Semantic Web Report

Baclé Lucas - Géhin Clément

November 2020

1 Introduction

Semantic and linked data web are domains becoming increasingly important in the IoT, aimed at easing the exchange of data coming from connected devices. During this course we have practically identified certain issues that IoT faces, and how semantic web responds to them.

2 Ontology design

During this first lab we defined an ontology of the weather, containing a notion of sensor using the Protégé software. Using the Hermit reasoner available with Protégé we observed the deductions which can be made from the represented knowledge.

2.1 Lightweight ontology design

Based on the given sentences we created the following classes to represent data, using semantic inheritance when required. Then, we did the same for the properties, adding characteristics to match the meaning of the sentences as best as possible. Data properties were used to describe relation between instances and values in defined ranges, whereas object properties link classes of concepts.

Figure 1: Classes & object properties & data properties created

2.2 Lightweight ontology settlement

Now that we have created our classes and properties, we need to populate them. We had sentences describing how to settle our ontology. Properties allow the reasoner to deduct data that is not explicitly expressed in those sentences. After adding the different individual entities, we observed the following basic deductions made by the HermiT reasoner:

As seen in figure 2, the reasoner was able to deduce that Toulouse is a place based on the fact that Toulouse is located in France and that this property links two places together.

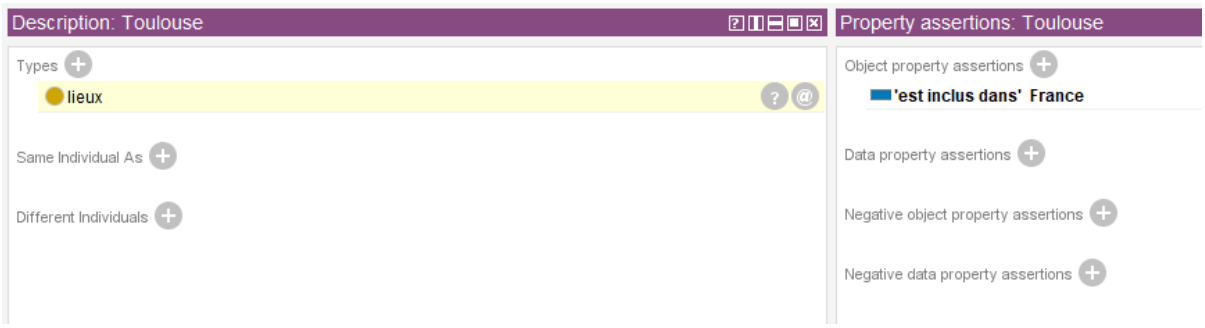

Figure 2: Toulouse is a place

Since France individual has Paris as capital, the reasoner was able to produce the deductions shown by figures 3 and 4.

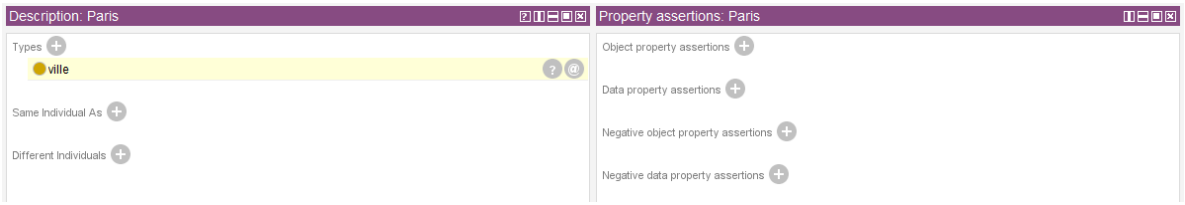

Figure 3: Paris is a city

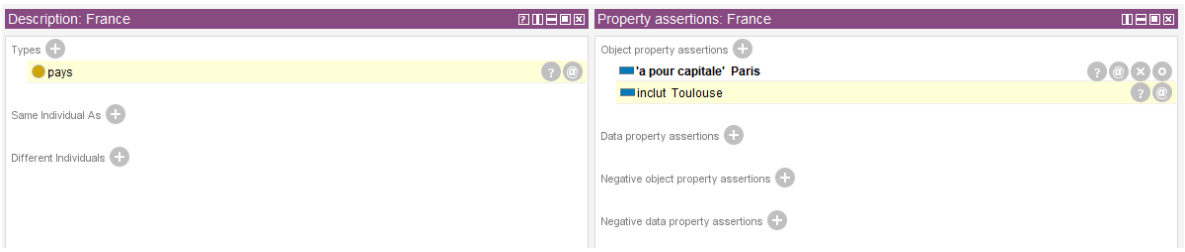

Figure 4: France is a country including Toulouse

From the facts 8. and 9. presented in the lab subject, the reasoner could deduct that A1 is a phenomena.

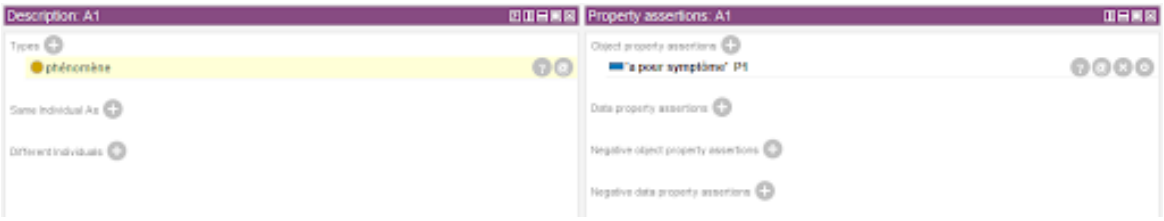

Figure 5: A1 is a phenomena

2.3 Heavy ontology

We then added more relations and meaning into our ontology. Firstly, we checked the characteristics of each properties to be sure their mathematical meaning was correct. For instance "is located in" property is transitive : if Toulouse is located in France and France is located in Europe then Toulouse is in Europe too.

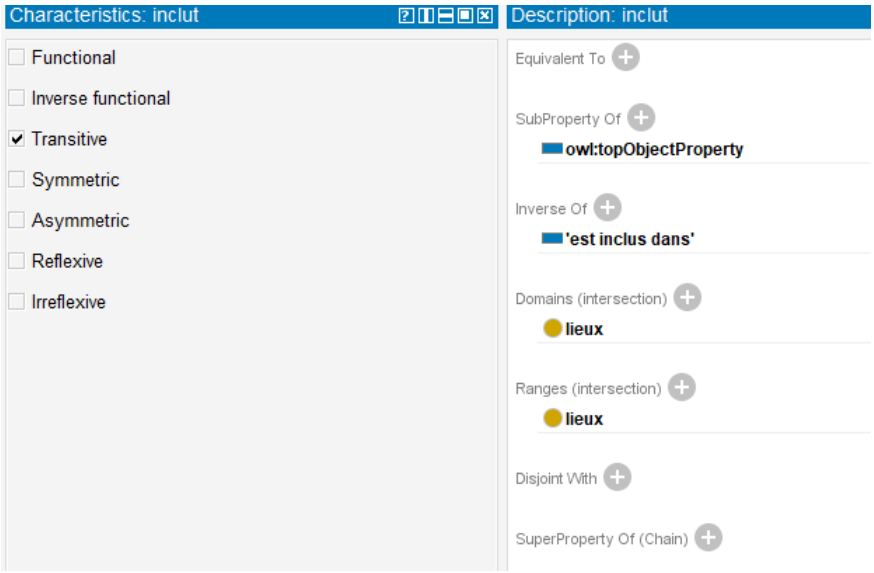

Figure 6: Example, transitive characteristic applies to "is located in"

We also completed few classes with OWL syntax. For instance "Rain" is a phenomenon that has an observation of type pluviometry and a value of type float greater or equal to 0.

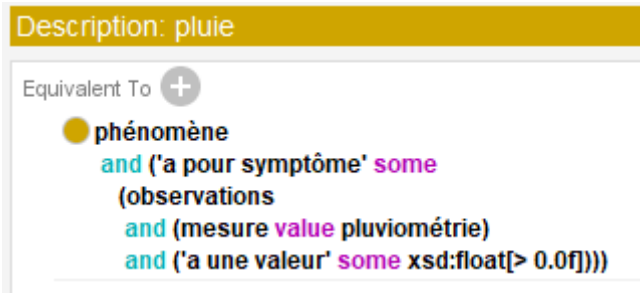

Figure 7: Rain phenomenon equivalent manchester syntax

Finally, we used the notion of sub-property to express the knowledge that if a country has a capital, it also includes this city.

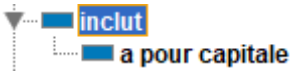

Figure 8: Example of use of the sub-property

2.4 Heavy ontology settlement

After adding the different knowledge presented above, we have observed the following deductions being made.

Because Paris is the french capital, the reasoner was able to deduce that it is included in France. Because La Ville Lumière is also the french capital, and that we have expressed that a capital is unique to a country (functional & inverse functional), the reasoner deduced that Paris is also represented by La Ville Lumière. It is the same for Toulouse if indicated as the French capital.

Because we have stated that countries set must be disjoint with cities one, the reasoner complained that Singapore cannot be both.

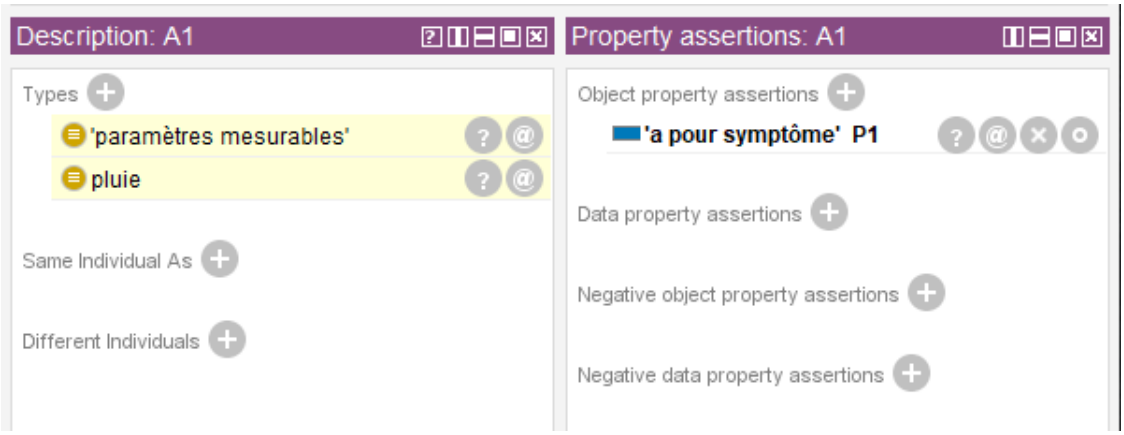

Figure 9: A1 has measurable parameter of type rain

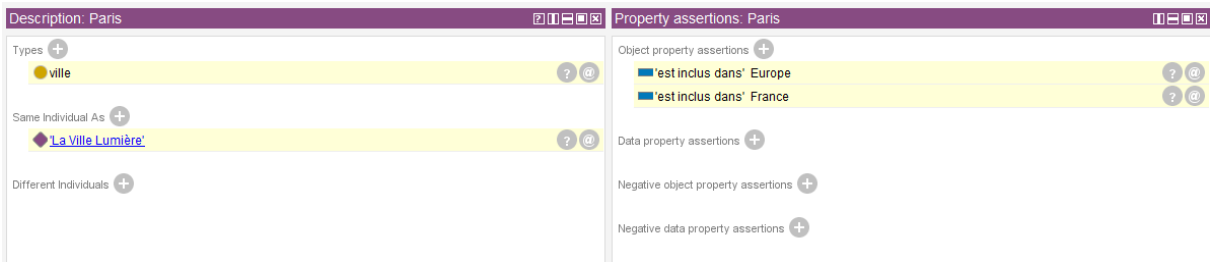

Figure 10: Paris is also "La Ville Lumière" and is included in Europe

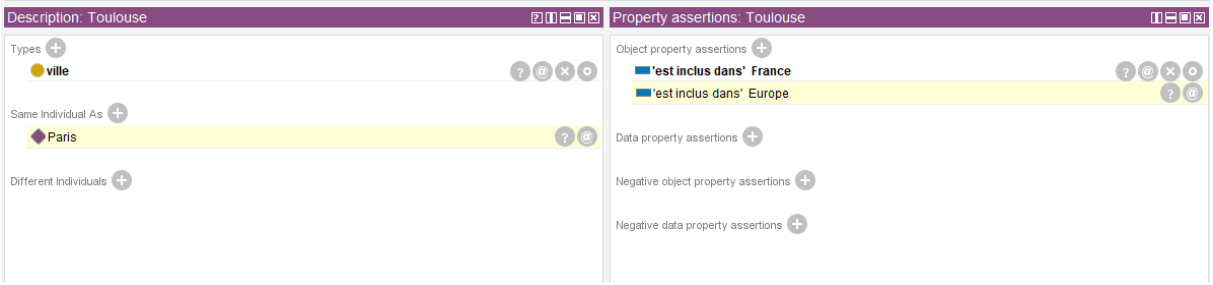

Figure 11: Logically Toulouse is also the same as Paris

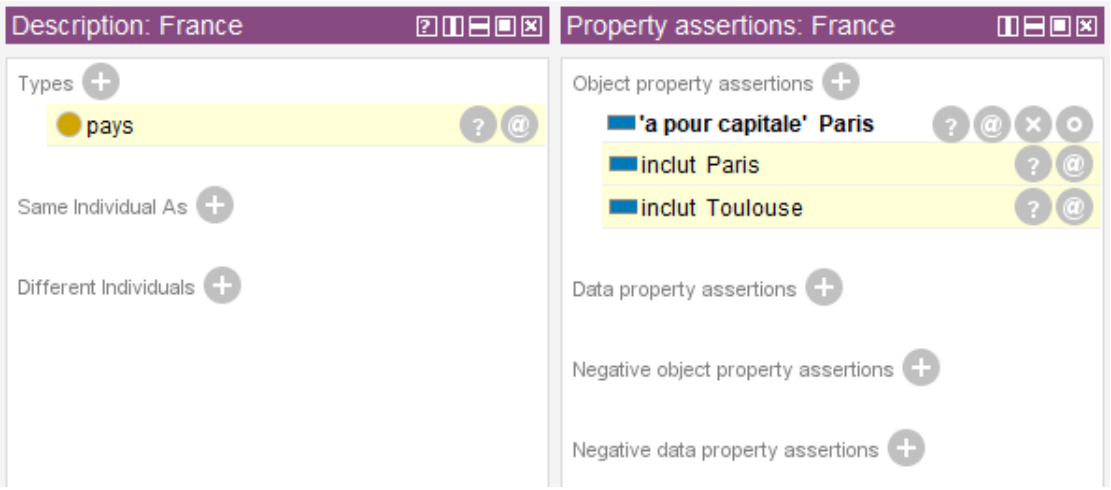

Figure 12: France also includes Paris

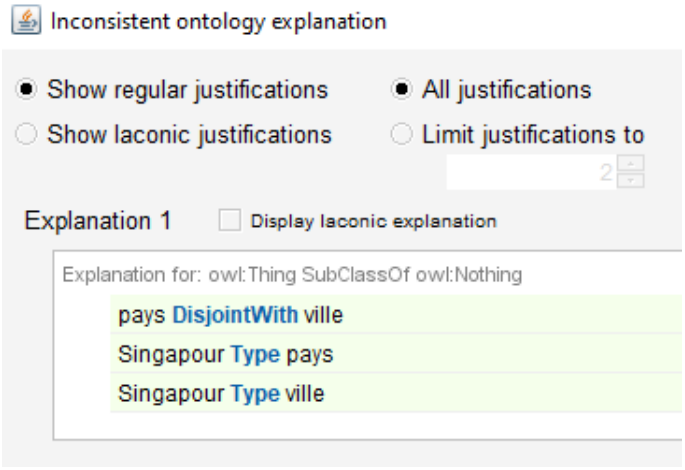

Figure 13: Contradiction, a city cannot be a country in our ontology

3 Data enrichment and use of reasoner in Java

Using a more complete version of our previous ontology we annotated a csv data-set released by the city of Aarhus in Denmark thanks to a Java program. With the RDF knowledge base resulting of this process, we would then be able to determined if we can trust all the measurements.

3.1 Program explanation

First we had to implements the interface IModelFunction to interact more easily with our knowledge-base. To do that we created functions to create or get an instance from our ontology.

The two following functions permit the creation of a new instance in our ontology. As you can see we can also bind data properties to the new instance.

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

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

Figure 14: Function to create instance in our ontology

The two following functions permit to retrieve an instance from the ontology. In those functions we retrieve all the instance and we return the one that match our criteria of comparison.

```
@Override
public String getInstantURI(TimestampEntity instant) {
    int i = 0;
    List<String> subjects = this.model.getInstancesURI(this.model.getEntityURI("Instant").get(0));
    while (i < subjects.size()) {
        if (this.model.hasDataPropertyValue(subjects.get(i), model.getEntityURI("a pour timestamp").get(0),
                instant.timestamp)) {
            return subjects.get(i);
        }
        i++;
    }
    return null;
}

@Override
public String getInstantTimestamp(String instantURI) {
    int i = 0;
    List<List<String>> properties = this.model.listProperties(instantURI);
    String propertyUri = model.getEntityURI("a pour timestamp").get(0);

    while (i < properties.size()) {
        if (propertyUri.equals(properties.get(i).get(0))) {
            return properties.get(i).get(1);
        }
        i++;
    }

    return null;
}
```

Figure 15: Function to get an instance in our ontology

The final function in our model permits to create an observation in our ontology. This function create an instance observation and add all the needed properties.

```java
@Override
public String createObs(String value, String paramURI, String instantURI) {
    String obsURI = this.model.createInstance("obs_"+instantURI, this.model.getEntityURI("Observation").get(0));

    String sensorURI = this.model.whichSensorDidIt(this.getInstantTimestamp(instantURI), paramURI);
    System.out.print("SENSOR: " + sensorURI);
    this.model.addObservationToSensor(obsURI, sensorURI);

    this.model.addDataPropertyToIndividual(obsURI, model.getEntityURI("a pour valeur").get(0), value);
    this.model.addObjectPropertyToIndividual(obsURI, model.getEntityURI("a pour date").get(0), instantURI);

    return obsURI;
}
```

Figure 16: Function to create an observation in our ontology

Now that we have our model, we can create our controller. In this controller we create observations in our ontology from a list of observation given in parameter. We use a map with a timestamp as key to avoid duplicates.

```java
@Override
public void instantiateObservations(List<ObservationEntity> obsList, String paramURI) {

    // The key of this map is the timestamp of instants individuals, to avoid
    // creating
    // multiple individuals for the same instant
    Map<String, String> instantsMap = new HashMap<String, String>();
    // For each element of the list, create its representation in the KB
    int i = 0;
    for (ObservationEntity oe : obsList) {
        i++;
        System.out.println("Instantiating observation " + i);
        String instantURI;
        if (!instantsMap.containsKey(oe.getTimestamp().getTimeStamp())) {
            instantURI = this.customModel.createInstant(oe.getTimestamp());
            instantsMap.put(oe.getTimestamp().getTimeStamp(), instantURI);
        } else {
            instantURI = instantsMap.get(oe.getTimestamp().getTimeStamp());
        }

        System.out.println("INSTANT: " + instantURI);
        System.out.println("PARAM: " + paramURI);
        System.out.println("VALUE: " + oe.getValue().toString());

        this.customModel.createObs(oe.getValue().toString(), paramURI, instantURI);
    }
```

Figure 17: Function of our controller

Finally the main program is responsible for the reading of the csv file and using our controller to populate the knowledge base.

4 Conclusion

In this lab we have been able to understand how an ontology is design using classes and object/data properties. We used the reasoner HermiT and saw how it was able to deduct properties and information on individual of the ontology. Finally, we used this knowledge in a IoT approach using data model and controller to interact with an ontology and use its information in an web application for example.