TP of semantic web of things report

Agathe Lièvre - Assia Nguyen
ISS - 2021/2022

1. Introduction

Ce TP nous offre une introduction au web sémantique. Nous avons pu mettre en place une ontologie sur le logiciel *Protégé*. Enfin, nous avons exploité cette ontologie grâce à une application Java.

2. Création de l'ontologie

Dans cette partie, nous avons créé une ontologie pour des données météorologiques grâce au logiciel *Protégé*.

2.1. L'ontologie légère

Tout d'abord, nous mettons en place une ontologie simple en créant des classes, des sous-classes et des propriétés basiques.

2.1.1. Conception

Nous avons commencé par créer des classes : 'Instant', 'Lieu', 'Observation', 'Paramètre mesurable', 'Phénomène'.

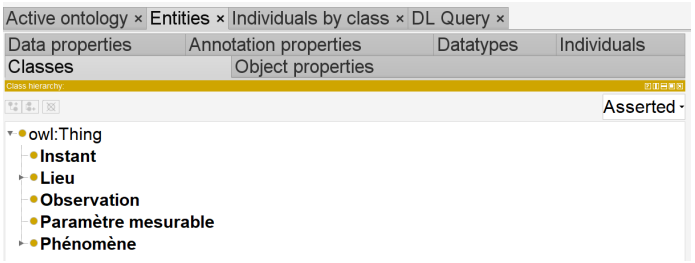


Figure 1 : classes de l'ontologie sur les données météorologiques

Ensuite, nous avons certaines classes qui possèdent des sous-classes, voire même des sous-sous-classes. Cela est fait afin de créer une hiérarchie.

Figure 2 : sous-classes de l'ontologie

Puis, nous avons ajouté des propriétés pour exprimer les relations entre les objets que nous venons de créer :

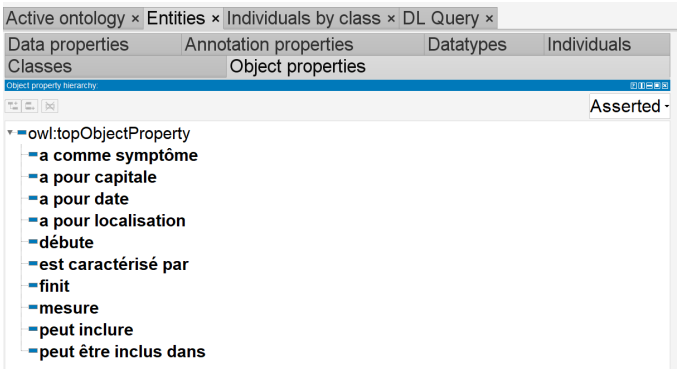


Figure 3 : propriétés des objets de l'ontologie

Pour les créer, nous avons dû leur donner un nom, une classe de départ ('Domains') et une classe d'arrivée ('Ranges'). Par exemple, si nous savons qu'un 'Phénomène a comme symptôme une Observation', cela revient à dire que :

- le nom de la propriété est 'a comme symptôme'
- la classe de départ est 'Phénomène'
- la classe d'arrivée est 'Observation'

Enfin, nous avons ajouté des attributs à des objets :

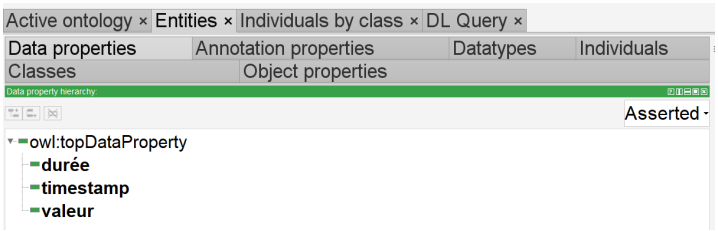


Figure 4 : propriétés des données de l'ontologie

Par exemple, pour dire qu'une 'Observation a une valeur', nous avons ajouté une data property 'valeur', sans unité, sur la classe 'Observation'. Pour ajouter une unité, pour 'timestamp' par exemple, nous lui ajoutons un 'Ranges' :

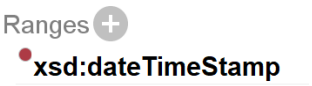


Figure 5 : ajout d'une unité à une propriété de données

2.1.2. Peuplement

Par la suite, nous avons peuplé l'ontologie, c'est-à-dire que nous avons créé des instances des classes mises en place dans la section précédente.

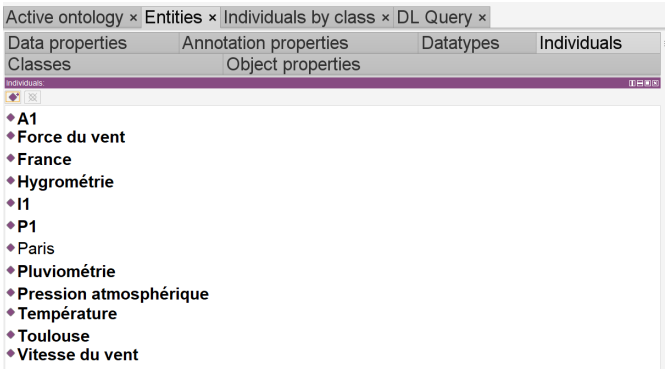
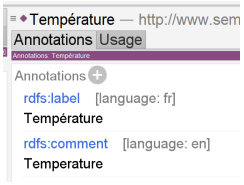


Figure 6 : instances des classes de l'ontologie légère

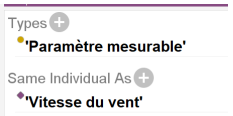
Nous avons donc pu répondre aux neuf questions en reliant les instances entre elles avec les propriétés créées précédemment.

1) Nous avons créé les instances suivantes : *température*, *hygrométrie*, *pluviométrie*, *pression atmosphérique*, *force du vent* et *vitesse du vent.* Elles sont toutes des instances de la classe 'Paramètre mesurable'.

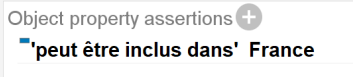
2) Nous avons rendu *temperature* et *température* synonymes.



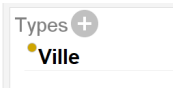
3) Nous avons précisé que *force du vent* et *vitesse du vent* sont deux instances similaires.



4) Nous avons utilisé la propriété 'peut être inclus dans' pour préciser que Toulouse se situe en France. Le raisonneur a donc déduit que Toulouse et la France sont des lieux car la propriété définit qu'un lieu peut être inclus dans un lieu.

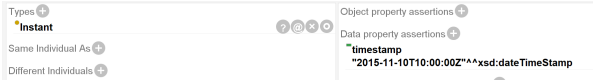


5) Nous avons défini le type de *Toulouse* comme étant une ‘Ville’.



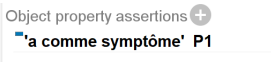
6) Si nous mettons que la France a pour capitale Paris, le raisonneur est capable de déduire que Paris est une ville alors c'était un individu non typé au départ. Le raisonneur a déterminé que la France était un pays car il a une capitale.

7) Nous avons créé une instance de la classe ‘Instant’ qui est un Timestamp. Nous devons utiliser un format particulier pour la définir.



8) Nous devons donner à *P1* plusieurs propriétés pour lui donner une ‘valeur’, un ‘Lieu’, un ‘Instant’ et un ‘Paramètre mesurable’.

9) Nous avons donné comme propriété ‘a comme symptôme’ pour associer l’Observation’ *P1* au ‘Phénomène’ *A1*.



En créant ces instances et après avoir lancé le raisonneur, nous avons remarqué la force du web sémantique. En effet, le raisonneur peut déduire des liens et des propriétés complémentaires à partir des informations de chaque instance.

2.2. L'ontologie lourde

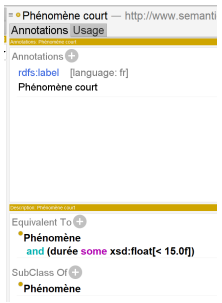
Après avoir instauré des relations simples, nous allons ajouter des liens entre les classes pour complexifier l'ontologie.

2.2.1. Conception

Pour mettre en place une ontologie lourde, nous avons d'abord utilisé le langage de Manchester. Ce langage permet d'écrire des règles sur les différents éléments de notre ontologie. Ces règles peuvent établir des conditions pour l'appartenance à une classe par exemple. Nous pouvons aussi jouer sur les caractéristiques (transitivité, unicité, etc.) des propriétés.

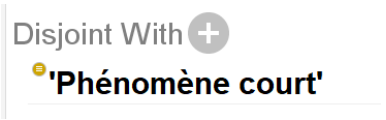
1) Nous avons séparé 'Ville' et 'Pays' pour qu'une ville ne puisse pas être un pays.

2) Nous avons ajouté une règle pour 'Phénomène court' pour définir une condition sur la 'durée'.

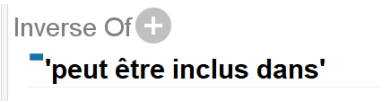


3) Nous avons fait de même pour 'Phénomène long'.

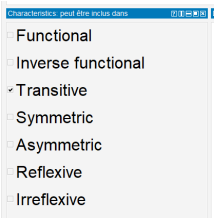
4) De même que pour 'Ville' et 'Pays', nous avons séparé 'Phénomène court' de 'Phénomène long'.



5) Nous avons précisé que la propriété 'est inclus dans' est l'inverse de 'peut être inclus dans'.



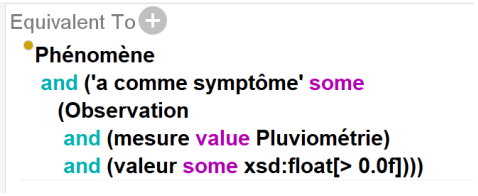
6) Pour dire qu'un lieu A situé dans un lieu B, lui-même situé dans un lieu C signifie que A est aussi dans C, nous avons défini que la caractéristique de la relation est 'Transitive'.



7) Nous avons précisé qu'un 'Pays' ne peut avoir qu'une 'Capitale'.

8) Pour dire qu'une 'Capitale' fait partie d'un 'Pays', nous avons précisé que la propriété 'a pour capitale' est une sous-propriété de 'peut inclure'.

9) Nous avons ajouté une règle à 'Pluie' pour préciser des conditions.



2.2.2. Peuplement

Après avoir ajouté ces nouvelles règles, nous pouvons à nouveau peupler l'ontologie.

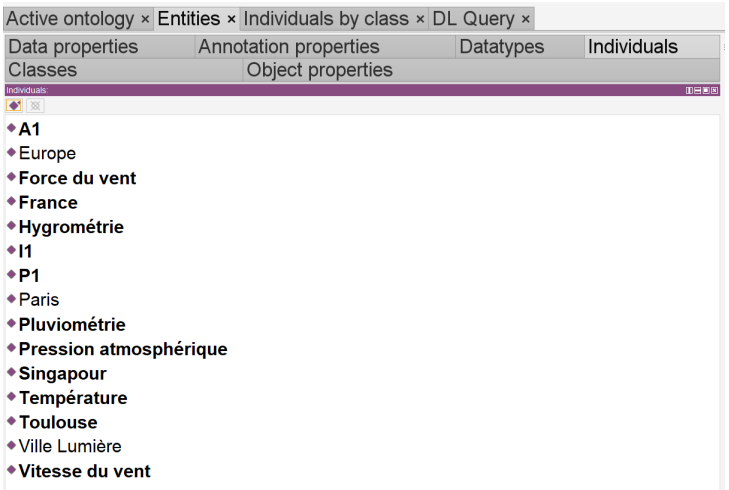


Figure 7 : instances des classes de l'ontologie lourde

1) A1 est un ‘Phénomène’ puisqu’il ‘a pour symptôme’ une ‘Observation’ (P1).
2) Ici, nous avons une ontologie inconsistante car nous avons dit qu’un ‘Pays’ ne peut avoir qu’une ‘Capitale’. Or, nous avons précisé que Paris et Ville Lumière sont des ‘Capitale’ de la France. L’ontologie est doublement inconsistante car nous avons dit qu’un ‘Pays’ ne peut pas être une ‘Ville’.
3) Le raisonneur ne comprend pas car nous avons déclaré qu’un ‘Pays’ ne pouvait avoir qu’une ‘Capitale’.

3. Exploitation de l’ontologie

Dans cette partie, nous allons mettre à profit l’ontologie que nous avons créé à travers une application sémantique Java. Par manque de temps, nous n’avons pas pu faire la partie sur *Protégé*.

Après avoir observé les spécifications de fonctions fournies dans le fichier *IConvenienceInterface*, nous avons pu implémenter les méthodes déclarées dans *IModelFunctions* dans l’interface *DoItYourselfModel* . Nous avons créé des “individuals” basés sur l’ontologie fournie à travers *IModelFunctions*, puis *DoItYourselfModel* utilise le modèle sémantique pour interagir avec l’ontologie. Enfin, le contrôleur *IControlFunctions* utilise des fonctions du modèle pour enrichir le data set. Dans ce fichier, nous instancions des observations en utilisant *IModelFunctions*.

Après avoir implémenté les méthodes manquantes dans l’interface *DoItYourselfModel,* nous les avons testées avec Maven. Le code de DoItYourselModel peut être visualisé ci-dessous:

```java
package semantic.model;

import java.util.List;

public class DoItYourselfModel implements IModelFunctions
{
    IConvenienceInterface model;

    public DoItYourselfModel(IConvenienceInterface m) {
        this.model = m;
    }

    @Override
    public String createPlace(String name) {
        List<String> instanceURI = this.model.getEntityURI("Place");
        String createdURI = this.model.createInstance(name, instanceURI.get(0));
        return createdURI;
    }
```

```java
@Override
public String createInstant(TimestampEntity instant) {
    String timestamp = instant.getTimeStamp();
    List<String> entitiesURI = this.model.getEntityURI("Instant");
    List<String> instancesURI = this.model.getInstancesURI(entitiesURI.get(0));

    for(int i=0; i < instancesURI.size(); i++) {
        if(this.model.hasDataPropertyValue(instancesURI.get(i), this.model.getEntityURI("a pour timestamp").get(0), timestamp)) {
            return null;
        } else {
            String createdURI = this.model.createInstance("Inst"+timestamp, entitiesURI.get(0));
            this.model.addDataPropertyToIndividual(createdURI, this.model.getEntityURI("a pour timestamp").get(0), timestamp);
            return createdURI;
        }
    }
    return null;
}
```

```java
@Override
public String getInstantURI(TimestampEntity instant) {
    String timestamp = instant.getTimeStamp();
    List<String> entitiesURI = this.model.getEntityURI("Instant");
    List<String> instancesURI = this.model.getInstancesURI(entitiesURI.get(0));

    for(int i=0; i < instancesURI.size(); i++) {
        if(this.model.hasDataPropertyValue(instancesURI.get(i), this.model.getEntityURI("a pour timestamp").get(0), timestamp)) {
            return instancesURI.get(i);
        } else {
            return null;
        }
    }
    return null;
}
```

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

```java
@Override
public String createObs(String value, String paramURI, String instantURI) {
    List<String> instanceURI = this.model.getEntityURI("Observation");
    String createdURI = this.model.createInstance("Obs"+value, instanceURI.get(0));
    String timestamp = getInstantTimestamp(instantURI);
    String sensorURI = this.model.whichSensorDidIt(timestamp, paramURI);

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

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

Figure 8 : code Java de DoItYourselfModel

Nous avons bien utilisé les classes créées sur Protégé dans le TP précédent comme ‘Lieu’, ‘Instant’ ou encore ‘Observation’. Nous avons donc pu implémenter une application sémantique java qui interagit avec notre ontologie à travers des méthodes utilisant les URI des objets créés dans Protégé.

Différence entre une object property et une data property : un attribut peut être une propriété de données (data property) ou une propriété d’objet (object property). Elles décrivent toutes deux le type de valeur qu'une déclaration doit avoir. Les object properties connectent deux individus: un sujet et un objet (ou un individu à un autre individu) avec un prédicat alors que les data properties, relient un seul sujet avec une forme de donnée d’attribut (attribute data). Les data properties possèdent des datatypes comme un string, un integer ou encore un datetime.

4. Conclusion

Grâce à ces deux travaux pratiques, nous avons pu appréhender les concepts de base du web sémantique. Dans un premier temps, la création d’une ontologie sur Protégé nous a permis de démarrer avec simplicité ce modèle avec un raisonneur. Le deuxième TP nous a fait utiliser l'ontologie précédente pour réaliser une API et nous a permis de mieux comprendre la manipulation de méthodes avec le framework Apache Jena. La complexité a légèrement augmenté car nous avons manipulé des Identifiants de Ressource Uniforme (URI) alors que dans Protégé, ils étaient implicites. Nous considérons avoir assimilé les principes du Web Sémantique à la fin de ces TP.