Deusto

Facultad de Ingeniería
Universidad de Deusto

Ingeniaritza Fakultatea
Deustuko Unibertsitatea

Ingeniero en Informática
Informatikako Ingeniaria

Proyecto fin de carrera
Karrera amaierako proiektua

Summary

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

List of Figures

Capítulo 1

Capítulo 3

List of Tables

Listings index

Capítulo 1

Capítulo 3

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. 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. [27]

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 [28].

SPARQL: the SPARQL Protocol and RDF Query Language is a language designed to make queries over resources in RDF format [31].

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.

1.1.1 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 [39].

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.

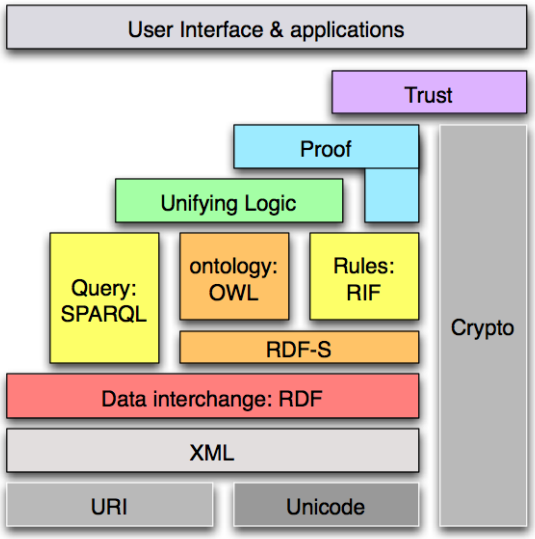


Figure 1.1: The semantic web architecture

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

```
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`

1.1.2 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 [38].

According to Tom Grubber [23] 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 [10]. 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. 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 [17] 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. [21]

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

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 different technologies used in the project are explained in detail, as well as the benefits of using them and the reasons for doing so.

The list of tools and frameworks used on the project is the following:

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

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 [39].

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 we can se an example of a set of triples in pseudo-code and the graph representing them.

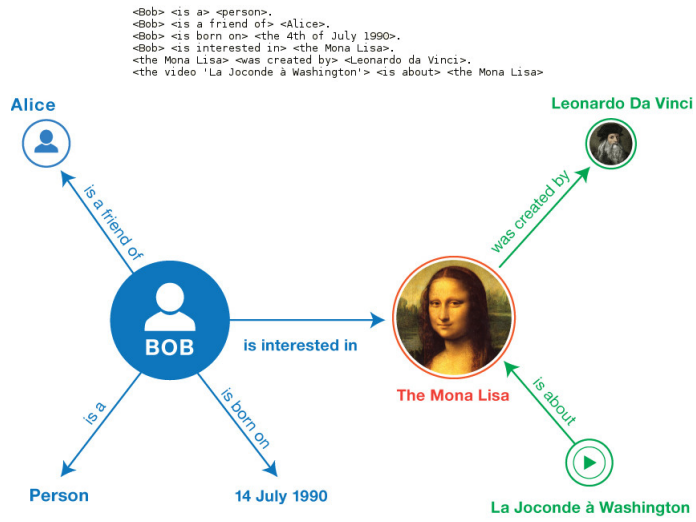


Figure 3.1: A set of triples and their corresponding graph

source: *http://www.w3.org/TR/rdf11-primer/*

There are three types of RDF data that can occur in triples: IRIs, Literals or Blank nodes [39].

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, 16](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[29].

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[12] 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.

To allow the definition of vocabularies, RDF-schema[7] is provided. This language allows the definition of RDF semantics, for example, one could define that the resource my:knows is a property or that the subjects of the previous triple are of type foaf:Person. RDF schema uses the concept of class, analogous to that on object oriented programming, to categorize resources. The property rdf:type is used when specifying a class. This way, hierarchies of classes and sub-classes, properties and sub-properties can be defined. Domains and ranges can be also specified for properties, which specify the type of the subject and the object on these properties respectively. An example of RDF-schema can be found in listing 3.3. There are alternatives for data modeling and inference to RDFS, such as OWL, however in many cases RDFS is enough to represent the logic on the system.

```
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>
6 :_b1 <rdf:type> <my:Arm>
```

Listing 3.3: Triple statements with a blank node.

RDF schema is not the only generally accepted vocabulary, there are several others that are currently considered standard. Among them we can find 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 [8].

Dublin Core: A metadata element set for describing a wide range of resources. Contains properties such as the creation date of the resource [24].

schema.org: A vocabulary for marking up web pages so that search engines can more easily find them [32].

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.

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[28]. 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.

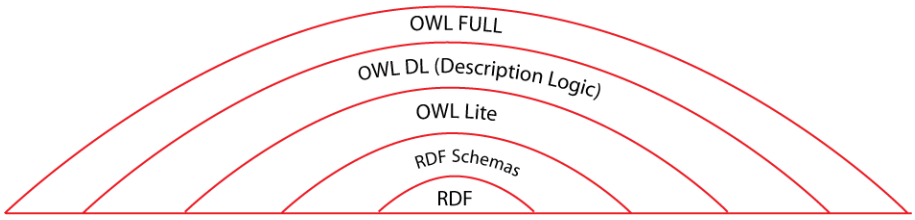


Figure 3.2: The OWL levels of expressivity

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.

```
<owlx:ObjectProperty owlx:name="adjacentRegion" owlx:symmetric="true">
  <owlx:domain owlx:class="#Region" />
  <owlx:range owlx:class="#Region" />
</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.

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

```
<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.

`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 [35].

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 [11].

SPARQL allows querying [31] and updating[33] 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.

```
1 SELECT ?s ?p ?o WHERE{
2         ?s ?o ?o .
3 }
```

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

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 database and will retrieve all triples.

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

```
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.

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, 30]. 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[14] string, as in the listing 3.9.

```
<my:Point> <rdf:type> <geo:Feature>
<my:Point> <geo:hasGeometry> <my:geom>
<my:geom> <rdf:type> <geo:Geometry>
<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

```
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.

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 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™[34] 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[37] for developing high-performance, concurrent programs that instead of relying on the mainstream multi-threading approach use asynchronous I/O[36].

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.

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.

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

Listing 3.11: NodeJS "hello world" web server.

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".

```
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.

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

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 [15, 22].

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 [19].

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.

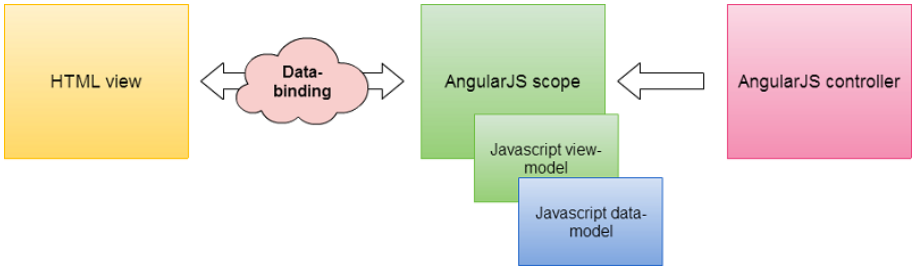


Figure 3.3: View-controller binding through the scope

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.

```
<div ng-controller="ExampleCtrl">
  <p>{{text}}</p>
  <button ng-click="change()">
    Change text
  </button>
</div>
<script type="text/javascript>
  var app = angular.module('exampleApp');
  app.controller('ExampleCtrl', function($scope){
    $scope.text = 'No text yet';
    $scope.change = function(){
      $scope.text = 'Button has been clicked';
    };
  });
</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[6] 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 [26]. 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[25].

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.

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

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.

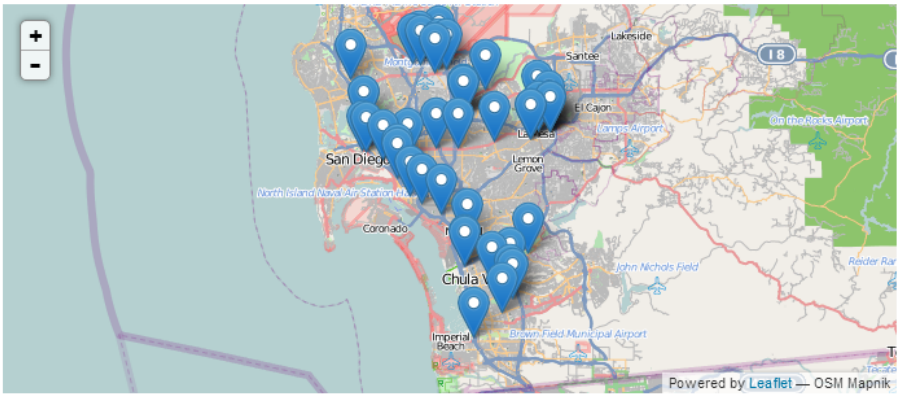


Figure 3.4: Leaflet with OSM mapnik tile layer

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

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.

```
1 // The moveend event is fired every time the
2 // map is moved, dragged or zoomed
3 Map.map.on('moveend', function(){
4   //Obtain the current bounding box of the map
5   var bbox = Map.map.getBounds().toBBoxString();
6   var feature = queryTheServer(bbox);
7   map.addLayer(features);
8 });
```

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 [20].

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[13] 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

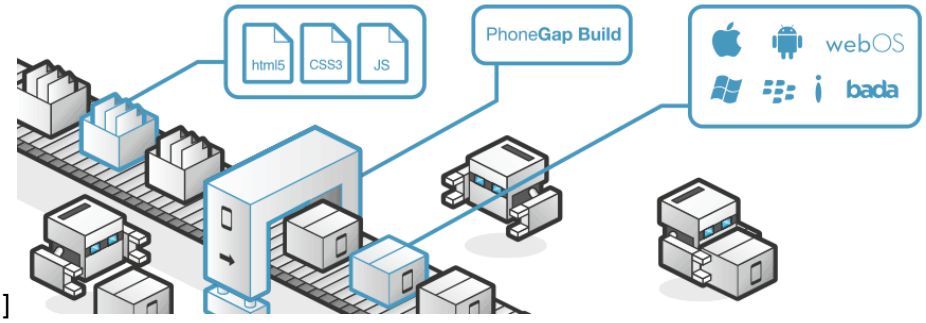
As convenient using HTML5 for developing cross platform applications is, there is still a need to configure the build for each of the platforms and to install the corresponding Software Development Kits (SDKs) which can be tedious and time consuming.

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.

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



]

Figure 3.5: Phonegap Build process

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

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[18]. 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

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

Listing 3.18: GPX way point representation.

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).

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.

```
<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.

```
<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[9]. 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` and `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.

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
- coordinates

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

- name
- description
- altitude
- category
- posts
- images
- author
- coordinates

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

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

RONT8 Resources of type Person must contain the following information:

- nickname
- email
- first name
- family name
- avatar
- description
- people in which is interested
- trail buddies
- optionally external homepage
- added trails
- added points of interest
- added notes
- added multimedia

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

- author
- content
- feature they belong to

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

- author
- download link
- addition date
- 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 data from Geonames and OpenStreetMap and aggregate it to the dataset.

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

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: Semant Web J Interoperability, Usability, Appl. http://www. semantic-web-journal. net/sites/default/files/swj176_0. pdf. Accessed 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] Bower - A package manager for the web. URL: `http://bower.io/` (visited on 06/17/2014).

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

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

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

[10] 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.

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

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

[13] Apache Cordova. About Apache Cordova. 2013.

[14] 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).

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

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

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

[18] Dan Foster. “GPX: the GPS exchange format”. In: Retrieved on 2 (2007).

[19] Adam Freeman. “The Anatomy of an AngularJS App”. In: Pro AngularJS. Springer, 2014, pages 207–231.

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

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

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

[23] Tom Gruber. What is an Ontology. 1993.

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

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

[26] Reuven M Lerner. “At the forge: Twitter bootstrap”. In: Linux Journal 2012.218 (2012), page 6.

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

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

[29] XML Schema Part. “XML Schema - 2: Datatypes”. In: W3C Recommendation 2 (2001).

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

[31] Eric Prud’Hommeaux, Andy Seaborne, et al. “SPARQL query language for RDF”. In: W3C recommendation 15 (2008).

[32] Jason Ronallo. “HTML5 Microdata and Schema. org.” In: Code4Lib Journal 16 (2011).

[33] 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).

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

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

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

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

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

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