A is related by transitive property P to B and B is related by P to C, then A will also be related by P to C.

**`SymmetrycProperty`** If a property is stated to be symmetric, then if A is related to B by the property, B will also by related to A by the same property.

**`FunctionalProperty`** If a property is functional, then it can have no more than one value for each individual. This mean that there will be no two triples of the same individual containing this property as a predicate.

**`InverseFunctionalProperty`** It states that the inverse of a property is a functional property.

```
<owlx:Class owlx:name="Wine" owlx:complete="false">
  <owlx:Class owlx:name="&food;PotableLiquid" />
  <owlx:ObjectRestriction owlx:property="#hasMaker">
    <owlx:allValuesFrom owlx:class="#Winery" />
  </owlx:ObjectRestriction>
</owlx:Class>
```

Listing 3.5: OWL property restriction example.

### Property restrictions

OWL Lite allows to place restrictions regarding on how properties can be used inside of instances of classes. One example can be found in listing 3.5

`allValuesFrom` This restriction is placed on a property in relation to a class. It indicates that a property on a particular class has a local range restriction associated with it. For example, the class *Person* may have a property *hasDaugther*, restricted to have all values from class *Woman*. This means that if an individual of type person is related by a relation *hasDaugther* to other individual, we can infer that this individual will be a woman.

`someValuesFrom` This restriction is placed on a property in relation to a class. It indicates that at least one value from the property is of a certain type. For example, the class *SemanticWebPaper* may have a property *hasKeyword* restricted to have some values from *SemanticWebTopic*. This would mean that a semantic web paper can have any number of keywords from any topic, as long as at least one of them belongs to a semantic web topic.

### Restricted cardinalities

OWL includes a limited form of cardinality restrictions. These are placed on properties with respect to classes, that is, the restrictions constrain the cardinality of the property in instances of a specific class.

`minCardinality` If a minimum cardinality of 1 is stated on a property with respect to a particular class, then any instance of that class will be related to at least one individual by that property. For example, a person must have some sort of identifying card, thus, a minimum cardinality of 1 should be placed on the relation *hasIDCard* with respect to *Person*.

`maxCardinality` If a maximum cardinality of 1 is placed on a property with respect to a particular class, then any instance of that class will be related to at most one individual by that property. For example, a person should have at most one ID card, thus, the relation *hasID* would have a maximum cardinality of 1.

`cardinality` It is used as a convenience to indicate that a property has both a maximum and a minimum cardinality with the same values.

### Intersection

OWL Lite offers a very limited expressibility for intersections through the property `intersectionOf`. With this expression one could express that a *EmployedPerson* is a intersection of a *Employee* and a *Person*. An example for this is shown in listing 3.6

```
<owlx:Class owlx:name="WhiteWine" owlx:complete="true">
  <owlx:IntersectionOf>
    <owlx:Class owlx:name="#Wine" />
    <owlx:ObjectRestriction owlx:property="#hasColor">
      <owlx:hasValue owlx:name="#White" />
    </owlx:ObjectRestriction>
  </owlx:IntersectionOf>
</owlx:Class>
```

Listing 3.6: OWL property restriction example.

### OWL DL and OWL Full

OWL DL is the second among the OWL sub-languages. It includes all the expressive power of OWL lite and adds on top of it. Similarly OWL Full adds on top of OWL DL. However, OWL Full is usually deemed as undecidable, in fact, there are no reasoners that can guarantee a solution in finite time. Due to this, many reasoners for OWL DL, such as Pellet try to find a way to convert OWL Full ontologies to a more a previous level of expressivity, since most ontologists use the highest expressive power when they only need OWL DL at most [49].

### Why OWL?

RDF schema provides a very limited way of expressing ontologies, it only allows to build classification hierarchies. Due to this, using OWL to describe the ontology would be the most recommendable thing. More specifically, the expression level of OWL DL, which allows to describe disjoint relations between classes (among other relation) plus all of the OWL Lite characteristics listed above, will allow to build a complete and correct ontology.

## 3.4. SPARQL

SPARQL is the recursive acronym for SPARQL Protocol and RDF Query Language. As its name indicates, it is a protocol and query language for RDF data. The idea behind this language is to allow the querying over RDF datasets in a similar language in which triples are expressed [16].

SPARQL allows querying [43] and updating [47] of RDF datasets. Similar to RDF, this language is composed of triples, however among the subjects predicates and objects, variables can be introduced. The logic behind SPARQL is stating a set of triples and introducing a variable somewhere among these triples. By doing so, the query engine will look and retrieve for all the cases which satisfy the set of triples in the query. Nothing impedes to query for a certain triple without specifying any variable, which can be useful when the goal is to know if the triple actually exists, but it is not the usual case.

In SPARQL prefixes can be used to indicate name spaces. By using the keyword prefix, it is possible to specify a URI which will be prepended to all triples using that resource. Variables are specified by using a interrogation mark (?) before an arbitrary name. Two examples can be found in listings 3.7 and 3.8.

This example is very basic, it does not actually specify any concrete triple it states subject object and predicate as variables. Due to this, the query engine will match every existing statement in the

```
SELECT ?s ?p ?o WHERE{
        ?s ?o ?o .
}
```

Listing 3.7: SPARQL query for retrieving all triples on the dataset.

database and will retrieve all triples.

```
PREFIX my: <http://example.org>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

SELECT ?name WHERE{
        ?person rdf:type foaf:Person .
        ?person foaf:name ?name .
}
```

Listing 3.8: SPARQL query for retrieving names on the dataset.

This case is slightly more elaborated than the previous one. First the keyword PREFIX appears, which specifies namespaces for the query, similar to how XML namespaces work. After the prefix declaration, there is a SELECT keyword. Here the operation to be done is specified, among the options of SELECT for regular queries, ASK for boolean queries, INSERT for data insertion and DELETE for data removal. After the select, the variables to be retrieved, bound in SPARQL terms, are specified. After that, the actual triples of the query are written. In this case a *person* variable is used to retrieve every resource of type *Person* in the dataset. Then if that resource contains a name, it is bound to the variable.

It is possible to use more advanced operations in SPARQL. Every triple statement is a implicit intersection, however, it is possible to use the keyword UNION as well. It is also possible to count the amount of occurrences of a variable, to order by variable values and even to filter with regular expressions. Utilities for retrieving only a certain subset of query matching elements are also provided, allowing to limit the result to a certain amount of triples or event to offset the start of the result to a certain triple.

### Why SPARQL?

Since the system will store data in RDF format, the use of SPARQL as a query language becomes obligatory. Still, there are several benefits for the use of SPARQL over a more traditional language such as SQL or over using a ORM. This protocol is designed for advanced queries, thus it allows to find complex relations among data in a relatively easy manner.

Besides, since SPARQL is a standard language, the same queries used for the system could be used to retrieve data from external datasets.

## 3.5. GEOSPARQL

RDF and SPARQL allow for reasoning in a large domain of applications, however, these reasoning is only concerned about relations explicitly represented in the datasets. This does not include the spatial relations we aim to represent in the system, such as nearby relations.

GeoSPARQL is a standard which aims to provide support from representing and querying spatial data over the semantic web. This standard has been presented by the Open Geospatial Consortium (OGC) and is designed modularly [1, 39]. The components that form the specification are listed below [2]:

- A vocabulary to represent spatial features, geometries and their relationships.
- A set of domain-specific spatial functions to use in SPARQL queries.
- A set of query transformation rules

## Vocabulary

The ontology used to represent spatial data is domain independent in the sense that it is only concerned in representing spatial objects and relations, without specifying the type of resources it represents. Thanks to this is can be used to model any kind of GIS system, whether it represents nature, a country's road infrastructure or cultural points of interest.

The ontology provides two type of top-level classes, *Features* and *Geometries*. Features are simply entities in the real world with some spatial location. A feature can have any kind of spatial form that cannot be precisely defined, for example, a lake or a forest. A *Geometry* on the other hand is any geometric shape, such as a point or a polygon used as a representation of the spatial location of a feature.

Together with these types, a medium for representing the exact geometries of the object is provided. By using the properties *asWKT* and *asGML* it is possible to provide the exact spatial information as a WKT or GML [19] string, as in the listing 3.9.

```
1 <my:Point> <rdf:type> <geo:Feature>
2 <my:Point> <geo:hasGeometry> <my:geom>
3 <my:geom> <rdf:type> <geo:Geometry>
4 <my:geom> <geo:asWKT> "POINT(0, 0)"^^geo:wktLiteral
```

Listing 3.9: A feature in the GeoSPARQL vocabulary.

Besides, there is also a way to provide represent non-exact relation, using qualitative properties, such as the *within* property.

## SPARQL extension

Exploiting the qualitative relations in the system is simple, a regular query can be used. However, in order to take advantage of the serialization of geometries, GeoSPARQL defines a series of functions that allow the manipulation and querying of spatial data. For example, the function *geof:distance* will return the shortest distance between two geometries.

This way, a extension for SPARQL is provided, which allows to find the implicit spatial relations in the dataset, as in the query shown in listing 3.10

## Why GeoSPARQL?

While there are some semantic storage systems that implement their own spatial indexes and representation system there has been no standard in the semantic web until the appearing of

```
SELECT ?p
WHERE{
  ?p a geo:Feature
  ?p geo:hasGeometry ?pgeo .
  ?pgeo geo:asWKT ?pwkt .
  ?w a geo:Feature .
  ?w a my:Park .
  ?w geo:hasGeometry ?wgeo .
  ?wgeo geo:asWKT ?wwkt .
  FILTER(geof:distance(?pwkt ?wwkt units:m)
    < 3000)
}
```

Listing 3.10: Spatial query in SPARQL.

GeoSPARQL.

However due to the innovative character of this standard, there are few systems which actually implement it. Anyway, the usage of this protocol in the system will benefit the system, for in the future, if the standard is adopted worldwide, the data will be more fitting to the Semantic Web.

#### Parliament

Parliament™[48] is a triple-store, SPARQL endpoint and reasoner created by SemWebCentral in 2009. It is one of the few existing semantic databases which support the GeoSPARQL protocol, and it does so without then need of a relational spatial database on the back-end.

Parliament provides a SPARQL endpoint over which selection or update queries can be done. Besides it provides a reasoner which allows to make spatial queries. This database-reasoner-endpoint bundle makes this software very adequate to support the semantic GIS system to be produced. In addition, the reasoner that the bundle provides supports OWL lite, thus it is not necessary to use any additional software to reason over the data.

## 3.6. NODEJS

NodeJs is a framework based on Chrome's runtime engine [52] for developing high-performance, concurrent programs that instead of relying on the mainstream multi-threading approach use asynchronous I/O [51].

The reason behind the development of this framework is mainly that Thread-Based networking is inefficient and difficult to use. Node will be much more memory efficient under high load that systems which allocate threads for each connection.

Node differs from other frameworks such as Django or Rails in that it uses a event driven programming model. In Node, the event model is taken further that in the rest of frameworks, the event loop is presented as a language construct instead of a library. In other languages the event loop is typically started through a blocking call such as `EventMachine.run()` however, in Node, there is no event loop start call, it simply enters it after executing the input script and exits it when there are no more callbacks to execute.

One example of Node's programming model is shown in listing 3.11. In the example, a web server listening on port 1337 is created. To when creating this server a callback is passed; every time a connection is made the web server responds with a "Hello world". The process will tell the operating systems to notify it when a new connection is made, and then go to sleep. When new connections are made the callback will be executed, each connection is just a small memory allocation.

```
1 var http = require('http');
2 http.createServer(function (req, res) {
3   res.writeHead(200, {'Content-Type': 'text/plain'});
4   res.end('Hello World\n');
5 }).listen(1337, "127.0.0.1");
6 console.log('Server running at http://127.0.0.1:1337/');
```

Listing 3.11: NodeJS "hello world" web server.

The framework is powered by a module system. Modules make it possible to include other JavaScript files into an application, using the keyword *require* to load them. This functionality allows easy use of external libraries, in fact, most of the core functionality is written using modules.

NodeJS treats HTTP as a first class protocol, attempting to correct most of the problems that arise in other web frameworks such as HTTP streaming. Thanks to this approach, the platform gives a good foundation for creating web development libraries such as Express.

### NPM

NPM stands for Node Package Manager. It is a tool built to allow for easy installation and control of NodeJS modules. The software offers a simple command line interface to work with. In addition, it offers a way of managing node projects by using JSON configuration files.

By creating a file named *package.json*, it is possible to specify the modules used on the project and their version, plus several other configuration options such as a remote repository.

When using Node to build applications it is almost unavoidable to use NPM to install the needed libraries.

### Express

Express is a minimalist and flexible web application development library built on top of NodeJS. It offers a URL routes, template engine utilities and middle ware among other things.

Express is currently the de facto standard in Node web application development, mainly due to the ease of use it offers. The library offers an `express` object on which callback functions for different HTTP requests can be registered. These callbacks are provided with a request and response object which can be used to analyse the request made to the server and to respond respestively. The example on listing 3.12 creates a express server on port 3000, which will respond to queries on the route "/hello".

Aside from the basic functionality, the library offers a series of shortcuts for rendering parameterized HTML templates, and for building JSON based APIs.

```
var express = require('express');
var app = express();
app.get('/hello', function(req, res){
  res.send('Hello World');
});
var server = app.listen(3000, function() {
 console.log('Listening on port %d', server.address().port);
});
```

Listing 3.12: Express ”hello world” program.

### Why NodeJS

There are several frameworks which could have been used for this project. Jena, a Java framework for Semantic Web application is typically used to manipulate data, and other frameworks such as Django offer very complete geospatial API. Node offers none of these, however, the advantages on performance and scalability it offers out weight other tools. Besides, the student's previous experience with Node has also been taken into account when choosing Node as the working platform.

## 3.6. ANGULARJS

In the last years, there has been a trend to develop web applications instead of developing natively for each operating system. The rise in computational power allows for complex processes to be executed in web browsers and makes the software installation times meaningless, which is causing a shift from native development to web development.

Developing applications for the web has several advantages: there is no need to think in multi-platform development, it is granted; there is no need for installation, etc. Still, there are several issues that arise when developing on the web. The main issue is related to complexity on big JavaScript apps; when projects grow enough it becomes nigh impossible to manage them, due to the lack of structure inherent to most JavaScript projects.

Due to this, tools have been developed to ease web application development. One of the most prominent among these tools is AngularJS, a framework with focus on SPA (Single Page Application) development following the MVC (Model-View-Controller) pattern [20, 27].

Angular has been developed with several goals in mind: First, reducing the amount of DOM manipulation on the code, separating the client side of the application from the server side, giving structure to web applications and improving the capability to test the code.

### Model View Controller

The MVC pattern is implemented in this framework by separating the application into three main components [24].

**The view:** The HTML templates which can be found in any web page are the view, however, unlike in regular applications, they are extended using directives provided by the framework.

**The controller:** Angular allows the definition of controllers in a application. They are used to define operation which will manipulate the data and the views. The views and the controllers are related through an object called `$scope` (see figure 3.3).

**The model:** Any data which belongs to the application domain is considered part of the model. Usually the data is obtained from the server, through REST APIs, since function to easy asynchronous requests are provided.

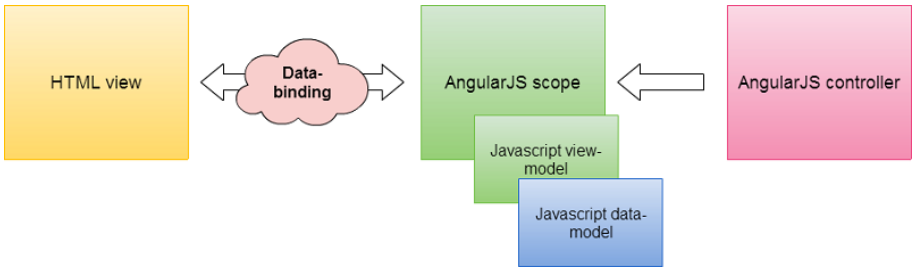


Figure 3.3: View-controller binding through the scope

*source: http://s.codeproject.com/*

In addition to the mentioned, the framework allows the definition of *Services* and *Directives*. Services are object that can be injected into a controller and have a wide array of uses, from code reusing to data sharing among controllers. Directives on the other side are utilities to manipulate the views and aren used in the HTML views. One example of such directives is `ng-click` which is used to specify which operation will be executed when a element on the DOM is clicked.

The MVC in Angular works using a two-way binding. Data is bound to the view through the `$scope` object; if a change happens in the view, the data in the models is automatically updated and if a controller modifies the data in the models, the view is automatically updated. An example is shown on listing 3.13. In this example. when the button is clicked, the function `change` will be executed, changing the value on the variable `text` which will automatically update the paragraph on the view.

```
1  <div ng-controller="ExampleCtrl">
2    <p>{{text}}</p>
3    <button ng-click="change()">
4      Change text
5    </button>
6  </div>
7  <script type="text/javascript>
8    var app = angular.module('exampleApp');
9    app.controller('ExampleCtrl', function($scope){
10     $scope.text = 'No text yet';
11     $scope.change = function(){
12       $scope.text = 'Button has been clicked';
13     };
14   });
15 </script>
```

Listing 3.13: AngularJS example.

## Modules

Similar to NodeJS, Angular provides it's own module system. Modules act as libraries, providing the application with services and directives. The framework does not provide with any tool for installation and management of modules, however, external package manages such as bower [9] have been built.

There are other tools that offer support for this framework. Yeoman is a widely used tool for scaffolding web applications. It is used to build skeletons for different types of projects such as Angular projects.

For the following modules have been considered:

***ui-router:*** Since angular is used for developing single page applications, a way to navigate through the views without reloading the full page is needed. Ui-router provides a more complete than the default way of doing this, offering functionality such as nested view and parametrized views.

***ui-bootstrap:*** Twitter Bootstrap is a CSS and JavaScript framework which aims at cutting the web application development time by reducing the amount of time dedicated to creating stylesheets [33]. This module provides angular directives and services for manipulating the elements provided by the framework.

***restangular:*** Angular offers a way of querying REST API through the *resources* module, however, it falls short when the functionality offered is more complex than simple CRUD (Create, Read, Update, Delete) operations. Restangular is a library that replaces the default module with a more complete functionality.

***flowjs:*** Uploading images and other kind of files is a harder task in SPAs than in regular web pages. This module helps in the taks of uploading files with a series of angular directives, although it can be used in other applications.

***leaflet-angular:*** Leaflet, detailed in section 3.7 is a library for creating interactive HTML5 based maps. This module is used to manipulate leaflet maps using a angular directives and services, which is more convenient than the usual way.

## Why AngularJS?

The project involves a good amount of client side JavaScript programming, so it is expected of the application to become quite difficult to manage. Due to this, there is no doubt that some kind of web application development framework is needed.

Some alternatives exist currently, such as Ember or Knockout, however Angular is the most mature of them. The huge community working with this library and the amount of exiting modules are some of the reasons that decline the balance in favor of angular. Other factors include the focus on developing a Single Page Application, which are more mobile friendly than regular web pages; and the amount of learning tools available for the framework.

## 3.7. LEAFLET

Leaflet is a JavaScript library for building mobile friendly interactive maps. It is designed with simplicity, usability and performance in mind and it works efficiently among all major mobile and desktop browsers, taking advantage of HTML5 and CSS3 when it can [32].

The library has quickly become one of the most if not the most popular mapping library since it's release in 2011, due to its small size, ease of use and extendability. The example in listing 3.14 shows the creation of a map.

```
1 var map = L.map('map');
2 var layer = L.tileLayer('http://{s}.tile.osm.org/{z}/{x}/{y}.png');
3 layer.addTo(map);
```

Listing 3.14: AngularJS example.

### Layers

Leaflet works with the concept of Layer. A layer is a visual representation of a geographic feature or set of features. There are three types of layers in the framework: tiles, vectors and layer groups.

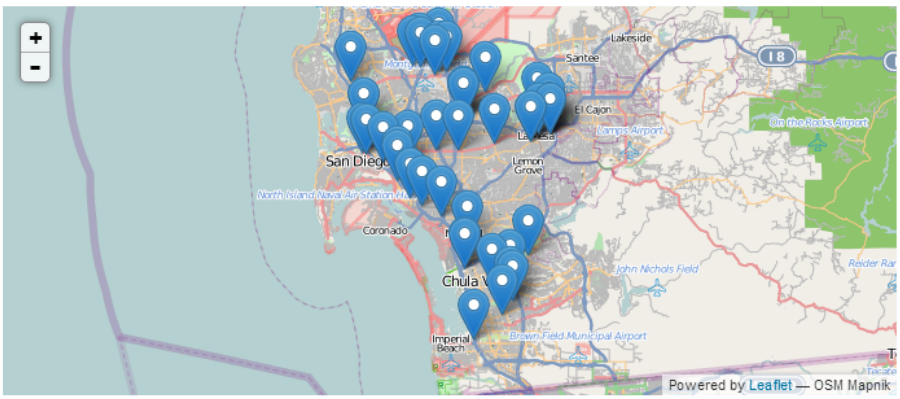


Figure 3.4: Leaflet with OSM mapnik tile layer

*source: http://drupal.org/*

**tiles:** Tiles are square bitmap graphics displayed on a grid fashion in order to show a map. Tile layers are defined by a URL from which the images will be obtained and the library will take care of displaying the appropriate grid depending on the zoom level and the view box. The tiles used change drastically the looks of the map, an example of the OSM mapnik tiles can be found in figure 3.4.

**vectors:** These layers are used to display information about features on the map in form of geometries. Vector layers can be classified into five categories, three of them corresponding to the typical vector representation on GIS systems: lines, polygons, points, rectangles and circles. Layers can be added and removed from the map easily as shown in example 3.15

**layer groups:** The last kind of layers are simply used as a convenience to manipulate more than one feature at the same time. One kind of layer group is the GeoJSON layer which allows the creation of layers from GeoJSON files, detailed in section 3.10.

```
var bounds = [[54.559322, -5.767822], [56.1210604, -3.021240]];
L.rectangle(bounds, {color: "#ff7800", weight: 1}).addTo(map);
var point = L.marker([50.5, 30.5]);
//A popup can be bound to be shown on click
point.bindPopup("This is a popup message");
```

Listing 3.15: Leaflet layers.

## Interaction

Leaflet is built with the idea of creating interactive maps, thus, it offers methods for manipulating the map and firing and receiving events. It is possible to register listener on map actions, such as when a map is moved or zoomed, or to register listeners on layers, to check when a layer is added, removed, clicked, etc.

This way, it is possible to respond to user actions via callbacks. The framework provides some shortcuts, for example, it is possible to bind the showing of a small popup to the click on a marker without the need to register a callback, as shown in listing 3.15

Another example on the usage of leaflet events can be found in listing 3.16. In the example, a callback is registered for the event of the user moving the map. This is useful for updating the map with information from the layers on the current view.

```
// The moveend event is fired every time the
// map is moved, dragged or zoomed
Map.map.on('moveend', function(){
  //Obtain the current bounding box of the map
  var bbox = Map.map.getBounds().toBBoxString();
  var feature = queryTheServer(bbox);
  map.addLayer(features);
});
```

Listing 3.16: Leaflet layers.

## Why Leaflet?

There are not many alternatives when it comes to create JavaScript based maps. Google maps can be used to embed a map on a web page, however, it does not offer much interaction and can hardly be used for other than displaying static data. OpenLayers is another alternative, however, it is known to be hard to learn and the community using it is smaller that Leaflet's one. The amount of developers using leaflet provides the framework with a good deal of extensions and its ease of use make it easy to work with even for novice developers.

## 3.8. PHONEGAP

Desktop computers are not the only devices that have seen a rise in computational power in the last years, smartphones and other mobile devices have also improved significantly.

Interaction, screens and even browser support are drastically different in desktop and mobile; and the latter still suffers from a lack of computational power. Due to this web development is not so easily applicable to these devices, building a mobile friendly web application is a difficult task.

However, due to the shift from native to web development happening on desktops, most developers have already learned the tools for developing HTML5 applications. This together with the popularity of mobile apps and difficulty of developing multi-platform applications, has given rise to a number of tools designed to code native mobile applications using HTML5, CSS3 and JavaScript.

The leading among these is Phonegap. Phonegap is a framework that allows to create mobile apps using standardized web APIs for all the major platforms [25].

### HTML5 development

The development tools used on a Phonegap application are identical to those used on a regular web page. HTML documents are used to structure the views, CSS stylesheets are used to define the appearance of the application and JavaScript is used to program the behavior.

This is possible because what the library does is opening a enhanced web view (a browser) on the mobile device and loading the files into it. Normally, this would mean that much of the sensors of the device, such as the camera and the gyroscope are not usable, however, the browser created by the framework is extended to allow access to all the hardware on the device.

Phonegap is based on Apache Cordova [18] which works with a set of plugins. These plugins provide the web view with the functionality it lacks, and expose it as JavaScript APIs. One example of this is the reading of the battery status, as shown on listing 3.17. Aside from the plugins that come shipped with the core libraries, the community has developed a considerable amount of them to provide functionalities that mobile browsers usually lack, such as HTML5 web socket support.

```
1  window.addEventListener("batterystatus", onBatteryStatus, false);
2
3  function onBatteryStatus(info) {
4      console.log("Level: " + info.level + " isPlugged: " + info.isPlugged);
5  }
```

Listing 3.17: Reading battery status with Phonegap.

### Phonegap Build

As convenient using HTML5 for developing cross platform applications is, there is still a need to configure the build for each of the platforms and to install the corresponding Software Development Kits (SDKs) which can be tedious and time consuming.

In the face of this issue, Phonegap offers a platform called Phonegap Build, a cloud based service which helps agile development and allows to compile applications for six of the seven platforms

supported by the framework. The Phonegap team illustrated this process in figure 3.5 on their web page.

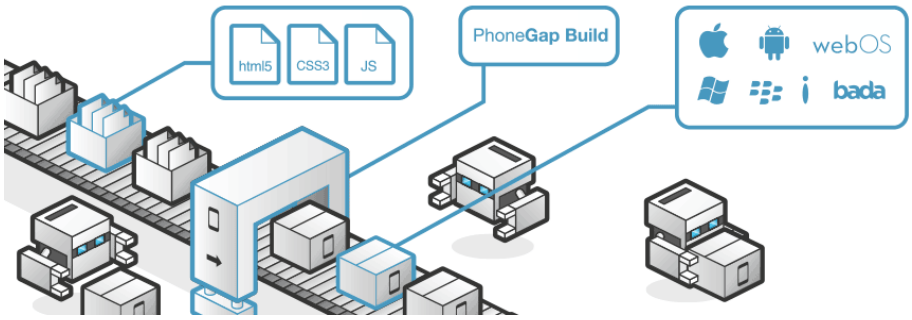


Figure 3.5: Phonegap Build process

*source:* `http://phonegap.com/about/`

### Why Phonegap?

In the world of mobile HTML5 development there are not many competitors. Phonegap is the leading platform, followed by Appcelerator and Ionic. Still, Phonegap is the most mature of them all and has the biggest community, which results in a higher amount of plugins offering more functionality.

On the other side, native development is out of consideration. One of the requirements of the project is for the mobile application to be multi-platform and while developing it natively may increase performance, the payback in development times would be too much. Besides, since the mobile application interfaces and interaction are heavily map based, the code used on other parts of the project can be reused.

Thus, the decision to use Phonegap is made, based on the cut on development times, the re usability of code and the fact that there is no need to learn other development languages.

## 3.9. GPX

GPX is a lightweight XML data format for the interchange of GPS data (tracks, routes and way points) between applications and web services on the Internet [23]. GPX was released in 2004 and has since then been the de facto standard when it comes to the encoding and interchange of GPS data.

The format represents three types of data, in the following manner:

**Way points:** A way point is a set of coordinates that identifies a point in space. They are represented using the `<wpt>` tag, as shown in listing 3.18

**Tracks:** A track is a segment containing way points, that is, a set of coordinates that describe a path. In GPX tracks are represented by the `trk` element and point inside one by `trkpt` elements (see listing 3.19).

```
<wpt lat="42.438878" lon="-71.119277">
  <ele>44.586548</ele>
  <time>2001-11-28T21:05:28Z</time>
  <name>Example waypoint</name>
</wpt>
```

Listing 3.18: GPX way point representation.

```
<trk>
  <name>Example GPX Document</name>
  <trkseg>
    <trkpt lat="47.644548" lon="-122.326897">
      <ele>4.46</ele>
      <time>2009-10-17T18:37:26Z</time>
    </trkpt>
  </trkseg>
 </trk>
```

Listing 3.19: GPX track representation.

**Routes:** A route is identical to a track in its encoding, with the difference that the tags `rte` and `rtept` are used (see listing 3.20). However, there is a difference in the meaning of this concepts; while a track is a record of where a person has been, a route is a suggestion about where someone might go in the future. For this reasons, tracks may have timestamps attached while routes will not.

```
<rte>
  <name> Example route </name>
  <rtept lat="47.644548" lon="-122.326897">
    <name>Example route point</name>
  </rtept>
</rte>
```

Listing 3.20: GPX route representation.

### Why GPX?

GPX has been the de facto standard when it comes to representing and interchanging trail and points of interest on the web for ten years. Because of this, it is unreasonable to think in a system which allows interchange of GPS data without using this format. Together with the completeness of the information provided by a GPX file, these are two solid reasons for the usage of GPX on the platform.

## 3.10. GEOJSON

GeoJSON is a format for encoding a variety of geographic data structures [13]. GeoJSON objects may represent a geometry, a feature or a collection of features (see section 3.5, GeoSPARQL). The following geometry types are supported:

- `Point`
- `LineString`

- `Polygon`
- `MultiPoint`
- `MultiLineString`
- `MultiPolygon`
- `GeometryCollection`.

More than a file format in itself, it is a specification of how to encode geographical features in JSON format. A complete GeoJSON data structure is always a JSON object, which consists in a collection of key-value pairs, being the key a string and the values any other kind of data.

The specification defines GeoJSON objects in the following way:

- A GeoJSON object may have any number of members (name/value pairs).
- The GeoJSON object may have a member with the name `type`. Its value will be a string that determines the type of the object.
- The value of the type member must be one of: `Point`, `MultiPoint`, `Polygon`, `MultiPolygon`, `LineString`, `MultiLineString`, `GeometryCollection`, `Feature` or `FeatureCollection`.
- A GeoJSON object may have an optional member of name `crs` which specifies the coordinate reference system.
- A GeoJSON object may have a `bbox` member, whose value must be a bounding box array.

## Geometries

Geometries are GeoJSON objects whose type is different from `Feature` or `FeatureCollection`. These elements, unless they are of type `GeometryCollection`, must have a member of name `coordinates` which specifies the coordinates of the object in an array of positions.

Positions are represented by an array of numbers, one for the longitude, one for the latitude and an optional one for the elevation, in that order. The coordinates of a object are formed by a array of positions, except for `Point` element, which have a single position.

A `GeometryCollection` must have a member named `geometries`, an array of GeoJSON geometry objects. An example of this can be found on listing 3.21.

```
{ "type": "GeometryCollection",
   "geometries": [
     { "type": "Point",
       "coordinates": [100.0, 0.0]
       },
     { "type": "LineString",
       "coordinates": [ [101.0, 0.0], [102.0, 1.0] ]
       }
   ]
 }
```

Listing 3.21: A GeoJSON `GeometryCollection` object.

## Features

A Feature is a object with some spatial representation. A feature must have two essential members, aside from the type, which are `geometry` and `properties`. The first of these members must contain

a GeoJSON geometry object, the spatial representation of the object.

The `properties` member on the other side, is a regular JSON object (a name/value dictionary) and contains all the non-spatial data belonging to the feature. A `FeatureCollection` does not contain a `geometry` member, instead it has a `features` object, an array of feature object in the collection. An example of a feature is shown on listing 3.22

```
{ "type": "Feature",
  "bbox": [-180.0, -90.0, 180.0, 90.0],
  "geometry": {
    "type": "Polygon",
    "coordinates": [[
      [-180.0, 10.0], [20.0, 90.0], [180.0, -5.0], [-30.0, -90.0]
      ]]
    },
    "properties": {
      "name": "Example feature"
    }
  }
```

Listing 3.22: A GeoJSON `Feature` object.

# 4. REQUIREMENTS SPECIFICATION

## 4.1. OVERVIEW

This chapter provides a specifications of the requirements, both functional and non-functional, that the project to be developed must satisfy, which define the overall functioning of the system to be produced. Each requirement will have a code associated for identification purposes.

This requirements specify the minimum possible constraints or characteristics that the system must fulfill. Most technology is kept outside of the requirement specification so that it does not affect the implementation or the design. To achieve a better understanding of these requirements, they are divided into the following sections, each corresponding to a different phase of product of the project.

- **Requirements of the ontology:** The requirements to be satisfied by the ontology as well as a minimum subset of classes and properties it must have are listed.
- **Requirements of the server:** The requirements to be satisfied by the server and the system overall are defined in this section.
- **Requirements of the web application:** The requirements to be satisfied by the web client are defined in this section.
- **Requirements of the mobile application:** The requirements to be met by the mobile client are defined in this section.
- **Non-functional requirements:** The requirements which don't specify any particular .

## 4.2. REQUIREMENTS FOR THE ONTOLOGY

The requirement for the ontology are the ones which define the characteristics that the vocabulary to be produced must meet. This requirements are listed below:

**RONT1** The ontology must support spatial reasoning.

**RONT2** The ontology must support the following inference mechanisms:

- disjoint classes
- subclasses
- subproperties
- inverse properties
- symmetric properties
- functional properties
- maximum cardinalities

**RONT3** The ontology must be designed following Linked Open Data best practices, so it should reuse existing vocabularies.

**RONT4** The ontology must represent the following types of resources:

- Features
  - Trails
  - Points of Interest
  - Geolocated Notes
- Persons
- Multimedia Resources
  - Text
  - Images
  - Video

**RONT5** The resources of type *Trail* must contain the information below:

- name
- description
- difficulty score
- maximum altitude
- minimum altitude
- ascending slope
- descending slope
- posts
- images
- author
- persons who traversed it
- spatial representation

**RONT6** The resources of type *Point of Interest* must contain the following information:

- name
- description
- altitude
- category
- posts
- images
- author
- spatial representation

**RONT7** The resources of type *Geolocated Note* must contain the following information:

- text
- multimedia
- author
- privacy level
- creation time
- duration
- action radius

- spatial representation

**RONT8** Resources of type *Person* must contain the following information:

- nickname
- email
- first name
- family name
- avatar
- description
- trail buddies
- optionally external homepage
- traversed trails
- added trails
- added points of interest
- added notes
- added multimedia

**RONT9** Multimedia resources of type *Text* must contain the following information.

- author
- addition date
- content
- feature they belong to

**RONT10** Multimedia resources of type *Image* and *Video* must contain the following information.

- author
- addition date
- download link
- feature they belong to

## 4.3. REQUIREMENTS SPECIFICATION FOR THE SERVER

The requirements for the server specify the functionality that the server must implement as well as the function and resources exposed by the API and overall inter-component communication, for the server is the central piece of the system.

**RSV1** The system must be able to operate on Points.

**RSV2** The system must be able to operate on LineStrings.

**RSV3** The server must be able to obtain the following data from a LineString representing a trail.

- distance
- difficulty
- ascending slope
- descending slope
- maximum altitude
- minimum altitude

**RSV4** The system must be able to create and send SPARQL queries to a data store.

**RSV5** The system must be able to retrieve, transform and aggregate it to the dataset, spatial data from the following sources:

- OpenStreetMap
- Geonames

**RSV6** The server must expose a SPARQL read-only endpoint.

**RSV7** The server must expose a public API.

**RSV8** The public API must follow the REST style.

**RSV9** The API must expose the following resources

- Trails
- Points of Interest
- Geolocated Notes
- Users

**RSV10** The resources must expose the following operations:

- Read
- Update (Except Geolocated Notes)
- Create

**RSV11** The resources must expose information about other related resources, such as posts, images, etc.

**RSV12** The API must offer additional operations which expose the following functionality:

- Search
- Features within a area
- Features near each other
- Information about the system
- Information about the API

**RSV13** The server must offer secure authentication.

**RSV14** The system must be able to provide recommendations based on preferences and location.

**RSV15** The system must be able to store user preferences, for recommendation and filtering purposes.

**RSV16** Signing in the system will be done via e-mail and password.

## 4.4. REQUIREMENTS OF THE WEB APPLICATION

The requirements for the web application detail the constraints that the web based client must fulfill, as well as the functionality that need to be implemented.

**RWEB1** The web application must work on most major browsers. The minimum browsers specified are the following:

- Google Chrome 26
- Mozilla Firefox 22
- Safari 6
- Internet Explorer 10
- Opera 12

**RWEB2** The web application must communicate with the server using the public API.

**RWEB3** The web application will have a homepage presenting the platform and linking to the rest of the sections.

**RWEB4** The web application must have a section allowing the following features:

- Explore data based on its location
- Draw and edit trails and points of interest

**RWEB5** The web application must have a section which provides the following functionality.

- Search over the data on the system
- Upload a trail or point of interest from a GPX file
- View detailed information about any data on the platform

**RWEB6** The web application must have a section which provides the following features:

- Register a user on the system
- Log in the system
- Modify the profile of a user

**RWEB7** All geographic data must be shown on a interactive map.

**RWEB8** All operations that require manipulation of spatial data have to be performed without any need of GIS knowledge from the user.

**RWEB9** Signing in only has to be needed for write operations.

**RWEB10** Registering a user via web platform will require the following information:

- User name
- Email
- Password

**RWEB11** Creating a trail via web platform will require the following information:

- Name
- Description
- Geographic representation

**RWEB12** Creating a point of interest via web platform will require the following information:

- Name
- Description
- Category
- Geographic representation

## 4.5. REQUIREMENTS OF THE MOBILE APPLICATION

The requirements to the mobile specify the operations that can be done with the mobile application and the functionality it must implement.

**RMOB1** The application must allow the recording of a trail.

**RMOB2** The application must allow exporting trails in GPX format or uploading them to the system when a network connection is available.

**RMOB3** The application must allow the creation of Geolocated Notes on the current position of the user.

**RMOB4** The application must keep track of the current position of the user at all times.

**RMOB5** The application must be able to receive real time notifications of nearby features.

**RMOB6** The application must be able to notify the user when a new feature is discovered, even if it is not currently active.

**RMOB7** The application must allow search and detailed view of information on the system.

**RMOB8** The application must provide log in and registration functionalities.

**RMOB9** The application can only be used by registered users.

**RMOB10** Registering a user via mobile platform will require the following information:

- User name
- Email
- Password

**RMOB11** Creating a trail via mobile platform will require the following information:

- Name
- Description
- Recording of the trail

**RMOB11** Creating a geolocated note via web mobile platform will require the following information:

- Text, Image or video
- Privacy level
- Range
- Coordinates

**RMOB12** Once signed in the mobile application, the system will store the password and username, for convenience purposes.

## 4.6. NON-FUNCTIONAL REQUIREMENTS

The requirements not belonging to a specific category are listed in this sections. This refers to information visibility and privacy, performance and user interaction issues mainly.

**RNF1** All the information, except a few sensible data, has to be published according to Linked Open Data best practices.

**RNF2** The following data must be kept private:

- User passwords
- The current location of the users

**RNF3** The server should be able to handle high loads of connection.

**RNF4** The interface on the web application must be responsive, that is, the interface must adapt to mobile and tablet devices.

**RNF5** The web application must have a fast loading time and small memory footprint, in order to be adapted to mobile devices.

**RNF6** The applications must have intuitive and easy to use and read interfaces.

**RNF7** The style of the mobile and web applications must be similar.

# 5. DESIGN SPECIFICATION

## 5.1. OVERVIEW

This section describes the design work done through the project as well as the tools used to carry that labor and the results of it. The chapter covers the following topics:

- **Architecture of the system:** The chosen architecture of the system is described, together with the reasons and advantages of the design.
- **Design of the ontology:** The final design of the ontology is described, as well as it's advantages in front of other data modeling paradigms.
- **Design of the central server:** The class design of the server is detailed in this section.
- **Design of the web application:** The class design of the web application is detailed in this section, as well as the interface design.
- **Design of the mobile application:** The class design of the mobile application is detailed in this section, as well as the interface design.
- **Development environment:** The list of technologies employed in the project is detailed, briefly explaining the role of each tool.

## 5.2. SYSTEM ARCHITECTURE

Figure 5.1 shows the architecture chosen for the system. This architecture comprises the following elements:

- A central server
  - Web server
  - SPARQL endpoint
  - REST API
  - Data agregator
  - Database connector
- A semantic spatial storage system which contains the ontology
- A web client used as the main interface of the system
- A mobile application to support the browser based one

### 5.2.1 Central server

The central server is the core component of the system. It implements most of the functionality on the system and provides data to the client. It also analyzes the trails uploaded to the system

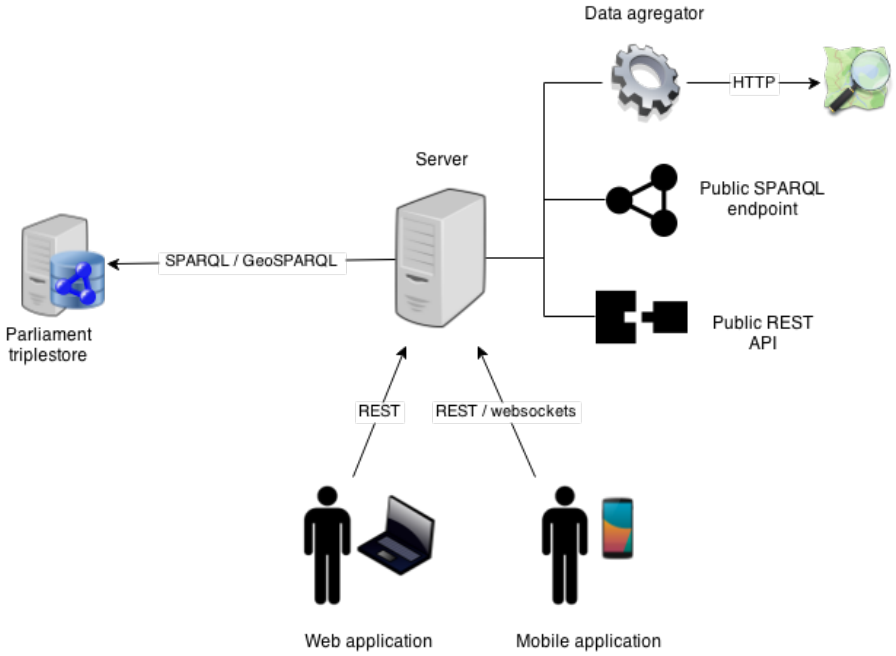


Figure 5.1: Architecture design of the system

and takes care of the communication with the database. The server is divided into several logical software pieces, which are described below.

### Database connector

The database connector implements the functionality needed to connect with the semantic data store. In essence, this means that it takes the requests from the users and transforms them into SPARQL queries. Then it takes the results provided by the database and converts them into a format that the server can process.

### The data agregator

This component has the function of querying external data sources and adapting the results to the data model of the system. It is independent of the rest of components of the server, since instead of being available to receive requests it will just run on demand. The component makes HTTP queries to the different APIs and endpoints on the web, for example OpenStreetMap and Geonames, and aggregates the data to the database.

### REST API

The API provides the basic means of accessing the information on the system. It is used by both clients to communicate with the server. It can be used for read, write and update operations, however, delete operations are not contemplated.

#### The SPARQL endpoint

This endpoint just routes SPARQL queries done to it to the database. Its role is related to filtering the queries that seek to write and to ensure that no petition is going to break the integrity of the data in the database.

### 5.2.2 Semantic spatial storage system

This component of the system is mostly third party software. It consists on a triple store, a data storage system that stores RDF data; a reasoner supporting OWL and GeoSPARQL and an HTTP SPARQL endpoint. This component also englobes the ontology designed to define the data model of the system. This ontology is described in detail in section 5.3.

### 5.2.3 Web client

The web client is the main interface though which the system is accessed. It communicates with the server through a REST API and offers most of the functionality of the system to the users.

The client may be used from desktop browsers or mobile browser with no restrictions. More details on the design of the client can be found on section **??**.

### 5.2.4 Mobile application

The mobile client provides the functionality that the web application cannot. It provides real time functions by using a websocket API that the server exposes. The design of this component is detailed in section **??**.

## 5.3. DESIGN OF THE ONTOLOGY

The ontology designed in the project represents specific knowledge related to the domain of Trails, Points of Interest and GPS data in general. It uses external ontologies as a base, in order to ease the sharing of information and the publishing of the dataset at Linked Open Data.

Figure 5.2 provides a graphical representation of classes on the SORELCOM ontology. As shown on the figure, classes from three vocabularies are reused. The FOAF vocabulary is used to obtain a representation of the users of the platform, the GeoSPARQL vocabulary is used as a base for all the resources with a spatial character and the W3C media vocabulary is used to represent the media resources on the platform.

The focus of the design is to reuse as many well known vocabularies as possible and defining the needed classes, relations and properties to satisfy the requirements of the ontology 4 to 10, defined in chapter 4.

### 5.3.1 Class hierarchy

The ontology defines a series of classes for the representation of specific spatial features present in the domain of the application. The classes are listed below:

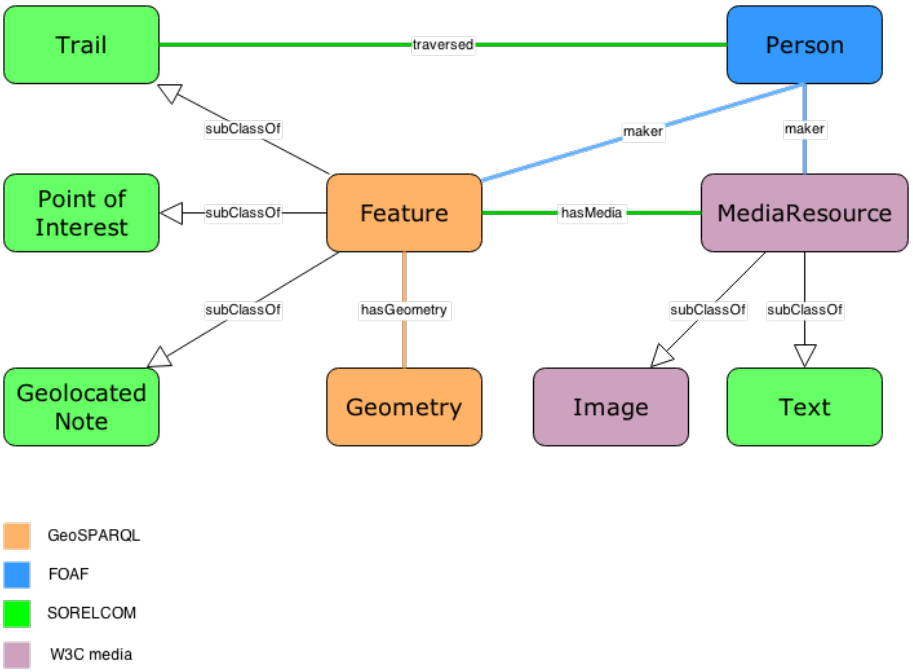


Figure 5.2: The SORELCOM ontology

**`sorelcom:Trail`** A trail represents a path in the world. In the context of the application, a trail can be interpreted as a path that a user has traversed or as a simple route that is indicated by a user, however, the ontology does not specify that a trail should have any of these meanings.

**`sorelcom:PointOfInterest`** A Point of Interest is a feature in the world that may result of some interest for a person. There is no specification on what can be the interest a person may have on the feature, thus a point of interest may be anything from a monument to a restaurant.

**`sorelcom:GeolocatedNote`** A Geolocated Note is a message with a spatial context. This message is left by a person at a specific location, and can be viewed by any person or just a specific group depending on its privacy settings. The message on a note may be formed by text, images, video or any combination of them.

In addition to the core types on the vocabulary, other classes obtained from external ontologies will be used or extended to represent different types of resources on the ontology. The additional classes used are the following:

**`foaf:Person`** This class is used to classify resources as persons. Any user on the system will be classified as person. The class is defined in the FOAF vocabulary.

**`geo:Feature`** A feature is any object in the real world that has a spatial representation. *Trails*, *Points of Interest* and *Geolocated Notes* are all types of Features. This class is defined in the GeoSPARQL vocabulary.

**`geo:Geometry`** In the GeoSPARQL ontology, a geometry is used to give a spatial representation to a Feature. Without a associated feature, geometries are not resources on the real world.

`media:MediaResource` A media resource is a representation of a media object, such as a image or a video. Media resources refer to the images, posts and videos saved in the system.

`media:Image` A image is a specialized media resource, which refers to a image. It is used in conjunction with the `sorelcom:Text` class to represent the specific type of a media in the system.

### 5.3.2 Properties

Most of the work on the design of the ontology consists on the creation of properties which relate the resources to the information that they must contain. Still, many of the properties used in the data model of the platform are obtained from external vocabularies.

The properties defined for the SORELCOM vocabulary are listed below, together with a table representing how the information about those resources is stored in the data model:

#### Feature properties

`sorelcom:name` Property representing the human readable name of a feature.

`sorelcom:description` Property representing a textual description of a feature.

`sorelcom:hasMedia` Property relating a feature to the media associated. It is the inverse of `sorelcom:mediaOf`.

#### Trail properties

`sorelcom:difficulty` Property representing the difficulty score of a trail. This score is a integer from 0 to 100, the higher the difficulty the higher the number. A trail may only have one difficulty.

`sorelcom:maximumAltitude` Property representing the altitude of the highest point of a trail.

`sorelcom:minimumAltitude` Property representing the altitude of the lowest point of a trail.

`sorelcom:totalDistance` Property representing the total distance in meters that must be traversed from the starting point of a trail to the end of it.

`sorelcom:ascendingDistance` Property representing the sum of the distance in meters of the segments of the trail which are ascendant .

`sorelcom:descendingDistance` Property representing the sum of the distance in meters of the segments of the trail which are descendant.

`sorelcom:circular` Property representing if the starting point and end point of a trail are the same.

`traversedBy` Property relating a trail to the users who have traversed it.

Information about how trails are represented on the system can be found on table 5.1.

Table 5.1: Trails on the SORELCOM data model.

| **Property** | *domain* | *range* | *information* |
|---|---|---|---|
| sorelcom:name | geo:Feature | string | name |
| sorelcom:description | geo:Feature | string | description |
| sorelcom:difficulty | sorelcom:Trail | integer | difficulty |
| sorelcom:maximumAltitude | sorelcom:Trail | float | maximum altitude |
| sorelcom:minimumAltitude | sorelcom:Trail | float | minimum altitude |
| sorelcom:ascendingDistance | sorelcom:Trail | integer | ascending distance |
| sorelcom:descendingDistance | sorelcom:Trail | integer | descending distance |
| sorelcom:hasMedia | geo:Feature | ma:Media | images, posts |
| foaf:maker | Thing | foaf:Person | author |
| sorelcom:traversedBy | sorelcom:Trail | foaf:Person | persons who traversed it |
| geo:hasGeometry | geo:Feature | geo:Geometry | spatial representation |

### Point of interest properties

`sorelcom:altitude` Property representing altitude of the point.

`sorelcom:category` Property representing the category of the point of interest.

Information about how points of interest are represented on the system can be found on table 5.2.

Table 5.2: Points of interest on the SORELCOM data model.

| **Property** | *domain* | *range* | *information* |
|---|---|---|---|
| sorelcom:name | geo:Feature | string | name |
| sorelcom:description | geo:Feature | string | description |
| sorelcom:altitude | sorelcom:PointOfInterest | float | altitude |
| sorelcom:category | sorelcom:PointOfInterest | string | category |
| sorelcom:hasMedia | geo:Feature | ma:Media | images, posts |
| foaf:maker | Thing | foaf:Person | author |
| geo:hasGeometry | geo:Feature | geo:Geometry | spatial representation |

### Geolocated Note properties

`sorelcom:range` Property representing range of action of a Geolocated Note.

`sorelcom:public` Property representing if the note is public.

`sorelcom:targets` Property relating the note to the persons that should receive it. It exists only when the note is not public.

Information about how geolocated notes are represented on the system can be found on table 5.3.

Table 5.3: Geolocated notes on the SORELCOM data model.

| **Property** | *domain* | *range* | *information* |
|---|---|---|---|
| dcterms:created | Thing | date | creation time |
| dcterms:valid | Thing | date or integer | duration |
| sorelcom:public | sorelcom:GeolocatedNote | boolean | privacy level |
| sorelcom:radius | sorelcom:GeolocatedNote | integer | action radius |
| sorelcom:hasMedia | geo:Feature | ma:Media | text, multimedia |
| foaf:maker | Thing | foaf:Person | author |
| geo:hasGeometry | geo:Feature | geo:Geometry | spatial representation |

#### Person properties

`sorelcom:hasTraversed` Property relating a Person to the trails it has traversed.

`sorelcom:trailBuddyOf` Property relating a Person to his/her trail buddies, that is, people who are usual sports or tourism mates.

Information about how persons are represented on the system can be found on table 5.4.

Table 5.4: Persons on the SORELCOM data model.

| **Property** | *domain* | *range* | *information* |
|---|---|---|---|
| foaf:nick | foaf:Person | string | nickname |
| foaf:mbox | foaf:Person | string | email |
| foaf:firstName | foaf:Person | string | first name |
| foaf:familyName | foaf:Person | string | family name |
| foaf:depiction | foaf:Person | foaf:Image | avatar |
| sorelcom:trailBuddyOf | foaf:Person | foaf:Person | trail buddies |
| sorelcom:hasTraversed | foaf:Person | sorelcom:Trail | traversed trails |
| foaf:weblog | foaf:Person | URI | external homepage |
| foaf:made | foaf:Person | Thing | features and media |

#### Media properties

`sorelcom:mediaOf` Property relating a media resource to the feature it belongs to.

Information about how media resources are represented on the system can be found on table 5.5. Media resources are divided into images, text and video, however, for convenience purposes all of them are presented in a single table. The only properties that are not shared among all media resource types are the content and the download link. The content is only used in the text, while the download link is used on media images and video.

### Inference mechanism

In order to get the maximum benefit from the semantic dataset the ontology has been defined using OWL. Due to this, several inference mechanism have been available during the design process,

Table 5.5: Media on the SORELCOM data model.

| **Property** | *domain* | *range* | *information* |
|---|---|---|---|
| foaf:maker | Thing | foaf:Person | author |
| ma:description | ma:MediaResource | string | content |
| ma:locator | ma:MediaResource | URI | download link |
| dcterms:created | Thing | datetime | addition date |
| sorelcom:mediaOf | ma:MediaResource | geo:Feature | feature |

however, only a subset of them have been used. The following inference mechanisms have been used:

**Sub classes** All resources which are instanced of a certain class A are also instances of the classes A is subclass of. This property is provided by RDF schema (see section 3.2 for more information). In RDF it is possible to use multiple inheritance, meaning that a class can be subclass of more than one class.

**Disjoint classes** A group of disjoint classes indicate that a resource of one class of the group cannot be of another class on the group

**Sub properties** Subproperty relations are the analogous of subclass relations when referring to resources of type property. It is also provided by RDF schema.

**Function properties** A functional property is a property that can only appear once in the triples of a certain resource. It is provided by OWL lite.

**Inverse properties** When a resource A is related to B by a property P1, then B is related to A by the property P2 inverse of P1. It is provided by OWL lite.

**Symmetric property** When a resource A is related to B by a symmetric property P, then B is related to A also by P.

Table 5.6 shows the inference rules used on the classes of the ontology and 5.7 shows the inference rules used on the properties of the data model.

Table 5.6: Inferences used on the SORELCOM ontology classes.

| **Class** | *inference type* | *Related class* |
|---|---|---|
| sorelcom:Trail | subclass of | geo:Feature |
| sorelcom:PointOfInterest | subclass of | geo:Feature |
| sorelcom:GeolocatedNote | subclass of | geo:Feature |
| sorelcom:Text | subclass of | ma:MediaResource |
| sorelcom:Trail | disjoint with | sorelcom:PointOfInterest |
| sorelcom:Trail | disjoint with | sorelcom:GeolocatedNote |
| sorelcom:GeolocatedNote | disjoint with | sorelcom:PointOfInterest |

Table 5.7: Inferences used on the SORELCOM ontology classes.

| **Property** | *inference type* | *Related property* |
|---|---|---|
| sorelcom:trailBuddyOf | sub property of | foaf:knows |
| sorelcom:trailBuddyOf | symmetric property | |
| sorelcom:hasTraversed | inverse property of | sorelcom:traversedBy |
| sorelcom:hasMedia | inverse property of | sorelcom:mediaOf |
| sorelcom:difficulty | functional property | |
| sorelcom:maximumAltitude | functional property | |
| sorelcom:minimumAltitude | functional property | |
| sorelcom:totalDistance | functional property | |

### 5.3.3 Advantages of ontology based design

The data model on the platform has been designed based on a ontology, mainly to follow the best practices of Linked Open Data. However, there are other alternatives, for instance, using a relational database and a RDF mapper. Mappers, for example, D2RQ expose the data on a relational database as a virtual RDF graph, allowing to query this data using SPARQL. This mappers exist to allow Semantic Web application to access the information stored on relational systems [8].

Even with this alternatives, the choice to use a completely semantic system over a relational database was made. This choice came with some disadvantages, the main of them being an increased of the complexity of the system. RDF lacks support on programming platforms compared to relational database systems, thus the amount of coding needed increases. Besides, the current standard for the creation of GIS systems, the PostGIS [40] extension for the PostgreSQL [41] cannot be used. The rest of the spatial databases, including the semantic ones lack support and tools to work with, which has caused an increase in development times.

However using ontologies to define the data model of the platform brings several benefits. The main benefits are detailed in this section.

#### Inference of new knowledge from the existing one

The main advantage of the ontology based data model design is the possibility of inferring knowledge, that is, creating new knowledge from the existing data. This process usually implies the creation on new triples from the existing triples on the dataset which can be done in two different manners. The first way consists on running the inference process every time a query is made and aggregating the new triples to the results returned. This saves space on the data store but in exchange it increases computation time, for which it is not a very used technique. The more popular approach is running the inference when data is added to the store and generating new statements which will also be saved. This increases space used and time for processing write operation, however, since read operations are usually way more common than read operations it pays off.

This inference is usually done by specialized reasoning software such as the Pellet OWL 2 reasoner [38]. The data store to be used, Parliament, provides its own reasoner. Parliament's reasoner supports OWL DL inference done when triples are inserted into the data store. The process exploits

inference properties like the ones detailed on the previous section to deduce new knowledge, for example, if a property is specified to have a certain domain, then when a triple with that property is inserted into the dataset it is possible to infer that the subject is of the types on the domain of the property. A graphical example of a simple inference can be found in figure 5.3. This example, shows how using inverse relation properties it is possible to obtain inverse properties needing only to specify one end of the relation.

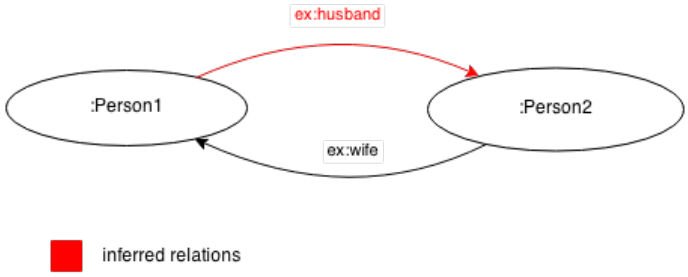


Figure 5.3: Inference of the husband relation as inverse of the wife relation

Many types of inferences are possible. The inference of the classes of a resource depending on a class hierarchy can be done; the example in figure 5.4 shows the usage of the SORELCOM ontology to infer the classes of a resource. Combinations of different inference properties can be used to express complex relations. In figure 5.5 inverse and transitive properties together with ranges and domains, are combined to infer knowledge on a small RDF graph.

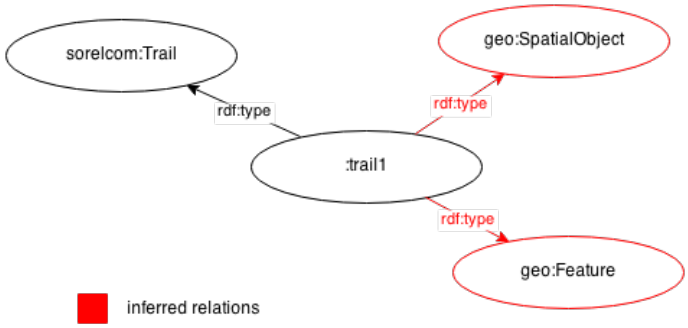


Figure 5.4: Inference of all the classes of a Trail

Depending on the describing language used, it is possible to create very complex rules, and the inference engine will automatically process all of them. This is one of the biggest advantages over relational database systems where a high amount of relations would need to be defined to encode the complex relations that can be represented in a ontology based system and allows to build complex queries.

### Schema and data separation and reusability

Ontologies provides mean for reusing other data models, since it is possible to import ontologies into another one. This way, it is possible to use already defined concepts without the need to develop vocabularies for them. One example of this can be found in the SORELCOM vocabulary, where the FOAF, GeoSPARQL, DC and W3C media ontologies are reused.

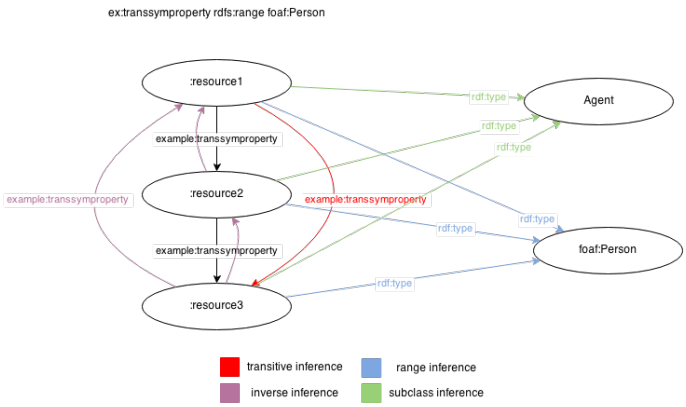


Figure 5.5: Inference of the husband relation as inverse of the wife relation

Besides, this allows separation between the data model and the actual data. In relational database management systems, the structure of the data is physically created on the system itself (for example, a file for each table) and it is not fully supported to import the model of one database to another. When using ontology based databases, the data model is represented by a ontology which can be imported into any data store, thus it is possible to reuse the schema among several different databases.

### Linked Open data

Ontologies are a tool that facilitate the publishing of Linked Open Data. The inference mechanisms used on these ontologies are not so relevant in this case, however the concepts expressed are crucial. Ontologies allow reusing other vocabularies to express the data model, which is a key point when developing a linked model.

Linked data refers to a set of best practices to publish and interlink structured data for access by both humans and machines via RDF. In order to do this it is necessary to make use of URIs as a real unique identifier, there is no point if a relation that expresses that two persons know each other has a different URI in every dataset. In order to avoid this, well known vocabularies, such as FOAF, and these concepts and relations are uniform among most if not all datasets. This is only possible when using ontology based design, for only ontologies allow this level of reusing.

## 5.4. DEVELOPMENT ENVIRONMENT

The tools and development environment used during the design process have been the following:

- Protégé-OWL 4.3
- UML 2
- Sass 3.8.8
- Twitter Bootstrap 3.1.1

### 5.4.1 Protégé-OWL

Protégé is an open-source ontology editor and framework for building intelligent systems[42]. The traditional architecture of the editor is based on frames [36], however, with the standardization of OWL (see section 3.3) support for this language was added to the framework [30].

The current version of Protégé can be used to edit classes and their characteristics, to access different reasoning engines, to edit and execute queries and rules and to visualize relationships between concepts. The tool has been widely adopted by the Semantic Web ontologists, for all their utilities and because it has been adopted among the W3C recommendations [31].

The tool provides a graphical user interface, which can be seen in figure 5.6. Through this editor, other vocabularies can be imported, in addition to create classes and properties for a new ontology and defining restrictions and relations among them. This tool has been used to create the SORELCOM ontology on the project.

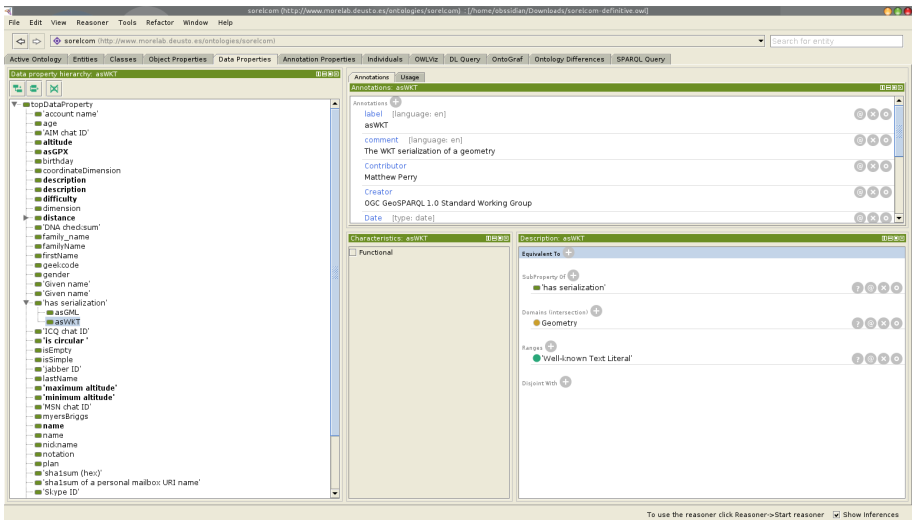


Figure 5.6: The Protégé OWL ontology editor

### 5.4.2 UML

The Unified Modeling Language (UML) is a general-purpose visual modeling language used to specify, visualize, construct and document the artifacts of a software system. It is used to capture decisions and provide understanding of the architecture and class design of the system [46].

UML can be used to capture information about the static design and dynamic behavior of the system and its components through different types of diagram. It is a standard used and accepted worldwide, thanks to which it can be used to expose and interchange information about system designs.

All the component and class diagrams presented in this document have been modeled according to UML standards.

### 5.4.3 SASS

CSS (Cascading Style Sheet) is a stylesheet language specification used to describe the presentation of documents written in a markup language. It is the technology used to describe the appearance of web pages [44]. SASS [14] which stands for Syntactically Awesome Style Sheets is a extension and preprocessor for the CSS language which aims to cut on developing times and improve stylesheet maintainability.

CSS preprocessors have become very popular lately among web developers for various reasons. First, they provide functionality that allows to programatically define styles for a web page instead of just describing, such as variables and functions. Second, these extensions improve code readability and reduce the size of the produced stylesheets. Finally, they follow the principle of DRY (dont repeat yourself), allowing to define the appearance of document without repeating directives. An example of SASS to CSS transformation can be found in figure 5.7.

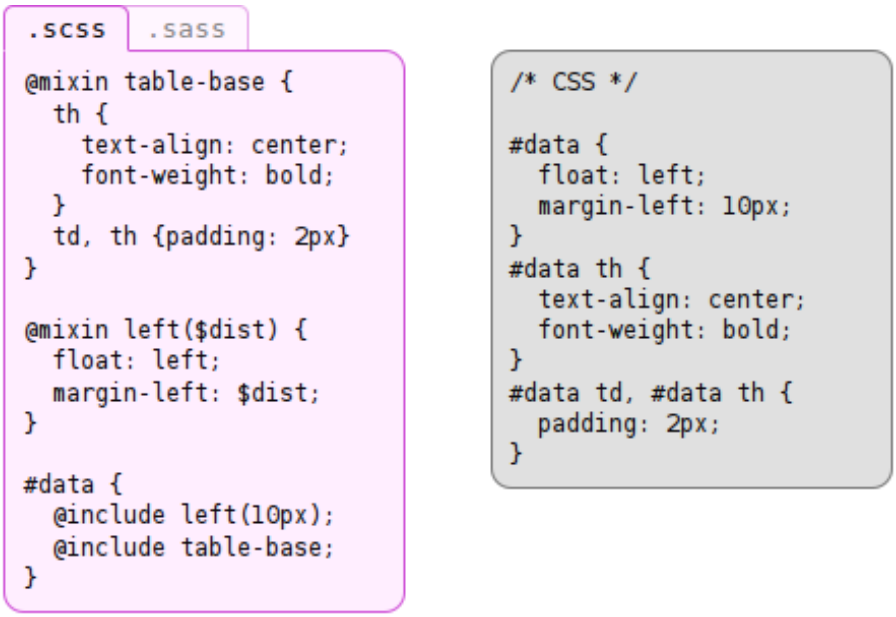


Figure 5.7: SASS to CSS transformation

This technology has been used in the project to define the appearance of both the mobile and the web interface.

### 5.4.4 Twitter Bootstrap

Even with the use of preprocessors, the time it takes to design the appearance of a HTML based application is considerable. In big enough project, specialized designers are employed for the interface design of the application, however on smaller scale projects it has become more usual to use a CSS framework to produce acceptable, if not original, appearances for the user interfaces.

Twitter Boostrap is one of the most, is not the most popular of these frameworks. It provides a set of CSS classes, default styles and JavaScript functions to develop responsive web applications. It has been used in this project to ease the adaptability of the web application interface on all kind of devices.

# 6. IMPLEMENTATION DETAILS

# 7. TESTING

# 8. USER GUIDE

# 9. CONCLUSIONS AND FUTURE LINES OF WORK

# Bibliography


[1] Robert Battle and Dave Kolas. “Enabling the geospatial semantic web with Parliament and GeoSPARQL”. In: *Semantic Web* 3.4 (2012), pages 355–370.

[2] Robert Battle and Dave Kolas. “Linking geospatial data with GeoSPARQL”. In: *Semantic Web Journal of Interoperability, Usability, Applicability* 24 (2011).

[3] Sean Bechhofer. “OWL: Web ontology language”. In: *Encyclopedia of Database Systems*. Springer, 2009, pages 2008–2009.

[4] Tim Berners-Lee, Roy Fielding, and Larry Masinter. “RFC 3986: Uniform resource identifier (uri): Generic syntax”. In: *The Internet Society* (2005).

[5] Tim Berners-Lee, James Hendler, Ora Lassila, et al. “The semantic web”. In: *Scientific american* 284.5 (2001), pages 28–37.

[6] Christian Bizer, Tom Heath, and Tim Berners-Lee. “Linked data: Principles and state of the art”. In: *World Wide Web Conference*. 2008.

[7] Christian Bizer, Tom Heath, and Tim Berners-Lee. “Linked data-the story so far”. In: *International journal on semantic web and information systems* 5.3 (2009), pages 1–22.

[8] Christian Bizer and Andy Seaborne. “D2RQ-treating non-RDF databases as virtual RDF graphs”. In: *Proceedings of the 3rd international semantic web conference (ISWC2004)*. Volume 2004. 2004.

[9] *Bower - A package manager for the web*. URL: `http://bower.io/` (visited on 06/17/2014).

[10] Dan Brickley and Ramanathan V Guha. “{RDF vocabulary description language 1.0: RDF schema}”. In: (2004).

[11] Dan Brickley and Libby Miller. “FOAF vocabulary specification 0.98”. In: *Namespace Document* 9 (2012).

[12] Jeen Broekstra, Michel Klein, Stefan Decker, Dieter Fensel, Frank Van Harmelen, and Ian Horrocks. “Enabling knowledge representation on the web by extending RDF schema”. In: *Computer networks* 39.5 (2002), pages 609–634.

[13] Howard Butler, Martin Daly, Allan Doyle, Sean Gillies, Tim Schaub, and Christopher Schmidt. *The GeoJSON Format Specification*. 2008.

[14] Hampton Catlin and Michael Lintorn Catlin. *Pragmatic guide to Sass*. Pragmatic Bookshelf, 2011.

[15] Balakrishnan Chandrasekaran, John R Josephson, V Richard Benjamins, et al. “What are ontologies, and why do we need them?” In: *IEEE Intelligent systems* 14.1 (1999), pages 20–26.

[16] Kendall Grant Clark, Lee Feigenbaum, and Elias Torres. “SPARQL protocol for RDF”. In: *World Wide Web Consortium (W3C) Recommendation* (2008).

[17] Open Geospatial Consortium et al. "OpenGIS® Implementation Specification for Geographic information-Simple feature access-Part 1: Common architecture". In: *OGC document* (2011).

[18] Apache Cordova. *About Apache Cordova*. 2013.

[19] Simon Cox, Adrian Cuthbert, Paul Daisey, John Davidson, Sandra Johnson, Edric Keighan, Ron Lake, Marwa Mabrouk, Serge Margoulies, Richard Martell, et al. "OpenGIS® Geography Markup Language (GML) Implementation Specification, version". In: (2002).

[20] Peter Bacon Darwin and Pawel Kozlowski. *AngularJS web application development*. Packt Publishing, 2013.

[21] Martin Dürst and Michel Suignard. *Internationalized resource identifiers (IRIs)*. Technical report. RFC 3987, January, 2005.

[22] ESRI. *What is GIS - Overview*. URL: `http://www.esri.com/what-is-gis/overview#overview_panel` (visited on 06/02/2014).

[23] Dan Foster. "GPX: the GPS exchange format". In: *Retrieved on* 2 (2007).

[24] Adam Freeman. "The Anatomy of an AngularJS App". In: *Pro AngularJS*. Springer, 2014, pages 207–231.

[25] Rohit Ghatol and Yogesh Patel. *Beginning PhoneGap: mobile web framework for JavaScript and Html5*. Apress, 2012.

[26] *GIS Wiki - GIS*. URL: `http://wiki.gis.com/wiki/index.php/GIS` (visited on 06/02/2014).

[27] Brad Green and Shyam Seshadri. *AngularJS*. O'Reilly Media, Inc., 2013.

[28] Tom Gruber. *What is an Ontology*. 1993.

[29] Dublin Core Metadata Initiative et al. "Dublin core metadata element set, version 1.1". In: (2008).

[30] Holger Knublauch, Ray W Fergerson, Natalya F Noy, and Mark A Musen. "The Protégé OWL plugin: An open development environment for semantic web applications". In: *The Semantic Web–ISWC 2004*. Springer, 2004, pages 229–243.

[31] Holger Knublauch, Matthew Horridge, Mark A Musen, Alan L Rector, Robert Stevens, Nick Drummond, Phillip W Lord, Natalya Fridman Noy, Julian Seidenberg, and Hai Wang. "The Protege OWL Experience." In: *OWLED*. 2005.

[32] *Leaflet - An Open-Source JavaScript Library for Mobile-Friendly Interactive Maps*. URL: `https://leafletjs.com/` (visited on 06/15/2014).

[33] Reuven M Lerner. "At the forge: Twitter bootstrap". In: *Linux Journal* 2012.218 (2012), page 6.

[34] Frank Manola, Eric Miller, Brian McBride, et al. "RDF primer". In: *W3C recommendation* 10 (2004), pages 1–107.

[35] Deborah L McGuinness, Frank Van Harmelen, et al. "OWL web ontology language overview". In: *W3C recommendation* 10.2004-03 (2004), page 10.

[36] Natalya Fridman Noy, Ray W Fergerson, and Mark A Musen. "The knowledge model of Protege-2000: Combining interoperability and flexibility". In: *Knowledge Engineering and Knowledge Management Methods, Models, and Tools*. Springer, 2000, pages 17–32.

[37] XML Schema Part. "XML Schema - 2: Datatypes". In: *W3C Recommendation* 2 (2001).

[38] *Pellet: OWL 2 Reasoner for Java*. URL: `http://clarkparsia.com/pellet/` (visited on 06/18/2014).

[39] Matthew Perry and John Herring. "OGC geosparql, a geographic query language for rdf data". In: *OGC Implementation Standard, ref: OGC* (2011).

[40] *PostGIS - Spatial and Geographic Object for PostgreSQL*. URL: `http://postgis.net/` (visited on 06/18/2014).

[41] *PostgreSQL: The world's most advanced open source database*. URL: `http://www.postgresql.org/` (visited on 06/18/2014).

[42] *Protégé*. URL: `http://protege.stanford.edu/` (visited on 06/18/2014).

[43] Eric Prud'Hommeaux, Andy Seaborne, et al. "SPARQL query language for RDF". In: *W3C recommendation* 15 (2008).

[44] Duncan Reed and Peter J Thomas. "Cascading Style Sheets". In: *Essential HTML fast*. Springer, 1998, pages 111–121.

[45] Jason Ronallo. "HTML5 Microdata and Schema. org." In: *Code4Lib Journal* 16 (2011).

[46] James Rumbaugh, Ivar Jacobson, and Grady Booch. *Unified Modeling Language Reference Manual, The*. Pearson Higher Education, 2004.

[47] Andy Seaborne, Geetha Manjunath, Chris Bizer, John Breslin, Souripriya Das, Ian Davis, Steve Harris, Kingsley Idehen, Olivier Corby, Kjetil Kjernsmo, et al. "SPARQL/Update: A language for updating RDF graphs". In: *W3C Member Submission* 15 (2008).

[48] SemWebCentral. *Parliament homepage*. URL: `http://parliament.semwebcentral.org/` (visited on 06/08/2014).

[49] Evren Sirin and Bijan Parsia. "Pellet: An OWL DL reasoner". In: *Proc. of the 2004 Description Logic Workshop (DL 2004)*. 2004, pages 212–213.

[50] *The Linking Open Data project*. URL: `http://www.w3.org/wiki/SweoIG/TaskForces/CommunityProjects/LinkingOpenData` (visited on 06/18/2014).

[51] Stefan Tilkov and Steve Vinoski. "Node.js: Using JavaScript to Build High-Performance Network Programs." In: *IEEE Internet Computing* 14.6 (2010).

[52] *v8 Javascript Engine*. URL: `https://code.google.com/p/v8/` (visited on 06/14/2014).

[53] W3C. *Ontologies - W3C*. URL: `http://www.w3.org/standards/semanticweb/ontology` (visited on 06/01/2014).

[54] W3C. *RDF 1.1 Primer*. Feb. 2014. URL: `http://www.w3.org/TR/rdf11-primer/` (visited on 06/01/2014).