Rapport de TP ISS: WEB Sémantique

Pierre Prie
Axel Bayle

January 24, 2019

1 Introduction

L'objectif de ces TPs est de nous faire manipuler la notion d'ontologie, dans le but de comprendre son fonctionnement général et ses enjeux. Le logiciel Protégé que nous avons utilisé nous donne la possibilité d'utiliser un outil appelé "raisonneur" pour tester et valider ou non notre conception et notre logique.

Notre cas se porte sur le développement d'une ontologie pour traiter des phénomènes météorologiques. Dans cet optique on doit traiter plusieurs types de données: lieux, instants, variables mesurées , mesure ...

2 Création de l'ontologie

2.1 L'ontologie légère

La première étape est de créer une ontologie légère, dans celle-ci on va créer les classes que l'on va manipuler ainsi que les propriétés majeures de ces classes.

2.1.1 Conception

Classes : Voici une capture d'écran montrant les différentes classes et sous classes de notre ontologie. Par exemple, on remarque que la classe lieu a trois sous classes: continent, pays et ville.

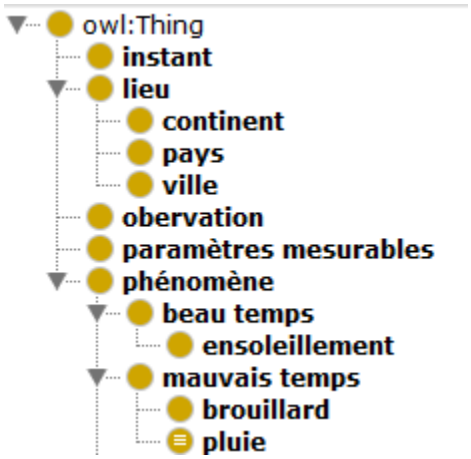

Figure 1: Les classes et sous-classes de l'ontologie

Propriétés : Dans ce type de modèle, les propriétés ne sont pas définies au même moment que les classes. En effet, on crée les propriété à part en définissant les classes manipulées et c'est la propriété qui va définir le type des objets lorsqu'elle est invoqué. C'est à dire, si une propriété dit : " un pays contient une ville" et si on utilise cette propriété de la sorte : " A contient B" alors on peut en déduire que A est un pays et B une ville.

On distingue deux types de propriétés :

- les datas properties : elles lient un objet avec un littéral.
- les object properties : elles associent des objets ensemble au travers d'une propriété.

Il faut par ailleurs noter qu'il n'y a pas de solutions types, en effet, selon la conception que l'on fait d'un même problème on peut avoir une propriété en data property ou en object property. Tout dépend de l'usage et du traitement qui vont être fait sur cette donnée.

Voici un exemple de data property que nous avons implémenté dans notre conception. Cette propriété nous dit qu'un objet Phénomène possède le champs 'a une durée en minutes' et que la donnée dans ce champs est de type integer.

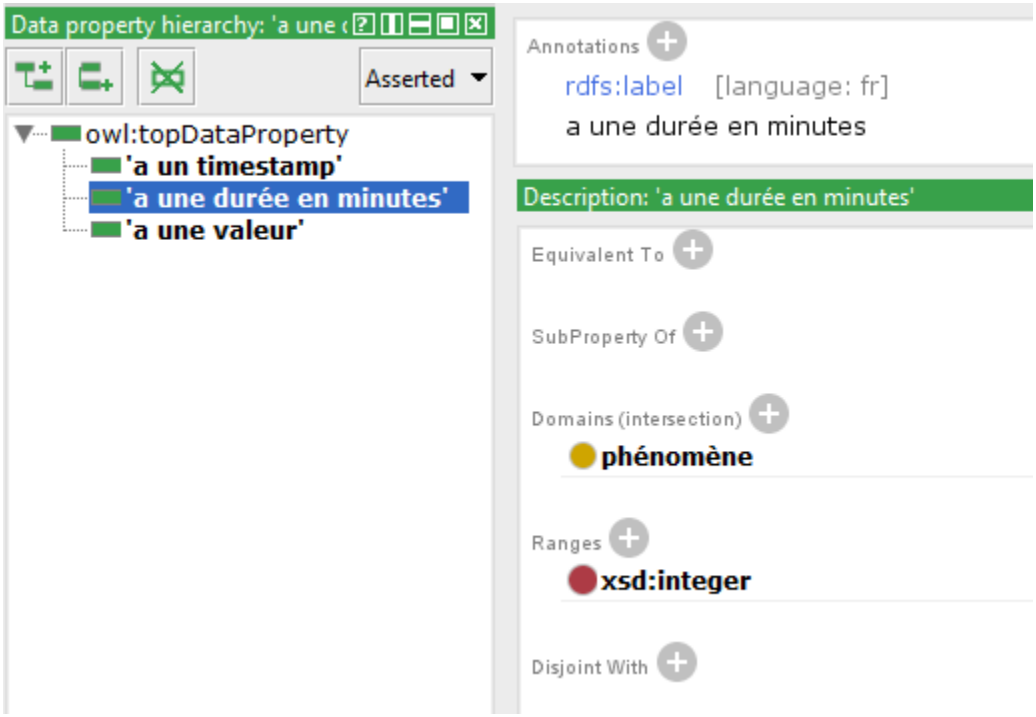

Figure 2: Les Data property de l'ontologie, exemple de 'a une durée en minutes'

Voici un exemple d'object proprerty que nous avons implémenté dans notre conception. Ici, l'object property "a pour date" est défini de la manière suivante: Elle possède un domaine "Observation" et une range "Instant", de cette manière on défini qu'une observation météo est faite à un instant. L'object property possède deux label, "a pour date" et "est faite à un", il est donc possible de l'appeler par l'un ou l'autre de ces noms.

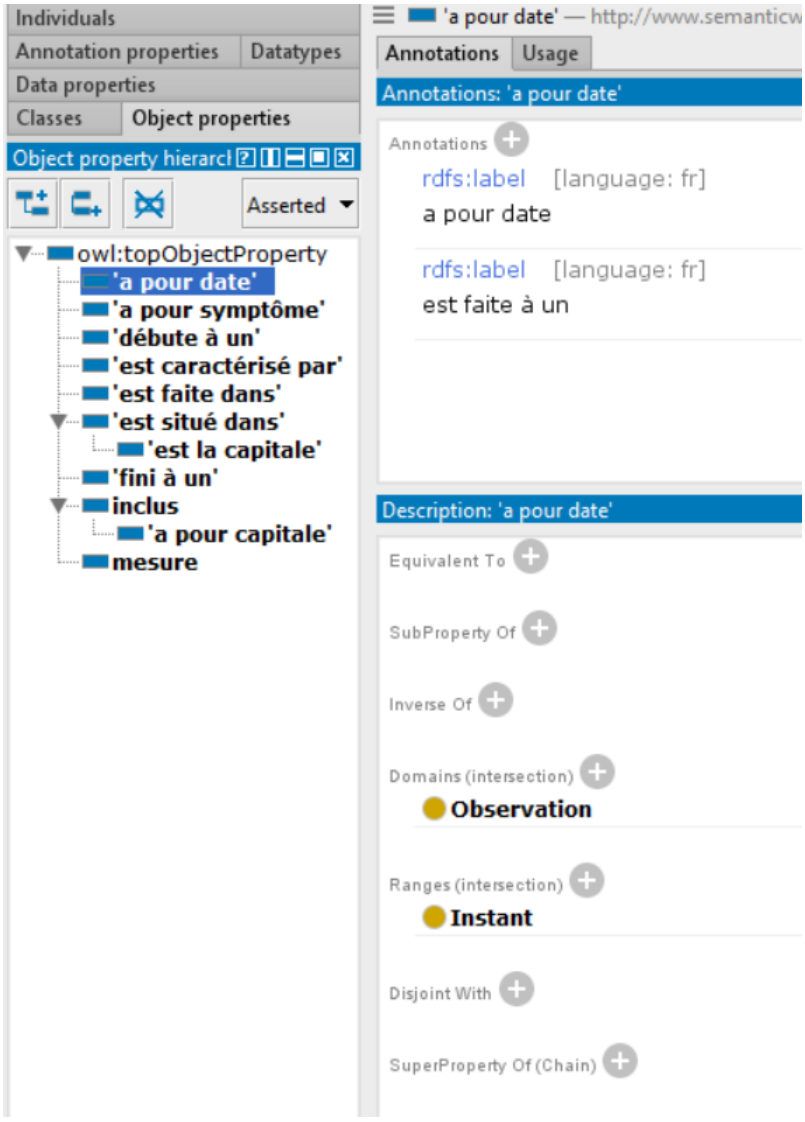

Figure 3: Les objects property de l'onthologie, exemple de 'a pour date'

2.1.2 Peuplement

Une fois le modèle décrit il vient l'étape de peuplement de l'ontologie. Par exemple on vient créer des individus : température, l'hygrométrie, pluviométrie, pression atmosphérique, vitesse du vent et force du vent comme des instances de la classe paramètres mesurables.

Sur cette capture on voit que l'individu force du vent est de type paramètres mesurables et que c'est le même individu que celui possédant le label vitesse du vent.

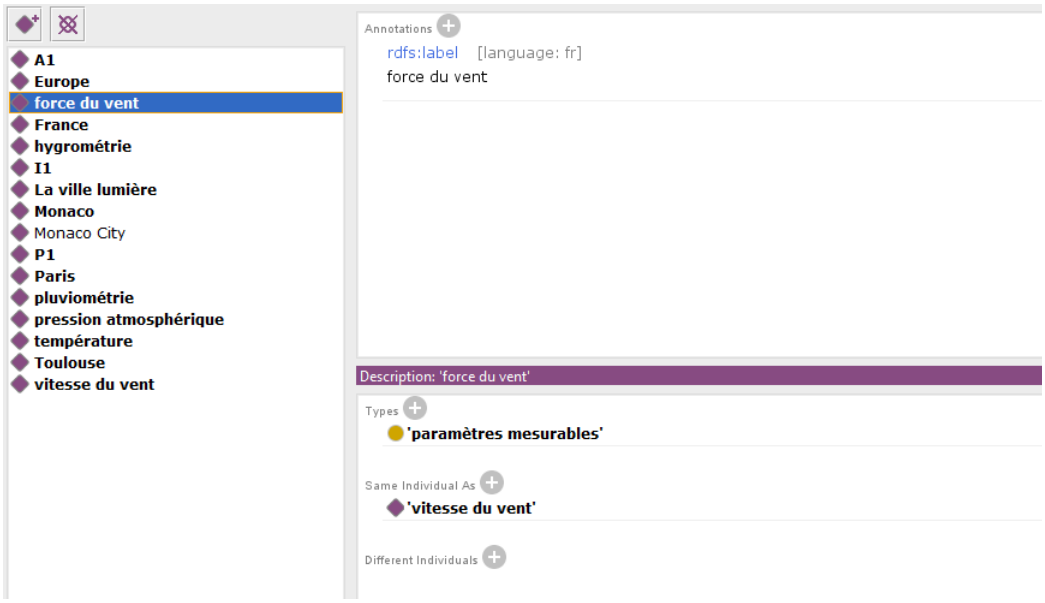

Figure 4: Les individus de l'ontologie, exemple de l'individu 'force du vent'

Une fois les individus crées on peut utiliser un raisonneur pour vérifier la justesse de notre création et d'éventuelles conflit de définition. Une autre fonction du raisonneur est de nous aider à définir les individus. Par exemple si nous avons défini l'individu France et l'individu Paris (sans leur donner de type) et que nous les avons associé avec la propriété "France a pour capitale Paris" alors le raisonneur comprendra et suggérera que France est un pays et Paris une ville.

2.2 L'ontologie lourde

L'objectif de cette partie est de définir plus spécifiquement notre ontologie en détaillant nos définitions notamment au niveau des classes et des objects properties.

2.2.1 Conception

Dans cette partie nous avons notamment défini la sous-classe "Pluie" de "Mauvais temps", elle même sous-classe de "Phénomène". En utilisant la syntaxe de Manchester nous avons pu spécifier un équivalent à cette classe "Pluie". Ainsi dans notre cas, toute observation de phénomène mesurant une pluviométrie et ayant pour valeur un float positive est assimilable à de la "Pluie". Il nous a également été possible de définir des sous-classes disjointes entre elles. Dans notre cas, "Pays" est disjoint de "ville" et "continent" par exemple.

Cette capture montre commet la sous-classe "Pluie" a été défini.

Nous avons également spécifié nos object properties. Nous avons par exemple défini une nouvelle object properties "inclus" comme étant l'inverse de "est situé dans". Nous avons également créer "est la capitale de" comme sous-object properties de "est situé dans" et "a pour capitale" comme sous-object properties de "inclus". Ces deux sous-object properties sont donc également inverses l'une de l'autre. Finalement un lieu

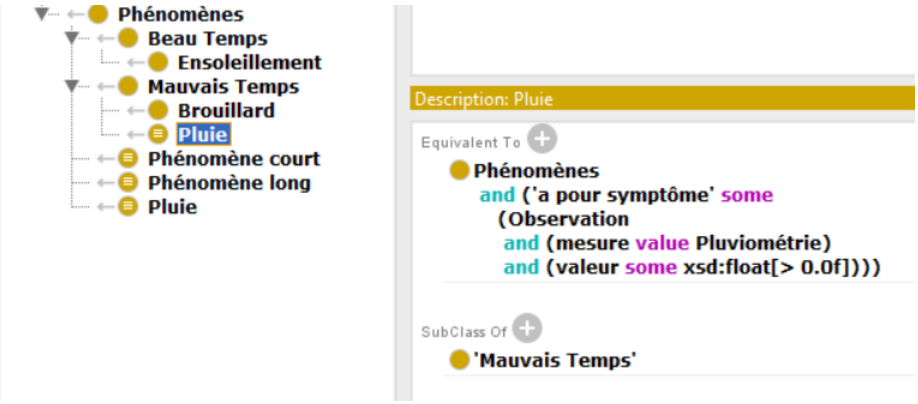

Figure 5: Exemple de la définition de la sous-classe ”Pluie” avec la syntaxe de Manchester

A peut donc avoir la propriété ” est inclus dans” un lieu B et en ”être la capitale”. De cette manière le lieu B obtient automatiquement les propriétés ”inclus” A et ”a pour capitale” A.

On peut aussi ajouter un caractère d'unicité à des propriété, c'est ce que l'on a fait pour ”a pour capitale”. De cette manière un pays ne peut avoir qu'une seule capitale.

2.2.2 Peuplement et validation par le raisonneur

Quand on définit à la fois Paris et La ville lumière comme capitale de la France, le raisonneur sait que paris est une ville et la France, un pays, mais il sait aussi que La ville lumière est une ville. Cependant le raisonneur va plus loin, nous avons spécifié l'unicité de la propriété ”a pour capitale” et, à ce titre, la France ne peut avoir deux capitales, c'est grâce à cela que le raisonneur déduit que Paris et La ville lumière sont un seul et même individu.

Si maintenant nous définissons une ville Toulouse et que l'on spécifie que Toulouse et Paris ne sont pas le même individu et que l'on essaye de définir Toulouse comme capitale de la France le raisonneur nous soulève une erreur car il y a violation de la règle d'unicité de la propriété.

Grâce au raisonneur on a pu aussi relever un défaut de conception: précédemment nous avons spécifié les ensemble Ville, Pays et Continent comme disjoint. Cependant si on essaye de définir un individu comme Monaco qui est à la fois ville et pays, le raisonneur ne va pas accepter car nous essayons de créer un individu à la jonction de deux ensembles disjoints. Il nous est donc nécessaires de créer deux entités différentes, par exemple Monaco et Monaco-city.

Le raisonneur est bien un outils puissant qui permet de caractériser au mieux nos données, on peut le noter à nouveau avec cette exemple, ou les lignes en jaunes correspondes à des déductions automatiques du raisonneur :

3 Exploitation de l'ontologie

On peut utiliser un moyen plus systématique de peupler une ontologie. En effet, on peut recueillir des grandes quantités de données et cela serait long et inintéressant de devoir rentrer chaque individu à la main.

Pour ce faire on peut définir des fonctions, dans notre cas ce sont des fonctions écrites en Java, qui vont permettre de traiter un fichier contenant les données en format JSON ou CSV pour créer des individus et les rentrer dans la base de données.

Le point majeur dans le traitement des données est de comprendre comment les objets (individus, classes et propriétés) sont gérés. On pourrait croire que ce sont les noms ou labels de ceux-ci qui les définissent, alors que c'est en fait l'URI qui les définit. Chaque objet possède sont propre URI et, c'est ce String, qui va permettre

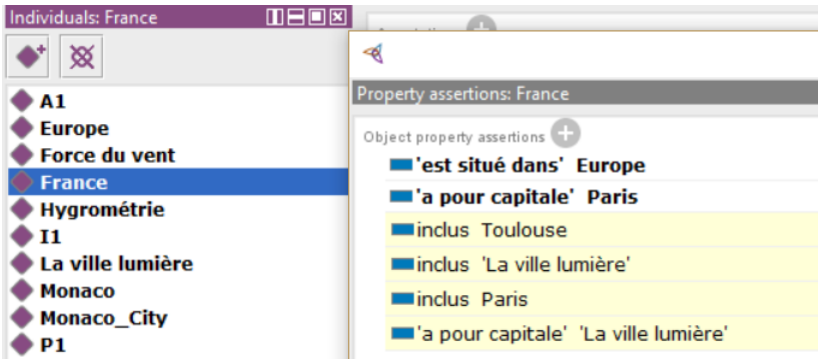

Figure 6: Exemple de déduction du raisonneur

d'appeler et manipuler l'objet. De cette manière on évite tout problème éventuel de confusion ou de conflit qui pourrait avoir lieu si deux objets avaient le même label.

L'analogie la plus représentative est celle de valeur et d'adresse. Par exemple, en C, si une variable a pour nom 'x' et une valeur '3' et que l'on veut utiliser et modifier cette variable dans une fonction on ne va pas utiliser 'x' pour y faire référence mais on va utiliser l'adresse de la variable qui est unique.

Ci dessous une des fonctions que nous avons codées, elle prend en paramètre l'URI d'un instant et retourne la valeur du timestamp de cet instant, ou null si il n'y a pas d'individu instant défini par cet URI.

```
@Override
public String getInstantTimestamp(String instantURI) {
    // TODO Auto-generated method stub
    String r = null;
    if (model.isOfType(instantURI, model.getEntityURI( label: "Instant").get(0))) {
        //System.out.println("if");
        List<List<String>> toto = model.listProperties(instantURI);
        for (int i = 0; i < toto.size(); i++) {
            if (toto.get(i).get(0) == model.getEntityURI( label: "a pour timestamp").get(0)) {
                r = toto.get(i).get(1);
                //System.out.println("for");
            }
        }
    } else {
        r = null;
    }
    return r;
```

Figure 7: Code de la fonction GetInstantTimestamp