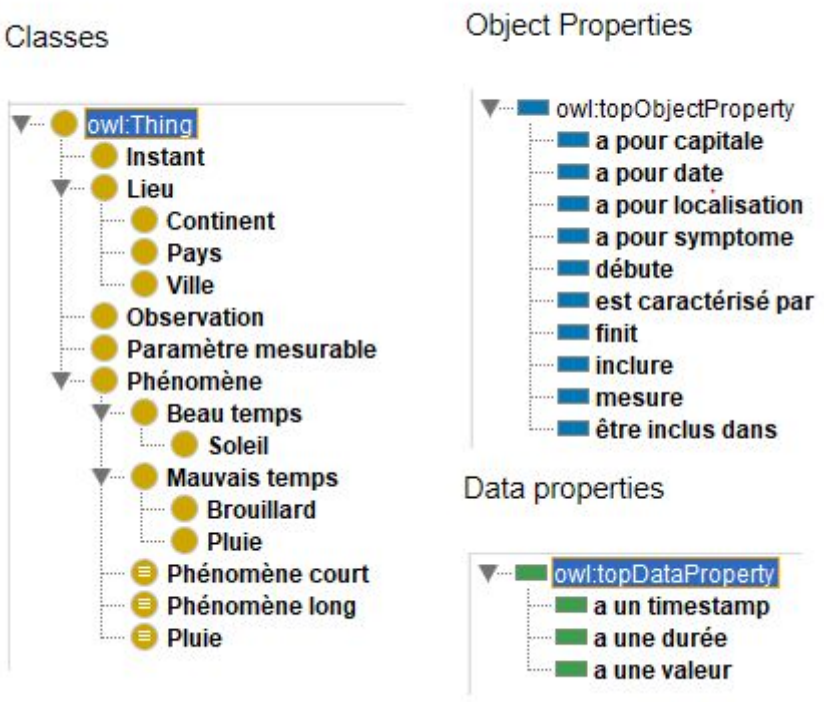
Semantic Web - Protégé - 5 ISS

Desportes Kilian
Imekraz Yanis
B1

TP1



2.2.2

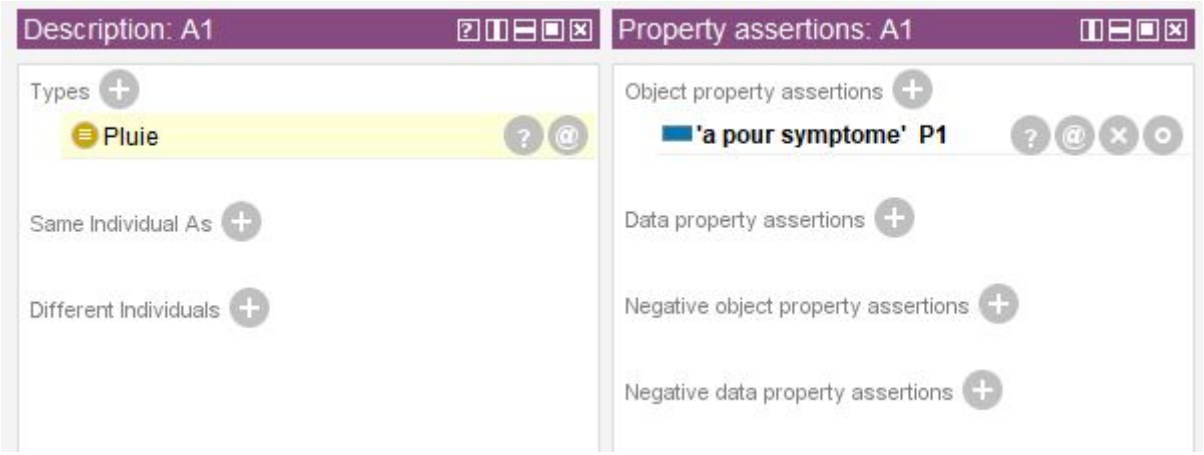
4) After we place Toulouse in France, the relation ‘être inclus dans’ having ‘lieu’ as type object, Toulouse and France instance will be classified as ‘Lieu’.

6) When we said that France has for capital Paris, the reasonner know that the has for capital relation is between a Country and a City. He can link these two and deduce that Paris is a city.

2.2.4

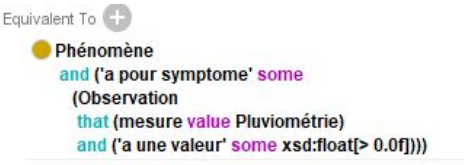
4) Singapour cannot be declared as Pays and Ville at the same time because we declared that a Ville cannot be a Pays (2.2.3.1).
The reasonner will reject this and crash.

1) The reasonner know on A1 :



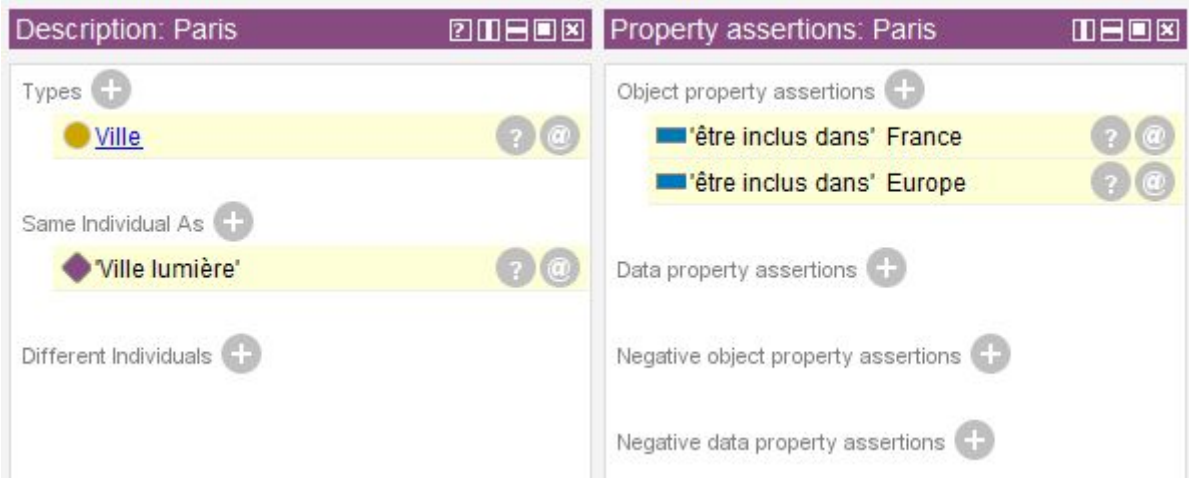
It know that A1 has the type ‘Pluie’ because :

- We said its symptom was A1;
- A1 had type ‘Observation’ and ‘3mm’ of ‘Pluviométrie’
- Pluie is, as we declared with Manchester syntax :



This corresponds to the values we give at P1, which is a symptom of A1.
So the reasonner will know that A1 is Pluie.

2) The reasonner know on Paris :



We declared France has capital Paris and Ville Lumière. So Paris will be know as ‘Ville’ (because a capital is a Ville) and as ‘same individual’ as Ville Lumière because a capital is unique.
It will also be included in France because ‘a pour capitale’ include ‘être inclus dans’ property dans this property is transitive. As we declared ‘France’ in ‘Europe’, Paris is also ‘inclus dans’ Europe.

3) Toulouse as capital of France

If we declare Toulouse as France Capital, is will make Toulouse ‘the same individual’ as Paris and ‘La ville lumière’ (And toulouse will be added in Paris and ‘La ville lumière‘ same individual’ part)

TP2

Java methods

Model :

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

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

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

```
@Override
public String getInstantTimestamp(String instantURI){
            String propertyURI =  this.model.getEntityURI("a pour timestamp").get(0);
            List<List<String>> prop = this.model.listProperties(instantURI);
            for(int i = 0 ; i < prop.size() ; i++) {
                  for(int y = 0 ; y < prop.get(i).size() ; y++) {
                        if(prop.get(i).get(y) == propertyURI) {
                              return prop.get(i).get(y+1);
            }}}
```

```
            return null;
}
```

```
@Override
public String createObs(String value, String paramURI, String instantURI) {
            String obsClassURI = this.model.getEntityURI("Observation").get(0);
            String obsURI =  this.model.createInstance("",obsClassURI);
            String timestampPropertyURI =  this.model.getEntityURI("a pour
date").get(0);
            String valuePropertyURI =  this.model.getEntityURI("a pour valeur").get(0);

            this.model.addDataPropertyToIndividual(obsURI, valuePropertyURI, value);
            this.model.addObjectPropertyToIndividual(obsURI, timestampPropertyURI,
instantURI);

            String timestamp = this.getInstantTimestamp(instantURI);
            String sensorURI = this.model.whichSensorDidIt(timestamp, paramURI);

            this.model.addObservationToSensor(obsURI, sensorURI);
            return obsURI;
}
```

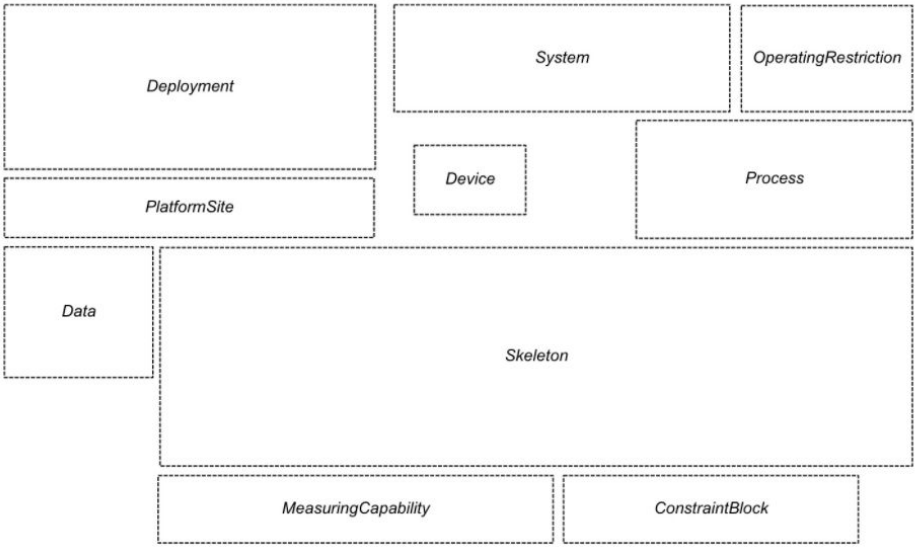
Controller :

```
@Override
public void instantiateObservations(List<ObservationEntity> obsList,String paramURI) {
      Map<String, String> map = new HashMap<String, String>();
      for(ObservationEntity oe : obsList){
            String instURI;
            if(!map.containsKey(oe.getTimestamp().getTimeStamp())){
                  instURI = this.customModel.createInstant(oe.getTimestamp());
                  map.put(oe.getTimestamp().getTimeStamp(), instURI);
            }else{
                  instURI = map.get(oe.getTimestamp().getTimeStamp());
            }
            this.customModel.createObs(oe.getValue().toString(), paramURI, instURI);
      }
}
```

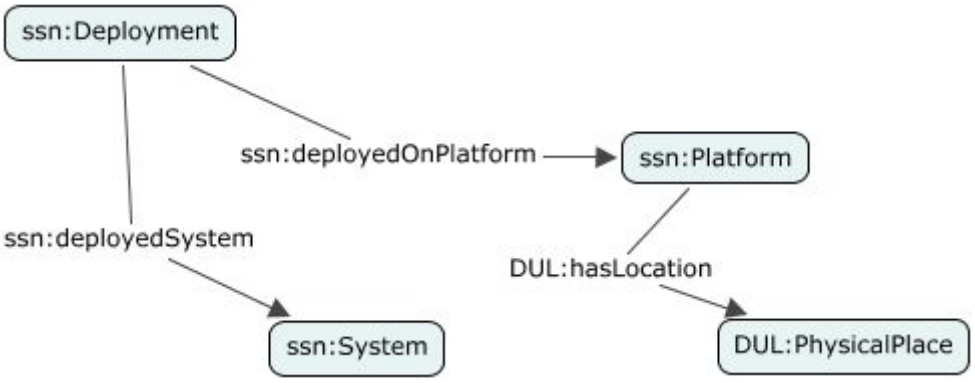
SSN :

SSN ontology describes sensors precisely and modularly.

Sensors will have a skeleton, common to every sensor, and different modules which are used in function of the environment they evolved in and the goals of the sensors, as well as the needs of the people that are using them.



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 :



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