INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES TOULOUSE
Université Fédérale Toulouse Midi-Pyrénées
LAAS
Département de Génie Électrique & Informatique

TP ISS exemple rapport

Emmanuel FERRANDI et Pierre LAURENS

19 décembre 2019

Table des matières

1 Introduction

Dans ce rapport, nous avons présenter le principe général de l'ontologie par une définition mais ensuite avec une application concrète en lien avec la météo. Nous avons utilisé le logiciel Protégé en version 5.5.0 pour faire nos expériences ainsi que Java8 et Maven pour la partie 2.

2 Création de l'ontologie

2.1 L'ontologie légère

Une définition d'une ontologie prise dans le livre Data Knowledge Engineering, publié en 1998, "An ontology is a formal, explicit specification of a shared conceptualisation". Autrement dit, une ontologie est un vocabulaire permettant la description des données par modélisation des connaissances. Prenons un exemple, un avocat correspond au nom d'un métier mais aussi d'un fruit. C'est là que l'ontologie va être utile pour mieux définir un nom ou un label pour qu'une machine puisse mieux le définir.
Ce principe est utile lors de l'utilisation d'un grand nombre de donnée différentes compréhensible par l'humain et la machine.

2.1.1 Conception

Dans un premier temps, nous avons créer les différentes classes qui vont nous être nécessaire tout au long de l'expérience. Dans le cadre de notre sujet, la météo, nous avons avoir besoin des classes suivantes :

- phénoménes :
  - beau temps :
    - ensoleillement
  - mauvais temps :
    - pluie
    - brouillard
- lieux :
  - continent
  - pays
  - ville
- paramètres mesurables
- observations
- instants

2.1.2 Peuplement

Ensuite on a ajouté des propriétés aux classes précédentes pour mieux les décrire. Prenons un exemple avant de lister toutes les propriétés utilisées. "Un phénomène est caractérisé par des paramètres mesurables", ce qui veut dire qu'un phénomène pour qu'il soit valide doit être décrit par un paramètres mesurables. Autrement dit, pour décrire la pluie, il va falloir y associer une pression atmosphérique ou la quantité d'eau tombée ou autres. Ainsi, nous avons détaillé chacune de nos classes pour éviter tout ajout de valeur erroné ou non correspondante. Nous avons souligné toutes les propriétés utilisées.

- un phénomène est caractérisé par des paramètres mesurables
- un phénomène a une durée en minutes
- un phénomène débute à un instant

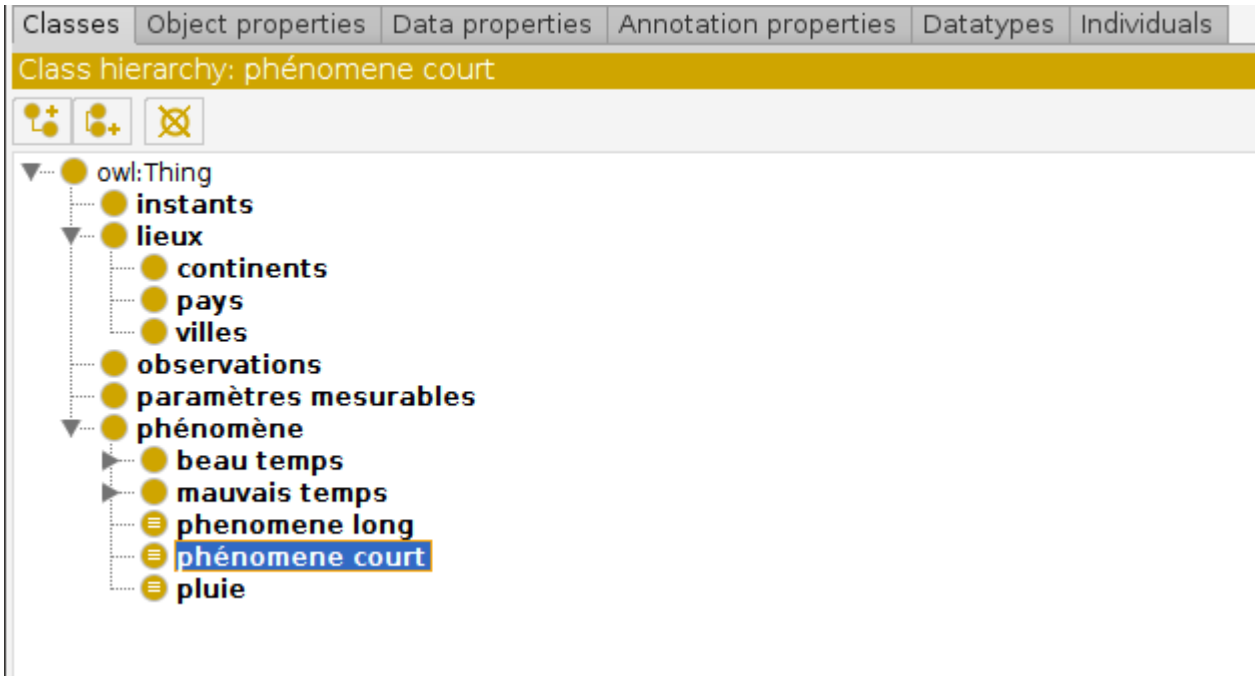

FIGURE 1 – Définition des différentes classes de notre ontologie

— un phénomène finit à un instant
— un instant a un timestamp de type wsd :dateTumeStamp
— un phénomène a pour symptôme une observation
— une observation météo mesure un paramètre mesurable
— une observation météo a une valeur pour laquelle vous ne représenterez pas l'unité
— une observation météo a pour localisation un lieu
— une observation météo a pour date un instant
— un lieu peut-être inclus dans une autre lieu
— un lieu peut inclure un autre autre lieu
— un pays a pour capitale une ville

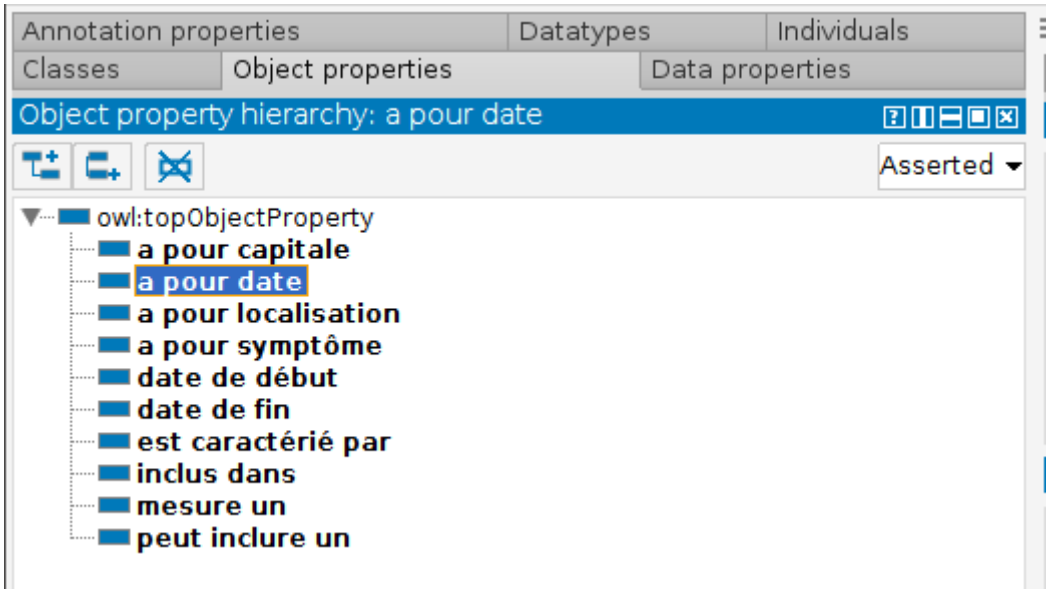

FIGURE 2 – Définition des différentes propriétés de notre ontologie

Dans la liste ci-dessus, il y a une subtilité que vous avez peut-être remarqué. Nous avons mis en même temps, les "Objects Properties" et les "Data Properties". Ces deux paramètres sont différents par rapport à leur description, un "Objects Properties" permet de décrire une classe comme un phénomène et "Data Properties" permet de décrire une donnée comme le timestamp. Nous avons voir dans la troisième partie l'importance de bien définir les données pour éviter toutes erreurs.

Lors ce que l'on run le raisonneur Hermit, le programme va compléter nos classes par les properties que l'on a remplis précédement. Il vérifie aussi la concordance des valeurs rentrées avec les propriétés en question, ce qui nous simplifie la tâche mais aussi évite tout erreur surtout quand nous avons un grand nombre de propriété à vérifier / traîter.

2.2 L'ontologie lourde

2.2.1 Conception

Dans cette partie, nous avons créer une description de nos éléments plus lourdes afin d'avoir et d'accepter des données plus précises.

- Toute instance de ville ne peut pas être un pays
- Un phénomène court est un phénomène dont la durée est de moins de 15 minutes
- Un phénomène long est un phénomène dont la durée est au moins de 15 minutes
- Un phénomène long ne peut pas être un phénomène court
- La propriété indiquant qu'un lieu est inclus dans un autre a pour propriété inverse la propriété indiquant qu'un lieu en inclue un autre.
- Si un lieu A est situé dans un lieu B et que ce lieu B est situé dans un lieu C, alors le lieu A est situé dans le lieu C
- A tout pays correspond une et une seule capitale
- Si un pays a pour capitale une ville, alors ce pays contient cette ville
- La Pluie est un Phénomène ayant pour symptôme une Observation de Pluviométrie dont la valeur est supérieure à 0.

Nous allons ensuite remplir avec des valeurs pour voir si nos connaissances sont bonnes. On fera un test avec le raisonneur de Protégé.

2.2.2 Peuplement

Voici les tests que l'on va effectuer :

- La France est située en Europe
- Paris est la capitale de la France
- La Ville Lumière est la capitale de la France
- Singapour est une ville et un pays

On peut constater que la Ville Lumière est devenue une sous-propriété de Paris car il n'est possible d'avoir seulement une capitale pour un pays et d'après la propriété la conception 2.2.1. Néanmoins, on constate que Toulouse s'est vu attribuer la même spécificité dû à notre façon de déclarer les propriétés. Ainsi, on peut en déduire que l'ontologie lourde peut aussi affiner le classement des valeurs et les réorganiser si besoin. Celà va être utile dans la partie 3, lorsque l'on importe des dates.

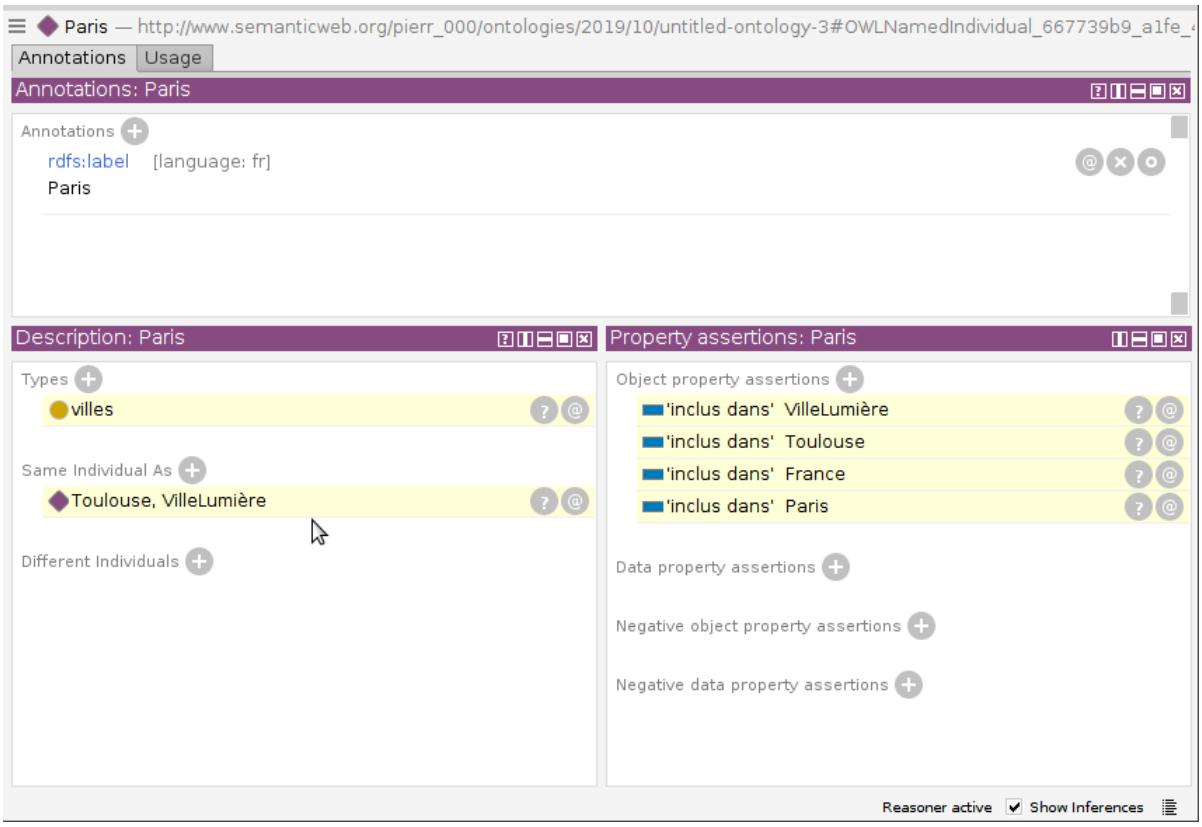

FIGURE 3 – Application du raissoneur sur la classe Paris

3 Exploitation de l'ontologie

Par rapport aux données brutes issues du dataset, l'ontologie joue le rôle de modèle de donnée représentant un ensemble de concepts d'un domaine et constitue des relations entre ces concepts.

Comme pour la manipulation via Protégé, l'API fait une distinction entre le label et l'URI : En effet, l'URI est un identifiant permettant de distinguer de manière unique un objet constituant une ontologie. Alors que le label peut représenter deux objets avec un nom identiques mais qui non pas la même signification. (par exemple, si nous choisissons un label "avocat", nous ne pouvons pas savoir si ce label se réfère au fruit ou à la profession). On a pu déduire que...

Ci-joint, le code. Nous avons développé, à partir d'une base de code donnée,un outil pour transformer des données CSV issues d'un open data mis en ligne par la ville d'Aarhus, au Danemark, en dataset RDF.
Cet outil est structuré avec le pattern MVC (Model, view, controller) :

- Model : c'est le coeur du pattern, la où se trouve la donnée, il se commpose de plusieurs objets java (décrit ci-aprrès) qui effectue des traitements sur la donnée de manière totalement transparente pour l'utilisateur.
- Controller : est un objet java qui permet de faire le lien entre la vue et le modèle lorsqu'une action utilisateur est intervenue sur la vue. C'est cet objet qui aura pour rôle de contrôler les données.
- view : l'interface IHM (terminal,...)

Code décrivant les méthodes du Modèle :

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

Listing 1 – Code pour CreatePlace

Le code ci-dessus nous permet de créer une instance qui porte le label "Lieu" sur Protege. Nous récupérons le nom que nous voulons donner à notre nouvelle instance ainsi que l'URI qui représente le chemin vers notre future instance. Nous préférons la distinguer grâce à l'uri et non un label pour la simple est bonne raison que l'URI est unique alors qu'un label peut se rapporter à plusieurs Instance.
Par exemple, si je prends un label "avocat". Sémantiquement, nous ne pouvons pas distinguer si "avocat" se réfère au fruit ou au métier.

```
public String createInstant(TimestampEntity instant) {

        for(String instance :
        ↪  this.model.getInstancesURI(this.model.getEntityURI("Instant").get(0)))
        ↪  {
                if(this.model.hasDataPropertyValue(instance,
                ↪  this.model.getEntityURI("a pour timestamp").get(0),
                ↪  instant.getTimeStamp())) {
                        return null; //already exists
                }
        }
        String instantClassURI = this.model.getEntityURI("Instant").get(0);
        String URI = this.model.createInstance("instant "+instant.getTimeStamp(),
        ↪  instantClassURI);
        String propertyURI =  this.model.getEntityURI("a pour timestamp").get(0);
        this.model.addDataPropertyToIndividual(URI, propertyURI,
        ↪  instant.getTimeStamp());
        return URI;
}
```

Listing 2 – Code pour createInstance

La fonction ci-dessus a pour but comme son nom l'indique de créer un Instant si celui-ci n'est pas déjà créer dans le passé.

```
public String getInstantURI(TimestampEntity instant) {

        String instantClassURI = this.model.getEntityURI("Instant").get(0);
        String propertyURI =  this.model.getEntityURI("a pour timestamp").get(0);

        for(String InstantIndivid: this.model.getInstancesURI(instantClassURI)){
        ↪  //parcourt de tout les URI qui existent
                if(this.model.hasDataPropertyValue(InstantIndivid, propertyURI,
                ↪  instant.getTimeStamp())){
                        return InstantIndivid;
                }
        }

        return null;
}
```

Listing 3 – Code pour getInstantURI

Le code ci-dessus permet d'implémenter la méthode qui parcourt la liste des URIs de chaque instant car nous savons que l'URI designe de manière unique un instant. Si l'URI est trouvé l'instant existe et est retouné sinon la méthode renvoie null.
Cette méthode est appelée dans pratiquement toutes les autres méthodes du modèles.

```
public String getInstantTimestamp(String instantURI){
        String propertyURI =  this.model.getEntityURI("a pour timestamp").get(0);

        for(List<String> propTuple : this.model.listProperties(instantURI))
        {
                if(propTuple.get(0).equals(propertyURI))
                {
                        return propTuple.get(1);
                }
        }
        return null;
}
```

Listing 4 – Code pour getInstantTimestamp

Cette méthode est similaire à celle du dessus sauf quel renvoie la date de création de l'instant si l'uri de celui-ci existe.

```
public String createObs(String value, String paramURI, String instantURI) {

        String obsURI = this.model.getEntityURI("Observation").get(0);
        String obsURIInstance = this.model.createInstance(instantURI+value,
        ↪  obsURI);
        String uri_dataValue = this.model.getEntityURI("a pour valeur").get(0);
        String uri_instantProp = this.model.getEntityURI("a pour date").get(0);
        String prop_param = this.model.getEntityURI("mesure").get(0);
        String sensorURI =
        ↪  this.model.whichSensorDidIt(getInstantTimestamp(instantURI),
        ↪  paramURI);


        this.model.addDataPropertyToIndividual(obsURIInstance, uri_dataValue,
        ↪  value);
        this.model.addObjectPropertyToIndividual(obsURIInstance, prop_param,
        ↪  paramURI);
        this.model.addObjectPropertyToIndividual(obsURIInstance, uri_instantProp,
        ↪  instantURI);
        this.model.addObservationToSensor(obsURIInstance, sensorURI);
        return obsURIInstance;
}
```

Listing 5 – Code pour createObs

Ci-dessus la méthode qui permet de créer, de donner une valeur à une observation et de lier une observation à un capteur (objet) et s'assure que l'observation vient bien du capteur selectionné.

Code décrivant la méthode que nous avons écrite sur le Contrôleur.

```
public void instantiateObservations(List<ObservationEntity> obsList,String
↪  paramURI) {

        Map<String, String> instantsMap = new HashMap<String, String>();
        // For each element of the list, create its representation in the KB
        int i =0;
        for(ObservationEntity obentity : obsList)
        {
                i++;
                System.out.println("Instantiating observation "+i);
                String instantURI;
                if(!instantsMap.containsKey(obentity.getTimestamp().
                ↪  getTimeStamp()))
                {
                        instantURI = this.customModel.createInstant(obentity.
                        ↪  getTimestamp());
                        instantsMap.put(obentity.getTimestamp().getTimeStamp(),
                        ↪  instantURI);
                }
                else
                {
                        instantURI = instantsMap.get(obentity.getTimestamp().
                        ↪  getTimeStamp());
                }
                this.customModel.createObs(obentity.getValue().toString(),
                ↪  paramURI, instantURI);
        }

}
```

Listing 6 – Code pour instantiateObservations

Le controller a pour rôle de lancer les méthodes du modèle. La fonction ci-dessus se trouve dans le dossier "contrôleur" et associe chaque observation de la liste des observations, placée en paramètre, à son instant quel créer si ce dernier n'existe pas.

4 Conclusion

Ces Travaux pratiques nous ont illustré le lien qui peut y avoir entre les objets en fonction des propriétés que nous leur atttribuons. En effet, le web sémantique structure les données formant une ontologie au travers d'un raisonnement qui se base sur les labels et les propriétés qui sont attachés à un objet. Ainsi, nous avons pu observer ces différents raisonnement au travers des observations que le logiciel nous propose. Ces raisonnements sont entièrement liés aux labels et aux propriétés que nous avons choisis.
En outre, nous en sommes venus à réfléchir au rôle que le web sémantique peut jouer au sein de l'Iot.
Ainsi, le web sémantique est une solution potentielle au problème d'interopérabilité que nous rencontrons dans l'internet des objets. Lorsque nous essayons de décrire un objet en service, cet objet peut possèder sa propre API. Ainsi deux objets similaires mais de marques différentes peuvent possèder des différences.
La sémantique apporte une solution à ce problème en rendant l'interface du service abstraite en la sémantisant ce qui permet non plus d'accéder à ce service par son nom mais par une description porteuse en lien avec ses propriétés. Le web sémantique permet par des ontologies de décrire le système IoT en lui même plutôt que sur le phénomène observé.
En général, les réseaux IoT sont amenés à être dynamique car ils sont modifiés régulièrement ou ils s'agrandissent régulièrement, avec l'ajout ou le retrait d'objet. Il peut être difficile de gérer manuellement ces réseaux. Ainsi, les constructeurs peuvent, en liant une "datasheet" à un objet, automatiser la gestion de ces réseaux en utilisant la sémantique. Autrement, nous pouvons également à l'aide de la sémantique, faire le lien entre un objet et son observation sans forcément connaitre la nature de cette observation.

List of source codes