

An OWL Demography Ontology made with Protégé, published on a Sesame repository and available through a web front-end made in PHP, XHTML and JavaScript.

Giuseppe Alessandro, Luciano De Franco, Carlo Leonardi

Dipartimento di Informatica e Telecomunicazioni, Facoltà di Ingegneria, Università di Catania, Catania, Italy

{alexgpeppe,luciano.defranco,carloleonardi83}@gmail.com

Abstract. This project implements an ontology that represents the population of each municipality belonging to each province in a given year. For each municipality in a given year, the amount of population partitioned by sex, age and marital status is stored in a database. The ontology has been written in OWL using Protégé and manual editing and was published with the Sesame framework. Therefore we have developed (using PHP, XHTML and JavaScript) a web front-end to query the ontology. The front-end allows you to perform a simple search based on key descriptive fields (*simple search*), a search based on location using Google map (*geo search*), a vision of the temporal evolution of the population (*growth*) and a free search using the SPARQL language (*free search*).

Keywords: Demography, Ontology, OWL, RDF, Protégé, Sesame, PHP.

1 Introduction

The Semantic Web is a mesh of information linked up in such a way as to be easily processable by machines, on a global scale. You can think of it as being an efficient way of representing data on the World Wide Web, or as a globally linked database.

The Semantic Web is not about links between web pages. The Semantic Web describes the relationships between things (like A is a part of B and Y is a member of Z) and the properties of things (like size, weight, age, and price).

The Semantic Web is generally built on syntaxes which use URIs to represent data, usually in triples based structures: i.e. many triples of URI data that can be held in databases, or interchanged on the World Wide Web using a set of particular syntaxes developed especially for the task. These syntaxes are called "Resource Description Framework" syntaxes.

Therefore the RDF is a language for describing information and resources on the web. Putting information into RDF files, makes it possible for computer programs ("web spiders") to search, discover, pick up, collect, analyze and process information from the web. For example if information about music, cars, tickets, etc. were stored in RDF files, intelligent web applications could collect information from many different sources, combine information, and present it to users in a meaningful way.

It has taken years to put the pieces together that comprise the Semantic Web, including the standardization of RDF, the W3C release of the Web Ontology Language (OWL), and standardization on SPARQL, which adds querying capabilities to RDF. So with standards and languages in place, we can see Semantic Web technologies being used by early adopters.

Semantic Web technologies are popular in areas such as research and life sciences where they can help researchers by aggregating data on different medicines and illnesses that have multiple names in different parts of the world. On the Web, *Twine* is offering a knowledge networking application which has been built with Semantic Web technologies. The *Joost* online television service also uses Semantic technology on the back-end: here Semantic technology is used to help Joost users to understand the relationships between pieces of content, enabling them to find the types of content they want most. Oracle offers a Semantic Web view of its Oracle Technology Network, called the *OTN Semantic Web* to name a few of those companies who are implementing Semantic Web technologies.

About our project, we created an ontology which describes the demographics of Italian municipalities. For each municipality in a given year, the amount of population partitioned by sex, age and marital status is stored in a database.

2 Ontology

Let's see in details all about our ontology: the schema, the languages and the tools we used to write it, the data sources, etc.

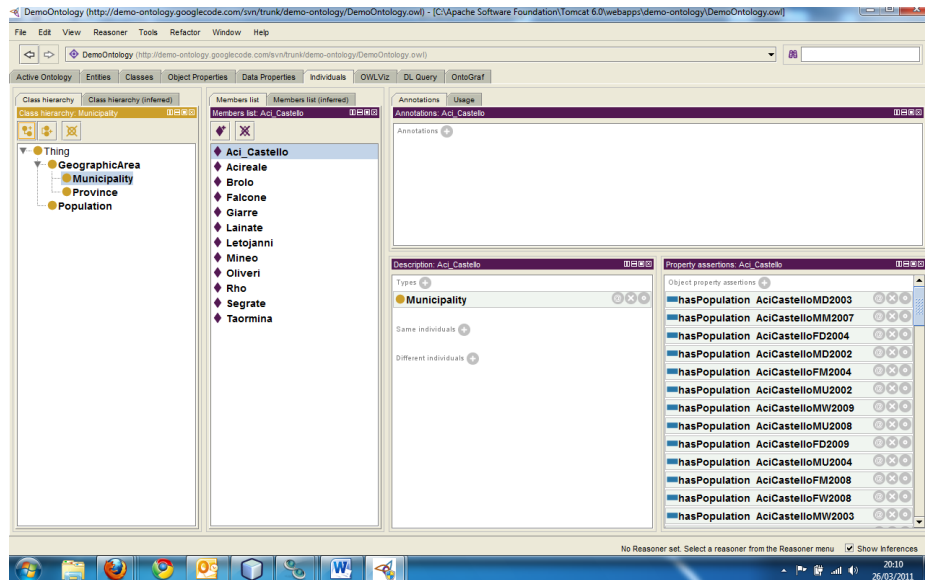
2.1 Preliminary choices

Before starting writing our ontology, we tried to decide what we had to represent exactly. Demography is the science studying in a quantitative way the phenomena pertaining to the state and the evolution of the population. In particular we distinguish two fundamental kinds of demography: the static one and the dynamic one. The first aims to take a picture of the population living in a certain region at a certain time. The latter instead focuses on estimating how population changes over time.

We decided to start from real data, so at first we looked for what the internet offered about. We found the Istituto Italiano di Statistica (ISTAT) provides on its website www.demoistat.it the most recent official data about population resident in Italian municipalities, coming from surveys performed by registry offices. In particular, the “resident population” section contains the number of people resident in all Italian municipalities on 1st January of every year, partitioned by age, sex and marital status. Information is stored in a traditional database you can query by means of a web front-end. Data from each municipality in a given year are also downloadable in cvs (comma-separated value) format. Therefore we decided to go by these data and we created our ontology schema to represent these pieces of information at best.

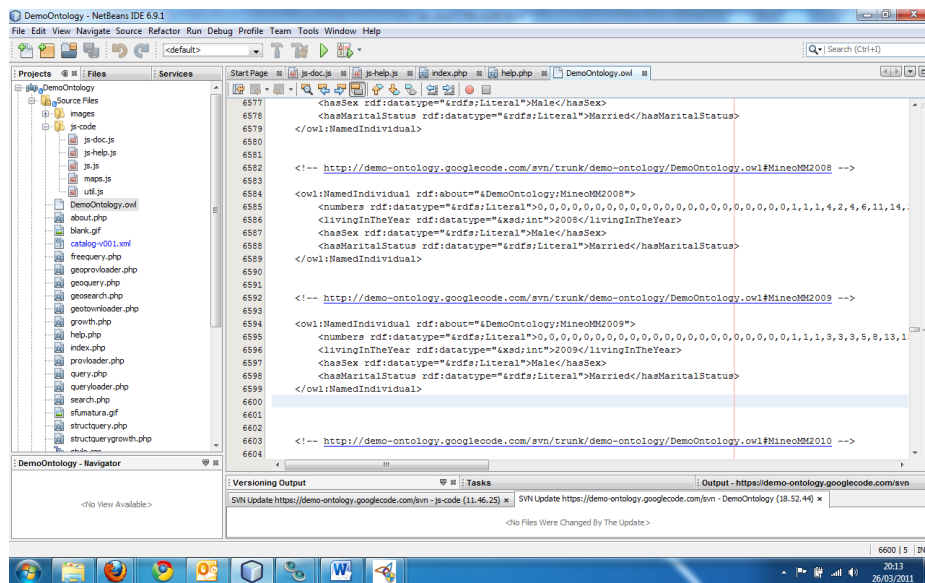
2.1 Used tools

To create our ontology schema and to insert the basic descriptive pieces of information and the first instances of population, we used Protégé framework (version 4.1.0 build 217). Protégé is a free, open source ontology editor developed at *Stanford University*. The Protégé platform supports two main ways of modeling ontologies via the Protégé-Frames and Protégé-editors. Protégé ontologies can be exported into a variety of formats including RDF(S), OWL, and XML Schema.



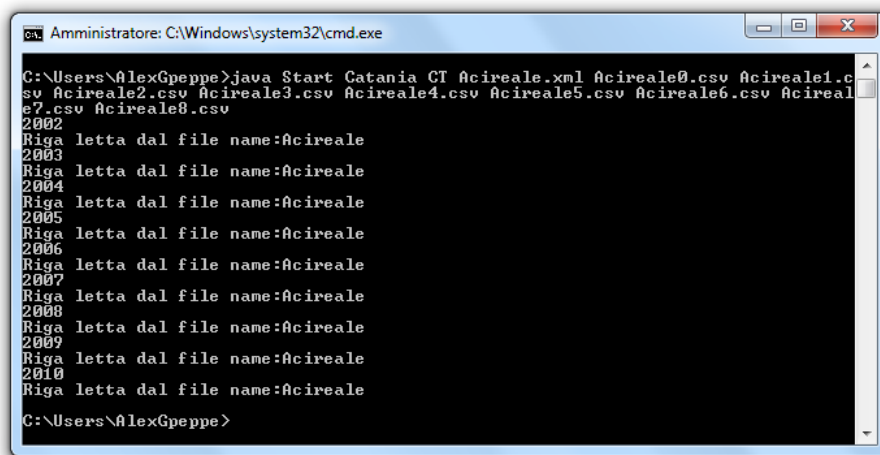
Picture 1 – Protégé usage

To insert further instances of population from ISTAT data, we manually edited by means of a textual editor the OWL file generated by Protégé.



Picture 2 – Ontology editing manually

To facilitate this boring operation of converting tabular data collected from ISTAT database to XML strings making up the OWL file, we created a simple Java utility which reads the cvs file got from www.demoistat.it for every municipality in a given year, processes it, and returns the corresponding XML strings which have to be inserted into the OWL file. This utility takes as input more csv files at the same time from the same municipality, so after having got the XML output you can easily insert into the OWL file the data of that municipality just in a single cut-and-paste operation.



```

C:\Users\AlexGeppe>java Start Catania CT Acireale.xml Acireale0.csv Acireale1.c
su Acireale2.csv Acireale3.csv Acireale4.csv Acireale5.csv Acireale6.csv Acireal
e7.csv Acireale8.csv
2002
Riga letta dal file name:Acireale
2003
Riga letta dal file name:Acireale
2004
Riga letta dal file name:Acireale
2005
Riga letta dal file name:Acireale
2006
Riga letta dal file name:Acireale
2007
Riga letta dal file name:Acireale
2008
Riga letta dal file name:Acireale
2009
Riga letta dal file name:Acireale
2010
Riga letta dal file name:Acireale
C:\Users\AlexGeppe>

```

Picture 3 – Java utility usage

2.3 Ontology structure

Our ontology plans the **GeographicArea** class, which represents a generic geographical area to which the geographical data will relate.

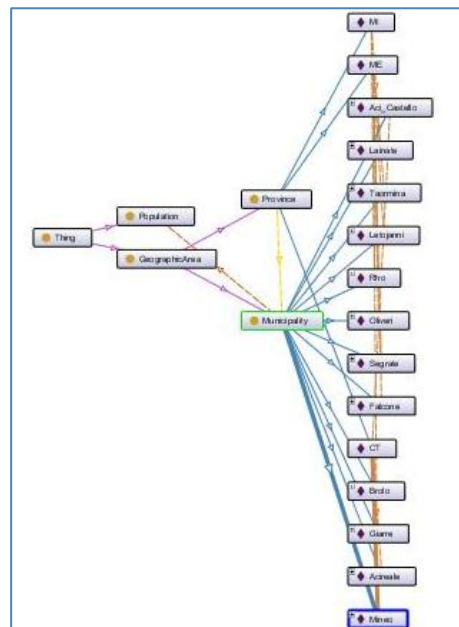
Any instance of the **GeographicArea** class has the following **DataProperties**:

- *extends*. This property expresses the extent of a geographical area. Its value is expressed in square kilometers.
- *hasName*. This property expresses the name of the geographical area.
- *isLocated*. This property expresses the geographic coordinates of a geographical location. Its values are expressed as a pair of decimals: the first one represents the longitude (positive for East and negative for West), and the second one represents the latitude (positive for Nord and negative for South).

The Ontology has two subclasses of **GeographicArea**, each of them represents the concret territorial entity we consider: **Province** and **Municipality**. The instances of the **Province** class will be bound to the ones of the **Municipality** class through the *hasMunicipality* **ObjectProperty**. The instances of the **Municipality** class instead will be bound to the instances of the **Population** class through the *hasPopulation* **ObjectProperty**. That way implements the hierarchical bond **Province** → **Municipality**

Now let's see how the Population class allows you to store demographic pieces of information. It has the following DataProperties:

- By these properties the Ontology can represent the whole information stored into the ISTAT's database, working to our advantage by allowing us to perform more powerful and effective queries based on the subject \rightarrow predicate \rightarrow object bonds and on the SPARQL query language, even from automatic processing tools having not an a priori knowledge about the content of the ontology.



Picture 4 – Ontology representation by using OntoGraph

2.4 Ontology publishing

We published our Ontology on a SESAME repository. SESAME is an open source framework for storage, inferencing and querying of RDF data. We used the 2.2.4 version: we installed it on an TOMCAT server just downloading the zip file of SDK from the sourceforge repository, and extracting two files into TOMCAT's webapps folder: `openrdf-workbench.war` and `openrdf-sesame.war`. The first one provides a web front-end which allows you to access the information on SESAME repositories through a browser; the second one provides access to SESAME server by dedicated clients using a specialized protocol based on HTTP requests.

We published our Ontology through the web front-end by creating a new repository we named *demography* and uploading the OWL file containing RDF data.

3 Front-End

With the aim to allow a more simple interaction with the Ontology even from not-skilled human users, we created a web front-end. We started from what the front-end of ISTAT's website already offers, and we enriched data presentation by means of explicative graphics and by inserting more powerful query features.

3.1 Used languages and tools

Because of dealing with a web front-end, the final language we used to represent information is obviously HTML. In particular, we based on the 1.1 Strict version of XHTML specification. By following this rules, we used HTML only to describe information, whereas we delegated to the CSS language everything dealing with the representation of the information.

For the whole part of information processing on the tomcat server, such as Ontology querying on SESAME, the language we used was PHP version 5.2.6.

For the client-side processing, such as editing forms or creating graphics, the language we used was JavaScript.

As our team is formed by three elements, we relied on a code repository to synchronize our project. In particular, we used the free SVN server provided by Google Code. All the front-end source code can be found at the web address <http://code.google.com/p/demo-ontology/>.

We used the NetBeans IDE 6.9.1 (Build 201007282301) development tool as source code editor and SVN client.

3.2 Provided features

Let's analyse the top-level features which are provided by the front-end, postponing the implementative details which make them possible.

Simple Search. This page allows you to query the SESAME repository to get information about population living in a given year in a given geographical area (a specific municipality, or a province, or the whole Italian territory), according to the selected sex and marital status. After having filled the form on the left side of the page, by clicking the "Start Query" button a graphic and a table appear: they show the result of user's query. The graphic shows the total amount of people which forms the population partitioned by age. The table shows the amount of people which forms the population partitioned by age, sex and marital status.

Geo Search. This page allows you to query the SESAME repository to get information about population living in a given year in a given province or municipality. After having filled the form on the left side of the page, by clicking the "Show" button the map shows the selected province with a markerplace located on the selected municipality. By clicking it, a window containing a summary of the municipal demographic information appears.

Growth. This page allows you to query the SESAME repository to get information about the growth of the population in a given municipality in a given period of time. After having filled the form on the left side of the page, by clicking the "Start Query" button a graphic appears showing the total amount of population living in each year in the selected period.

Free query. This page allows you to query the SESAME repository in a direct way by editing yourself the query. After having edited the query and having clicked the "Start Query" button, the result of your submission is shown in a table.

3.3 Implementative details

Now let's focus on the implementative details underlying the features provided by the front-end.

All the queries to the SESAME repository are submitted to the server via HTTP GET requests. The PHP interpreter builds the query string according to the piece of information it has to get from the repository, then it creates an HTTP GET request using the string just built and adds an header specifying the results has to be in the SPARQL Query Results XML format, and it sends it to the SESAME server. After having received the response, the PHP interpreter reads it as a string and creates a SimpleXML element from it, to be able to parse the content of the response.

All interactions to the SESAME server are based on this mechanism. They aren't used just to query the repository to retrieve the data the user wants to get, but they are performed also to get pieces of information while rendering the options in the forms: this allows the web page to let the user submit always consistent queries, and to prevent him from querying the repository about not stored data.