Semantic Web
Practical Work - Report
5ISS-A1
INSA Toulouse

Cécile DUTHOIT, Linn MJELSTAD

January 2017

1 Introduction

This practical work aimed to allow us to manipulate the notion of ontology, see the major aspects of this notion, and understand how a reasoner can understand an ontology. This document will describe what we did and what we retain from our work sessions.

The context of the project we worked on during these sessions is to develop a smart weather application.

2 Using Protégé

During these work sessions, we used Protégé, a free, open source ontology editor and a knowledge management system, developed at Stanford University. It provides a graphic user interface to define ontologies and includes deductive classifiers to validate that models are consistent and to infer new information based on the analysis of an ontology.

3 Build an ontology from a set of assertions

3.1 Types and instances

From a set of assertions, we used Protégé to build our weather ontology. The purpose was to establish a vocabulary to describe data in a semantic way. Here are the assertions we traduced:

1. "Nice weather and bad weather are two types of phenomenons."
2. "Rain and fog are types of bad weather and sun is a nice weather phenomenon."
3. "Temperature, hydrometry, rainfall, atmospheric pressure and wind speed are measurable parameters (not types of parameters but instances of parameters)."

4. "The word température is a French synonym of temperature."
5. "Wind strength is similar to wind speed."
6. "Cities, countries and continents are locations."

Figure 1 represents the way we translated all of these assertions.

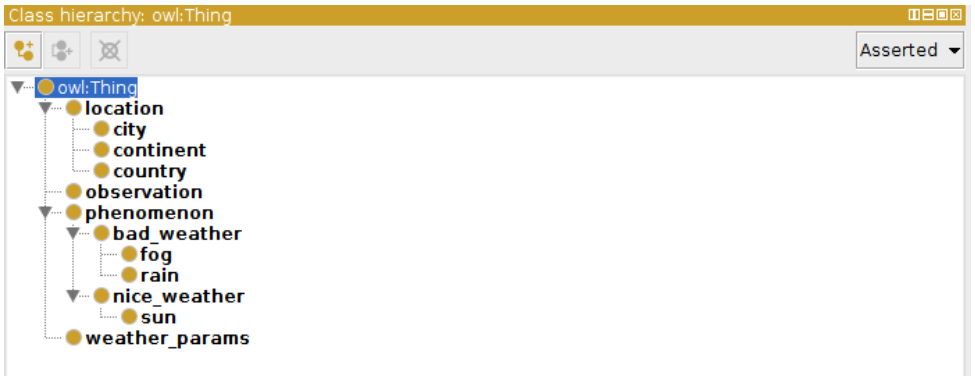


Figure 1: Our implementation of the assertions.

Protégé allows us to create different labels for the same object in different languages. Temperature is an instance of the class measurable parameters, and we added a label "Température" to be used in french. We did the same with the wind strength, which is another way to mention the wind speed.

3.2 Properties

Then we added properties that will not be exhaustively described -in order to make this document humanly readable in a reasonable period. Two examples of the properties we implemented are described below.

We distinguish between data properties and object properties:

- **Data properties:** links a data to an object.
  Example: "A phenomenon lasts for a period in minutes."
  We created a data property called "lasts for" to link an object Phenomenon to a Double value.

- **Object properties:** links an object to another object.
  Example: "A phenomenon leads to an observation"
  We created an object property called "whom symptom is" to link an object Phenomenon to an object Observation.

3.3 Populating our light ontology

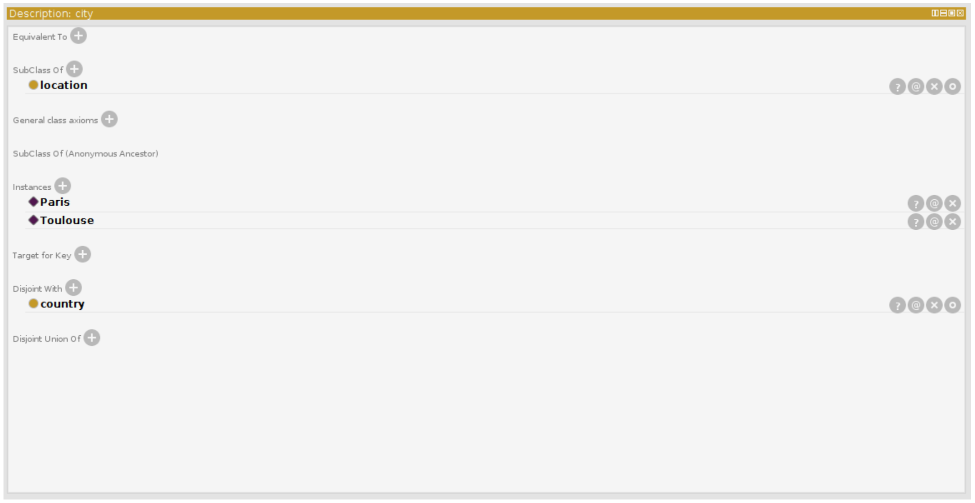
We populated our ontology from another set of assertions, such as:

- ”Toulouse is a city”
  We created an instance of City (which is a children of Location) called ”Toulouse”.
- ”Toulouse is located in France.”
  We added an instance of Location called ”France”. Then we created an instance of the object property ”is located in” and use it to link the two locations ”Toulouse” and ”France”.

3.4 Draft of an heavy ontology

In this part, we forced and limited some types on the basis of a set of assertions, such as:

- ”A city cannot be a country.” (in practice, what about Singapore, Vatican, etc.?)
  As shown in Figure 2, we disjointed both objects City and Country so that a city could not be a country (and vice versa).



Description: city

Equivalent To

SubClass Of
- location

General class axioms

SubClass Of (Anonymous Ancestor)

Instances
- Paris
- Toulouse

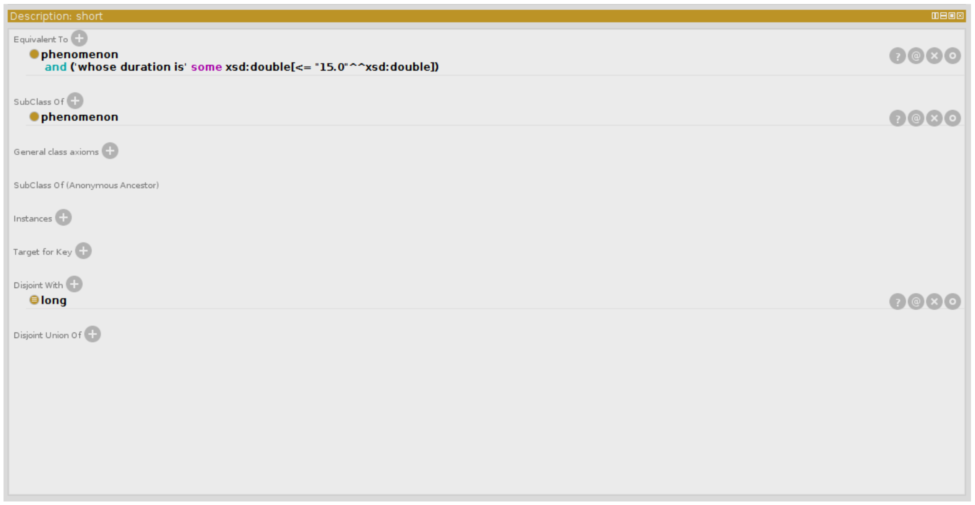
Target for Key

Disjoint With
- country

Disjoint Union Of

Figure 2: A city is characterised as disjoint from a country.

- ”A short phenomenon is a phenomenon that lasts for less than 15 minutes.”
  As shown in Figure 3, we created a child ”Short” of the object Phenomenon and we specified a maximum for it.


Description: short

Equivalent To

```
phenomenon
 and ('whose duration is' some xsd:double[<= "15.0"^^xsd:double])
```

SubClass Of

```
phenomenon
```

General class axioms

SubClass Of (Anonymous Ancestor)

Instances

Target for Key

Disjoint With

```
long
```

Disjoint Union Of

Figure 3: A short phenomenon lasts for less than 15 minutes and is disjoint from a long phenomenon.

3.5 Populating our heavy ontology

We enriched our ontology on the basis of a set of assertions. What the reasoner understood when we started it is described below each assertions:

- **"France is located in Europe."**
  Europe is a location because the property "is located in" links two Locations, but it does not know if Europe is a City, a Country, or a Continent since the property does not precise anything else. It also learns that Toulouse is located in Europe (transitivity).
- **"Paris is the capital of France."**
  Paris is located in France because the capital of a country is located in this country and also in Europe (transitivity). Paris is a city since a capital is a City.
- **"The *City of Lights* is the capital of France."**
  Because we precised that a Country has only one capital, the reasoner understands that the City of Lights is another name of Paris. That means that the City of Lights is a City, located in France, in Europe, etc.
- **"Monaco is a city and a country."**
  Since we precised that City and Country are disjoint, the reasoner refuses to accept the second type we add.

3.6 Questions

3.6.1 What does the reasoner know about A1?

The reasoner only knows that A1 is a Thing because we created it at the top level of the ontology since we had no info about its type. We list all the info that the reasoner can get about A1:

- P1 is a symptom of A1
- One of A1's symptoms measured a rainfall of 3 mm in Toulouse at instant I1
- One of A1's symptoms was observed in France (in Europe)
- And so on (with all info the reasoner knows about Toulouse, I1, etc.)

3.6.2 What does the reasoner know about Paris?

Because we included that each country has one unique capital, the reasoner knows that Paris and the "City of Lights" are the same city. We list all the info that the reasoner can get about Paris:

- Paris is the capital of France.
- Paris is a city (because capitals are cities).
- Paris is located in France (since a capital is included in its country).
- Paris is located in the same country as Toulouse (since Toulouse is located in France).
- Paris is also called the "City of Lights" (since the "City of Lights" is also the capital of France and this capital is unique).
- Paris is located in Europe (since France is located in Europe and a location included in another is also included in the location in which the first one is located).
- Paris is not a country (since an instance of a city cannot be a country).

3.6.3 What is the reasoner's reaction if Toulouse is declared as the capital of France?

Since we forced a unique capital for each country, the reasoner will understand that "Toulouse" is another name of Paris, just like the "City of Lights" was another name of Paris.

4 Data enrichment and Java reasoner

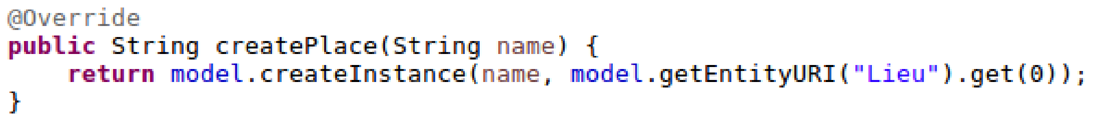
The purpose of this part was to reuse the ontology we made previously and use an open-source data set to enrich these data. We worked on completing a java project that would use the structure we created in Protégé.

Unfortunately, due to the shortness of time of the practical work sessions, we only implemented the missing functions in the IModelFunctions file in the Java project. Afterwards, we ran the Java reasoner to check if it understood everything correctly, and we corrected the errors that occurred before moving on.

We implemented functions that could create instances of a type (create place, create instant) and functions that would retrieve information (get timestamp of an instant, get the URI of an instant).

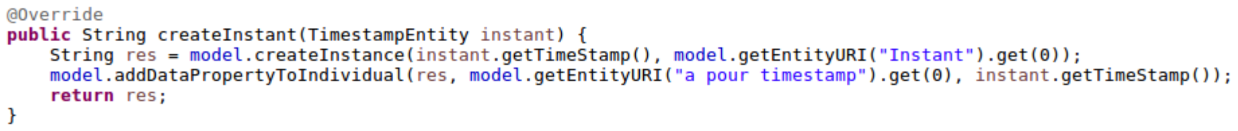
We wanted to use our ontology to describe, in a semantic matter, the data that we got from a open data dataset. The data comes in a CSV format, and it would have to be converted by our application into 5 star classified data; *”link your data to other data to provide context”*.

In our case, it is the DoItyourselfControler that would take the data from the dataset and use the functions we created in the IModelFunctions to populate the ontology. The main function of the DoItYourselfControler is to instantiate an observation, so it would have to create all the different types of instances (place, phenomena, instant, etc.) and link them all together using the object proprieties and data proprieties we have already defined.



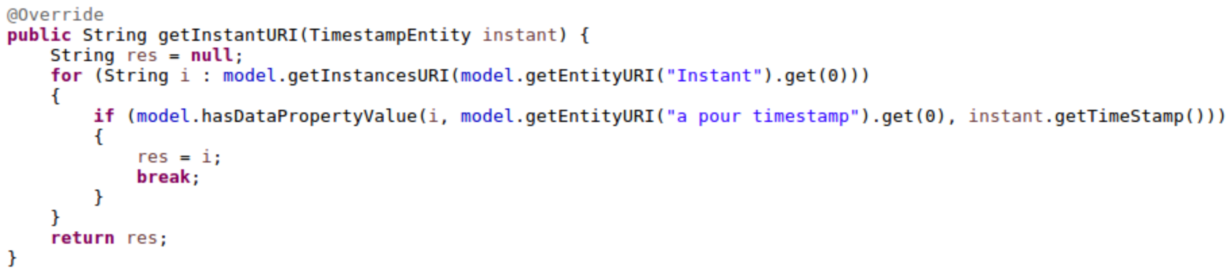
```java
@Override
public String createPlace(String name) {
    return model.createInstance(name, model.getEntityURI("Lieu").get(0));
}
```

Figure 4: Create a place.



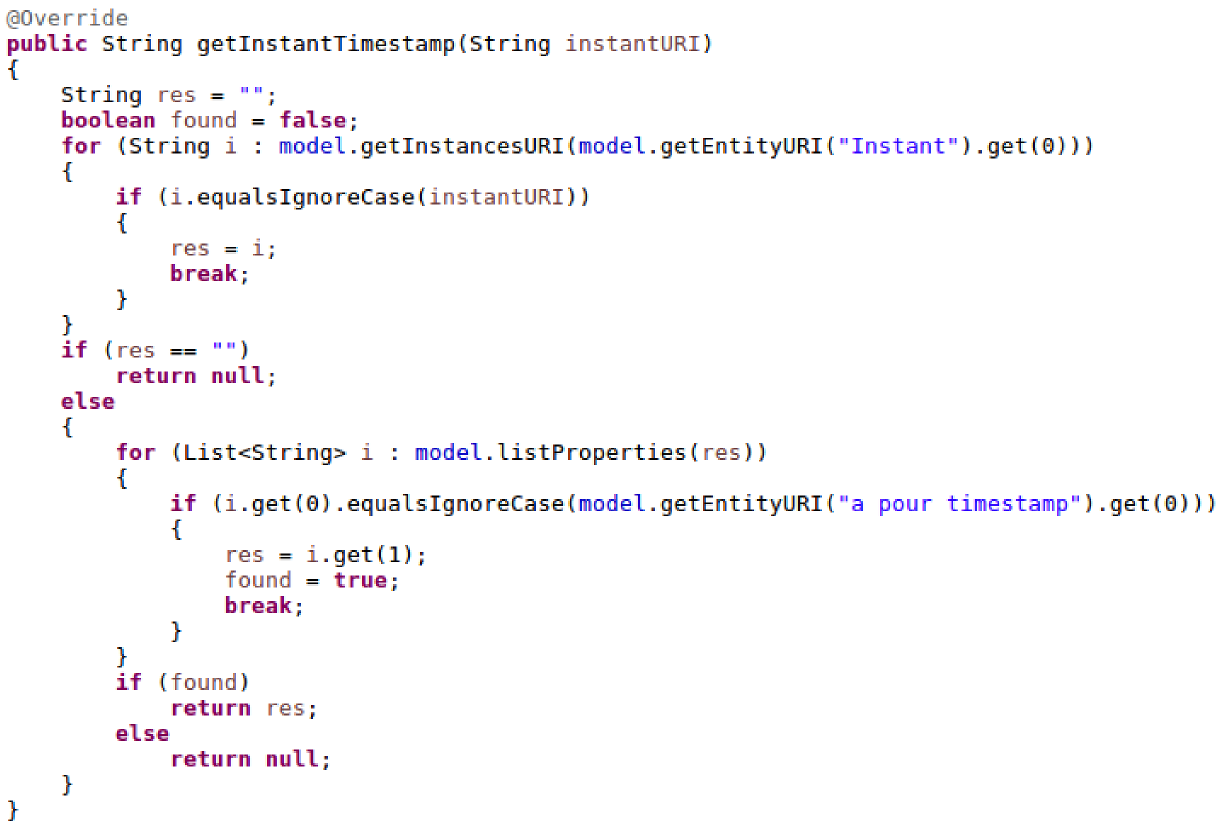
```java
@Override
public String createInstant(TimestampEntity instant) {
    String res = model.createInstance(instant.getTimeStamp(), model.getEntityURI("Instant").get(0));
    model.addDataPropertyToIndividual(res, model.getEntityURI("a pour timestamp").get(0), instant.getTimeStamp());
    return res;
}
```

Figure 5: Create an instant.



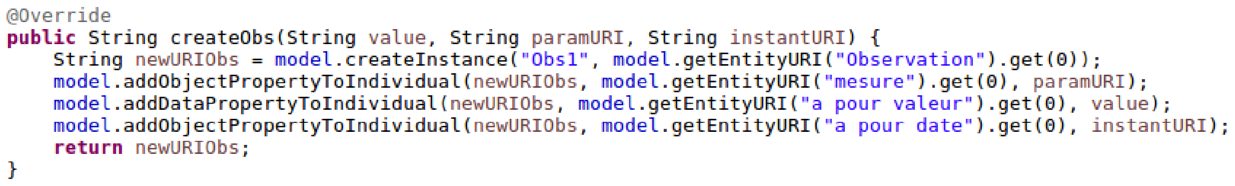
```java
@Override
public String getInstantURI(TimestampEntity instant) {
    String res = null;
    for (String i : model.getInstancesURI(model.getEntityURI("Instant").get(0)))
    {
        if (model.hasDataPropertyValue(i, model.getEntityURI("a pour timestamp").get(0), instant.getTimeStamp()))
        {
            res = i;
            break;
        }
    }
    return res;
}
```

Figure 6: Get the URI of a given instant.



```java
@Override
public String getInstantTimestamp(String instantURI)
{
    String res = "";
    boolean found = false;
    for (String i : model.getInstancesURI(model.getEntityURI("Instant").get(0)))
    {
        if (i.equalsIgnoreCase(instantURI))
        {
            res = i;
            break;
        }
    }
    if (res == "")
        return null;
    else
    {
        for (List<String> i : model.listProperties(res))
        {
            if (i.get(0).equalsIgnoreCase(model.getEntityURI("a pour timestamp").get(0)))
            {
                res = i.get(1);
                found = true;
                break;
            }
        }
        if (found)
            return res;
        else
            return null;
    }
}
```

Figure 7: Get the timestamp of a given instant.



```java
@Override
public String createObs(String value, String paramURI, String instantURI) {
    String newURIObs = model.createInstance("Obs1", model.getEntityURI("Observation").get(0));
    model.addObjectPropertyToIndividual(newURIObs, model.getEntityURI("mesure").get(0), paramURI);
    model.addDataPropertyToIndividual(newURIObs, model.getEntityURI("a pour valeur").get(0), value);
    model.addObjectPropertyToIndividual(newURIObs, model.getEntityURI("a pour date").get(0), instantURI);
    return newURIObs;
}
```

Figure 8: Create an observation.