TP ISS Web Sémentique rapport

Ting Chen, Tehema Teiti

December 7, 2018

1 Introduction

Le cours "De l'Internet of Things au Web Semantic of Things" introduit les principes et technologies du web sémantique. Une série de travaux pratiques en deux parties est réalisée afin de manipuler et de comprendre les concepts du web sémantique afin de l'utiliser dans le cadre de l'Internet des Objets. La première partie se concentre sur la création d'une ontologie liée à l'observations de phénomènes météorologiques à l'aide du logiciel Protege. La deuxième partie permet, à travers une API Java, de manipuler de façon simple les concepts du web sémantique en réutilisant l'ontologie créée dans la première partie. Ce rapport présente la réalisation de ces travaux pratiques en expliquant les concepts étudiés du web sémantique.

2 Création de l'ontologie

2.1 L'ontologie légère

2.1.1 Conception

Nous avons représenté la connaissance décrite à la question 2.2.1 par la création d'une hiérarchie de classe présentée dans la figure 1.

2.1.2 Peuplement

1. 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 ?
   Le raisonneur définie les individus **Toulouse** et **France** comme des lieux. Cela s'explique par le fait que la propriété est situé met en relation un lieu avec un autre lieu.

2. Toulouse est une ville.
   Le raisonneur n'infère plus que **Toulouse** est un lieu car on le définit explicitement comme une ville. La classe ville étant une sous-classe de lieu, Toulouse est implicitement un lieu.

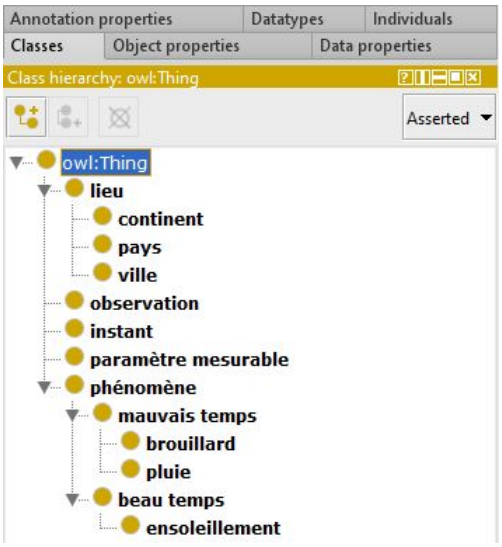

Figure 1: Classes représentant la connaissance

3. La France a pour capitale Paris. Ici aussi, Paris est un individu non typé. Le raisonneur infère que **France** est un pays et **Paris** est une ville car elles sont liées par la propriété **a pour capitale** qui met en relation un pays et une ville.

4. A1 a pour symptôme P1.
   Le raisonneur infère que A1 est un phénomène car la propriété **a pour symptôme** met en relation un **phénomène** et une **observation**.

2.2 L'ontologie lourde

2.2.1 Conception

A l'issue de ce TP, nous avons conçu l'ébauche d'une ontologie lourde de plusieurs façons, selon les faits devant être modélisés.
La première façon est de passer par une expression suivant la **syntaxe Manchester** afin de spécifier des contraintes sur une classe. Dans la question 2.2.3.2, on modélise le fait qu'un **phénomène court** doit avoir une durée inférieure à 15 minutes en utilisant la syntaxe Manchester : phénomène and (′ a une durée en minutes ′ some xsd:int <15 and xsd:int])
La deuxième façon est l'utilisation de **caractéristiques de relation** afin de poser des conditions sur un classe. Dans la question 2.2.3.7, on modélise le fait qu'à tout pays correspond une et une seule capitale en définissant la propriété **a pour capitale** avec une relation **functional**.
La troisième façon est l'utilisation de **sous-propriétés** pour définir des relations entre les classes. Dans la question 2.2.3.8, on modélise le fait que "si un

pays a pour capitale une ville, alors ce pays contient cette ville" en définissant qu'une **capitale** est une sous-classe de **ville**.

2.2.2 Peuplement

1. Que sait le raisonneur de A1 ?
   Le raisonneur infère que A1 a pour type **pluie**. En effet, on définit que la pluie est un phénomène ayant pour symptôme une observation de **pluviométrie** dont la valeur est supérieure à 0. De plus, A1 a pour symptôme P1, et P1 a pour valeur 3. Donc A1 est correspond au phénomène de pluie.

2. Que sait le raisonneur de Paris ?
   Le raisonneur infère que **Paris** égal **la Ville Lumière** car on définit à tout pays correspond une et une seule capitale. Paris est **la capitale** de la France et la Ville Lumière est **la capitale** de la France, alors Paris et **la Ville Lumière** sont égales.

3. Constatez la réaction du raisonneur si Toulouse est déclarée comme le capitale de la France ?
   Le raisonneur infère qu'il y a une erreur dans l'ontologie. Parce qu'on définit **à tout pays correspond une et une seule capitale**. Et puis **Paris** est la capitale de la France, ensuite si on définit **Toulouse** est la capitale de la France. Ce sera une erreur pour l'ontologie.

3 Exploitation de l'ontologie

3.1 Compréhension du sujet

Le but de la deuxième séance de ce TP en Web sémantique est de décrire sémantiquement des données brutes issues d'un dataset à travers une API Java, Jena.
Par rapport aux données brutes issues du dataset, l'ontologie permet de définir une hiérarchie de classes qui permettront de donner une sémantique à ces données. L'API Java propose des fonctions afin de manipuler les principes du web sémantique, comme la création d'instances.
Comme pour la manipulation via Protégé, l'API fait une distinction entre le label et l'URI. La fonction **createInstance(label, type)** permet de créer une instance en lui spécifiant un label et une classe : **label** qui représente le label de l'instance, et **type** qui représente l'URI de la classe auquelle doit appartenir l'instance qui sera créée. Cette fonction retourne alors l'URI de la nouvelle instance créée.
Afin de modéliser les données brutes provenant des datasets, la classe **DoItYourselfModel** contient des fonctions que nous devons implémenter et qui permettent de créer des instances à partir de ces données. La classe **DoItYourselfControl** propose une fonction **instantiateObservations(obsList, paramURI)** que nous devons implémenter et qui permet d'instancier un ensemble

d'observations, liées à un paramètre observable, à partir des données brutes de dataset. La classe **Controler** possède une fonction **main()** qui implémente un programme récupérant les données brutes d'un fichier texte et appelle les fonctions que nous avions implémentées précédemment afin de générer un modèle qui sera ensuite exporter au format **ttl**. Ce fichier peut être alors exploité par Protégé afin de vérifier que les données brutes issues des datasets sont bien décrites sémantiquement.

3.2 Implémentation

Dans cette partie, nous présentons la façon dont nous avons implémenter les fonctions des classes **DoItYourselfModel** et **DoItYourselfControl**.

1. La fonction **createPlace()** prend en paramètre un nom, doit créer une instance de type Lieu et doit retourner l'URI de cette instance. Pour créer une instance de type Lieu, on peut utiliser la fonction **createInstance()** de la classe **IConvenienceInterface** qui prend en paramètre un label et l'URI d'une classe (dans notre cas, la classe Lieu). Pour récupérer l'URI à partir d'une chaîne de caractère, on utilise la fonction **getEntityURI()** de la classe **IConvenienceInterface** qui retourne une liste d'URIs. Dans notre cas, notre ontologie ne contient qu'une URI de classe Lieu. On peut donc directement le récupérer avec **list.get(0)**. Enfin, on passe en paramètre cet URI dans la fonction **createInstance** qui retourne alors l'URI de la nouvelle instance.

2. La fonction **createInstant()** prend en paramètre un objet de type TimeStampEntity et doit retourner l'URI d'un nouvel instant. On doit alors créer une instance de type Instant lié à une valeur représentant un "timestamp" par la dataProperty **a pour timestamp**. Comme pour la fonction précédente, on utilise les deux fonctions **getEntityURI()** pour récupérer l'URI de la classe Instant et **createInstance()** pour retouner l'URI de l'instance que l'on vient de créer. De plus, il faut récupérer l'URI de la dataProperty **a pour timestamp** afin de lier l'instant avec sa valeur. Enfin, on lie l'URI de l'instant et l'URI de la dataProperty avec la fonction **addDataPropertyToIndividual()**.

3. La fonction **getInstantURI()** prend en paramètre un objet de type TimestampEntity et doit retourner l'URI de l'instant dont le "timestamp" correspond à ce paramètre. Tout d'abord, on récupère l'URI de tous les instants avec la fonction **getInstancesURI()** en précisant l'URI de la classe Instant en paramètre. Ensuite, on parcours la liste des URIs des instants récupérés puis on récupère la valeur liée à la dataProperty **a pour timestamp** : si la valeur liée correspond au timestamp passé en paramètre, on retourne l'URI de cet instant. Sinon, on retourne null.

4. La fonction **getInstantTimestamp()** prend en paramètre l'URI d'un instant (une instance de la classe Instant) et doit retourner sa propre valeur

timestamp. Pour ce faire, on récupère toutes les properties de cet instant avec la fonctions **listProperties()** en passant en paramètre l'URI de l'instant. Chaque élément de cette liste contient un couple **(property, object)** avec **property** le nom d'une property et **object** la valeur associée. On vérifie alors pour chaque couple si la property correspond à la dataProperty **a pour timestamp**. Si oui, on retourne sa valeur timestamp. Sinon, on retourne null.

5. La fonction **createObs()** prend en paramètre une valeur, l'URI d'un paramètre observable et l'URI d'un instant. Cette fonction doit alors instancier une observation à partir de ces paramètres et retourner l'URI de cette nouvelle instance. De plus, afin de valider les tests unitaires, nous devons associer le bon capteur à cette observation. Dans un premier temps, comme pour les questions précédentes, nous instancions l'observation en récupérant les URIs des différentes ressources puis nous associons les propriétés à cette observation. Ensuite, la classe **IConvenienceInterface** propose les fonctions **whichSensorDidIt()** et **addObservationToSensor()** afin de récupérer l'URI du capteur adapté à un paramètre observable et de l'associer à une observation.

6. La fonction **instanciateObservation()** prend en paramètre une liste d'ObservationEntity et l'URI d'un paramètre observable. Elle doit ensuite instancier toutes les observations de cette liste en réutilisant la fonction **createObs()** que nous avions précédemment implémenté.

3.3 Code source

Ci-joint, le code.

```
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) {
            // labels of places
            String place = model.getEntityURI("Lieu").get(0);
```

```
        String URI = model.createInstance(name, place);
        return URI;
}

@Override
public String createInstance(TimestampEntity instant) {

        String instantURI = model.getEntityURI("Instant").get
            (0);
        String dataPropertyURI = model.getEntityURI("a pour
            timestamp").get(0);

        String URI = model.createInstance(instant.
            getTimeStamp(), instantURI);

        model.addDataPropertyToIndividual(URI,
            dataPropertyURI, instant.getTimeStamp());

        return URI;

}

@Override
public String getInstantURI(TimestampEntity instant) {

        String instantURI = model.getEntityURI("Instant").get
            (0);
        String dataPropertyURI = model.getEntityURI("a pour
            timestamp").get(0);

        List<String> instances = model.getInstancesURI(
            instantURI);

        for (int i = 0; i < instances.size(); i++) {
                if(model.hasDataPropertyValue(instances.get(i),
                    dataPropertyURI, instant.getTimeStamp())) {
                        return instances.get(i);
                }

        }

        return null;
}

@Override
public String getInstantTimestamp(String instantURI)
```

```
{
        List<List<String>> listProperties = model.
            listProperties(instantURI);
        String dataPropertyURI = model.getEntityURI("a pour
            timestamp").get(0);

        for (int i = 0; i < listProperties.size(); i++) {
                if(listProperties.get(i).get(0) ==
                    dataPropertyURI) {
                        return listProperties.get(i).get(1);
                }
        }
        return null;
}

@Override
public String createObs(String value, String paramURI, String
    instantURI) {
        String obeservationURI = model.getEntityURI("
            Observation").get(0);

        String objectPropertyURI = model.getEntityURI("a pour
            date").get(0);
        String objectPropertyURI2 = model.getEntityURI("
            mesure").get(0);
        String dataPropertyURI = model.getEntityURI("a pour
            valeur").get(0);
        String dataPropertyURI2 = model.getEntityURI("a pour
            timestamp").get(0);

        String uri = model.createInstance("observation",
            obeservationURI);

        model.addObjectPropertyToIndividual(uri,
            objectPropertyURI, instantURI);
        model.addObjectPropertyToIndividual(uri,
            objectPropertyURI2, paramURI);

        model.addDataPropertyToIndividual(uri,
            dataPropertyURI, value);

        String sensorURI = null;
        List<List<String>> listProperties = model.
            listProperties(instantURI);

        for (int i = 0; i < listProperties.size(); i++) {
```

```
                if(listProperties.get(i).get(0) ==
                    dataPropertyURI2) {

                    sensorURI = model.whichSensorDidIt(
                        listProperties.get(i).get(1),
                        paramURI);
                }
            }

            model.addObservationToSensor(uri, sensorURI);
            return uri;
    }
}


public class DoItYourselfControl implements IControlFunctions
{
    private IConvenienceInterface model;
    private IModelFunctions cusotmModel;

    public DoItYourselfControl(IConvenienceInterface model,
        IModelFunctions customModel)
    {
            this.model = model;
            this.cusotmModel = customModel;
    }

    @Override
    public void instantiateObservations(List<ObservationEntity>
        obsList,
                String paramURI) {
            for (ObservationEntity obs : obsList) {

                String instantURI = cusotmModel.createInstant(
                    obs.getTimestamp());
                cusotmModel.createObs(obs.getValue().toString(),
                     paramURI, instantURI);
            }

    }
}
```