UNIVERSITÀ DEGLI STUDI DI TRENTO
DISI - Dipartimento di Ingegneria e Scienza dell'Informazione

Master of Science in Computer Science

Final Thesis

Spacetime: A Two Dimensions Search and Visualisation Engine Based on Linked Data

---

1st Reader:
Marco RONCHETTI

Graduant:
Fabio VALSECCHI
ID: 154931

Academic Year 2012/2013

Contents

Chapter 1
Introduction

The following thesis project can be collocated in the domain of Semantic Web, Web of Data and Linked Data. The project has concerned the design and development of a web application based on DBpedia that represents one of the most interesting part of the Linked Data project. DBpedia is a community with the aim of extracting structured information from Wikipedia and make it available to users. Moreover they can query the structured data, gaining more information than the one provided by Wikipedia . The problem behind DBpedia is that there is not an usable and performant user interface for extracting data.

The developed application, during the stage at the department of Information Engineering and Computer Science, tries to provide a tool that allows users to perform geo-temporal queries on the DBpedia knowledge base derived from Wikipedia. The application, called Spacetime, is a two dimensions search engine that provides a simple visualisation user interface for presenting the geo-temporal results. This user interface follows exactly the dimensions on which the application is designed. In fact it is constituted by a map and a timeline that correspond respectively to the space and time dimension.

For making more attractive the application, Spacetime has been equipped by a set of features such as heat map, time sliding animation, map aggregation, icon map customisation and map saving and loading.

Thus the aim of this thesis are following:

- Construct an application able to solve the user interface usability problem of the existing applications based on DBpedia;
- Construct a new application, different from the existing ones, that provides a possible real tool for users.

The work started with a preliminary research phase on the DBpedia community, described in Chapter 2. In particular, this first step concerned the studying of the dynamics behind DBpedia especially regarding the mechanisms of the knowledge base, the structure of the ontology and the knowledge extraction framework.

Chapter 3 illustrates the investigation performed on the Sparql Language that is the query language able to interrogate RDF datasets such as the one used by DBpedia and queried by Spacetime. The chapter starts explaining how the Sparql language works and which are the query forms available. Then it includes describing the main operators provided by the language and some step by step query construction.

Chapter 4 provides a classification of the existing applications based on DBpedia and describes in details the most important of them. Moreover it includes an evaluating comparison of these applications based on some specific criteria. This comparative analysis has been performed for understanding the strength and weakness of the evaluated applications and for subsequently using it in the Spacetime development.

Chapter 5 is the chapter dedicated to Spacetime and it illustrates the characteristics of the application starting from the idea behind it. Then it continues describing the Spacetime categories, the use case diagram, the main Spacetime functionalities, the GUI design and it finish with a technical description related to the architectural structure, the timeline, the technologies used, the included Sparql queries and the problems of the application.

Chapter 6 proposes some Spacetime use cases for making more clear which are the capabilities of the application providing some of the most significant features and researches available in Spacetime. Furthermore it includes also a section in which some of the possible future implementation are discussed.

Chapter 2
DBpedia

2.1 DBpedia Project

2.1.1 What is DBpedia?

In this section a general and initial description of DBpedia is given for introducing this project.

DBpedia is a community effort to extract structured information from Wikipedia[1] and to make this information available on the Web. DBpedia allows you to ask sophisticated queries against datasets derived from Wikipedia and to link other datasets on the Web to Wikipedia data [1]. Moreover DBpedia has been described by Tim Berners-Lee[2] as one of the more famous parts of the Linked Data project [9].

At first sigh you may be wondering which is the need of having a system that extracts information from Wikipedia making it available on the Web. This is a task already fulfilled by Wikipedia and it seems to be useless. Rather the reason of the existence of DBpedia is that it provides to the users more information than Wikipedia. In fact the strength of DBpedia is the capability of answering complex user requests.

[1]Wikipedia is a collaboratively edited, multilingual, free Internet encyclopedia supported by the non-profit Wikimedia Foundation. Its 25 million articles, over 4.1 million in the English Wikipedia alone, are written collaboratively by volunteers around the world. Almost all of its articles can be edited by anyone with access to the site,[3] and it has about 100,000 active contributors.

[2]Sir Timothy John "Tim" Berners-Lee is a British computer scientist, best known as the inventor of the World Wide Web and as an expert of the Semantic Web and Linked Data.

For making more clear the concept some possible user queries are listed below:

- *"Which european countries have a capital with more than 3 million people in which flows a river longer than 300 kilometres?";*
- *"Which bands are composed by 4 members and played prog/psych music in Great Bretagne in the 60s?";*
- *"Which movie directors have done spy movies with Robert Redford in the cast before 1976?".*

Taking the first example in the list, it is easily comprehensible that Wikipedia can not provide the set of european countries that have a capital with more than 3 million people, in which flows a river longer than 300 kilometres. Though Wikipedia does not have a page that directly describes this complex set of countries, it contains all the data required for retrieving it. In fact Wikipedia has a page dedicated to London, Berlin, Paris and all the other european capitals in which population and river characteristics are normally available to users. DBpedia extracts all the information contained in special containers, called infoboxes, from Wikipedia pages and creates a knowledge base of entities.

Hence the advantage, that DBpedia provides to users, is that they have no needs to navigate all the european capital pages, for assembling the interested information, but they can ask to DBpedia to perform this costly task for them.

The knowledge base entities are described by a globally unique identifier that can be decomposed in different parts located in different locations on the Web. Currently, the Web of interlinked data sources around DBpedia provides approximately 4.7 billion pieces of information and covers domains such as geographic information, people, companies, films, music, genes, drugs, book and scientific publications [3].

2.1.2 DBpedia Knowledge Base

The DBpedia Knowledge Base describes more than 2.6 million entities [3]. Each entity is defined by a Uniform Resource Identifier (URI) which is described by a common pattern: `http://dbpedia.org/page/Name`, where *Name* is taken from the corresponding Wikipedia article URL.

For instance, the Wikipedia article page `http://en.wikipedia.org/wiki/Pink_Floyd` describes the entity of Pink Floyd band. DBpedia creates the corresponding URI as `http://dbpedia.org/page/Pink_Floyd`.

Each entity is composed by a set of Resource Description Framework[3] (RDF) triples. The DBpedia Knowledge Base is composed around 274 million RDF triples which have been extracted from 35 different Wikipedia language versions. A triple can be viewed as an expression in the form *Subject-Predicate-Object* and it can be represented by a node-arc-node link [8] as shown in figure 2.1. Each RDF triple represents a statement of a relationship between the *things* denoted by the nodes that it links.

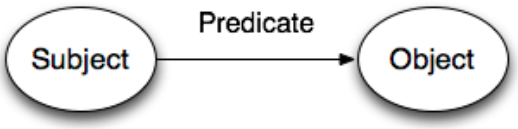

Figure 2.1: The graphical representation of a RDF triple.

2.1.3 DBpedia Ontology

As described in the previous section DBpedia provides a dataset of RDF triples based on Wikipedia data. This information is represented by DBpedia using an ontology[4] manually created by the members of the community. The ontology is composed by 359 classes and 1775 properties. The ontology is based on the Ontology Web Language[5] (OWL) and its classes are organised in a hierarchy where "Thing" representing the top most level. The need of having an ontology comes from the fact that the Wikipedia infobox template system evolved without using a central schema for describing entities and their properties. This aspects lead to a situation in which, for instance, the entity Person has an attribute for describing the place in which the person was born that can be either "birthplace" or "placeofbirth". DBpedia ontology wants to centralise the equivalent property names in a unique property label.

The following example describes a portion of the RDF triples list that compose the entity of the Pink Floyd band:

Subject: `<http://dbpedia.org/resource/Pink_Floyd>`
Predicate: `<http://dbpedia.org/ontology/name>`

[3]RDF is a language for representing information about resources in the World Wide Web and it can be viewed as a directed, labeled graph containing Linked Data.

[4]The DBpedia ontology is available at `http://mappings.dbpedia.org/server/ontology/classes/`.

[5]The Ontology Web Language (OWL) is the markup language for explicitly representing the meaning and the semantic of terms using vocabulary and relationships between them.

Object: `"PinkFloyd"@en`

Subject: `<http://dbpedia.org/resource/Pink_Floyd>`
Predicate: `<http://dbpedia.org/ontology/hometown>`
Object: `<http://dbpedia.org/resource/London>`

Subject: `<http://dbpedia.org/resource/Pink_Floyd>`
Predicate: `<http://xmls.com/foaf/0.1/homepage>`
Object: `"http://www.pinkfloyd.com/"`

The RDF triples in the example respect the *subject-predicate-object* approach and use the properties **name**, **hometown** and **homepage** for characterizing the Pink Floyd band. Each of these properties corresponds to an *object* that can be a literal such as **"Pink Floyd"@en**, a link such as **"http://www.pinkfloyd.com/"** or a reference to another entity (e.g. another *subject*) like **London**.

Therefore the DBpedia knowledge base is a set of entities, identified by URIs and described by a set of RDF triples. These aspects about the DBpedia architecture respect the four principles[2] about Linked Data drawn by Tim Berneers-Lee:

- Use URIs as names for things;
- Use HTTP URIs so that people can look up those names;
- When someone looks up a URI, provide useful information, using the standards (RDF*, SPARQL);
- Include links to other URIs. So that they can discover more things.

2.1.4 DBpedia Knowledge Extraction Framework

In this section we describe the mechanism of data extraction that DBpedia adopts for constructing its knowledge base. As we know DBpedia retrieves information parsing the Wikipedia pages and extracting data especially from special containers called infoboxes. Since infoboxes contain different kind of data, the DBpedia knowledge extraction framework provides a set of 11 extractors, one for each of the following type of information:

- *Labels*: the title of the WIkipedia article;
- *Abstracts*: a short and a long abstract for each Wikipedia articles;

- *Interlanguage links*: the links that connect equivalent articles in different languages;
- *Images*: links to Wikimedia Commons images;
- *Redirects*: the redirect strings used by Wikipedia to identify synonymous terms;
- *Disambiguation*: pages that explain the different meanings of homonyms;
- *External links*: links to external Web resources;
- *Pagelinks*: links between Wikipedia articles;
- *Homepages*: web links to home pages of entities such as companies and organisation;
- *Categories*: the category of the article;
- *Geo-coordinates*: geographical coordinates are expressed using Geo Vocabulary and GeoRSS.

In order to extract, store and keep update its knowledge base the DBpedia knowledge extraction framework uses two different kind of extractions:

- *Dump-based extraction*: it updates the DBpedia knowledge base with the dumps[6] published monthly by the Wikimedia Foundation. This procedure allows DBpedia to have an updated knowledge base in term of structure since the dump provides the last version of Wikipedia structure;
- *Live Extraction*: it uses the *Wikipedia OAI-PMH live feed* that instantly reports all Wikipedia changes. Using these live feeds the live extraction can update the knowledge base with new RDF triples each time a feed about a changed article is received. The update process deletes old RDF triples inserting the new ones. The time lag for DBpedia to reflect Wikipedia changes lies between one or two minutes[3].

However the most important source of information inside a Wikipedia page is the infobox shown in figure 2.2. This object is a table containing a list of attribute-value pairs and it is located on the top right side of every Wikipedia page. There are two different extraction algorithm:

[6]A database dump contains record of the table structure and/or the data from a database and is usually in the form of a list of SQL statements.

- *Generic Infobox Extraction*: it processes each infobox within a Wikipedia page creating RDF triples taking the corresponding DBpedia URI of the Wikipedia page as *subject* (e.g. Berlin), the concatenation of the namespace and the infobox attribute as *predicate* (e.g. http://dbpedia.org/property/country) and the attribute value as *object* (e.g. Germany);

- *Mapped-based Infobox Extraction*: it tries to solve the problem of attribute name synonymous and multiple template using an ontology that maps the 350 most used infobox templates in 170 ontology classes and 2350 attributes within these templates to 720 ontology properties.


```
{{About|the capital of Germany}}
{{Use dmy dates|date=July 2012}}
{{pp-move-indef}}
{{Infobox German state
|Name   =Berlin
|German_name=
|image_photo=Overview Berlin.jpg
|image_caption=Left to right: [[Berliner Fernseh
|state_coa =Coat of arms of Berlin.svg
|coa_size =70
|map   =Berlin in Germany and EU.png
|map_size =270
|map_text =Location within [[European Union]] an
|flag   =Flag_of_Berlin.svg
|area   =891.85
|area_source=
|population=3538652
|pop_ref =<ref name="Population11">{{cite web|ur
|pop_date =31 October 2012
|elevation=34
|demonym =Berliner
|GDP   =101.4
|GDP_year =2011
|GDP_ref=<ref>{{cite web|url=http://www.berlin.d
|Website =[http://www.berlin.de/international/in
|leader_title=[[Governing Mayor of Berlin|Govern
|leader =Klaus Wowereit
|leader_party=SPD
|ruling_party1=[[Social Democratic Party of Germ
|ruling_party2=[[Christian Democratic Union (Ger
|votes   =4
|divisions =12 [[Boroughs and localities of Berl
|NUTS   =DE3
|State   =Berlin
|Vorwahl =030
|Kfz   =B<small> (for earlier signs see note)</sm
|iso region=DE-BE
|PLZ   =10001-14199
|coordinates_display=display=inline, title
|latd=52 |latm=30 |lats=2 |longd=13 |longm=23 |l
|date   =September 2010
}}
```

Figure 2.2: A portion of the infobox of the Berlin Wikipedia article and its corresponding code.

Chapter 3

Sparql Language

Sparql is a query language designed specifically to query RDF databases [6]. Since the aim of this thesis is designing and developing an application based on DBpedia data, an important issue for our project is understand the capabilities of the Sparql language. More precisely it is necessary to figure out which queries the language is able to express and which results correspond to them. This chapter describes the Sparql language in such a way to understand its power and its expressive potentialities.

First of all, Sparql works sending queries from a client to a service, named as Sparql endpoint, using a http connection. A Sparql endpoint is an interface used for querying a specific RDF knowledge base associated to the endpoint. There exists different DBpedia Sparql endpoints such as *SNORQL Query Builder*[1] and *Virtuoso SPARQL Query Editor*[2].

3.1 The First Sparql Query

In order to start dealing with Sparql, we start from the construction of a simple query. For understanding its composition we need the following concepts:

- *Basic Graph Pattern*: it is a triple pattern contained in most of Sparql queries. It is like RDF triple but each subject, predicate and object can be substituted by a variable. As we can see in the example below, each of the RDF terms composing the triple can be substituted by a variable;

[1] http://dbpedia.org/snorql
[2] http://dbpedia.org/sparql

Subject: `<http://dbpedia.org/resource/Pink_Floyd>` or **?var1**
Predicate: `<http://dbpedia.org/ontology/hometown>` or **?var2**
Object: `<http://dbpedia.org/resource/London>` or **?var3**

- *Pattern Matching*: given a basic graph pattern, we can state that it matches a subgraph of RDF data when RDF terms from that subgraph may be substituted by the variables and the result is a RDF graph equivalent to the subgraph. For instance looking at the below basic graph pattern, we say that it matches all RDF triples in the dataset that have an equivalent predicate and object. Hence the set of matches it is composed by all the resources in the dataset that have an attribute hometown with value equal to London.

  Subject: **?var1**
  Predicate: `<http://dbpedia.org/ontology/hometown>`
  Object: `<http://dbpedia.org/resource/London>`

It is possible now to write a simple Sparql query, based on the previous example of *Pink_Floyd - hometwon - London*, as follows:

```
PREFIX : <http://dbpedia.org/resource/>
PREFIX dbp: <http://dbpedia.org/ontology/>

SELECT ?x
WHERE {
        ?x dbp:hometown :London
}
```

The query contains two PREFIX statements for mapping labels into IRIs. The first one maps the label ":" to `<http://dbpedia.org/resource/>` and the second one, "dbp:" to `<http://dbpedia.org/ontology/>`.

Then the query is composed by a SELECT statement for selecting which data has to be put in the results. Finally the WHERE command defines a criteria for refining the query. Note that this criteria is exactly the basic graph pattern previously defined. In fact it is composed following the *subject-predicate-object* form. It differs only for the use of the prefixes:

Subject: **?x**
Predicate: **dbp:hometown**
Object: **:London**

The query result describes the set of music bands which have London as hometown. Actually, the results include all the resources contained in the DBpedia dataset that has a triple defined with *hometown* as predicate and London as object. Some of these resources are those that belong to *Band* or *Music Artist* category.

3.2 Sparql Query Forms

The Sparql language has four different query forms:

- **SELECT**: it is equivalent to the SQL SELECT and it returns a table containing the results of the query. In particular it returns all, or a subset of, the variables involved in a query pattern match. For instance, the following sparql query returns a table that have as columns, the variables *?place* and *?population* ignoring *?country*, and as rows the corresponding values for each query pattern match. That is, each value of *?place* and *?population* that have a match with an RDF triple in the dataset.

```
SELECT ?place ?population
WHERE {
        ?place dbp:country ?country ;
               dbp:populationTotal ?population.
        FILTER ( ?country = :Italy &&
                 ?population > 500000 )
}
```

- **CONSTRUCT**: it returns an RDF graph, extracting a specific subset of the dataset. The CONSTRUCT query returns an RDF graph constructed by substituting the variables, involved in the WHERE statement, in a set of triple templates. In the example below, the RDF graph is composed by a set of triples, defined following the triple template *?place :pop ?population*, whose RDF terms are given by pattern matches.

```
CONSTRUCT   { ?place :pop ?population }
WHERE {
        ?place dbp:country ?country ;
               dbp:populationTotal ?population.
        FILTER ( ?country = :Italy &&
                 ?population > 200000 )
}
```

- **ASK**: it returns a boolean value (true or false) indicating whether a query pattern matches or not. For instance the following ASK query returns *true* because exists at least a pattern match. It means that it exists at least an RDF triple in the dataset that fulfil the specifications of the WHERE statements.

```
ASK WHERE {
        ?place dbp:country ?country ;
               dbp:populationTotal ?population.
        FILTER ( ?country = :Italy &&
                 ?population > 200000 )
}
```

- **DESCRIBE**: it returns an RDF graph that describes the resources found. The DESCRIBE query example that follows below, returns a description of the *country* property that links *Rome* to *Italy*.

```
DESCRIBE ?x
WHERE {
   :Rome ?x :Italy .
}
```

3.3 Sparql Operators

This section is dedicated to the most important Sparql operators. These commands are points of strength for the language and they make it more powerful allowing the construction of complex queries. Moreover using these operators, the query expressiveness degree increases. The following list illustrates each operator:

- ***SUBQUERY***:

  It is a way for embedding a query inside another one. Using subqueries makes possible to limit the results of a query putting it inside an external query.

```
SELECT ?user ?email
WHERE {
    ?user dbp:knows :Ginevra .    {
       SELECT ?user
       WHERE {
           ?user dbp:mbox ?email .
 } } }
```

- ***UNION***:

  It is a function that allows to express the possibility of matching several alternative patterns. In the following example the result of the query is composed by the solutions that match one of the two patterns that the *UNION* operators connects.

```
SELECT ?x
WHERE {
   { ?x rdf:type xsd:boolean }
    UNION
   { ?x rdf:type xsd:integer }
}
```

- ***OPTIONAL***:

  Basic queries produce results in which each solution match all the patterns inside the query. The *OPTIONAL* operator allows to add a solution when a certain information is available, but not reject the solution if it does not match all the patterns in the query. The queries adopting *OPTIONAL* operator, does not eliminate solutions for which the optional part does not match.

  For instance, the following query returns the users that have a property name. The use of *OPTIONAL* allows to includes the users, that have no mail box, in the results.

```
SELECT ?user ?email
WHERE {
    ?user dbp:name  ?name .
    OPTIONAL { ?user  dbp:mbox  ?email }
}
```

- ***IF (expression1, expression2, expression3)*:**

  It evaluates the first argument of the function, interpreting it as an effective boolean value (i.e. true or false). It returs *expression2* when the effective boolean value is true and *expression3* otherwise.

- ***rdfTerm IN (expression, ...)***:

  It tests whether the RDF term on the left-hand side is found in the values of the expressions list on the right-hand side, returning a boolean value.

  Similarly the *NOT IN* operator tests whether the RDF term is not found in the values of the expressions list.

- **Negation**: *MINUS* and *NOT EXISTS* operators represent two ways of expressing negation in Sparql, based on different approaches:
  - ***NOT EXISTS***: it is a filter expression that tests if a RDF triple pattern does not match the entries of the dataset. The results of a query is composed by the entries of the dataset for which does not exist the pattern specified in the *NOT EXISTS*.

```
SELECT * {
    ?a ?b ?c
    FILTER NOT EXISTS { ?x ?y ?z }
}
```

  - ***MINUS***: it is a different approach that calculates the results in the left-hand side that are not compatible with the solutions on the right-hand side. This function removes the solutions of the query that match the expression in the *MINUS* function.

```
SELECT * {
    ?s ?p ?o
    MINUS { ?x ?y ?z }
}
```

3.4 Complex Queries

3.4.1 Building a complex query

In this section we explain how to write more complex queries than the ones seen in the previous sections. We take as example the user question, *"Which bands are composed by 4 members and played prog/psych music in Great Bretagne in the 60s?"*, and we try to translate it in Sparql language. The construction of the query proceeds step by step:

Step 1: the following query returns the set of Bands coming from the United Kingdom. As seen in section 3.1, PREFIX statements allows to map labels with IRIs, while SELECT and WHERE allow respectively to define the desired data and refine them with some criteria.

```
PREFIX : <http://dbpedia.org/resource/>
PREFIX dbp: <http://dbpedia.org/ontology/>
PREFIX dbprop: <http://dbpedia.org/property/>

SELECT DISTINCT ?band
```

```
WHERE {
        ?band dbpprop:name ?band_name ;
              dbp:hometown ?city .
        ?city dbp:country :United_Kingdom .
}
```

Step 2: this query refines the results, obtained in step 1, presenting only the bands that starts playing in the 60s. As we can see below, the query presents a new basic graph pattern, *dbp:activeYearsStartYear ?time*. Then the FILTER operator selects only the triples with a specific value of *?time* (i.e. the date variable *?time* must have a value of year, greater and equal than 1960, and less and equal than 1970). Starting from this step the queries avoid to include each time the prefix statements.

```
SELECT DISTINCT ?band
  WHERE {
        ?band dbpprop:name ?band_name ;
              dbp:hometown ?city ;
              dbp:activeYearsStartYear ?s_time ;
              dbp:activeYearsEndYear ?e_time ;
        ?city dbp:country :United_Kingdom .
        FILTER ( ?e_time >= "1960"^^xsd:date &&
                 ?s_time <= "1970"^^xsd:date )
  }
```

Step 3: this query filters the bands that played a genre different from psychedelic or progressive rock. Hence the result is composed by the bands of the United Kingdom that played psych/prog rock in the 60s.

```
SELECT DISTINCT ?band
  WHERE {
        ?band dbpprop:name ?band_name ;
              dbp:hometown ?city ;
              dbp:activeYearsStartYear ?s_time ;
              dbp:activeYearsEndYear ?e_time ;
              dbp:genre ?genre ;
        ?city dbp:country :United_Kingdom .
        FILTER ( ?e_time >= "1960"^^xsd:date &&
                 ?s_time <= "1970"^^xsd:date )
        FILTER ( ?genre = :Psychedelic_rock ||
                 ?genre = :Progressive_rock )
  }
```

Step 4: in this step we refine the query providing only the United Kingdom bands that played psych/prog rock in the 60s and that are composed by exactly 4 members. For expressing the query, the Sparql language provides the aggregate function COUNT for retrieving the number of members in each band. Then using the GROUP BY and HAVING operators is possible to obtain the set of bands composed by 4 members.

```
SELECT DISTINCT ?band ((COUNT(?member) AS ?count))
WHERE {
        ?band dbpprop:name ?band_name ;
              dbp:hometown ?city ;
              dbp:activeYearsStartYear ?s_time ;
              dbp:activeYearsEndYear ?e_time ;
              dbp:genre ?genre;
              dbp:formerBandMember ?member .
        ?city dbp:country :United_Kingdom .
        FILTER ( ?e_time >= "1960"^^xsd:date &&
                 ?s_time <= "1970"^^xsd:date )
        FILTER ( ?genre = :Psychedelic_rock ||
                 ?genre = :Progressive_rock )
} GROUP BY ?band
  HAVING (COUNT(?member) = 4)
```

Step 5: this query provides the same results of the previous one filtering the bands that contains the particular character "%", using the MINUS command. This is done because the results containing this character are doubled.

```
SELECT DISTINCT ?band ((COUNT(?member) AS ?count))
WHERE {
        ?band dbpprop:name ?band_name ;
              dbp:hometown ?city ;
              dbp:activeYearsStartYear ?s_time ;
              dbp:activeYearsEndYear ?e_time ;
              dbp:genre ?genre;
              dbp:formerBandMember ?member .
        ?city dbp:country :United_Kingdom .
        FILTER ( ?e_time >= "1960"^^xsd:date &&
                 ?s_time <= "1970"^^xsd:date )
        FILTER ( ?genre = :Psychedelic_rock ||
                 ?genre = :Progressive_rock )
        MINUS {
```

```
                    ?band dbpprop:name ?band_name
                    FILTER (REGEX(STR(?band), "%", "i"))
        }
} GROUP BY ?band
  HAVING (COUNT(?member) = 4)
```

3.4.2 Other complex queries

After the step by step construction of a complex query, some other complex queries are listed as examples:

- *"Which movie directors have done spy movies with Robert Redford in the cast before 1976?".*

```
PREFIX : <http://dbpedia.org/resource/>
PREFIX dbp: <http://dbpedia.org/ontology/>
PREFIX dbprop: <http://dbpedia.org/property/>

SELECT DISTINCT ?director_name ?movie
   WHERE {
      ?director dbprop:name ?director_name.
      ?movie dbp:director ?director ;
             dbp:releaseDate ?date ;
             dbp:starring :Robert_Redford .
      FILTER (?date <= "1976"^^xsd:date)
      FILTER (REGEX(STR(?director_name), ",", "i"))
   }
```

- *"Which are the italian writer born between the 1850 and 1900 and dead after the 1950 in a city different from the birth place?"*

```
PREFIX : <http://dbpedia.org/resource/>
PREFIX dbp: <http://dbpedia.org/ontology/>
PREFIX dbprop: <http://dbpedia.org/property/>

SELECT DISTINCT ?writer
   WHERE {
      ?writer rdf:type dbp:Writer;
              dbp:birthDate ?birthDate;
              dbp:deathDate ?deathDate;
              dbp:birthPlace ?birthPlace;
              dbp:deathPlace ?deathPlace.
```

```
    ?birthPlace dbp:country :Italy.
    FILTER (?birthDate >= "1850"^^xsd:date &&
            ?birthDate <= "1900"^^xsd:date)
    FILTER (?deathDate >= "1950"^^xsd:date)
    FILTER (?birthPlace != ?deathPlace)
}
```

3.5 Problems And Limits

This section illustrates the problems regarding the construction of an application that wants to map the Sparql language with an usable and performant user interface. The following examination is focused on the limits and the problems related to the DBpedia, in particular the ontology and the data of DBpedia.

The result of the analysis, performed in this chapter, about the query language shows that Sparql is a powerful instrument that allows the construction of complex queries through its set of commands and operators, introduced in section 3.3. However the real problem, emerging from the investigation, is the DBpedia ontology and the data inside its datasets. In order to understand better which are the limits behind these aspects, we divide the problems in two categories:

Ontology limits:

This category regards the problems related to the internal structure of the Dbpedia ontology.

- As described in chapter 2.1.3 the DBpedia ontology is composed by 359 classes and 1775 properties. Moreover, these classes have a different internal definition that consists of a set of properties directly depending from them. For instance, the class *Person* has particular properties that describe a person like first name, surname, age, birth date, death date, hometown and so on. While the class *Organisation* has other properties that describe all the characteristics of an organisation such as name, foundation date, hometown, founders and so on. It is clear that a software developer that has to deal with this ontology, has also to know exactly which are the properties of the ontology classes, if he wants to write Sparql queries. Hence the developer requirement of having a depth knowledge about the ontology, is the first limit that characterises the development of an application based on DBpedia.

Investigating the ontology, its classes and its properties remain a costly task, but it is the only way for exceeding this problem;

- Another problem is the internal composition of the ontology that appears not balanced. In fact navigating the ontology, emerges that some classes are developed in depth rather than other ones. For instance, taking the class *Event* and analysing their subclasses (i.e. Convention, Election, FilmFestival, MilitaryConflict, SpaceMission and SportsEvent), we can notice the presence of the class *SpaceMission*. First, you may be wondering if this class is really needed in an ontology that tries to summarise the huge amount of information of Wikipedia. Second, *SpaceMission* class has 40 properties differently from *Election* and *FilmFestival* that have respectively 9 and 15 properties. It is now clear the imbalance of the ontology that attributes 40 properties to a class like *SpaceMission*, in despite of more significant classes such as *Election* and *FilmFestival*.

Resource limits:

This class of problems is related to the resource data inside the Dbpedia dataset.

- The big problem, related to the result quality of a Sparql query that interrogates DBpedia, is the uniformity of the data. If we analyse the resources dataset, belonging to a specific class, we can notice that not all of them hold the entire set of properties that their own class have.

  For instance, supposing we are interested in extracting from DBpedia the movies released during 1989. The query can be easily written as follows:

```
SELECT DISTINCT ?sub ?date
WHERE {
    ?sub rdf:type dbp:Film;
         dbp:releaseDate ?date .
     FILTER ( ?date >= "1989-01-01"^^xsd:date &&
              ?date < "1990-01-01"^^xsd:date )
}
```

  The problem is the quality of the results, because not all the movies belonging to the *Film* class have the attribute *releaseDate*. In fact the property *releaseDate* is hold from 30943 resources of the 71715

belonging to the class *Film*. This specific example shows that more than half of the resources interested in the query can not be included in the final result set. This situation is quite common inside the DBpedia dataset and it represents a problem for Spacetime because half of the results that the application would be provide to the user are filtered by this limitation;

- Another problem related to the resource data of Dbpedia is the resource class attribution. In fact sometimes happens that a resource is stored in the dataset using a class that is the too generic for it and for which exists a more specific subclass suitable for it.

  For instance, taking the Pink Floyd music band, we notice that its class is *Organisation* that is the parent class of *Band*. It would been better to classify the Pink Floyd resource as a *Band* and not as a generic *Organisation*. This problem engraves the query results quality because if a query is written for extracting a set of resources belonging to a certain class, the resources classified using a wrong class will be not included in the final result set of the query.

Chapter 4
Applications

The challenge of exploiting the vast amount of DBpedia information is the purpose of each application that is interested in using DBpedia data. DBpedia datasets contain a huge amount of knowledge that regards everything that can be thought. In fact this information are taken from Wikipedia pages that host a wide range of topics related to literature, music, cinema, history, geography, biology, computer science, technology and so on. Paradoxically the problem of constructing a user interface, that is capable to make users able to ask complex questions like *"Which european countries have a capital with more than 3 million people in which flows a river longer than 300 kilometres?"*, is the quantity of the data. In fact managing this enormous amount of information, makes very difficult the creation of a system able to handle it. Even if DBpedia has built a "lighter" ontology (see section 2.1.3) for classificating data using classes and properties, the problem still remain extremely difficult. Several approaches to the problem have been developed, during the last years, by different companies but this issue still remains open.

This chapter describes the state of the art of the applications based on the structured knowledge of DBpedia, illustrating the different approaches for accessing data. Since applications provide different methodologies to access the data, the following classification provides a list of techniques, for retrieving data from DBpedia, ordered from basic procedures to more sophisticated systems.

The first two approaches for accessing DBpedia data are not real applications but they are important for understanding the following classification:

- The most basic "application" is described by a user that download[1] the

[1] http://wiki.dbpedia.org/Downloads

DBpedia dataset and browses through it for looking at RDF triples. The problem of this approach is that it is improbable considering the amount of datasets and the amount of megabytes of data contained in them. Moreover, using this method, it is not possible to connect different resources for obtaining meaningful information;

- A more high-level approach but not suitable to be used by normal users is the execution of SPARQL queries for interrogating the DBpedia dataset. The problem of this method is that users require a deep knowledge of the dataset and its properties in order to be able to write queries. This is a task even difficult for the experts of the DBpedia ontology considering the size of the ontology and the number of its properties. The main query interfaces are *SNORQL Query Builder*[2] and the *Virtuoso SPARQL Query Editor*[3] and they both interrogate DBpedia using the SPARQL query language. Moreover a good knowledge of this language is required for writing queries over DBpedia.

There exist more powerful and usable methods, that are more similar to normal search engines that takes care about their user interface.

- **Entity Search, Find, and Explore by OpenLink Virtuoso**[4]:
  It is a tool for navigating DBpedia data using three different kind of research. It is possible to perform research starting from a keyword, an URI or a label. The text search requires the insertion of a text pattern to look for. Then a finder shows a list of entities with the text occurring in any literal property value or label. The entity URI lookup is used inserting entity URI that are recognised by the autocomplete feature of the tool. This simple text search is better than the previous approaches but it is not able to exploit the huge amount of knowledge contained in DBpedia dataset and to make available the possibility of asking complex queries;

- **gFacet**[5]:
  gFacet is a browser for the Web of data that adopts a particular approach that combines graph-based and facet-based approaches [7]. GFacet represents RDF triples using a graph where nodes are RDF subjects or objects and edges are RDF predicates;

[2]http://dbpedia.org/snorql/
[3]http://dbpedia.org/sparql
[4]http://dbpedia.org/fct/
[5]http://www.visualdataweb.org/gfacet.php

- **LodLive**[6]:
  LodLive is a graph-based linked open data live browser. It is a navigator of RDF resources based on SPARQL endpoints. It represents data using a graph-based approach that allows the representation of relations between concepts. It provides different kind of user visualisation such as textual data, images, links to additional resources and geo-localisation on map. LodLive makes the users able to explore a linked dataset moving from one concept to another by browsing a graph;

- **Faceted Wikipedia Search**:
  It is a research system based on the faceted search paradigm [5]. Users can perform their research combining text search with additional restriction properties called facets. For instance, taking the european capital example of section 2.1.1, a facet of the entity "city" could be "population amount" while for the entity "river" the "length". Faceted Wikipedia Search provides facets for addressing user researches in the correct direction filtering undesired results.

4.1 gFacet

It is an application based on graph and facet approach. The user operations for finding information are the following:

- Initially the application provides a textual search from which users express a context for the research.
  Supposing a user insert the keyword "italian";

- After a search is performed the application visualises a list of classes related to the textual insertion. Each class has the number of objects that contains (i.e. Italian films, 2345 objects)
  The application, for "italian" keyword, provides a list that contains Italian singers, Italian footballers, Italian films, Italian actors and so on;

- The user chooses the interested class.
  Supposing the user chooses the class of Italian actors;

- Once a class is selected, the list of object contained in the class, and ordered in alphabetical order, is presented to the user.
  The list of italian actors is listed in the appropriate panel: Alberto Sordi, Anita Caprioli, Antonio Albanese...;

[6] http://en.lodlive.it/

- After that it is possible to connect the class chosen to other classes of objects, through class relations.
  The user selects the birth place relation associated with the class of Roman tows and cities in Italy as shown in figure 4.1;

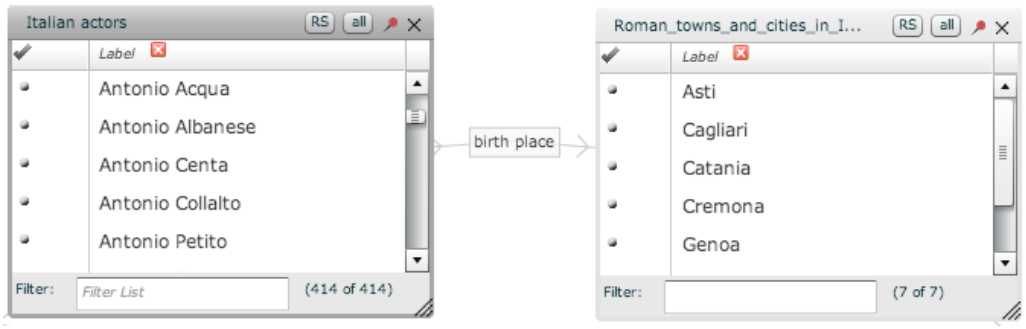

Figure 4.1: A relation, between two class of objects, expressible using gFacet.

- Finally, once at least a relation is established, it is possible to select an object from a class for obtaining an updated list of the other classes. In this way the application shows the result of the relation utilisation.
  Selecting Genova, the application updates the list of actors, presenting only the actors with birth place in Genova, as shown in figure 4.2;

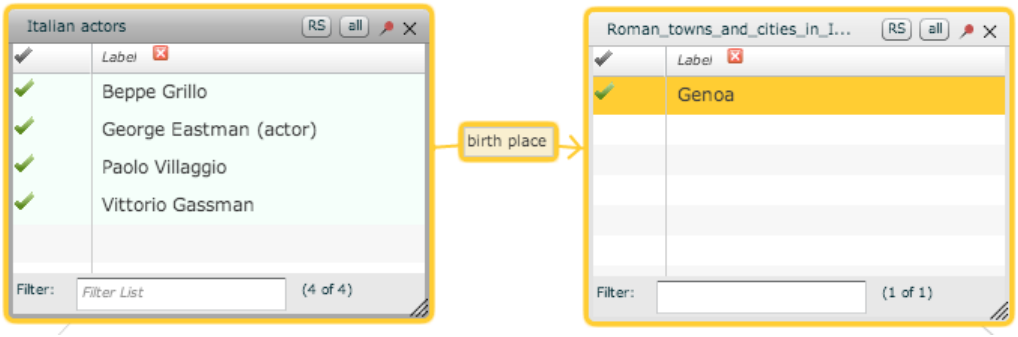

Figure 4.2: The result of the birth place relation for the italian city of Genova.

- This process can be performed many times creating a graph where nodes are classes of objects (i.e., Italian actors) and edges relation between classes of objects (i.e., birth place).

4.2 LodLive

LodLive is a graph-based linked open data live browser. It is a navigator of RDF resources based on SPARQL endpoints. The application proposes three different kind of research:

- Simple search: it is a textual research that starts from a user insertion of a keyword for finding an URI;
- URI-based search: the user holding a URI can start the graph browsing inserting the URI;
- Endpoint search: the user, once has chosen an endpoint and a class belonging to it, can insert a keyword.

Once the search is terminated the application shows the searched concept as a circle surrounded by many smaller ones that represent concept properties. Expanding the smaller circles is possible to create a graph that connects different concepts.

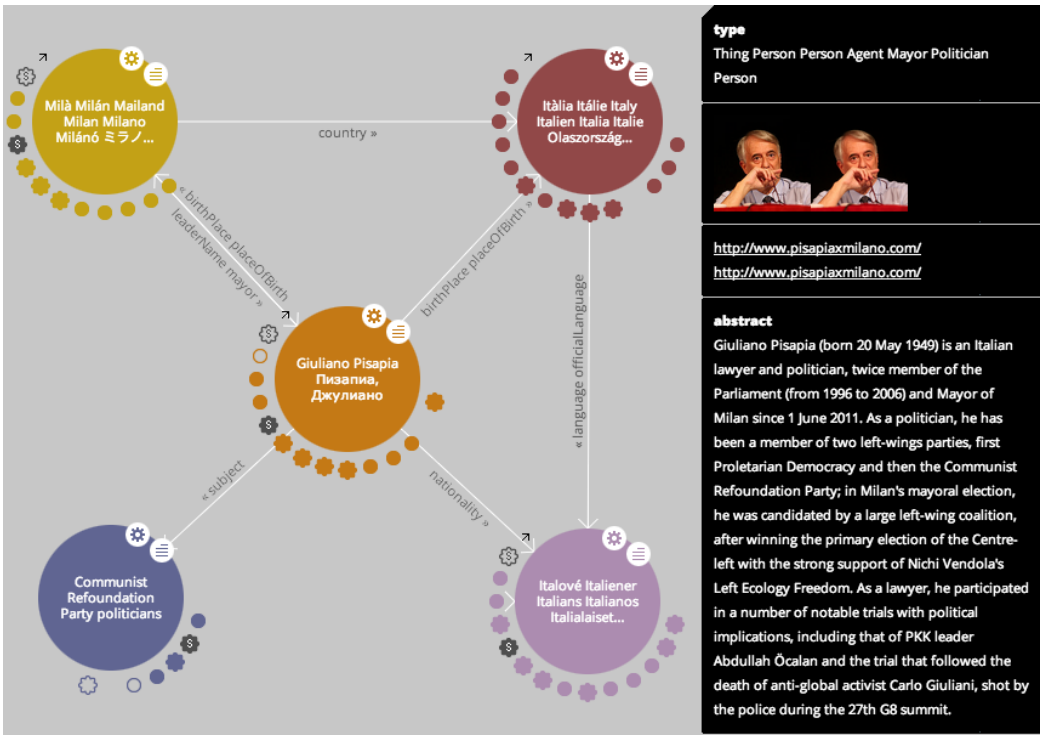

Figure 4.3: A portion of LodLive application that shows the graph of concepts and the additional information right panel.

The figure 4.3 shows the LodLive graph applied to an example started by the *simple search* of "Milan" as keyword. The graph contains the concept of the city of Milan and other concepts linked to it such as Italy and Giuliano Pisapia that are respectively the country and the mayor of the city. The edges of the graph represents the relations between concepts for which LodLive is able to calculate the inverse relations.

For each concept, LodLive allows user to open a map panel for geolocalizing concepts, an image panel for looking at concept images and an additional information panel for detailed information (See Figure 4.3).

LodLive is composed of a J-Query plug-in (named LodLive core), a JSON configuration map (named LodLive-profile), an HTML5 page, some images and some other public J-Query plug-ins [4].

4.3 Faceted Wikipedia Search

The Faceted Wikipedia Search is a different way of searching information. The project totally relies on DBpedia Knowledge base from which retrieve data. The application tries to provides to the users a tool for expressing and asking complex questions. For achieving its aim the the Faceted Wikipedia Search adopts a faceted search paradigm. This approach enables users to compose complex questions step by step using facets. A facet is a component of the user interface for refining user researches. Facets exploit the properties of an entity for making more precise the result of a user question. For instance, as shown in figure 4.4, the facets of an entity *river* are *has mouth at*, *length* and *watershed*. These three facets are only a little part of the properties set, but they can improve the result of the user questions.

The figure 4.4 shows all the components in the user interface of the application:

- *Textual Search*: it is a simple text search;
- *Facets Panel*: it displays the most relevant facets used by users;
- *Filters Panel*: it visualises the facets used by user and allows to remove them;
- *Results Panel*: it lists the results of a user research.

Faceted Wikipedia Search was online for several years at `http://dbpedia.neofonie.de`, but is now offline because Neofonie has stopped to maintain

the server. Moreover there are currently no concrete plans of getting the application online again. Faceted Wikipedia Search is now a dead application but it has one of the most interesting approach among the application based on DBpedia.

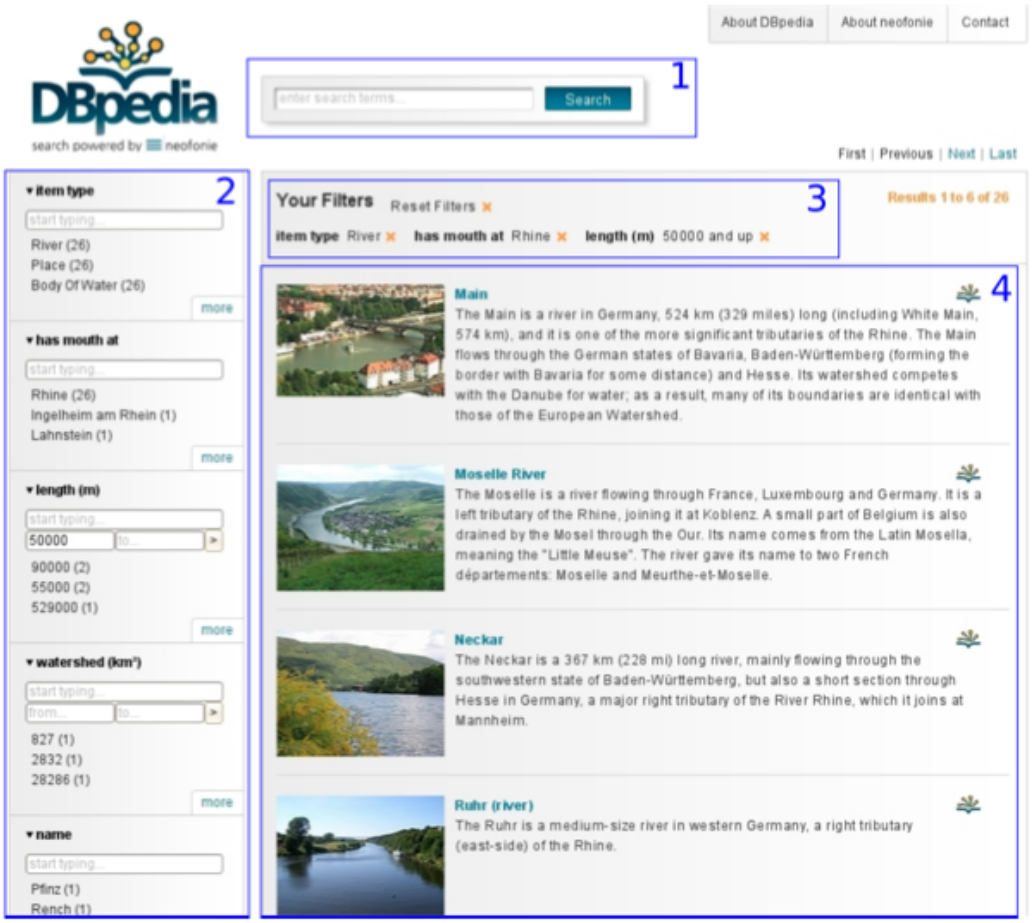

Figure 4.4: Screen shot of Wikipedia Faceted Search showing textual search in area 1, facets in area 2, filter panels in area 3 and the results in area 4.

4.4 Applications Comparison

This section presents a comparison between the main applications illustrated in the previous section using several criteria for evaluating them. The application chosen for this comparative analysis are gFacet, LodLive and Faceted Wikipedia Search (FWS). The parameters chosen for the evaluation are the following:

- **Queries Complexity:** it defines the complexity expressible using a specific application;
- **Interface Usability:** it describes the usability grade that a certain application provides to users;
- **User Typology:** it defines the typology of user that can utilise a specific application;
- **Technologies used:** it lists the technologies adopted for the implementation of a certain application.

The table 4.1 shows, for each criteria, the corresponding value for each application.

	gFacet	**LodLive**	**FWS**
Queries Complexity	quite complex	not complex	complex
Interface Usability	not so good	not so good	good
User Typology	expert user	expert user	normal user
Technologies Used	Flash	JQuery, HTML5	?

Table 4.1: The summary of the criteria analysis of the DBpedia based application.

The result of the comparative research, about the most important application based on DBpedia, shows that LodLive is the most innovative application in term of technologies used while GFacet allow to express quite complex query. Finally Faceted Wikipedia Search can represents complex queries, providing a good usability that allows to normal users to utilise the application. Unfortunately, Faceted Wikipedia Search, that appears to be the most interesting Dbpedia based application, is no longer available on the web.

4.5 Other Applications

The complexity of constructing an user interface able to exploit the huge amount of information contained in the DBpedia knowledge base suggests to

develop more restricted application focused on smaller contexts. For instance there are application like:

- **DayLikeToday**[7]: it is a web application that allows to navigate, through a time line, all the events happened in a certain day of the year;
- **AboutThisDay**[8]: it is a tool for finding events, related to several categories (i.e. media, war, literature, politics, business, art, religion, science, society and sport), happened in a specific day of the years;

[7] http://el.dbpedia.org/apps/DayLikeToday/
[8] http://www.aboutthisday.com/

Chapter 5
Spacetime

This chapter treats the description of Spacetime, the application developed for this thesis project. The name of the application wants to highlight the fact that it is based on two dimensions, the time and the space. Spacetime can be defined as a two dimensions search and visualisation engine based on the DBpedia knowledge base.

Spacetime tries to exploit the huge amount of data contained in the DBpedia dataset. Since DBpedia contains an enormous amount of information, it is difficult to entirely manage it. Spacetime uses a subset of this database taking only the resources that have a spatial and a temporal indication in their attributes. This choice allows to address the focus of the application only on a specific and more restricted set of data.

Spacetime provides a search engine for identifying, in a specific space area and in a certain range of time, entities belonging to the categories taken from the DBpedia ontology. Part of these categories are the following: organisations, bands, companies, sport teams, persons, architects, writers, journalists, politicians, events, elections, film and music festivals, military conflicts, sport events, architectural structures, historic places, monuments, natural places, works, movies, books, softwares, paintings ...

The following treatment is divided in two different sections: the first one is a description of the application from the point of view of the user while the second one is a technical analysis of the application from the point of view of the developer.

5.1 Spacetime

This section is dedicated to Spacetime. In particular it explains which is the idea behind the application, which are its functionalities and how the application works.

5.1.1 Spacetime Idea

The idea behind Spacetime can be simply found inside its name. It is based on two main concepts: space and time. Two dimensions that are the base of a search engine that takes its knowledge from the DBpedia dataset. As said in the previous chapters the big challenge, of managing the huge amount of data hosted in DBpedia, is find a method for providing the data in a simple and clear way to the user. The real problem is that the DBpedia data are organised in an ontology (described in chapter 2.1.3) composed by classes defined using a complex set of attributes, that are not easy to handle. Then, constructing an application for retrieving information from this ontology become very difficult, since it is required to know exactly which properties correspond to each class in the ontology.

The data extraction operation performed by Spacetime is different from the normal techniques adopted by the existing application based on DBpedia. Spacetime takes all the resources in the DBpedia dataset that have at least one spatial and one temporal indication without considering the other attributes of the resources. This approach allows to overcome the ontology complexity problem and to succeed in extracting useful information for the application.

The resulting application is a search engine that provides a set of DBpedia resources mapped on a Google map and on a timeline. The researches can be effected using three parameters:

- **Space**: the space parameter can be a country, a continent or the entire globe;
- **Time**: the time parameter is a range of time defined by two dates;
- **Spacetime category**: this parameter defines the category of the DBpedia resources.

In the next section the Spacetime categories are described for understanding what is the third parameter on which the application is based.

5.1.2 Spacetime categories

The Spacetime categories are a subset of the hierarchical structure of the DBpedia ontology. They are composed by five macro categories: *Organisation*, *Person*, *Event*, *Place* and *Work*. In order to understand better which subcategories compose these macro categories, we list part of them below:

- **Organisation**: this class contains Band, Broadcaster, Clerical Order, Comedy group, Company, Government agency, Military unit, Political party, Sports league, Sports team, ...
- **Person**: this class contains Architect, Artist, Athlete, Economist, Journalist, Monarch, Philosopher, Politician, Religious, Scientist, Sports manager, ...
- **Event**: this class contains Convention, Election, Film festival, Military conflict, Music festival, Space mission, Sport events, ...
- **Place**: this class contains Architectural place, Historic place, Monument, Natural place, Populated place, Protected area, Ski area, ...
- **Work**: this class contains Artwork, Cartoon, Film, Musical, Album, Single, Song, Software, Television show, Website, Written Work, ...

The choice of the macro categories has been performed following two criteria:

- A macro category must have a category of resources that the user is interested in;
- A macro category must have a considerable amount of resources with a spatial and temporal indication in their attributes for allowing the user research over them.

Regarding the second criteria, an examination of the number of resources has been made for perceiving the amount of information that the selected categories would have contained. The following table shows for each of the macro categories chosen the number of resources, with a spatial and temporal indication, contained in it.

As table 5.1 shows, all the categories have a considerable amount of resources.

	N of resources
Organisation	29158
Person	5265804
Event	77613
Place	225161
Work	125107

Table 5.1: The number of resources with a spatial and temporal indication for each Spacetime categories.

5.1.3 Use Case Diagram

The use case diagram in figure 5.1 represents the user interactions with Spacetime. It depicts only one kind of actor that is *User* that represents the application utiliser. There are five different use cases that are the following:

- *Perform a research*: the research operation requires the insertion of three parameters by the user: a spatial indication (i.e. a country, a continent or the entire earth), a temporal indication (i.e. a range of time defined by two dates) and a Spacetime category (i.e. a category between Organisation, Person, Event, Place, Species, Work or one of their subclasses). This use case provides to the user a set of DBpedia results mapped on a Google maps and at the same time ordered on a timeline. Moreover the user can activate additional features, described in details in the next section, such as time sliding animation, heat map visualisation, map aggression, numbered researches and icon customisation;

- *Save a map*: the saving operation requires the insertion of a text string corresponding to the name of the map file. Then it is required also to select a file format for determining the the output format of the map. This operation is available only after the user has performed a research or has loaded a map. In general the operation can be done only when the Google map is not empty;

- *Load a map*: the loading operation requires that the user must select a Spacetime map file from its local machine. After a map is loaded is not allowed to perform the operation another time;
- *Modify a map*: the process of modifying a map consists of some customisation operations such as the events remotion and the change of the icon of the events mapped.
- *Consult a map*: this use case allows the user to navigate a Google map populated with a set of clickable markers. The on click event create an info window popup containing the abstract of the resource clicked in different languages, and two resource links to Wikipedia and DBpedia for additional and more specific information.

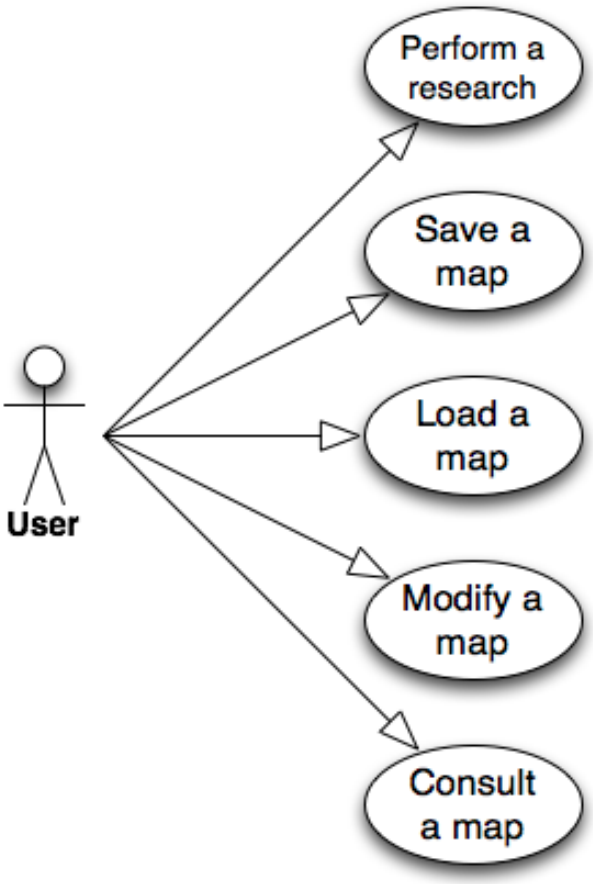

Figure 5.1: The use case diagram of Spacetime.

5.1.4 Spacetime Functionalities

In the previous section the use case diagram provides a first view of the functionalities of the application. In this section instead some specific aspects about them are analysed for understanding the potentialities of Spacetime. In particular the additional features of the application and the saving and loading operations will be discussed.

Spacetime data visualisation

An important set of Spacetime functions are the different kind of data visualisation, that allow to represents the results set of a research in different way. Spacetime provides four different data visualisation:

- **Density visualisation:** The first Spacetime feature, called density visualisation, provides a heat map that is a map that depicts the density of the data following the geographical concentration of the resources. When this feature is enabled, a coloured overlay appears on the map, indicating the high density areas with red and the low density areas with blue. The colours in between red and blue are defined by a colour gradient that is a set of RGB colours.

- **Time Sliding animation:** This feature allows the user to create a time animation in which the events appear sequentially on the map, following a temporal order. This feature becomes very useful if combined with the *density visualisation* function, because it provides an animation in which the heat map changes during the time providing the evolution of the events.

- **Map aggregation:** The map aggregation provides a very powerful feature because it allows the creation of more complex maps that include different researches based on different parameters. For instance this function allows to create maps that contains researches performed on different countries, different time ranges or different categories. Thus it is possible to create maps containing resources from *Writer* and *Book* category, *Athletes* and *Sports events*, *Elections* performed in Italy, France and Germany or *Music festival* placed in United Kingdom and United States of America.

  Moreover the possibility of modifying the icon markers of the resources, gives more power to the map in term of meaning and clearness. This feature can be used with two loaded map, a loaded map and a new research and viceversa.

- **Numbered research:** This function is thought for helping users in numbering results. In fact, normally when a research is performed without the use of this feature, the results are shown on the map with a generic marker. This default indicator tells no information to the user about the temporal position on the timeline. The numbered research proposes a more significant map composed by markers labeled with numbered icons that provides an additional temporal information on the map. An example of this functionality is described in chapter 6.

- **Icon modifier:** The last feature is the icon modifier that allows user to create more meaningful maps. This function provides several icons divided in eight categories: *colours*, *numbers*, *letters*, *people*, *culture*, *events*, *transportation* and *sports*. The user, before performing a research, can chose one icon that becomes the default icon marker for the results of the research on the map. This feature becomes very useful when it is used in common with the aggregation function described above. In fact, collectively using these two features, allows to distinguish different researches on the same map just by using different icons that differentiate the resources of a research from the the resources of the other. An example of this functionality is described in chapter 6.

Saving and Loading Map

We analyse now the saving and loading map operations because they have an important role in the application. During the utilisation of Spacetime, these two functionalities become very useful in some situations. We analyse them below:

- As described in section 5.2.5, the Sparql endpoint is a point of failure because it connects Spacetime to the DBpedia dataset. Then if this interface has a problem, Spacetime can not retrieve the information for the user. Hence supposing the Sparql endpoint is not working, the Spacetime research function is not available because it can not retrieve the information passing through the endpoint. In this particular case the saving and loading operations become extremely important and useful because they are independent from the DBpedia endpoint. Then, they can provide to the user its maps even if, the Sparql endpoint service is not working. In general these two operations allows the user to load their saved maps in Spacetime and work on them without performing a second time the research corresponding to the saved map. The aim of the saving and loading operations is to provide a faster way to retrieve maps avoiding the Sparql endpoint execution time.

- Another situation in which these two functions become very useful is when the user wants to perform a research with a lot of results. This researches take a considerable long time to be executed. Then it is convenient for the user to save the map after the first execution and retrieve it, in less time, in a second time using the loading process.

- The saving and loading operations can be very helpful for using the aggregation feature described in the previous section. In fact the user can save two different maps, loads and aggregates them in a second moment without repeating the two researches. Moreover a single stored map can be loaded on Spacetime and expanded, just by aggregating it with a new Spacetime research.

5.1.5 GUI Design

This section provides a description of the graphical user interface (GUI) of Spacetime. The GUI design has taken into account two main aspects for building the user interface:

- **Input mechanisms**: they regard all the web controls (e.g. text box, check box, buttons, calendar fields, ...) that the user can use for insert data;

- **Output mechanism**: they regard all visual output and animation that the application provides to the user.

Taking care about these two important aspects, the application has been designed for trying to be:

- *Graphical nice to the eye*;

- *Intuitive and simple to use*;

- *Effort minimiser*.

The Spacetime interface is composed by three main components:

- A **control panel** for performing research, saving and loading maps;

- A **map** for visualising on the space the events found through a research;

- A **timeline** for visualising during the time the events found through a research.

The control panel is the most important part of the user interface because it contains all the components for performing a research, activating features and saving/loading maps. As we can see from figure 5.2 the control panel it is *tab-based menu* composed by three tabs: the first is used for searching data, the second for the saving and loading map operations and the third for accessing to the Spacetime demo area.

The search tab has been designed in four different area. Three of them are used for constructing a research and the last one is a *features bar* used for activating the available features. From the point of view of the user the research construction consist of three steps:

Step 1: select a DBpedia category from the list;

Step 2: insert a country or a continent and select an entry from the space filter;

Step 3: insert a two date values from the calendars and select an entry from the time filter.

After these three steps the research construction is finished and the user can activate one or more functions from the *features bar*. For making more clear the data insertion, the search tab is divided in three area denoted by a sequential number that corresponds exactly to the three steps just described.

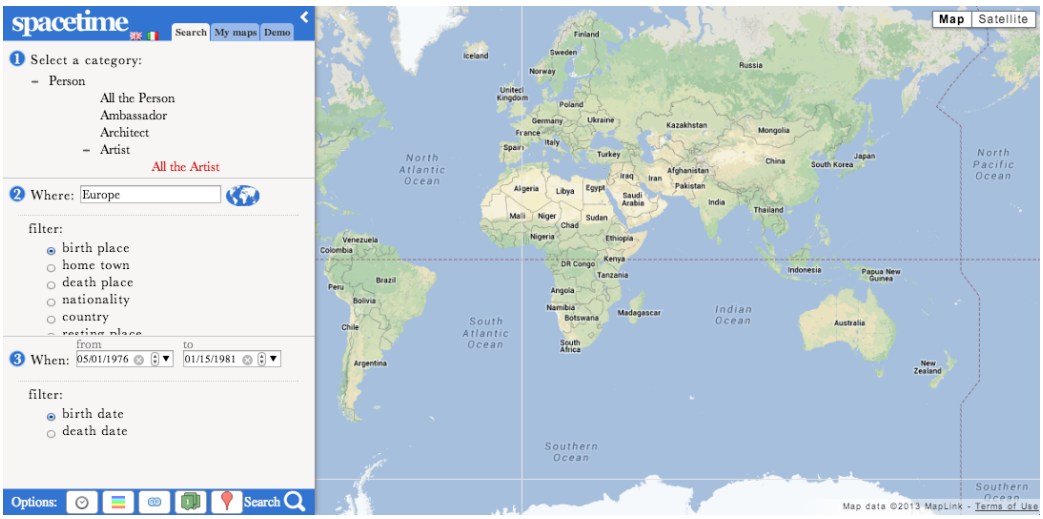

Figure 5.2: A screenshot of Spacetime showing the map and the control panel.

Spacetime has been designed with a structure that leave most of the screen size to the map and the timeline. This choice has been done for facilitate the

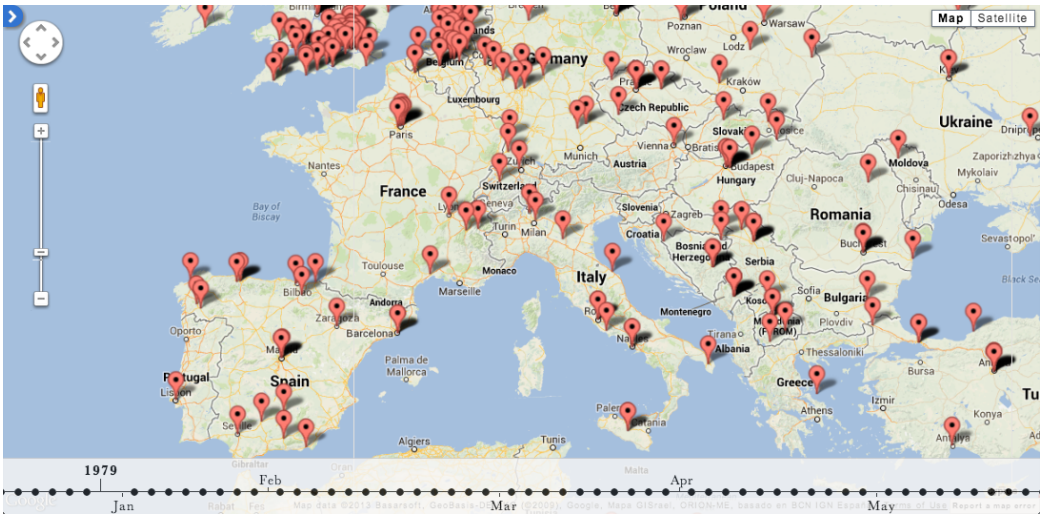

Figure 5.3: A screenshot of Spacetime showing the results set on the map and the timeline.

user in the consultation of the Spacetime map and timeline. In fact when a user performs a research or loads a map, the smart panel, shown on the left in figure 5.2, disappears for leaving the entire space of the screen to the map and the timeline as we can se in figure 5.3.

5.2 Spacetime Technical Description

This section, despite of the previous one, is more technical in fact it describes the Spacetime implementation details, illustrating the components of the architectural structure of the application, the timeline component, the technologies used for the development process, the sparql queries adopted for retrieving the data from Dbpedia and finally the problems of Spacetime.

5.2.1 Architectural Structure

This section is dedicated to the architectural structure of Spacetime. It describes each part that compose the application that is mainly constituted by five components:

- **DBpedia**: it is the knowledge base from which Spacetime extracts data for providing them to the user;
- **Sparql endpoint**: it is an interface that connects Spacetime to DBpedia. This web service is responsible of the Sparql queries execution.

- **Google Maps**: it allows to render the data retrieved from DBpedia on a map, showing a spatial representation of them to the user;
- **Javascript**: it is the core of Spacetime and contains its application logic. It is responsible of all the interactions between the application and the other components;
- **HTML**: it defines the graphical structure of the Spacetime and the dynamic content of the application.

The Javascript core functions, as shown in figure 5.4, is the central component of Spacetime and it manages the interactions with all the other application components. The following analysis illustrates, in details, each of this interaction for making more clear how Spacetime works: (For a better understanding of the following analysis is suggested to look at figure 5.4)

1. **Javascript - Sparql**: The Spacetime data retrieving is performed extracting data from DBpedia through a Sparql endpoint that is an interface used for querying a specific RDF knowledge base associated to the endpoint. Normally when the application retrieves data, it constructs a sparql query, starting from the user input, and it sends it to the sparql endpoint. Hence, Spacetime delegates the sparql endpoint to extract the requested information from DBpedia.
2. **Javascript - HTML**: This kind of interactions are related to the user requests that start from the html application page. In fact each of the request that a user performs such as a research, a map saving, a map loading or a content change, starts from the Spacetime html page and then it is processed by a specific javascript function.
3. **Javascript - Google Maps**: Another important category is the one that involved interactions starting from the Google map. These interactions concern for instance the rendering of the retrieved DBpedia data, the remotion or the editing of a Google marker and the request of the information related to a specific Spacetime events. These interactions are important because they form, in common with the interactions of the previous point, the set of user requests.

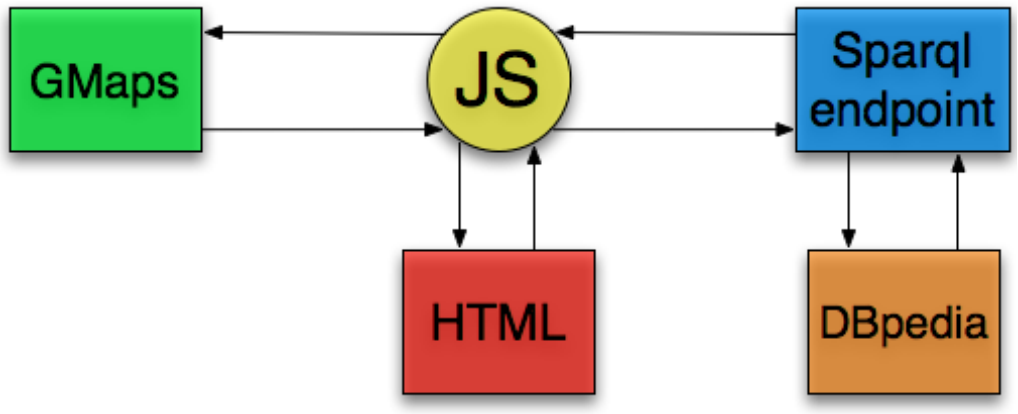

Figure 5.4: The architectural structure of Spacetime.

5.2.2 Timeline

The timeline and the Google map are the most important logical component of the application. They are the base of Spacetime, in fact the Google map represents the space since it provides the spatial dimension while the timeline defines the time dimension. Even if the timeline is a very important component of the application is not mentioned in the architectural structure described in section 5.2.1 differently from the map. This because the timeline is part of the HTML block. This section propose a description of most used timelines found on the web and shows the Spacetime timeline.

The best timelines found on the web are the following:

- **TimelineJS**[1]: the timeline in figure 5.5 is an open-source tool that enables to build interactive timelines that contain media from different sources and it is built in support for Twitter, Flickr, Google Maps, YouTube, Vimeo, Vine, Dailymotion, Wikipedia and SoundCloud.
- **Timeliner.js**[2]: it is a simple, interactive, historical timeline built using HTML, CSS and JQuery. Figure 5.6 shows a screenshot of the timeline.
- **Lateral On-Scroll Sliding**[3]: it is a vertical timeline that can embed links to web site resources. It works on a vertical scrolling and uses animation for the time events appearance implemented using JQuery. Figure 5.7 shows a screenshot of the timeline.

[1] `http://timeline.verite.co/`
[2] `http://www.technotarek.com/timeliner/`
[3] `http://tympanus.net/Tutorials/LateralOnScrollSliding/`

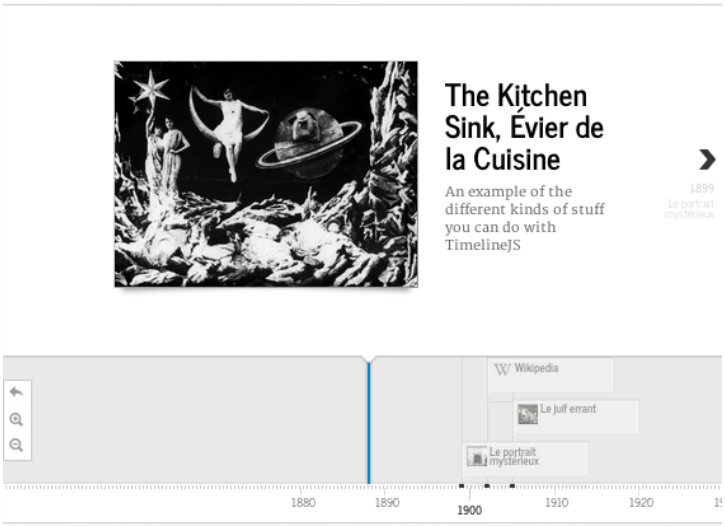

Figure 5.5: A screenshot of the TimelineJS.

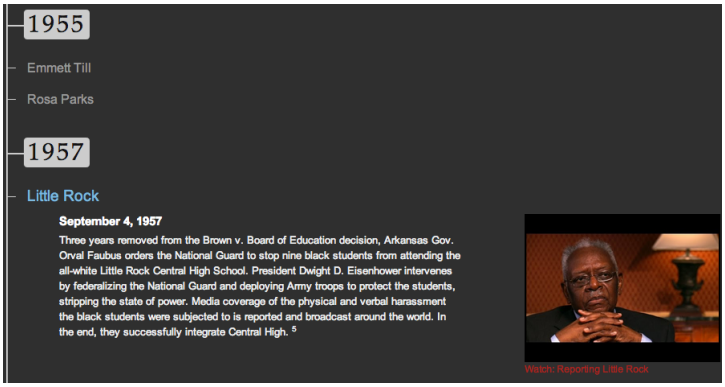

Figure 5.6: A screenshot of the Timeline.js.

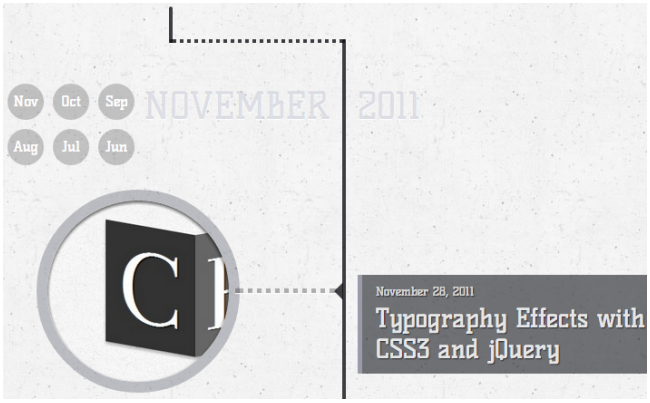

Figure 5.7: A screen shot of the Lateral On-Scroll Sliding.

The problem of using the above timelines is that they are hardly embeddable inside the structure of Spacetime. In fact the timelines seen in the previous classification are very large in term of dimensions and they are made for contains a lot of data such images, videos and long text.

Hence Spacetime adopts a more compact timeline with a minimal design look. It is built using HTML, CSS and JQuery. Building an ad-hoc timeline, as the one shown in figure 5.8, instead of using an existing plug-in, allows to adapt the timeline to the application in the best way. In fact the Spacetime timeline has specific requirements:

- Simple and intuitive for the user;
- It must be dimensionally compact and able to contain a considerable high amount of events;
- It must show a popup containing the space and time information of an events;
- It must contains only the years and months for which exists at least an event.

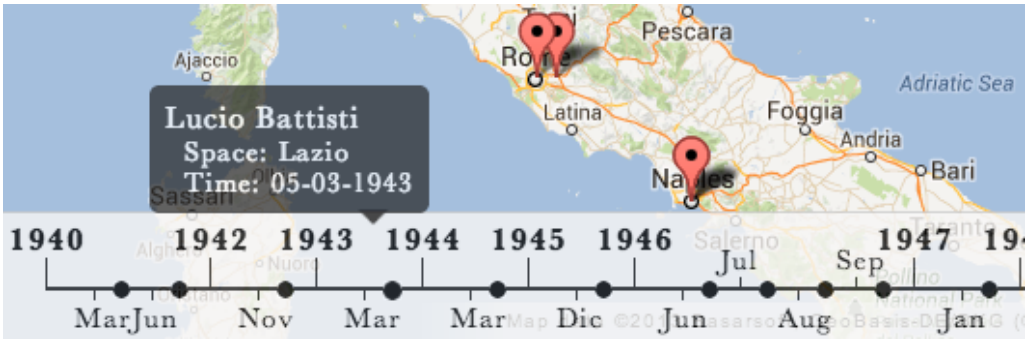

Figure 5.8: A screen shot of the Spacetime timeline.

5.2.3 Technologies used

The technologies utilised in the implementation of Spacetime are the following:

- **Sparql Query Language for RDF**: the query language is used in Spacetime for interrogating the DBpedia dataset through a Sparql endpoint that takes as input a Sparql query and returns as output a JSON file containing the query results;

- **Google Maps JavaScript API v3**: the Google Maps API are used for populating a map with the data contained in the JSON file returned by the Sparql endpoint. Moreover the Google Maps has been very important in the implementation of some additional feature of Spacetime. In fact these API allows to populate the map as an heat map and also to easily change the marker icons of the results on the map;

- **Javascript and JQuery library**: the scripting language and its library define a set of functions that are the core of the application. Javascript represents the core of Spacetime. In particular the JQuery library allows the insertion of animation inside the application;

- **Asynchronous JavaScript and XML (AJAX)**: this technique is very important because permits to send asynchronous request to the Sparql endpoint for retrieving the data requested by the user;

- **JavaScript Object Notation (JSON)**: this text format is handled by the javascript functions for managing the results of a certain Sparql query and for the map saving and loading operations;

- **Cascading Style Sheets (CSS)**: the style sheets language is used for designing and implementing the graphical aspect of Spacetime;

- **HyperText Markup Language 5 (HTML5)**: the markup language is used for developing certain part of the application such as the map saving operation, implemented using the *Blob* object available only using HTML5. Moreover the language allows to insert rounded corners using the *border-radius* CSS property;

- **HTML**: the markup language is used for defining the structure of the application pages.

5.2.4 Sparql queries

This section describes which are the Sparql queries embedded inside the code of the application for retrieving the data corresponding to the user requests. We can separate the queries in three main different groups:

- **Filter queries**: these queries provide the data for the space and time filters in the user interface of the application. These two filters, allows the user to select an attribute inside a set of properties for refining their research. For instance, if a user is performing a research on the *Companies* category, the space filter is composed by a list of attributes

such as city, country or foundation place of a certain company. The time filter instead provides a list of properties like founding date, extinction date or opening date.

The following query is used for building the time filter. It extracts the labels of the DBpedia resources of a specific category, that have an attribute of type *xsd:date*. As we can see from the code, in the *SELECT* statement there are two elements: the *?label* of each attribute and a count expression *(count(?label) as ?cont)*. This expression is used for calculating the occurrences of each *?label* and then ordering them according to a descending order.

```
PREFIX dbp: <http://dbpedia.org/ontology/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
SELECT DISTINCT ?label (count(?label) as ?cont)
WHERE {
    ?event rdf:type dbp:Category;
           ?pred ?x.
    ?pred rdfs:range xsd:date;
          rdfs:label ?label.
FILTER(langMatches(lang(?label), "EN"))
}
ORDER BY DESC(?cont)
```

The following query is used for building the space filter and it extracts the labels of the DBpedia resources of a specific category, that have an attribute with the following characteristics: it is a DBpedia resource and it has a correspondence in latitude and longitude in the Basic Geo (WGS84 lat/long) Vocabulary. Hence the query is finding all the attribute labels that represents a spatial indication.

```
PREFIX dbp: <http://dbpedia.org/ontology/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT DISTINCT ?label (count(?sub) as ?cont)
WHERE {
    ?sub rdf:type dbp:Category;
         ?pred_place ?place.
    ?pred_place rdfs:range ?place_type;
                rdfs:label ?label.
    ?place_type rdfs:subClassOf* dbp:Place.
```

```
FILTER(langMatches(lang(?label), "EN"))
}
ORDER BY DESC(?cont)
```

- **Research queries**: these queries are the most important in the application because they retrieve the data related to the user researches. The queries are built taking the user input: a Spacetime category, a range of time composed by two dates and a spatial indication such as a country, a continent or the entire earth. The following query shows how to retrieve the resources of the *Person* category that are born in Europe between the 1960 and 1970.

```
PREFIX dbp: <http://dbpedia.org/ontology/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX wgs: <http://www.w3.org/2003/01/geo/wgs84_pos#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
SELECT DISTINCT ?label ?sub ?date ?place ?lat ?lon
WHERE {
?sub rdf:type dbp:Person;
     rdfs:label ?label;
     ?pred_place ?place;
     dbp:birthDate ?date;
     dbp:birthPlace ?place.
?place rdf:type ?class ;
       wgs:lat ?lat ;
       wgs:long ?lon.
?class rdfs:subClassOf* dbp:PopulatedPlace .
FILTER ( ?date >= '1960-01-01'^^xsd:date &&
         ?date <= "1970-01-01"^^xsd:date )
FILTER ( ?lat >= "27.6363"^^xsd:float &&
         ?lat <= "81.0088"^^xsd:float &&
         ?lon <= "39.8693"^^xsd:float &&
         ?lon >= "-31.2660"^^xsd:float )
FILTER ( langMatches(lang(?label), "EN"))
}
ORDER BY ASC(?date)
```

As we can see in the above query, the code contains a filter with the dates *1960-01-01* and *1970-01-01* but it does not contain *Europe*. In fact for specifying a certain spatial area we used the bounding box technique. This method associates two values of latitude and two values

of longitude to each country or continent. These coordinates define an area that encloses a country or continent. Hence the second filter in the above query selects only the resources that are inside this area.

- **Resource queries**: these queries are needed for retrieving the information related to a single resource. This data are composed by the abstract of the resource and the list of the language in which the abstract is available for that specific resource.

  The following query is used for recovering the abstract of the city of Berlin but it can be used for any resource in the DBpedia dataset just by substituting *Berlin* with the corresponding name of the desired resource.

```
PREFIX : <http://dbpedia.org/resource/>
SELECT ?abstract
WHERE {
    <http://dbpedia.org/resource/Berlin>
    <http://dbpedia.org/ontology/abstract>
    ?abstract .
    FILTER ( langMatches(lang(?abstract), "EN"))
}
```

  This query instead return the list of languages in which the abstract of the city of Berlin is available.

```
SELECT DISTINCT lang(?abstract) as ?lang
WHERE {
    <http://dbpedia.org/resource/Berlin>
    <http://dbpedia.org/ontology/abstract>
    ?abstract .
}
```

  As we can see from the code the query is very simple and it uses the *lang* operator for extracting only the language of each abstract associated to a specific DBpedia resource.

5.2.5 Problems

This section explains which are the problems, in term of performance, and the limits of Spacetime.

Bounding box

The problem related to the bounding box is it a problem that affects the performance of Spacetime, in particular the quality of the results that it proposes to the users. In fact the bounding box technique is used for filtering the results of a Sparql query and obtaining a set of resources that belong to the same country or continent. First of all, a bounding box is composed by two latitude values and two longitude values. The intersection of these four values describes four geographical points that define a rectangular area that contains a certain country or continent. For instance, as we can see from figure 5.9, Italy is contained in the area defined by the following values of latitude and longitude: +47°5′25.95”, +6°37′12.62”, +36°39′2.98”, +18°30′38.99”.

Figure 5.9: The bounding box of Italy.

The bounding box problem regards the shape that a bounding box defines. In fact these area are rectangular and then they contains more than one country or continent. Hence the Spacetime research, produces a set of results that is not totally correct. In fact part of this results set is composed by

resources that belong to a different country respect to the one chosen by the user for its research. This because the bounding box, filters the results outside the area that it defines, but it considers all the results inside it, including those that belong to a different country from the one chosen by the user. For instance, in figure 5.9 we can see that inside the Italy bounding box are contained part of other countries such as France, Switzerland, Austria, Slovenia, Hungary, Croatia, Bosnia Herzegovina, Tunisia and Algeria.

This is a problem because Spacetime provides a results set composed by two parts, a correct subset and a wrong surplus results subset.

Sparql endpoint

The Sparql endpoint problem regards the weakness of this web service. As described in chapter 3, a sparql endpoint is an interface used for querying a specific RDF knowledge base associated to it. The main problem is that this service represents the only entity that connects Spacetime with the DBpedia dataset. Hence, if the endpoint has a problem, Spacetime can not retrieve the information that it needs. Then the Sparql endpoint becomes one point of failure for Spacetime, and makes the application functions set not entirely available, in the case the endpoint does not work. In fact, when the endpoint has a problem, the users can not perform a research but they can only load their own maps and work on them.

The second problem related to the Sparql endpoint regards the query execution, in fact it can happen that the following error is returned from the endpoint:

Virtuoso 42000 Error SQ200: The memory pool size 80871424 reached the limit 80000000 bytes, try to increase the MaxMemPoolSize ini setting.

In this case Spacetime recognises the anomaly and it shows an error message to the user. The problem is that this error is generated performing researches using certain DBpedia categories, hence the user can not perform researches using them. This is a limitation for the application but it does not depend from Spacetime but from the Sparql endpoint memory pool size.

Resource limits

This section concerns the resource limits problem, already discussed in chapter 3.5. This problem regards the uniformity of the DBpedia data and affects the result quality of the Sparql queries. There are two different problems that are the following:

- The first problem regards the resource properties. If we take the set of the DBpedia resources related to a specific category, we notice that not all of them have all the properties defined in the category. For instance, taking the *Film* category, only 30943 resources of the 71715 belonging to it, have the property *releaseDate*. Therefore the quality of the query results is very affected by this aspects because the 43% of the resources belonging to this specific category can not be queried by Spacetime. This because these resources don't have the property needed for being queried.

- The second problem regards the membership resource category. As described in chapter 2.1.3 DBpedia uses an ontology that is a hierarchical representation of categories. Hence each resource belongs to a category of the ontology. The problem is that sometimes the resources are associated to a category that is too generic. For instance, the Pink Floyd music band has *Organisation* as category instead of *Band* that is an ontology child of *Organisation*. Analysing this case, we can understand that this is a problem for Spacetime because the users can perform a research on *Band* and expects to find Pink Floyd in the results without finding it.

Chapter 6
Use Cases

This chapter is dedicated to some Spacetime use cases and it proposes some hypothetical future implementations. The following description contains some significative functions included in the application and some features that could be added to Spacetime in the future implementations.

The first section related to the use cases is also part of the application. In fact Spacetime includes an area, called *Spacetime Demo*, that shows to the user what are the main functionalities of the application providing a demo for testing them.

6.1 Use cases description

A use case describes a specific use of the application and it gives a useful example of what an application provides to the user. For this reason, a list of the main Spacetime use cases is provided below.

- **Temporal ranking:** Normally when the user performs a research using Spacetime, it obtains a map containing the results corresponding to the research and a timeline in which the event are ordered following the time. In these kind of research there is no indication about the time in the map, but it is present only in the timeline. The *temporal ranking* use case wants to exceed this weakness and provide an order indication directly in the map. Thus this feature permits to obtain more meaningful maps and in the same time to create a temporal ranking of the events resulting from a research.

  For instance, figure 6.1 and 6.2 show respectively the result of the same query using the *temporal ranking* feature or not. It is now clear, the

difference between a normal research and the temporal ranking one. The former does not provide any time indication on the map while the latter suggests for each resource on the map its position in the ranking using a numbered icon.

In particular, figure 6.1 shows the results of a query performed over *University* as category, United States of America as space and 1900 and 1910 as time range. It is important to note that the time filter is set to *founding date*, then the results on the timeline are ordered exactly following the university founding date. The temporal ranking feature allows to bring this temporal order also on the map and improve the visual meaning of the map. The final result, shows in figure 6.1, is the founding temporal ranking of the universities in the United States of America between 1900 and 1910.

The user can activate this feature just by clicking on the temporal ranking button collocated in the Spacetime function bar on the bottom of the Spacetime control panel.

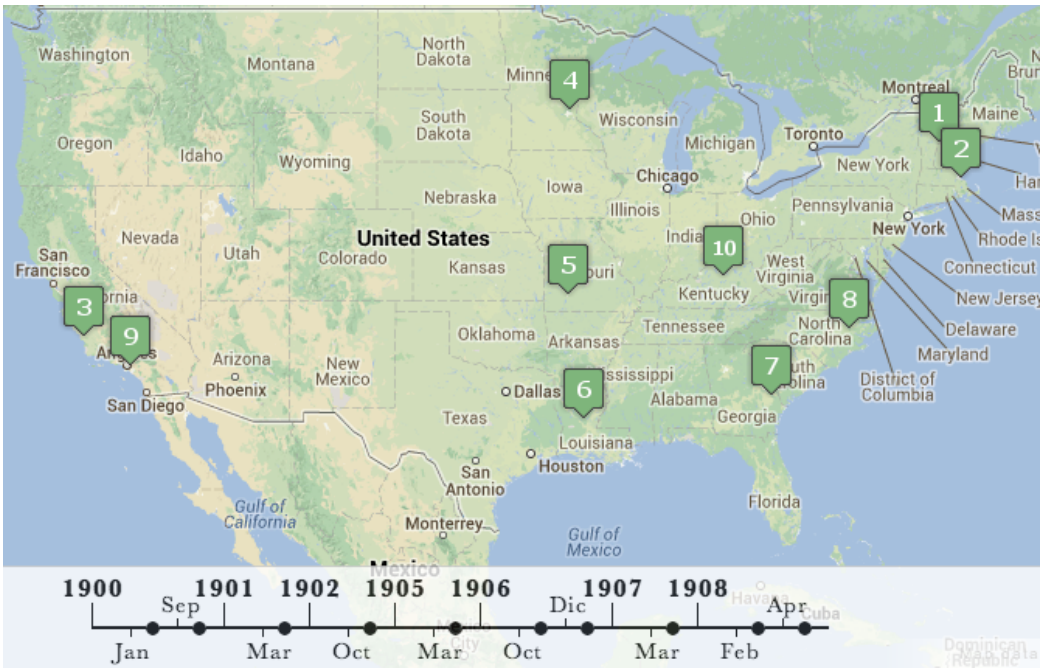

Figure 6.1: The universities founding between 1900 and 1910 in the USA using the temporal ranking feature.

Figure 6.2: The universities founding between 1900 and 1910 in the USA without using the temporal ranking feature.

- **Heat map evolution:** This use case, called *heat map evolution*, shows how the events resulting from a certain research change during the sliding of a particular time range. Spacetime implements this feature using the heat map functionality of Google Maps API v3, that allows to transform a Google map from the normal markers visualisation to a heat map that indicates the density of the markers using a colour gradient. The power of this functionality with Javascript can produce an animation in which the heat map density changes according to the number of events in a certain time instant.

  As we can see from figure 6.3, during the animation, Spacetime provides only the map and a time slider. In fact the timeline disappears but a time slider indicating the sliding of the time is included in the top right of the map. In this way the user can see the geographical density variation of the events taking into account also the time, looking at the map and at the time slider.

  The user can activate this feature just by clicking on heat map and the time sliding button collocated in the Spacetime function bar on the bottom of the Spacetime control panel.

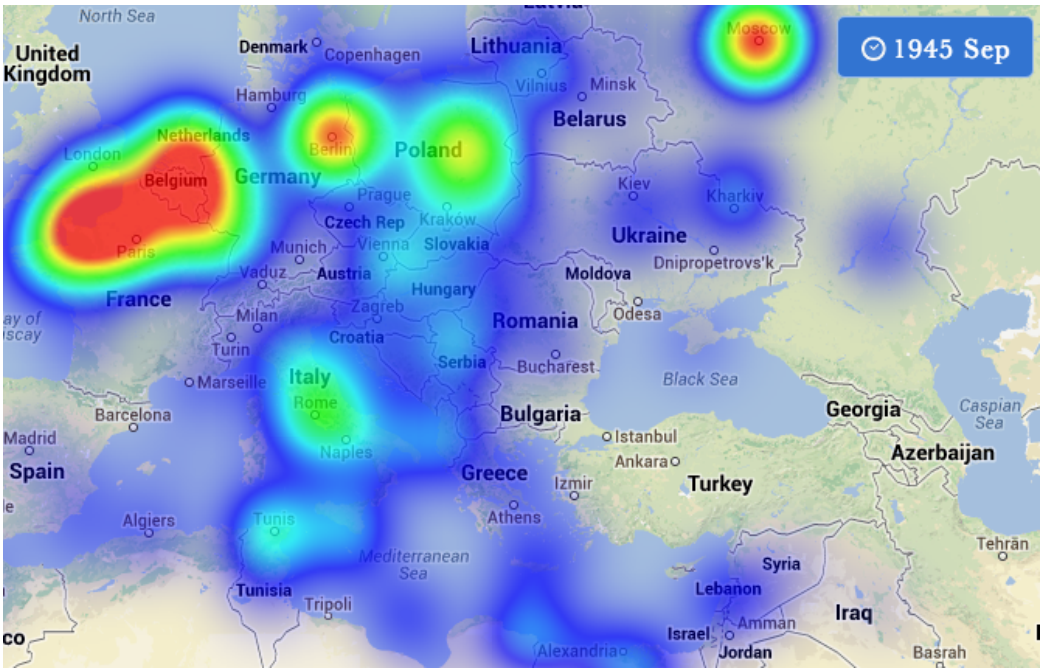

Figure 6.3: A screen shot of the heat map animation of the european military conflict during the Second World War.

- **Time sliding animation:** This use case consists of a time animation, in which the events belonging to a certain research are added to the map one by one following their temporal order. In this use case, the user interface is equipped by a map, a timeline and a time slider. These three elements are coordinated for providing a temporal animation in which each time a marker is added to the map, the corresponding event on the timeline is highlighted and the time value in the time slider is updated. As we can see from figure 6.4, the time slider in the top right corner indicates the current year and the month in which the animation is arrived, the timeline highlights in red colour the events already added to map while the map render each marker with a top down animation.

  The time sliding animation is a useful tool because it provides additional information to the user. In fact looking at the animation the user can understand which events happens before the other and remembers where they are collocated on the map before the map is totally ready. Thus the user can take some indications about some of the events inside a map before consulting it. Hence this functionality can be used only for making animations but also for helping the user in its researches.

  Figure 6.4 is a screenshot of the animation that shows the persons born

in India from 1869 and 1875. On the top right of the figure there is a time slider indicating August 1873 that is the current time denoting the point in which the animation is arrived. The timeline is perfectly synchronised with the time slider in fact it shows as last red highlighted point the event of August 1873.

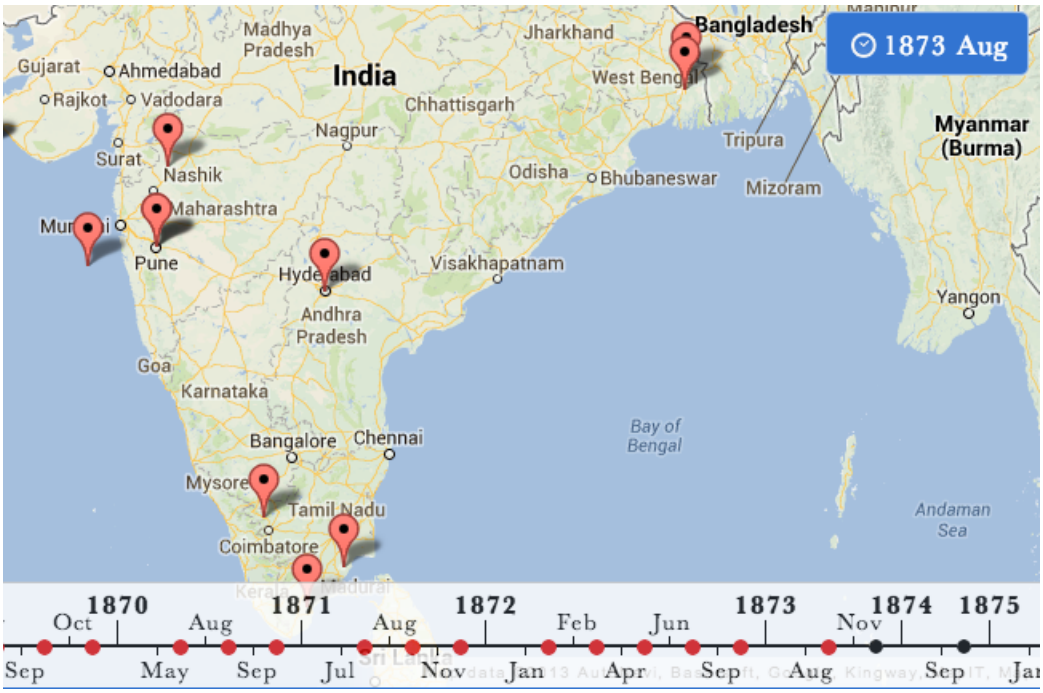

Figure 6.4: A screenshot of the animation map of the persons born in India between 1869 and 1875.

- **Map Aggregation:** Tipically, performing a research with Spacetime means selects a Spacetime category, a country or a continent and a time range indicated by two dates. The result set is composed by the events of the category, country and time range choosen. This is exactly what the application wants to provide to the user but sometimes this is not enough. In fact it could happen that the user wants to display on the same map the results of two different researches. For instance, what if a user wants to join a research performed using time range 1500-1525 with another one performed using time range 1575-1600, without include the events between 1525-1575? The answer is map aggregation, a feature for aggregating different researches in the same map.

  This functionality allows user to create maps that contains for instance: events belonging to different time range, events residing in

different countries or continents and events regarding different categories. These maps could be composed by *writers* and *books*, *athletes* and *sports events*, *elections* performed in Italy, France and Germany or *music festival* placed in United Kingdom and United States of America. Moreover it is possible to join maps with different category, time range and country such as the map containing the italian *movies* of the 90' and the french *film festival* of the 80'.

Figure 6.5 shows the aggregation of several maps that differs from the country on which the relative research was performed. The map in the figure is the result of the aggregation of seven maps performed respectively using China, Vietnam, Taiwan, Philippines, Japan, North Korea and South Korea as country in the research. As we can see from the figure, the map aggregation used in combination with the possibility of changing the icon of a specific research, allows to distinguish better the results on the map.

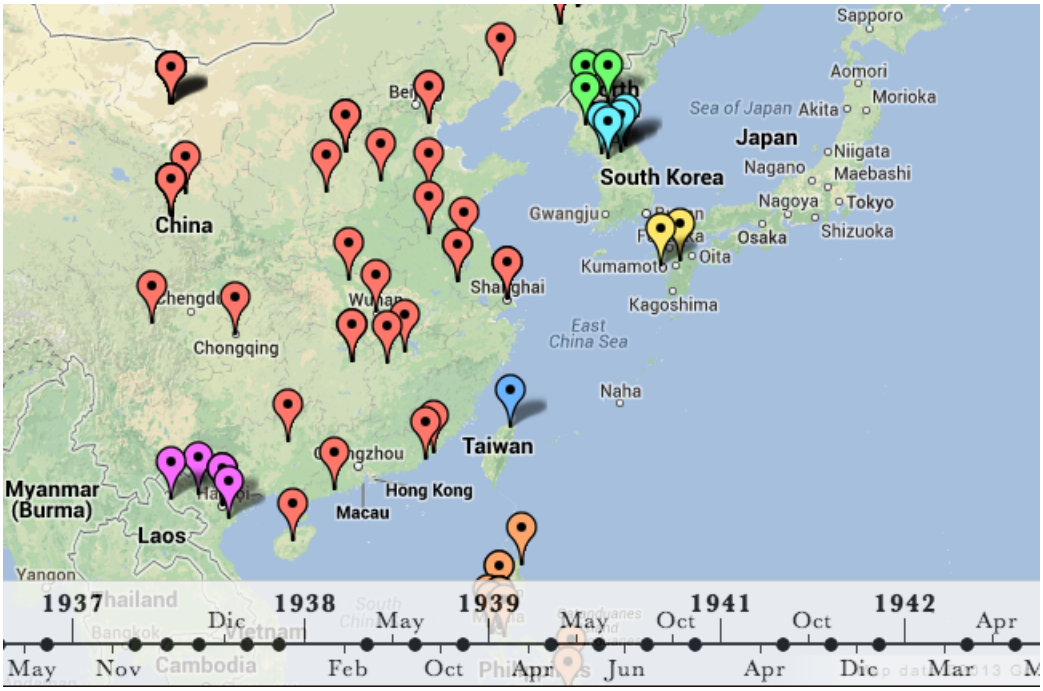

Figure 6.5: A screenshot of the aggregation map resulting by the joining of the military conflict of China, Vietnam, Taiwan, Philippines, Japan, North Korea and South Korea.

- **Map aggregation 2:** We described in the previous use case what is the map aggregation, we analyse now another example of map aggrega-

tion that is quite different from the previous one. The example shown in figure 6.6 represents a summary of all the events associated to the football player Zinedine Zidane. The map is composed by the aggregation of several maps that differs by their category, time range and country. In fact as we can in the figure there are events placed in Italy, France, Spain, Germany, Netherlands and United Kingdom. Moreover these events belong to different category that are *Person* in the case of the marker indicating the born of the player, *soccer team* in the case of the foundation event of one of the team of the Zidane and *football match* for the remain markers indicating some of the most important victories and loss of the player.

The particularity of the map is given by the complexity of building it. In fact it is not possible to create automatically a map that contains all the events related to a person using Spacetime because not all the resources contains all the references of all the person that are involved in. Then constructing this kind of map needs more time and a significant background by the user about the context on which is going to perform its researches.

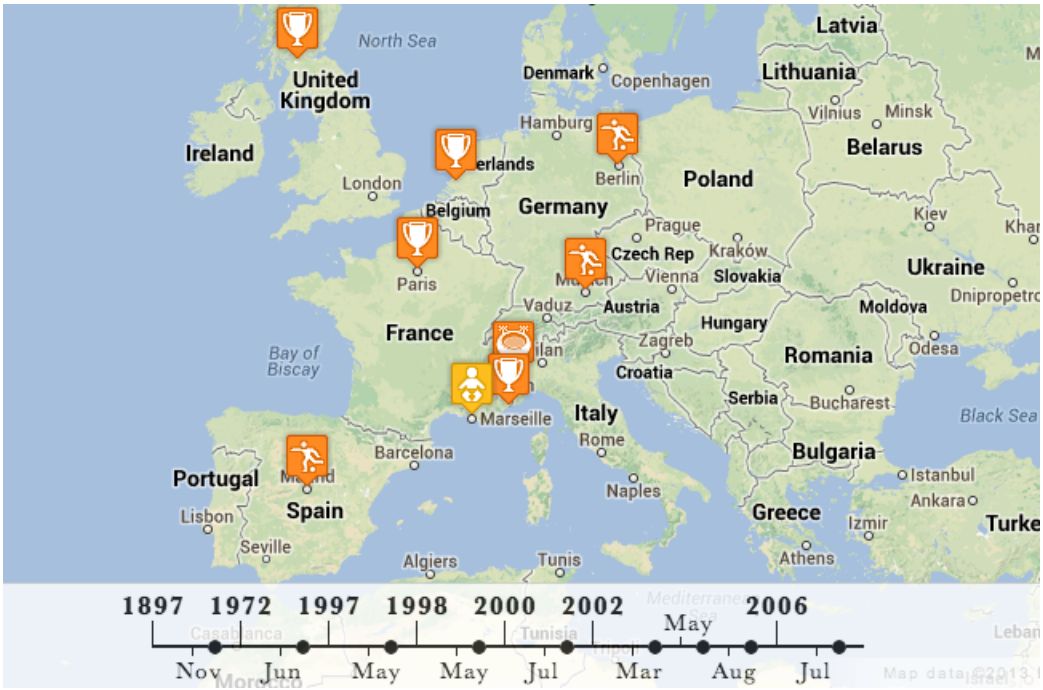

Figure 6.6: The more important events of the Zinedine Zidane soccer carrier.

6.1.1 Future Implementations

This section illustrates some future implementations that could be extensions of the existing version of the application.

- **Add-hoc extension:** Once the application is ready and equipped with its set of features, it is possible to include some additional add-hoc functionalities. For instance, supposing we are performing a research on *Writers* and consulting a certain writer in the result set, supposing Pier Paolo Pasolini. It would be interesting for the user to know who are the writers influenced by Pasolini and who have influenced Pasolini.

  This task is easily implementable in Spacetime but it remains an add-hoc functionality available only for *Writers*. Then the idea behind the add-hoc extensions is to select the most interesting categories in Spacetime and extend the application for making possible interaction such as the influences of a writer, the commander of a battle, the stadium of a sportive competition or whatever is not included in the resource abstract showed by Spacetime.

Chapter 7
Conclusions

The aim of the application developed during the internship was:

- Construct an application able to solve the user interface usability problem of the existing applications based on DBpedia;
- Construct a new application, different from the existing ones, that provides a possible real tool for users.

The resulting application provides a method for extracting data from DBpedia using a simple interface based on two main objects: a Google map and a timeline. This interface shows in a clear way the results obtained by the user requests.

Moreover it uses only the resources in the DBpedia dataset that have a spatial and temporal property. In this way the application restricts the huge amount of data inside the DBpedia dataset.

Finally Spacetime provides a new tool, different from the existing applications based on DBpedia, that could be a new way of searching information.

Bibliography

[1] Sören Auer et al. "Dbpedia: A nucleus for a web of open data". In: *The Semantic Web* (2007), pp. 722–735.

[2] Tim Berners-Lee. "Linked data-design issues". In: (2006).

[3] Christian Bizer et al. "DBpedia-A crystallization point for the Web of Data". In: *Web Semantics: Science, Services and Agents on the World Wide Web* 7.3 (2009), pp. 154–165.

[4] Diego Valerio Camarda, Silvia Mazzini, and Alessandro Antonuccio. "LodLive, exploring the web of data". In: *Proceedings of the 8th International Conference on Semantic Systems*. ACM. 2012, pp. 197–200.

[5] Rasmus Hahn et al. "Faceted wikipedia search". In: *Business Information Systems*. Springer. 2010, pp. 1–11.

[6] Steve Harris and Andy Seaborne. "SPARQL 1.1 query language". In: *W3C Working Draft* 14 (2010).

[7] Philipp Heim, Jürgen Ziegler, and Steffen Lohmann. "gFacet: A Browser for the Web of Data". In: *Proceedings of the International Workshop on Interacting with Multimedia Content in the Social Semantic Web (IMC-SSW'08)*. Vol. 417. Citeseer. 2008, pp. 49–58.

[8] Graham Klyne, Jeremy J Carroll, and Brian McBride. "Resource description framework (RDF): Concepts and abstract syntax". In: *W3C recommendation* 10 (2004).

[9] Talis podcasts.s3.amazonaws.com. *Sir Tim Berners-Lee Talks with Talis about the Semantic Web*. 2012. URL: `http://talis-podcasts.s3.amazonaws.com/twt20080207_TimBL.html`.