Institut National des Sciences Appliquées
PTP Innovative Smart System

Rapport Web Sémantique

Schersach Baptiste - Arens Maxime

Toulouse, le 16 décembre 2019

1 Introduction

Le web sémantique permet de rajouter du sens aux informations sur le web. Derrière le mot sémantique se cache plusieurs technologies, tel que RDF(Ressource Description Framework). Ces données augmentées sont contenues dans des ontologies, des vocabulaires decrivant des données et les liens entre celles-ci.

Grâce au web sémantique et aux méta-données, la qualité des moteurs de recherche à grandit ces dernières années. Le web sémantique permet de rajouter un sens au contenu du web, interprétable non seulement par l'homme, mais aussi par une machine. Cette meilleure compréhension permet une plus grande performance et robustesse. Le language RDF permet de définir les ressources en triplets : sujet, prédicat et objet. C'est par exemple cette répartition en triplets qui rend la lecture de l'information plus facile.

Aujourd'hui, nous pouvons surfer sur le web, aller sur des forums, faire des recherches, le web sémantique permet que les machines puissent mieux comprendre ces ressources destinés à des humains et donc nous aider à trouver ce qui nous intéresse.

2 Création de l'ontologie

L'objectif de cette partie est de créer une ontologie, un vocabulaire décrivant des données issues du milieu météorologique avec l'aide du logiciel Protégé.

2.1 L'ontologie légère

Nous commencerons par mettre en place des objets et des relations simples.

2.1.1 Conception

Pour représenter des objets nous avons commencé par faire des classes , chaque classes peut avoir des sous classes afin de créer une hiérarchie. En structurant les données entre elles nous leurs rajoutons déjà du sens.

En créant des classes (dans l'onglet entities/class de Protégé) nous avons pu répondre aux questions 1 à 4 de cette partie. Nous pouvons voir ci-dessous l'état final de nos classes.

Nous pouvons voir que la pluie et le brouillard sont des sous-classes de mauvais temps, qui est elle même une sous classe de phénomène (météorologique).

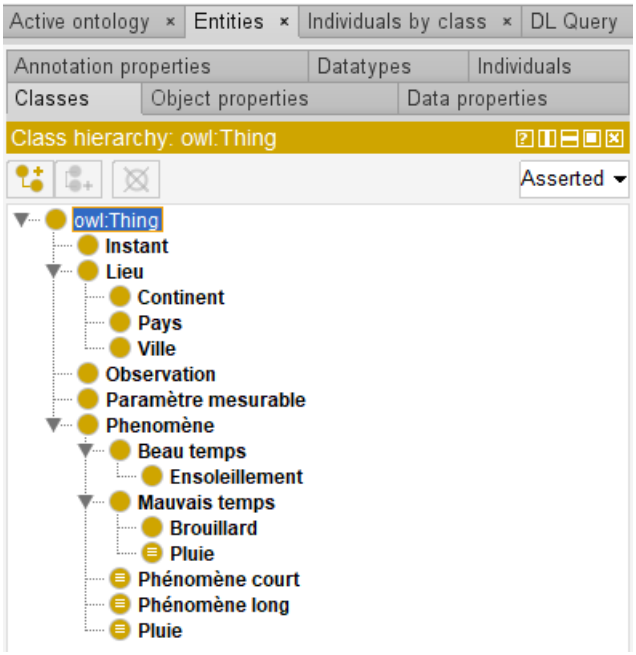

FIGURE 1 – Classes de l'ontologie météo sur Protégé

Ayant désormais des objets existants dans notre ontologie, il était temps de leur rajouter des propriétés si nous voulions exprimer des relations entre eux. Il est intéressant de noter qu'une propiété d'un objet se transmet à ses sous-classes.

Pour cela nous avons créé de nouvelles propriétés dans entities/ObjectProperties en renseignant dans ces propriétés : leur nom, leur classe de départ et leur classe d'arrivée. Par exemple, exprimer qu'un *Phénomène débute à un Instant* s'implémente à travers une propriété de classe ayant pour nom **débute à**, pour classe de départ **Phénomène** et pour classe d'arrivée **Instant**.

En reprenant cette idée nous avons pu repondre aux questions de 1 à 13 décrivant une relation entre deux classes.

Ci-dessous l'état final de nos object properties. Nous pouvons rajouter que certaines propriétés peuvent êtres des sous-propriétés : être la capitale d'un pays est une sous-propriété d'être la ville d'un pays.

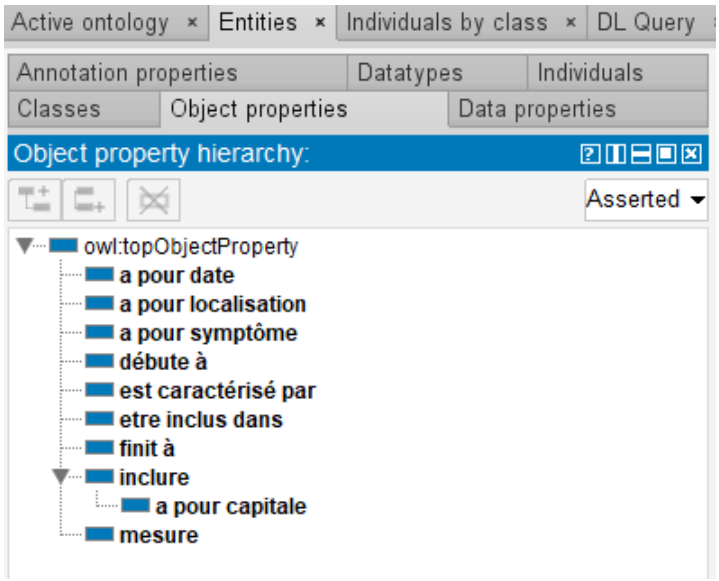

FIGURE 2 – Relation des objets de l'ontologie météo

Pour répondre aux restes de questions qui portaient sur le rajout d'attributs à un objet nous avons utilisé des data properties (dans entities/DataProperties). Par exemple pour représenter le fait qu'*un instant à une date* nous avons rajouté une propriété **Timestanp** sur la classe **Instant**.

Ci-dessous les quelques propriétés que nous avons rajouté sur certaines de nos classes.

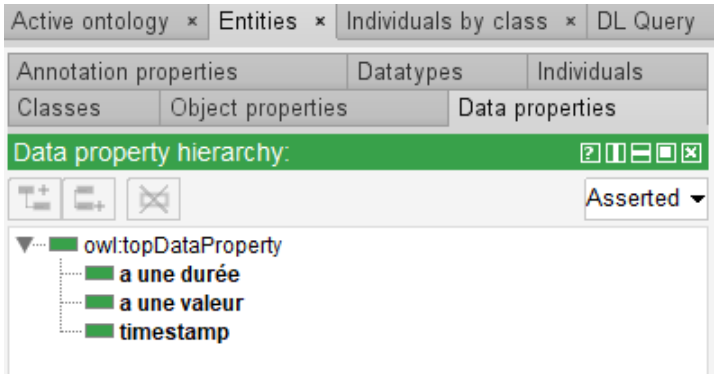

FIGURE 3 – Propriétés des classes de l'ontologie météo

2.1.2 Peuplement

Peupler une ontologie consiste à créer des instances des classes créées précédement.

Ci-dessous les différentes instances que nous avons créées pour répondre aux questions 1 à 9 de cette partie.

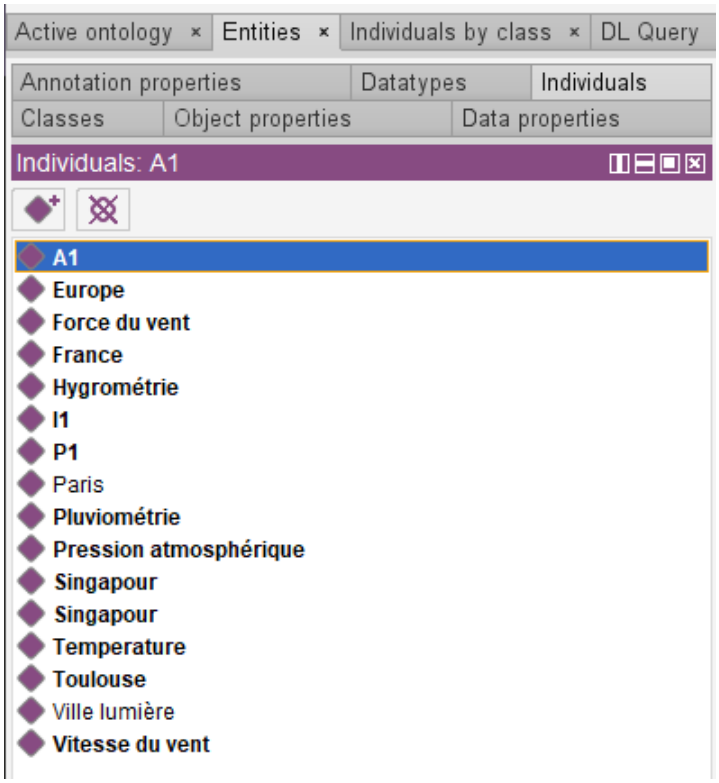

FIGURE 4 – Instances des classes de l'ontologie météo

Nous pouvons relier ces instances entre elle avec les propriétés de relation vu précédemment. Grâce aux propriétés sur les données nous rajoutons des propriétés à ces instances.

Une fois que nous avions répondu aux questions, nous avions des instances avec des liens entre elles et des informations complémentaires sur certaines de ces instances. Nous avons pu constater la force du web sémantique quand nous avons lancer le raisonneur sur cette ontologie.

En effet, le raisonneur à inféré certains liens ou propriétés sur des instances à partir des informations que nous lui avions donné. Par exemple en notant que la France à pour capital Paris, le raisonneur à pu inférer que Paris est une ville.

2.2 L'ontologie lourde

Maintenant que nous avons pu voir comment des objets aux relations simples peuvent former une ontologie intéressante nous allons complexifier les choses et voir quelques applications supplémentaires d'une ontologie.

2.2.1 Conception

Afin de concevoir une ontologie lourde nous avons d'abord utiliser le langage de Manchester afin d'écrire des règles sur les différents éléments de notre ontologie. Ces règles permettent d'établir une condition pour l'appartenance à une classe : suivant la valeur d'un attribut ou un vérifiant l'absence d'appartenance à une classe spécifique (et contradictoire).

Ensuite pour compléter cette ontologie nous avons joué sur les caractéristiques des propriétés sur les relations que nous avions créé dans la première partie. En rajoutant par exemple des propiétés de transitivité, d'unicité sur ces relations.

Lorsque l'on lance le raisonneur désormais on peut voir que grâce à la condition d'unicité sur le fait d'avoir une capitale, le fait que Paris et la Ville-Lumière sont deux instances qui sont capitales de la France signifie pour le raisonneur que Paris et Ville-Lumière sont la même instance.

Si on ajoute Toulouse comme capitale, alors on obtient que Toulouse est équivalent à Paris et Ville-Lumière. Si on avait rajouter une condition sur la région (une ville ne peut appartenir qu'à une région), alors le fait que Toulouse et Paris n'appartiennent pas à la même région aurait déclenché un conflit pour le raisonneur.

3 Exploitation de l'ontologie

Après voir cloné le repository, nous nous sommes attelés à l'implémentation des interfaces. Nous avons eu beaucoup de mal à faire ce TP, du fait que trouver les bonnes informations était assez difficile. Un membre du groupe vient de Toulouse Business School et n'avait pas forcément les connaissances en Java requises pour ces travaux pratiques. Nous avons donc préféré faire moins mais en prenant le temps de comprendre ce que nous faisions. Nous n'avons donc notamment pas pu réaliser la partie sur Protégé.

La première interface *IModelFonctions* est utilisé pour créer des *individuals* basés sur notre ontologie. L'interface implémentée *DoItYourslef* utilise le modèle sémantique pour facilement intéragir avec notre ontologie.

La deuxième interface, *IControlFonctions* a été utilisé pour créer des observations tout en utilisant l'interface *IModelFonctions*. Le rôle du controller étant d'établir un lien entre le dataset et le meta-data.

Nous avons donc jonglé entre ces interfaces et le fichier de test (un petit peu de rétro-engineering) afin de pouvoir remplir les méthodes manquantes.

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

public String createInstant(TimestampEntity instant) {
   List<String> listInstant =
       model.getInstancesURI(model.getEntityURI("Instant").get(0));
   for (int i = 0; i < listInstant.size(); i++) {
      if (model.hasDataPropertyValue(listInstant.get(i),
          model.getEntityURI("a pour timestamp").get(0) ,
          instant.getTimeStamp())) {
         return null;
      }
      else {
         String newInstance = model.createInstance("Instant",
             model.getEntityURI("Instant").get(0));
         model.addDataPropertyToIndividual(newInstance,
             model.getEntityURI("a pour timestamp").get(0),
             instant.getTimeStamp());
         return  newInstance;
      }
   }
   return null;
}

public String getInstantURI(TimestampEntity instant) {
   List<String> listInstant =
       model.getInstancesURI(model.getEntityURI("Instant").get(0));
```

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

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

public String createObs(String value, String paramURI, String
    instantURI) {
   String newObs = model.createInstance("Observation",
       model.getEntityURI("Observation").get(0));
   model.addObjectPropertyToIndividual(newObs, model.getEntityURI("a
       pour timestamp").get(0), instantURI);
   model.addDataPropertyToIndividual(newObs, model.getEntityURI("a
       pour valeur").get(0), value);
   model.addObjectPropertyToIndividual(newObs,
       model.getEntityURI("mesure").get(0), paramURI);

   String sensor =
       model.whichSensorDidIt(this.getInstantTimestamp(instantURI),
       paramURI);
   model.addObservationToSensor(newObs, sensor);
   return newObs;
}
```

Nous retrouvons les classes de Place, Instant et Observator que nous avions créées sur Protégé. Cela nous permets de voir que nous pouvons intéragir avec notre ontologie à travers une API Java ce qui ouvre les horizons d'implémentation par rapport à un logiciel comme Protégé.

Durant la réalisation de ce second travaux pratique nous ne sommes plus rendu compte de l'importance de l'URI des objets que nous créions dans Protégé.

Ce sont ces URI qui permettent l'identification et la sélection de nos objets (ce qui se faisait de façon "invisible" sur Protégé).

4 Conclusion

Nous avons pu découvrir différents aspects du web sémantique, à travers l'utilisation d'ontologie avec notamment Protégé. Nous avons appris comment créer une ontologie, la peuplé et l'utilisé avec un reasoner. Bien que la 2ième partie du travaux pratique avec l'utilisation d'une API Java ait été beaucoup plus difficile que la première, l'objectif du cours de créer, d'implémenter et de manipuler les ontologies a été atteint.