

Semantic Web of Things

Yosra ZEYRI NEMRI

INSA TOULOUSE

MSIoT B1

1. INTRODUCTION

The semantic web enables us to structure data that are on the web to better exploit them. It provides a set of functions that offers a common structure to all data on the web. It enables an automatic treatment (enables computers, software systems, and people to work together in a network). The semantic web aims at converting the current web, with unstructured and semi-structured documents into a "web of data".

All the languages of this architecture are standardized by the World Wide Web consortium. In these lab sessions, we focused on three technologies:

- **OWL** - Ontology Web Language: logic-based language, designed to be interpreted by machines so that they can automatically process and reason through the information it describes
- **RDF** - stands for Resource Description Framework and is a method for publishing and linking data
- **SPARQL** - stands for Protocol and RDF Query Language and is the way data stored in RDF format gets queried, retrieved and manipulated on the Semantic Web

FIRST LAB SESSION

(Conception of an ontology and use of the reasoner in protégé)

Objective

We use the ontology editor “Protégé” to describe an ontology of meteorological phenomenons; this ontology will have to represent the knowledge related to meteorological phenomena (rain, storm, ...), the measurable parameters that characterize them (temperature, humidity, ...), but also the sensors that allow to make these observations. We then use a reasoner (HermiT) to validate and complete assertions. We are asked for light and heavy ontologies (*Ontologie légère/Ontologie lourde*). Thanks to the concepts available in protégé (classes, individuals, object properties, data properties), we created relation between the different “objects”.

2. Creation of ontology

2.1. Light ontology

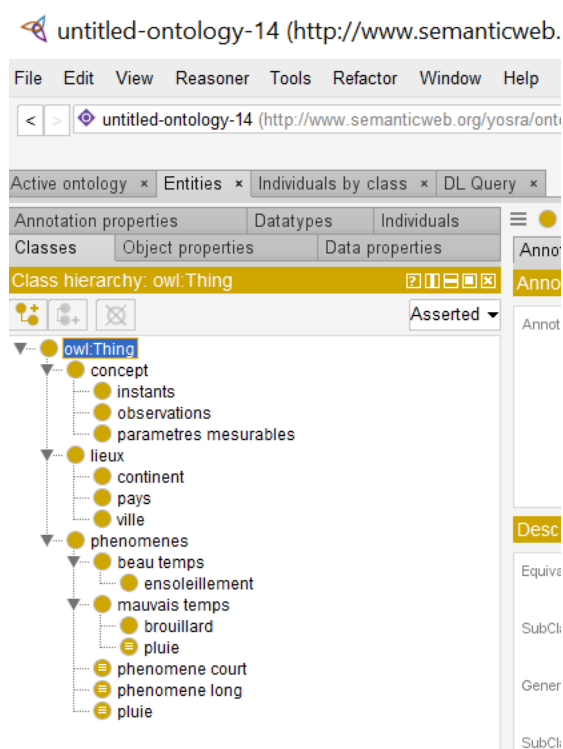
2.1.1. Conception

Creation of classes and subclasses:

First, we express knowledge by considering the following information:

1. Le beau temps et le mauvais temps sont deux types de phénomènes.
2. La pluie et le brouillard sont des types de phénomènes de mauvais temps, l'ensoleillement est un type de phénomène de beau temps
3. Les paramètres mesurables sont une classe de concept, ainsi que les instants et les observations
4. Une ville, un pays et un continent sont des types de lieux

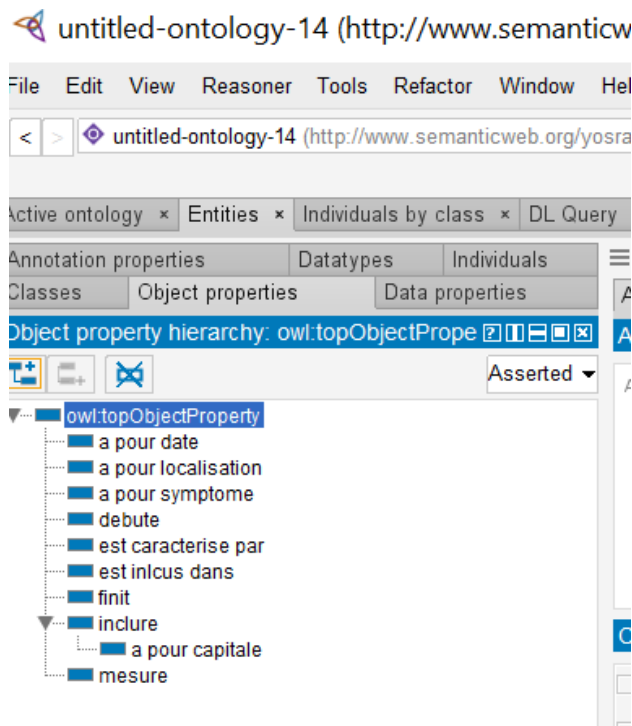
We then create classes and subclasses as shown in the figure below:



Creation of properties:

1. Un phénomène est caractérisé par des paramètres mesurables
2. Un phénomène a une durée en minutes
3. Un phénomène débute à un instant
4. Un phénomène finit à un instant
5. Un instant a un timestamp, de type xsd : dateTimeStamp
6. Un phénomène a pour symptôme une observation
7. Une observation météo mesure un paramètre mesurable
8. Une observation météo a une valeur pour laquelle vous ne représentez pas l'unité
9. Une observation météo pour localisation un lieu.
10. Une observation météo a pour date un instant
11. Un lieu peut être inclus dans un autre lieu
12. Un lieu peut inclure un autre lieu
13. Un pays a pour capitale une ville

Object properties:



To link objects with a relation we use the Domain and Range fields of the properties. For example, to express the relation « Un phénomène a pour symptôme une observation » :

untitled-ontology-14 (<http://www.semanticweb.org/yosra/ontologies/2022/0/untitled-c>)

The screenshot shows the Protégé interface for 'untitled-ontology-14'. The 'Object properties' tab is active, displaying the 'Object property hierarchy: a pour symptome'. The hierarchy includes 'owl:topObjectProperty' and its subclasses: 'a pour date', 'a pour localisation', 'a pour symptome' (highlighted), 'debute', 'est caractérisé par', 'est inclus dans', 'finit', 'inclure', 'a pour capitale', and 'mesure'. The right panel shows the 'Annotations: a pour symptome' section with the annotation 'rdfs:label [language: en] a pour symptome'. Below this, the 'Description: a pour symptome' section lists various property characteristics (Functional, Inverse functional, Transitive, Symmetric, Asymmetric, Reflexive, Irreflexive) and domain/range restrictions. The domain is set to 'phenomenes' and the range to 'observations'.

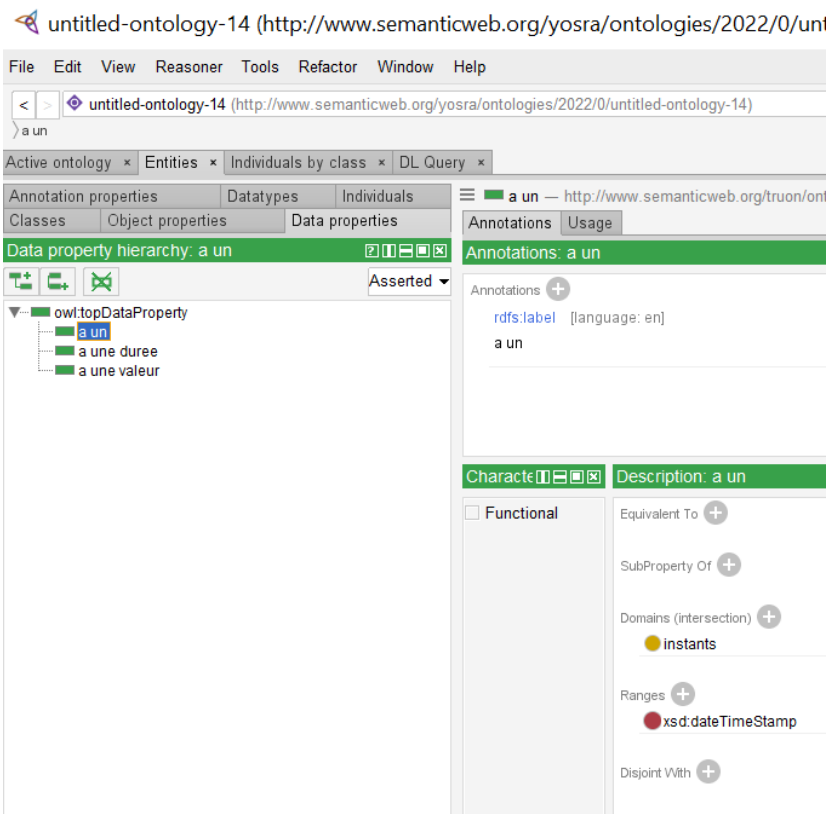
Data properties:

untitled-ontology-14 (<http://www.semanticweb.org/yosra/ontologies/2022/0/untitled-c>)

The screenshot shows the Protégé interface for 'untitled-ontology-14'. The 'Data properties' tab is active, displaying the 'Data property hierarchy: owl:topDataProperty'. The hierarchy includes 'owl:topDataProperty' and its subclasses: 'a un' (highlighted), 'a une durée', and 'a une valeur'. The right panel shows the 'Annotations: a un' section with the annotation 'rdfs:label [language: en] a un'. Below this, the 'Description: a un' section lists various property characteristics (Functional, Inverse functional, Transitive, Symmetric, Asymmetric, Reflexive, Irreflexive) and domain/range restrictions. The domain is set to 'phenomenes' and the range to 'observations'.

PS: The difference between object and data properties comes from the fact that an object property links two classes, while a data property links a class with a raw data to be usually typed.

For instance, the data property “a un” links “instants” to a timestamp (in our case of type `dateTimeStamp`), but a timestamp is not a class.

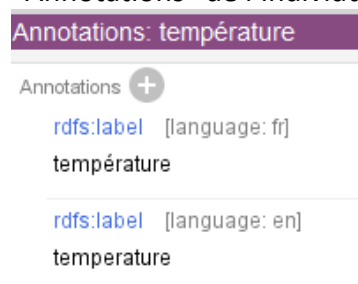


2.1.2. Settlement (fr. Peuplement)

1. La température, l'hygrométrie, la pluviométrie, la pression atmosphérique, la vitesse du vent et la force du vent sont des paramètres mesurables (Attention, pas des types de paramètres, mais des instances de paramètres)



2. Le terme temperature est un synonyme anglais de température. Examinez les "Annotations" de l'individu.



3. La force du vent est similaire à la vitesse du vent

Description: force du vent

Types +

- parametres mesurables

Same Individual As +

- vitesse du vent

4. Toulouse est située en France. Remarquez que les individus dans cette phrase ne sont pas typés : créez Toulouse et France non pas comme une ville et un pays, mais comme des individus sans classe. Comment les classifie le raisonneur ?

⇒ **Toulouse and France instance are deducted as 'Lieux' by the reasoner HermiT (because the relation "est inclus dans" has "lieux" as type object)**

5. Toulouse est une ville

Description: Toulouse

Types +

- ville

6. La France a pour capitale Paris. Ici aussi, Paris est un individu non typé

⇒ **"Paris" is deducted as "ville" and "France" is "pays" and that "Paris" is the capital of "France"**

7. Le 10/11/2015 à 10h00 est un instant que l'on appellera I1 (noté 2015-11-10T10:00:00Z)

Description: I1

Types +

- instants

Same Individual As +

Different Individuals +

Property assertions: I1

Object property assertions +

Data property assertions +

- 'a un' "2015-11-10T10:00:00Z"^^xsd:dateTime

8. P1 est une observation qui a mesure la valeur 3 mm de pluviométrie à Toulouse à l'instant I1 (pas besoin de représenter l'unité)

Description: P1

Types +

- observations

Same Individual As +

Different Individuals +

Property assertions: P1

Object property assertions +

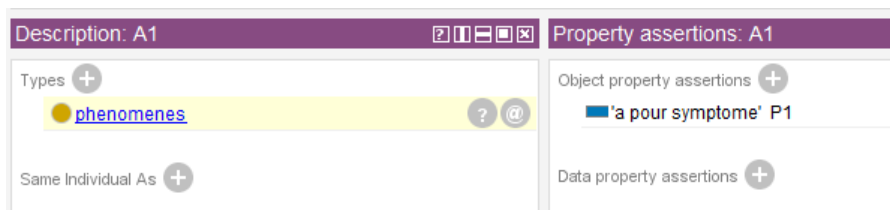
- 'a pour date' I1
- mesure pluviometrie
- 'a pour localisation' Toulouse
- 'a pour localisation' Paris
- 'a pour localisation' 'La Ville Lumiere'

Data property assertions +

- 'a une valeur' 3

9. A1 a pour symptôme P1

⇒ **A1 is deducted as « phenomenes »**



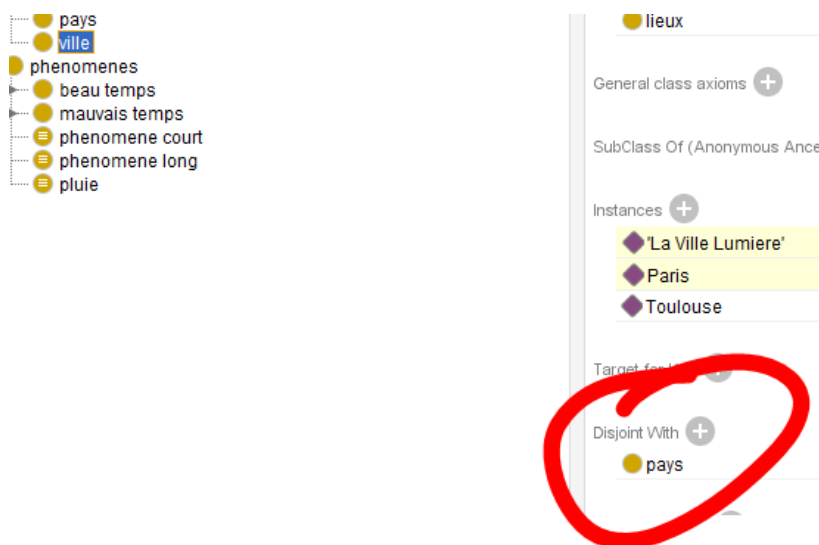
2.2. Heavy ontology

2.2.1. Conception

Based on the light ontology we have just created; we add properties in this second part (logical axioms). These logical axioms will aim at enriching our data by deducing information. To do so, we add the following properties:

1. Toute instance de ville ne peut pas être un pays

⇒ **We created a constraint "Disjoint with" in "Ville" for "Pays". Automatically the inverse constraint appears in "Pays"**



2. Un phénomène court est un phénomène dont la durée est de moins de 15 minutes — En syntaxe de Manchester : Phénomène that « a une durée » some xsd:float [< 15]

⇒ **We created the subclass “phenomene court” and complete their "Equivalent To" with the expression bellow**



3. Un phénomène long est un phénomène dont la durée est au moins de 15 minutes

⇒ **We created the subclass “phenomene long” of class “phenomenes” and complete their "Equivalent To" with the expression bellow**

Description: phenomene long

Equivalent To +

phenomenes and ('a une duree' some xsd:integer[> 15])

SubClass Of +

phenomenes

4. Un phénomène long ne peut pas être un phénomène court

⇒ We reuse the constraint "Disjoint with". Automatically the inverse constraint appears in "phenomene court"

- phenomene court
- phenomene long
- pluie

General class axioms +

SubClass Of (Anonymous Ancestor)

Instances +

Disjoint With +

phenomene court

5. La propriété indiquant qu'un lieu est inclus dans un autre a pour propriété inverse la propriété indiquant qu'un lieu en inclue un autre.

⇒ We use the "Inverse Of" Object Property Description

Description: est inclus dans

Equivalent To +

SubProperty Of +

Inverse Of +

include

6. Si un lieu A est situé dans un lieu B et que ce lieu B est situé dans un lieu C, alors le lieu A est situé dans le lieu C (utilisez les caractéristiques de la relation)

⇒ We just activated the transitivity of the relation "est inclus dans"

Characteristics

☐ Functional

☐ Inverse functional

☒ Transitive

☐ Symmetric

☐ Asymmetric

Description: est inclus dans

Equivalent To +

SubProperty Of +

Inverse Of +

include

7. A tout pays correspond une et une seule capitale (utilisez les caractéristiques de la relation).

protegeproject.github.io/protege/views/object-property-characteristics/

Available characteristics

- **Functional** - asserts that the selected property is Functional. Intuitively, this means that for any given individual, the property can have at most one value. In other words, there can be at most one outgoing relationship along the property for that individual. Note that, if multiple individuals are specified as values for the property then these values will be inferred to denote the same object.
- **Inverse Functional** - asserts that the selected property is InverseFunctional. Intuitively, this means the inverse property of the selected property (whether it explicitly declared or not) is Functional. In other words, there can be at most one incoming relationship along the property for that individual. Note that, if multiple individuals are specified as incoming values for the property then these values will be inferred to denote the same object.

☐ Asymmetric
☐ Reflexive
☐ Irreflexive

Source : <http://protegeproject.github.io/protege/views/object-property-characteristics/>

⇒ This is the bijectivity. So, by referring to the explanation above, we activate **Functional** and **Inverse Functional** for the Object Property “a pour capitale”.

Characteristics: ☒ Functional ☒ Inverse functional ☐ Transitive ☐ Symmetric ☐ Asymmetric

Description: a pour capitale

Equivalent To +

SubProperty Of +

include

inverse ('est inclus dans')

8. Si un pays a pour capitale une ville, alors ce pays contient cette ville (utilisez la notion de sous-propriété).

⇒ We set the Object Property “a pour capitale” as a SubProperty of “include”.

Description: a pour capitale

Equivalent To +

SubProperty Of +

include

inverse ('est inclus dans')

9. La Pluie est un Phénomène ayant pour symptôme une Observation de Pluviométrie dont la valeur est supérieure à 0. — Phénomène that « a pour symptôme » some (Observation that ('mesure' value Pluviométrie) and ('a pour valeur' some xsd:float [> 0]))

⇒ We just add the specifications above in the description of the subclass “pluie”

Description: pluie

Equivalent To +

phenomenes and ('a pour symptome' some (observations and (mesure value pluviometrie) and ('a une valeur' some xsd:float [> 0.0f])))

2.2.2. Settlement (fr. Peuplement)

1. La France est située en Europe

- ⇒ **Addition of the individual “Europe” and the property “est inclus dans” between France and Europe**

The screenshot shows a knowledge graph editor interface. On the left, a list of entities is visible, including 'Europe', 'France', 'Paris', 'La Ville Lumiere', and 'Toulouse'. The main area displays the 'Property assertions: France' panel, which lists several assertions, including 'est inclus dans' Europe (highlighted in yellow). Below this, the 'Description: Europe' panel shows the type 'lieux' (highlighted in yellow). To the right, the 'Property assertions: Europe' panel lists several assertions, including 'inclure France' (highlighted in yellow). The interface also includes a 'Data property assertions' panel at the bottom right.

2. Paris est la capitale de la France

- ⇒ **Adding the property “a pour capitale” between Paris and France**

The screenshot shows the 'Property assertions: France' panel in the knowledge graph editor. It lists several assertions, including 'a pour capitale' Paris (highlighted in yellow). The panel also includes a search bar and a list of entities to choose from.

3. La Ville Lumière est la capitale de la France

- ⇒ **Add the individual “La Ville Lumiere” and the property “a pour capitale” between “La Ville Lumiere” and France**

Individuals: France

- A1
- Europe
- force du vent
- France
- hygrometrie
- La Ville Lumiere
- P1
- Paris
- pluviometrie
- pression atmospherique
- Singapour
- Singapour
- temperature
- Toulouse
- vitesse du vent

Annotations: France

rdfs:label [language: en]
France

Description: France

Types
pays

Property assertions: France

Object property assertions

- 'a pour capitale' 'La Ville Lumiere'
- 'est inclus dans' Europe
- 'a pour capitale' Paris
- 'a pour capitale' Toulouse
- 'inclure' Paris
- 'inclure' 'La Ville Lumiere'
- 'inclure' Toulouse

4. Singapour est une ville et un pays

- ⇒ **Singapour cannot be declared as Pays and Ville at the same time because we declared the relation “Toute instance de ville ne peut pas être un pays”. The reasoner will reject this; So, the solution is to create two different individuals called Singapour**

Inconsistent ontology explanation

☒ Show regular justifications ☒ All justifications
☐ Show laconic justifications ☐ Limit justifications to

Explanation 1 ☐ Display laconic explanation

Explanation for: owl:Thing SubClassOf owl:Nothing

- pays **DisjointWith** ville
- Singapour **Type** pays
- Singapour **Type** ville

Active ontology * Entities * Individuals by class * DL Query *

Annotation properties Datatypes Individuals

Object properties Data properties

Individuals: Singapour

- A1
- Europe
- force du vent
- France
- hygrometrie
- I1
- La Ville Lumiere
- P1
- Paris
- pluviometrie
- pression atmospherique
- Singapour
- Singapour
- temperature
- Toulouse
- vitesse du vent

Annotations: Singapour

rdfs:label [language: en]
Singapour

Description: Singapour

Types
pays

Same Individual As +

Different Individuals +

Observation and Justification :

1. Que sait le raisonneur de A1 ?

⇒ **A1 is a phenomenon of type « pluie » (because « Pluviométrie » not null [> 0])**

2. Que sait le raisonneur sur Paris ?

⇒ **Paris is in Europe via transitivity: the reasoner knows that Paris is the capital of France and that France is in Europe. Moreover, since a “pays” has a unique capital, the reasoner understands that Paris is “la ville lumière”.**

3. Constatez la réaction du raisonneur si Toulouse est déclarée comme la capitale de la France

⇒ **If we declare "Toulouse" as the capital of France, "Toulouse" is noted as the same individual as "Paris" and "La ville Lumiere" by the reasoner.**

Second LAB SESSION (semantic-aware application)

3. Ontology exploitation

- Clone the following git repository: `git clone https://framagit.org/nseydoux/iss-semantic-lab`

```
C:\Windows\System32\cmd.exe
Microsoft Windows [version 10.0.19043.1415]
(c) Microsoft Corporation. Tous droits réservés.

C:\Users\choua\Desktop>git clone https://framagit.org/nseydoux/iss-semantic-lab
Cloning into 'iss-semantic-lab'...
warning: redirecting to https://framagit.org/nseydoux/iss-semantic-lab.git/
remote: Enumerating objects: 99, done.
remote: Counting objects: 100% (99/99), done.
remote: Compressing objects: 100% (63/63), done.
remote: Total 99 (delta 29), reused 81 (delta 23), pack-reused 0
Receiving objects: 56% (56/99), 604.00 KiB | 1.05 MiB/s
Receiving objects: 100% (99/99), 1.46 MiB | 1.60 MiB/s, done.
Resolving deltas: 100% (29/29), done.

C:\Users\choua\Desktop>
```

> mvn compile

```
C:\Windows\System32\cmd.exe
remote: Total 99 (delta 29), reused 81 (delta 23), pack-reused 0
Receiving objects: 56% (56/99), 604.00 KiB | 1.05 MiB/s
Receiving objects: 100% (99/99), 1.46 MiB | 1.60 MiB/s, done.
Resolving deltas: 100% (29/29), done.

C:\Users\choua\Desktop>cd iss-semantic-lab
C:\Users\choua\Desktop\iss-semantic-lab>mvn compile
[INFO] Scanning for projects...
[INFO]
[INFO] -----< fr.insa.iss:semantic >-----
[INFO] Building semantic 0.0.1-SNAPSHOT
[INFO] -----[ jar ]-----
[INFO]
[INFO] --- maven-resources-plugin:2.6:resources (default-resources) @ semantic ---
[WARNING] Using platform encoding (Cp1252 actually) to copy filtered resources, i.e. build is platform dependent!
[INFO] skip non existing resourceDirectory C:\Users\choua\Desktop\iss-semantic-lab\src\main\resources
[INFO]
[INFO] --- maven-compiler-plugin:3.3:compile (default-compile) @ semantic ---
[INFO] Changes detected - recompiling the module!
[WARNING] File encoding has not been set, using platform encoding Cp1252, i.e. build is platform dependent!
[INFO] Compiling 11 source files to C:\Users\choua\Desktop\iss-semantic-lab\target\classes
[INFO]
[INFO] BUILD SUCCESS
[INFO]
[INFO] Total time: 3.366 s
[INFO] Finished at: 2022-01-11T09:29:53+01:00
[INFO]
C:\Users\choua\Desktop\iss-semantic-lab>
```

3.1. Implementing of IModelFunctions

These are the java methods implemented:

3.1.1 createPlace method

We use the [getEntityURI](#) method to retrieve the URI of the Place class. We use the [createInstance](#) method to create the instance from the previously retrieved URI.

```
@Override
public String createPlace(String name) {
    // TODO Auto-generated method stub

    return model.createInstance(name, model.getEntityURI("Place").get(0));
}
```

3.1.2. createInstant method

For the instantiation of an Instant, we put the timestamp as a label of the object, but we also attach to the object the timestamp with a Data Property "a pour timestamp".

To check the uniqueness of the Instant we have the choice to use the label or the property relation, we opt for the label.

To instantiate the object, we reuse the methods [getEntityURI](#) and [createInstance](#) but also the methods [addDataPropertyToIndividual](#) and [getInstancesURI](#) (to retrieve the instances of Instant).

```
@Override
public String createInstant(InstantEntity instant) {
    // TODO Auto-generated method stub

    boolean found=false;
    List<String> liste=model.getInstancesURI(model.getEntityURI("Instant").get(0));
    for(Iterator<String> it=liste.iterator(); it.hasNext();)
    {
        if(model.listLabels(it.next()).get(0).equals(instant.getTimeStamp()))
        {
            found=true;
        }
    }

    if (!found)
    {
        String Uri = model.createInstance(instant.getTimeStamp(), model.getEntityURI("Instant").get(0));
        model.addDataPropertyToIndividual(Uri, model.getEntityURI("a pour timestamp").get(0), instant.getTimeStamp());
        return Uri;
    }

    return null;
}
```

3.1.3. getInstantURI method

To retrieve the URI linked to a timestamp, we browse the Instances of the Instant object and check if they match the timestamp we are looking for. To do this, we use [hasDataPropertyValue](#) method.

```

@Override
public String getInstantURI(TimestampEntity instant) {
    // TODO Auto-generated method stub

    boolean found=false;

    List<String> liste=model.getInstanceURI(model.getEntityURI("Instant").get(0));
    for(Iterator<String> iter=liste.iterator(); iter.hasNext();)
    {
        String list=iter.next();
        if(model.hasDataPropertyValue(list, model.getEntityURI("a pour timestamp").get(0),instant.getTimeStamp()))
        {
            found=true;
            return list;
        }
    }
    return null;
}

```

3.1.4. getInstantTimestamp method

This method looks for the Timestamp of the instant whose address is set as a parameter. It has as output the value of the Timestamp. We first check if the URI corresponds to an Instance `getInstanceURI`, and if it is there, let's retrieve its properties `listProperties` and return the data linked to the property "a pour timestamp".

```

@Override
public String getInstantTimestamp(String instantURI)
{
    // TODO Auto-generated method stub
    List<String> listeInstances=model.getInstanceURI(model.getEntityURI("Instant").get(0));
    if (!listeInstances.contains(instantURI))
    {
        return null;
    }
    else{
        List<List<String>> liste=model.listProperties(instantURI);
        for(Iterator<List<String>> iter=liste.iterator(); iter.hasNext();)
        {
            List <String>list=iter.next();
            if (list.get(0).equals(model.getEntityURI("a pour timestamp").get(0)))
            {
                return list.get(1);
            }
        }
    }
    return null;
}

```

3.1.5. createObs method

```

@Override
public String createObs(String value, String paramURI, String instantURI) {
    // TODO Auto-generated method stub
    String instance = model.createInstance(paramURI.concat(instantURI),model.getEntityURI("Observation").get(0));

    model.addObjectPropertyToIndividual(instance, model.getEntityURI("a pour date").get(0), instantURI);

    model.addDataPropertyToIndividual(instance, model.getEntityURI("a pour valeur").get(0), value);

    model.addObservationToSensor(instance,model.whichSensorDidIt(getInstantTimestamp(instantURI), paramURI));

    return instance;
}

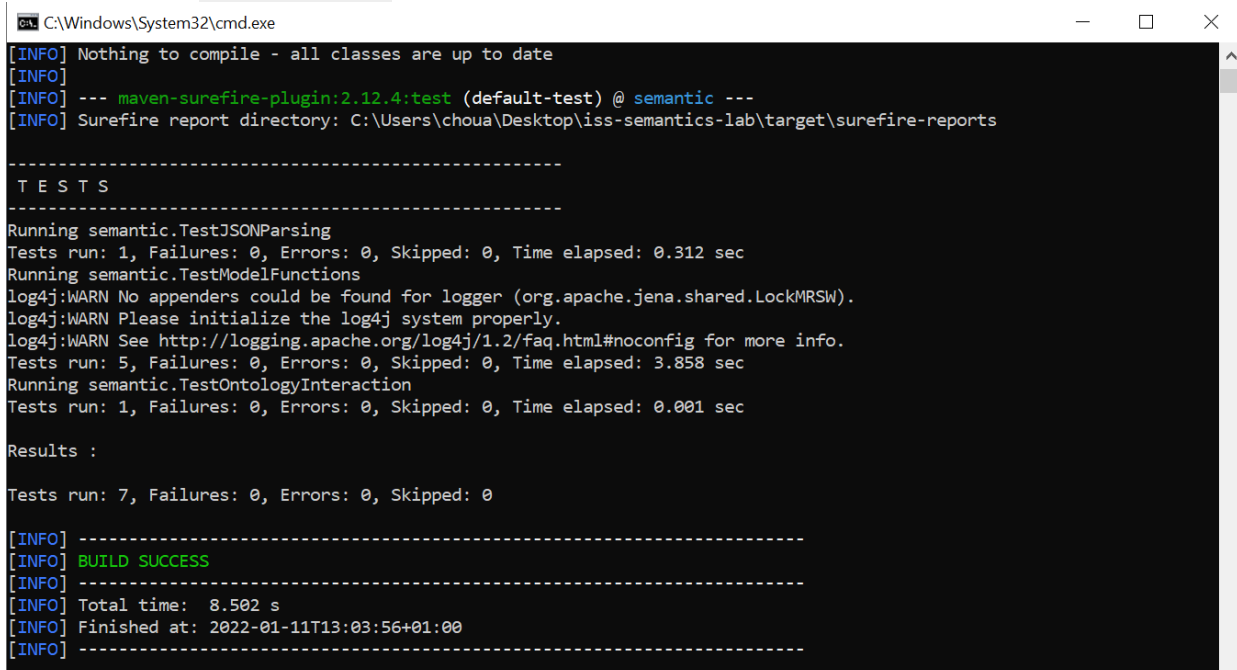
```

3.2. Implementing of IControlFunctions

```
@Override
public void instantiateObservations(List<ObservationEntity> obsList, String paramURI) {
    // TODO Auto-generated method stub

    Map<String, String> map = new HashMap<String, String>();
    for (ObservationEntity oe : obsList) {
        String instURI;
        if (!map.containsKey(oe.getTimestamp().getTimestamp())) {
            instURI = this.cusotmModel.createInstant(oe.getTimestamp());
            map.put(oe.getTimestamp().getTimestamp(), instURI);
        } else {
            instURI = map.get(oe.getTimestamp().getTimestamp());
        }
        this.cusotmModel.createObs(oe.getValue().toString(), paramURI, instURI);
    }
}
```

> mvn test



C:\Windows\System32\cmd.exe

```
[INFO] Nothing to compile - all classes are up to date
[INFO]
[INFO] --- maven-surefire-plugin:2.12.4:test (default-test) @ semantic ---
[INFO] Surefire report directory: C:\Users\choua\Desktop\iss-semantic-lab\target\surefire-reports

-----
T E S T S
-----
Running semantic.TestJSONParsing
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.312 sec
Running semantic.TestModelFunctions
log4j:WARN No appenders could be found for logger (org.apache.jena.shared.LockMRSW).
log4j:WARN Please initialize the log4j system properly.
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more info.
Tests run: 5, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 3.858 sec
Running semantic.TestOntologyInteraction
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.001 sec

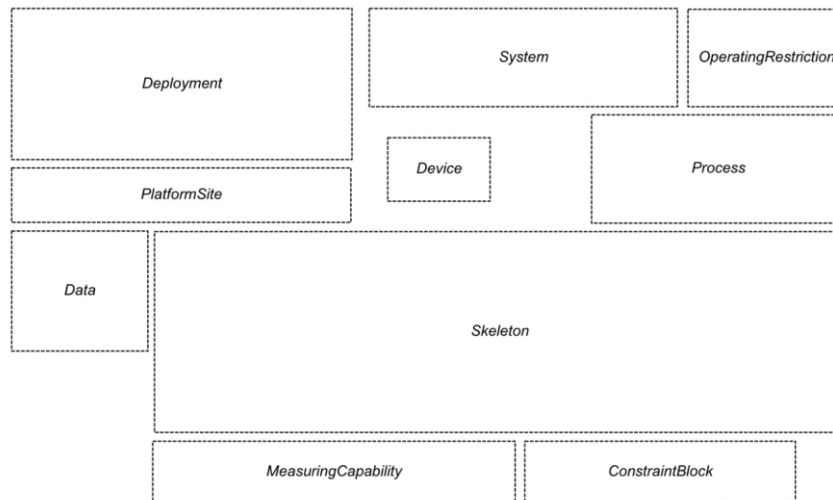
Results :

Tests run: 7, Failures: 0, Errors: 0, Skipped: 0

[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 8.502 s
[INFO] Finished at: 2022-01-11T13:03:56+01:00
[INFO] -----
```

3.3. Semantic Sensor Network Ontology

SSN is a modularly ontology for sensors (SSN ontology describes sensors precisely and modularly). Here is a schema that explains its architecture:



Source : https://www.w3.org/2005/Incubator/ssn/XGR-ssn-20110628/#Ontology_structure

Sensors will have a skeleton, common to every sensor, and we can describe its characteristics thanks to others modules (Process, DATA, PlatformSite, Deployment etc...). For example, if someone wants information about the platform where the sensor is, we simply have to use the PlatformSite module. This module is linked with deployment and system modules. PlatformSite will allow to represent location:

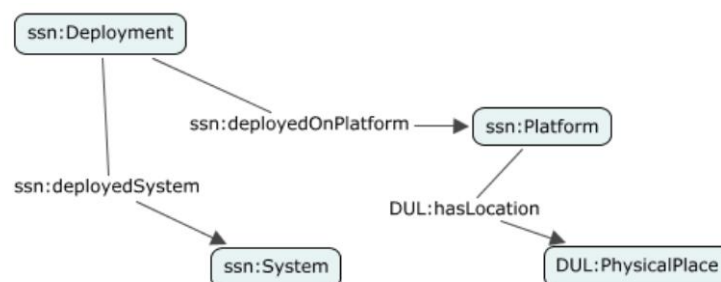


Fig: Using ssn:Platform in conjunction with dul:PhysicalPlace ([PlatformSite module](#))

Source : https://www.w3.org/2005/Incubator/ssn/XGR-ssn-20110628/#Ontology_structure

All these available properties for a sensor will allow to describe precisely what a sensor is, how it works, where it is, ...etc.

3.4. What is the difference between object property and data property?

⇒ **Data property:**

- Links individuals to data (e.g. Number, String, etc ...)
- Allows to add a quantitative relation, by adding a data to a class. This data or "variable" is added as a class attribute finally.

⇒ **Object property:**

- Links individuals to other individuals.
- Allows to create a qualitative relation between two types.