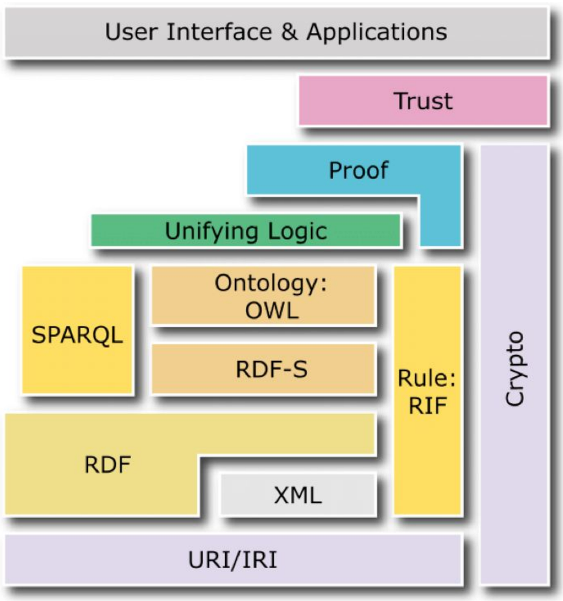
Treatment of semantic data

Abderrazzak Mahraye-Ilhame Hadouche

INSA TOULOUSE ISS B1

I Introduction

The semantic web enables us to structure data that are on the web to better exploit them. The semantic web provides a set of functions that offers a common structure to all data on the web. It enables an automatic treatment. 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 theses lab sessions, we focused on three languages :

• OWL - Ontology Web Language

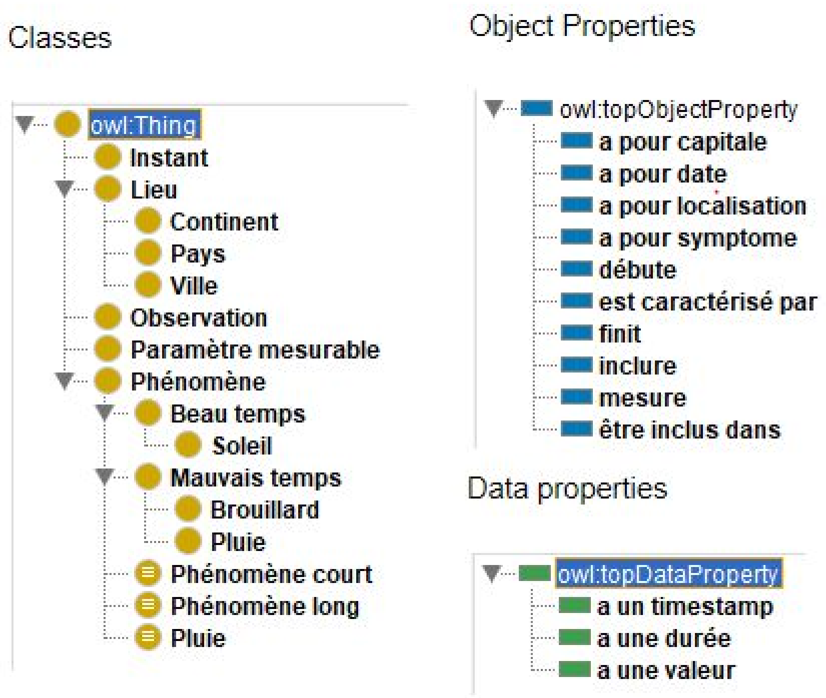
• RDF - Resource Description Framework

• SPARQL - SPARQL Protocol and RDF querying language

II FIRST LAB SESSION: PROTEGE

II.1 Creation of lightweight ontology

During the first session, we defined an ontology of meteorological phenomenons using the OWL ontology development environment in the software "Protégé". To do that, we created relation between the different “objects” thanks to the concepts available in protege(class, individuals, object property, data property).



Here is our results:

For instance, to create the relation “le beau temps et le mauvais temps sont deux types de phénomènes”, we created a class :

• phénomène And two sub-classes :

• beau temps

• mauvais temps

We did the same for the classes "pluie" and "brouillard" that are subclasses of "mauvais temps" and "soleil" a subclass of "beau temps".

In addition, to create the relation “un phénomène est caractérisé par des paramètres mesurables”, we created an object property “est caractérisé par” and we linked “phénomène” and “paramètres mesurables” to it.

Finally, to create the relation “un instant est caracérisé par un timestamps de type xsd:dataTimeStamp”, we created a data property “a pour timestamp” and we linked it to “un instant” and “xsd:dataTimeStamp”

II.2 Deductions by Hermit : lightweight ontology

After the setting up the ontology, the reasoner Hermit can deduct new knowledge. In fact, After we place Toulouse in France, the relation ‘être inclus dans’ having ‘lieu’ as type object, Toulouse and France instance are deducted as ‘Lieu’ by the reasoner. Samilary, it deduced that "Paris" is a city and "France" is a country and that "Paris" is the capital of "France".

II.3 Creation of heavyweight ontology

A heavyweight ontology is like the lightweight ontology with an addition: logical axioms. There are various object property axioms : SubObject-PropertyOf, EquivalentObjectProperties, DisjointObjectProperties, InverseFunctionalObjectProperty…

For instance, to create the relation :

- “Toute instance de ville ne peut pas être un pays”, we created a constraint "disjoint" in "Ville" for "Pays". Automatically the inverse constraint appears in "Pays".
- "Si un lieu A est situé dans un lieu B c’est qye le lieu B est situé dans le lieu C, alors le lieu A est situé dans le lieu C": we just activated the transitivity of the relation "est inclus dans".
- "Un phénomène court est un phénomène dont la durée est de moins de 15 minutes": we created a sub class of "phénomène" called "phénomène court" equivalent to "phénomène and ‘a pour durée’ some xsd:int[<= "15"8sd:int]". We defined the "phénomène long" as the opposite and use the disjoint function
-

II.2 Deductions by Hermit : heavyweight ontology

After having created the relations "Paris est la capitale de France" and "la ville Lumière est la capitale de France" and also that "à tout pays correspond une et une seule capitale", the reasoner deduced that "Paris" and "La ville Lumière" are the equals.

Moreover, 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 reasonner will reject this.

Finally, if we declare "Toulouse" as the capital of France, "Toulouse" is noted as the same individual as "Paris" and "La ville Lumière" by the reasoner.

III SECOND LAB SESSION: CREATION OF A SEMANTIC AWARE APPLICATION IN JAVA

III.1 Implementation of IModelFunctions

There are thee java methods implemented:

```
public String createPlace(String name) {

        // TODO Auto-generated method stub
        String type =model.getEntityURI("Lieu").get(0);
        String place = model.createInstance(name, type);
        return place;
    }
```

```
@Override
    public String createInstant(TimestampEntity instant) {
        // TODO Auto-generated method stub
        String type =model.getEntityURI("Instant").get(0);
        String propertyURI = model.getEntityURI("a pour timestamp").get(0);

        String Instant = model.createInstance(instant.timestamp, type);
        model.addDataPropertyToIndividual(Instant, propertyURI, instant.timestamp);
        return Instant;
    }
```

```
Override
    public String getInstantURI(TimestampEntity instant) {
        String type =model.getEntityURI("Instant").get(0);
        List<String> ListInstants = model.getInstancesURI(type);
        String propertyURI = model.getEntityURI("a pour timestamp").get(0);
        for(String strInstant : ListInstants)
        {

            if(model.hasDataPropertyValue(strInstant, propertyURI, instant.timestamp))
                return strInstant;
        }
        // TODO Auto-generated method stub
        return null;
    }
```

```
@Override
	public String getInstantTimestamp(String instantURI)
	{
		List<List<String>> properties = model.listProperties(instantURI);
		String propertyURI = model.getEntityURI("a pour timestamp").get(0);
		for (List<String> property : properties)
		{
			if(property.get(0).equals(propertyURI))
			{
				return property.get(1);
			}

		}

		return null;
	}
```

```
public String createObs(String value, String paramURI, String instantURI) {
		DateFormat df = new SimpleDateFormat("yyyy-MM-dd'T'HH:mm:ss");
		String type =model.getEntityURI("Observation").get(0);
		String obsURI = model.createInstance("obs", type);
		String propertyURIts = model.getEntityURI("a pour timestamp").get(0);
		String propertyURIv = model.getEntityURI("a pour valeur").get(0);
		String datePropertyURI =model.getEntityURI("a pour date").get(0);
		String sensorURI =model.whichSensorDidIt(getInstantTimestamp(instantURI), paramURI);

		model.addDataPropertyToIndividual(obsURI, propertyURIv, value);
		model.addDataPropertyToIndividual(obsURI, propertyURIts, getInstantTimestamp(instantURI));

		model.addObjectPropertyToIndividual(obsURI, datePropertyURI, instantURI );
		model.addObservationToSensor(obsURI, sensorURI);

		// TODO Auto-generated method stub
		return obsURI;
	}
```

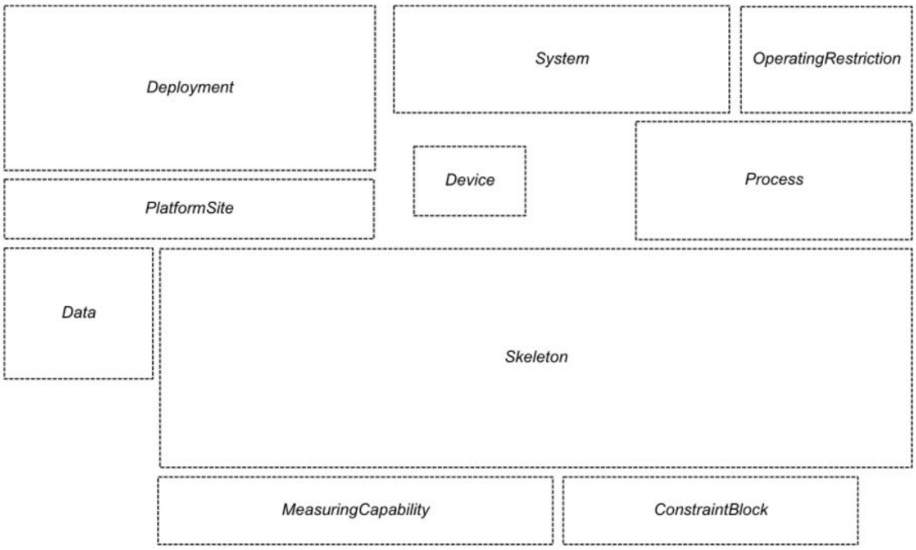
III.2 Implementation of IControlFunctions

```
@Override
	public void instantiateObservations(List<ObservationEntity> obsList,String paramURI) {
		Map<String, String> map = new HashMap<String, String>();
		for(ObservationEntity obsE : obsList){
			String instURI;
			if (map.containsKey(obsE.getTimestamp().getTimeStamp())){
```

```
                instURI = this.customModel.createInstant(obsE.getTimestamp());
                map.put(obsE.getTimestamp().getTimeStamp(), instURI);
            }else{
                instURI = map.get(obsE.getTimestamp().getTimeStamp());
            }
            this.customModel.createObs(obsE.getValue().toString(), paramURI, instURI);
        }
}
```

III.3 Semantic Sensor Network Ontology

SSN is a modulary ontology for sensors. Here is a schema that explains its architecture:



Each sensor must have a Skeleton, and we can describe its characteristics thanks to others modules (Process, DATA, PlatformSite, Deployment etc…).