INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES TOULOUSE

WEB SEMANTICS

Rapport sur les travaux pratiques

Module Data and Application

SOUMIS PAR

Yacine BENCHEHIDA 5ISS-A2
Hugo LE BELGUET 5ISS-A2
Sami BEYAH MSIoT-A2

SOUS LA DIRECTION DE

Prof. **Nicolas SEYDOUX**

Année scolaire : 2021-2022
Janvier, 2021

Lien Git : https://github.com/sami-mkb/Web_Semantic

Sommaire

INTRODUCTION

Dans le cadre de notre **5ème année** de formation à l'INSA de Toulouse, spécialité **Innovative Smart Systems (ISS),** nous avons suivi le cours de Traitement des Données Sémantiques de l'Unité de Formation Analyse et traitement des données, applications métier. C'est à la suite de deux travaux pratiques ainsi que des cours magistraux que nous avons produit un rapport synthétisant tous les concepts que nous avons appris.

L'**objectif** de ces travaux pratiques était de :

- **Concevoir une ontologie** à l'aide de Protégé constituant un modèle de données dans le domaine des observations météorologiques,
- **Développer un outil basé** sur la libraire Java Jena permettant d'intégrer à notre ontologie des données CSV issues d'un open data set mis en ligne par la ville d'Aarhus, au Danemark

TP1 – Création de l'ontologie

Ce TP a pour objet de nous faire **manipuler concrètement la notion d'ontologie**, de voir les aspects principaux et les déductions que peut faire un raisonneur à différentes étapes du développement de l'ontologie.

1. Conception de l'ontologie légère

Dans un premier temps, nous allons travailler sur **la conception de l'ontologie légère** et pour cela on va commencer par créer les classes appropriées, en accord avec les connaissances suivantes :

1. Le beau temps et le mauvais temps sont deux types de phénomènes.

2. La pluie et le brouillard sont des types de phénomènes de mauvais temps, l'ensoleillement est un type de phénomène de beau temps

3. Les paramètres mesurables sont une classe de concept, ainsi que les instants et les observations

4. Une ville, un pays et un continent sont des types de lieux

Après analyse, on identifie **5 classes** (Instants, Lieu, Observations, Paramètres mesurables, Phénomènes). La classe ***Lieu*** contient **3 sous classes** que l'on identifie dans la 4ème phrases (Continent, Pays, Ville).

On observe aussi dans la 1ème phrase que **Beau temps et Mauvais temps** sont des sous classes de **Phénomènes**. Pour finir ces deux sous classes ont-elles même 2 sous classes (ensoleillement et brouillard).

Pour créer ces classes dans *Protégé* on se rend dans l'onglet « Entities -> Classes », c'est là qu'elles vont apparaitre.

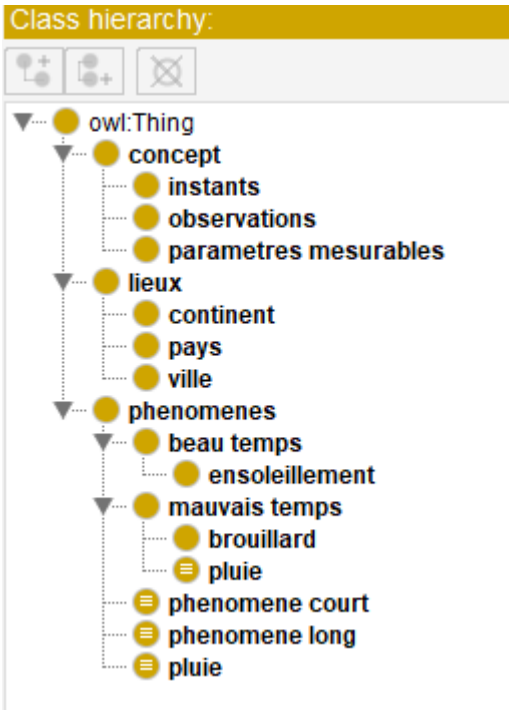


Figure 1 : Classes

Maintenant que nous avons ajouté les diverses classes et sous classes, nous allons passer **aux propriétés et concepts, cela va permettre d'exprimer des relations entre les classes.**

On veut donc exprimer les phrases suivantes :

1. Un phénomène est caractérisé par des paramètres mesurables

2. Un phénomène a une durée en minutes

3. Un phénomène débute à un instant

4. Un phénomène finit à un instant

5. Un instant a un timestamp, de type xsd :dateTimeStamp

6. Un phénomène a pour symptôme une observation

7. Une observation météo mesure un paramètre mesurable

8. Une observation météo a une valeur pour laquelle vous ne représenterez pas l'unité

9. Une observation météo a pour localisation un lieu.

10. Une observation météo a pour date un instant

11. Un lieu peut être inclus dans un autre lieu

12. Un lieu peut inclure un autre lieu

13. Un pays a pour capitale une ville

On procède donc à créer les « *Object Properties* » et « *Data Properties* » dont le résultat apparait comme suit :

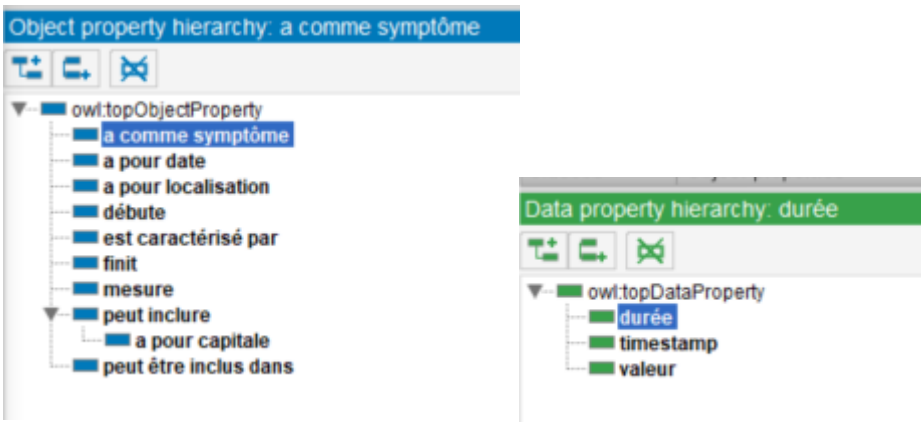


Figure 2 : Object Properties & Data Properties

Plus en détails, on peut voir ici comment on utilise les champs *Domain et Range* pour lier des objets avec une relation, pour « débute » :

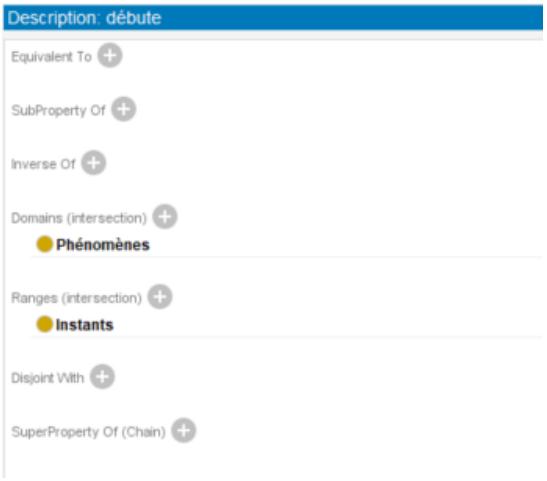


Figure 3 : Object properties début

2. Peuplement de l'ontologie légère

Pour cette partie on va se rendre dans l'onglet « Entities -> Individuals » puis on va remplir les faits décrit dans les phrases suivantes :

1. La température, l'hygrométrie, la pluviométrie, la pression atmosphérique, la vitesse du vent et la force du vent sont des paramètres mesurables (Attention, pas des types de paramètres, mais des instances de paramètres)

2. Le terme temperature est un synonyme anglais de température. Examinez les "Annotations" de l'individu.

3. La force du vent est similaire `a la vitesse du vent

4. Toulouse est située en France. Remarquez que les individus dans cette phrase ne sont pas types : créez Toulouse et France non pas comme une ville et un pays, mais comme des individus sans classe. Comment les classifie le raisonneur ?

5. Toulouse est une ville

6. La France a pour capitale Paris. Ici aussi, Paris est un individu non type

7. Le 10/11/2015 `a 10h00 est un instant que l'on appellera I1 (noté 2015-

11-10T10 :00 :00Z)

8. P1 est une observation qui a mesure la valeur 3 mm de pluviométrie à Toulouse à l'instant I1 (pas besoin de représenter l'unité)

9. A1 a pour symptôme P1

On observe ici le résultat général :

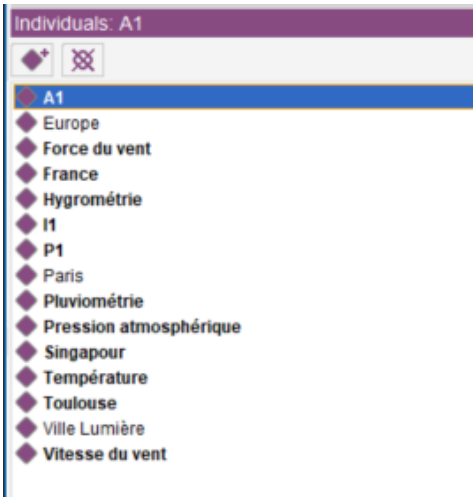


Figure 4 : Individuals

On peut prendre quelques exemples pour illustrer ce que nous avons fait, pour la phrase 1 et 3 voici comment on a créé « force du vent » :

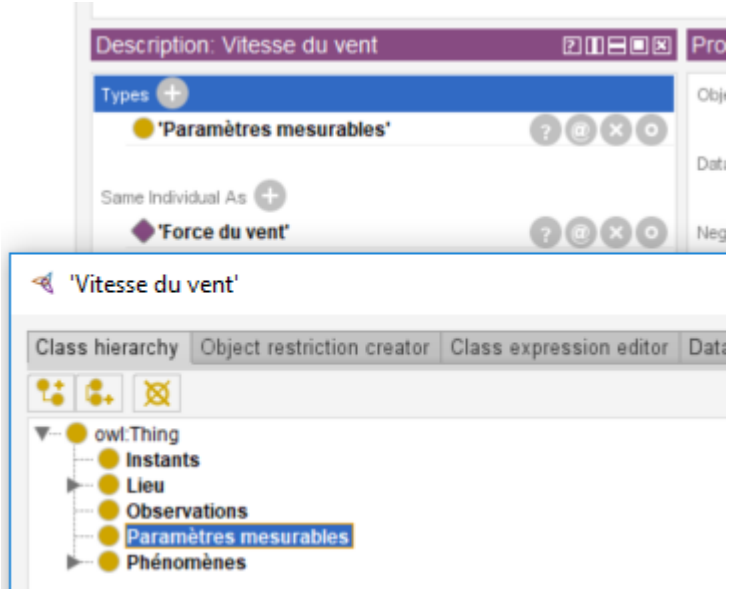


Figure 5 : Force du vent

On a rajouté le type et l'équivalence. Il en va de même pour les autres.

Nous allons maintenant lancer le raisonneur Hermit.

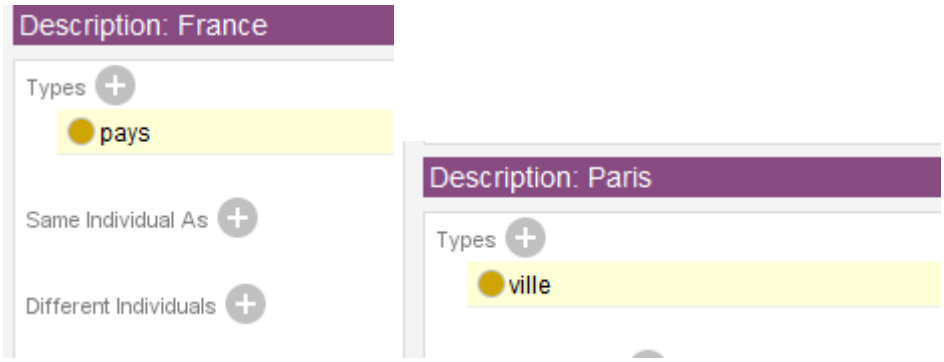


Figure 6 : déduction du raisonneur sur France et Paris

On observe ici qu'il a déduit le type de France et Paris que nous avions intentionnellement rentré sans le spécifier. C'est dû au fait dans France on a précisé « est inclus dans Europe » et « a pour capital Paris », de plus Europe a automatiquement été associé à un type lieux pour les mêmes raisons.

3. Ebauche de l'ontologie lourde

Ici on veut exprimer les connaissances suivantes :

1. Toute instance de ville ne peut pas être un pays

2. Un phénomène court est un phénomène dont la durée est de moins de 15 minutes

— En syntaxe de Manchester : Phénomène that 'a une durée' some xsd:float [< 15]

3. Un phénomène long est un phénomène dont la durée est au moins de 15 minutes

4. Un phénomène long ne peut pas être un phénomène court

5. 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.

6. 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 (utilisez les caractéristiques de la relation)

7. A tout pays correspond une et une seule capitale (utilisez les caractéristiques de la relation).

8. Si un pays a pour capitale une ville, alors ce pays contient cette ville (utilisez la notion de sous-propriété).

9. La Pluie est un Phénomène ayant pour symptôme une Observation de Pluviométrie dont la valeur est supérieure à 0.

— Phénomène that 'a pour symptôme' some (Observation that ('mesure' value Pluviométrie) and ('a pour valeur' some xsd:float [>0]))

Pour réaliser tout cela voici comment on s'y est pris phrase par phrase :

1 – Dans la classe « ville » on utilise *Disjoint with* où on ajoute « Pays »

2 & 3 – Dans la classe phénomène long et court, on rajoute à *« Equivalent to »* les formules spécifiés dans l'énoncé.

4 – De la même manière que pour la phrase 1, on utilise *Disjoint with* dans phénomènes court et long : l'un pour l'autre.

5 – Il s'agit de se rendre dans *« Object Properties »* puis dans « est inclus dans » ici on trouve la propriété *« inverse of »* où on va rajouter *Inclure* pour répondre à l'exigence.

6 – Cette phrase décrit un **principe de transitivité**. Pour la mettre en œuvre, on va donc cocher *« transitive »* dans le même objet que pour la phrase 5.

7 – Cette phrase décrit le **principe de bijectivité**. Pour le mettre en œuvre, on va dans la propriété de l'objet « a pour capital » et on va cocher les caractéristiques *« Functional »* et *« Inverse Functional »,* la première correspond à une notion d'injectivité, la deuxième à la bijectivité.

8 – Pour répondre à cette connaissance, on va mettre *« a pour capital »* en sous propriété de *« Inclure »*

9 – On va compléter la classe Pluie en rajouter le formulaire proposé à *« Equivalent to ».*

4. Peuplement de l'ontologie lourde

On rajoute les éléments suivants dans Individuals pour peupler l'ontologie lourde :

1. La France est située en Europe

2. Paris est la capitale de la France

3. La Ville Lumière est la capitale de la France

4. Singapour est une ville et un pays

Lorsqu'on lance le raisonneur on observe beaucoup de déductions.

Pour commencer à cause de la 4eme phrase on commence avec cette erreur.

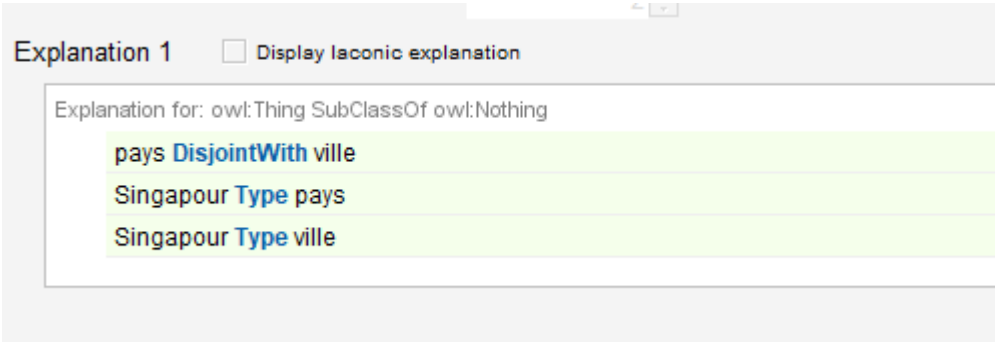


Figure 7 : Erreur raisonneur Singapour

C'est dû au fait qu'on a précisé qu'une ville ne peut pas être un pays. Pour régler ce problème, on est forcé de créer deux individus « Singapour », une ville et un pays.

Un type **« Phénomène »** a été attribué à A1, c'est dû au fait qu'on a précisé qu'il a pour symptôme P1, or P1 a pour propriété mesure pluviométrie et on a précisé que la pluie a pour symptôme une observation qui mesure et qui a pour valeur, pluie étant équivalent de phénomène, voilà pourquoi le raisonneur en a déduit ça.

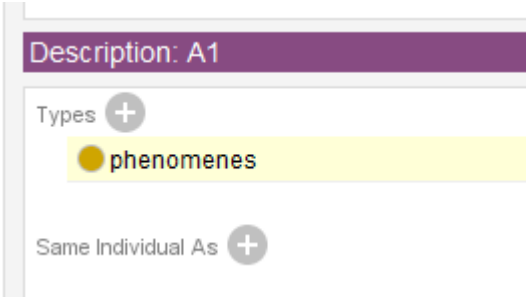


Figure 8 : déduction du raisonneur pour A1

Concernant Paris, le raisonneur a rajouté qu'il s'agit d'une ville mais ça c'était déjà le cas avant, néanmoins on voit aussi maintenant *« est inclus dans France »* et *« Est inclus dans Europe ».*

C'est dû au fait qu'on ait précisé que *Si un pays a pour capitale une ville, alors ce pays contient cette ville* et que la capitale de la France est Paris. De plus on a dit que la France est en Europe et si Paris est en France alors le raisonneur en déduit que Paris est en Europe.

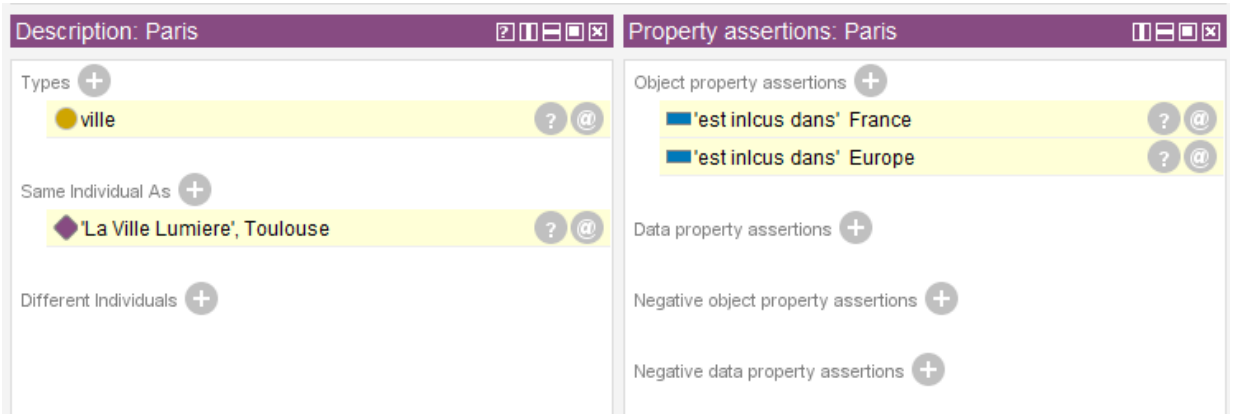


Figure 9 : Déduction raisonneur sur Paris

Finalement, on peut aussi observer *« Same Individual As »,* concernant la ville Lumière : c'est parce qu'on a précisé qu'il s'agit de la capitale de la France, or il ne peut y avoir qu'une seule capitale par pays donc le raisonneur en déduit qu'il s'agit des mêmes individus.

Il en va de même lorsqu'on ajoute *« a pour capital »* Toulouse à l'individu France, ça change également dans l'individu Toulouse qui devient équivalent à Paris et La ville Lumière.

TP2 – Implémentation et Exploitation de l'ontologie

L'objectif de ce TP a été de **manipuler une ontologie à l'aide d'un code source pour construire une application sémantique**. Après avoir développé notre application, nous avons rédigé un jeu de données produit par la ville d'Aarhus, au Danemark.

Ces données sont collectées à partir de capteurs de température, et elles sont stockées dans des fichiers CSV. Nous avons utilisé des tests unitaires pour valider au fur et à mesure notre implémentation.

L'API programmée fonctionne sous le modèle Model-View-Controler (MVC). Son principe est le suivant :

- Le modèle, la vue et le contrôleur sont **trois services distincts**
- Le modèle **contient la majorité des algorithmes**, compilant les données au plus bas niveau du modèle
- La vue s'occupe de **l'affichage des données compilées**, et ore à l'utilisateur la possibilité d'interagir avec le programme
- Le contrôleur fait **l'interface entre la vue et le modèle**, et permet une interaction entre les deux services

Ci-dessous une illustration du **modèle MVC** avec les interactions entre les différents services :

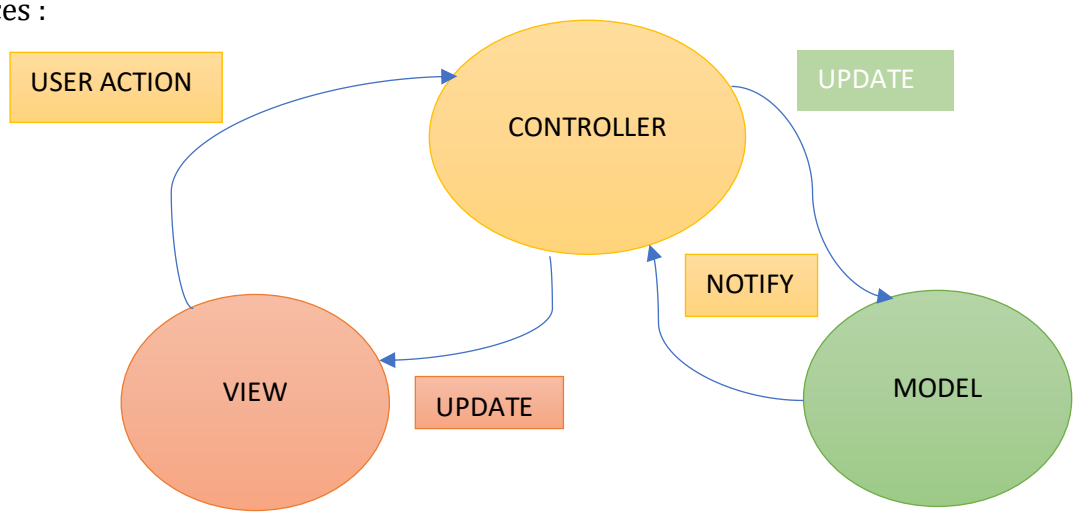


Figure 10 : Le modèle MVC

On a à notre disposition des classes contenant :

- Des classes d'objets et leur URI
- Des méthodes d'instanciation d'objet d'une certaine classe et la mise en relation avec des propriétés

- Des méthodes “parsant” du CSV
- Les squelettes des méthodes à développer
- Les tests pour les méthodes à développer

Modèle

Classe - “DoItYourselfModel”

i) Méthode createPlace

L’objectif de cette méthode est :

- De **récupérer l’URI** de la classe Place via l’appel à la méthode *getEntityURI*
- De **créer l’instance** à partir de l’URI récupéré précédemment via la méthode *createInstance*

Cette fonction, qui prend en paramètre un String, crée **une instance de la classe Lieu** ayant pour label le nom entré en paramètre, et renvoie l’URI de l’instance de classe créée.

Voici le code que nous avons implémenté :

```
@Override
public String createPlace(String name) {
    // Auto-generated method stub
    return model.createInstance(name, model.getEntityURI("Place").get(0));
}
```

Nous avons fait le choix de concevoir et de récupérer le premier indice de la liste retourné par la méthode en question. En effet, on récupère le premier indice de la liste (get(0)) car l’ontologie est conçue de cette façon.

ii) Méthode createInstant

La fonction *createInstant* crée une instance de la classe Java *Instant*, et lui associe une data property “a pour timestamp" ayant pour valeur le timestamp passé en paramètre.

Pour instancier l'objet, nous réutilisons les méthodes *getEntitURI* et *createInstance* mais aussi les méthodes *addDataPropertyToIndividual* et *getInstancesURI* (pour récupérer les instances d'Instant).

```
@Override
public String createInstant(TimestampEntity instant) {
    boolean found=false;
    List<String> liste=model.getInstancesURI(model.getEntityURI("Instant").get(0));
    for(Iterator<String> it=liste.iterator(); it.hasNext();)
    {
        if(model.listLabels(it.next()).get(0).equals(instant.getTimeStamp()))
        {
            found=true;
        }
    }

    if (!found)
    {
    String Uri = model.createInstance(instant.getTimeStamp(), model.getEntityURI("Instant").get(0));
    model.addDataPropertyToIndividual(Uri, model.getEntityURI("a pour timestamp").get(0), instant.getTimeStamp());
    return Uri;
    }

    return null;

}
```

iii) Méthode *getInstantTimestamp*

L'objectif de cette méthode est **de vérifier**, à partir d'un URI, si c'est une instance d'Instant et si oui, nous retournons son *timestamp*. En effet, nous vérifions d'abord :

- si l'URI correspond bien à une Instance (*getInstancesURI*),
- si elle est présente, nous récupérons ses propriétés (*listProperties*) et retournons la Data *properties* liée à la propriété "a pour timestamp"

```
@Override
public String getInstantTimestamp(String instantURI)
{
    List<String> listeInstances=model.getInstancesURI(model.getEntityURI("Instant").get(0));
    if (!listeInstances.contains(instantURI))
    {
        return null;
    }
    else{

        List<List<String>> liste=model.listProperties(instantURI);
        for(Iterator<List<String>> it=liste.iterator(); it.hasNext();)
        {
            List <String>aux=it.next();
            if (aux.get(0).equals(model.getEntityURI("a pour timestamp").get(0)))
            {
                return aux.get(1);
            }
        }

    }
    return null;
}
```

iv) Méthode *getInstantURI*

Cette fonction retourne l'URI de la classe *Instant* ayant pour *timestamp* celui passé en paramètre.

Pour ce faire, nous parcourons les instances de l'objet Instant et vérifions s'ils correspondent au timestamp recherché.

```
@Override
public String getInstantURI(TimestampEntity instant) {
    // TODO Auto-generated method stub
    boolean found=false;

    List<String> liste=model.getInstancesURI(model.getEntityURI("Instant").get(0));
    for(Iterator<String> it=liste.iterator(); it.hasNext();)
    {
        String aux=it.next();
        if(model.hasDataPropertyValue(aux, model.getEntityURI("a pour timestamp").get(0),instant.getTimeStamp()))
        {
            found=true;
            return aux;
        }
    }
    return null;
}
```

v) Méthode *createObs*

La **méthode** « *createObs* »crée une instance de la classe Observation avec les paramètres suivants :

- La valeur passée en paramètre est la mesure effectuée
- L'URI *paramURI* correspond à l'URI du capteur utilisé pour la mesure
- L'URI *instantURI* correspond à l'URI de l'Instant auquel à été prise la mesure et retourne l'URI associé à l'observation créée :

Nous donnons à l'objet un Label étant **la concaténation de l'URI de la mesure physique** et l'URI du timestamp.

Nous lui attachons une date avec (*addObjectPropertyToIndividual*) et une valeur avec (*addDataPropertyToIndividual*). Finalement nous relions cette observation au sensor en utilisant les méthodes *addObservationToSensor* et *whichSensorDidIt*.

```
@Override
public String createObs(String value, String paramURI, String instantURI) {
    // TODO Auto-generated method stub
    String instance=model.createInstance(paramURI.concat(instantURI),model.getEntityURI("Observation").get(0));
    model.addObjectPropertyToIndividual(instance, model.getEntityURI("a pour date").get(0), instantURI);
    model.addDataPropertyToIndividual(instance, model.getEntityURI("a pour valeur").get(0), value);
    model.addObservationToSensor(instance,model.whichSensorDidIt(getInstantTimestamp(instantURI), paramURI));
    return instance;
}
```

Contrôleur

Classe - "DoItYourselfControl"

i) Méthode *instantiateObservations*

La fonction *instantiateObservation* analyse la liste d'observations extraite du dataset et les instancie dans notre base de connaissance. Concrètement, nous utilisons cette méthode pour instancier les objets dans le fichier CSV.

```
@Override
public void instantiateObservations(List<ObservationEntity> obsList,
        String paramURI) {
    int total=obsList.size();
    int count=0;
    for(Iterator<ObservationEntity> it=obsList.iterator(); it.hasNext();)
    {   count ++;
    ObservationEntity aux=it.next();
    String bloup =customModel.createObs(aux.getValue().toString(), paramURI, customModel.createInstant(aux.getTimestamp()));
    System.out.print("Element ");
    System.out.print(count);
    System.out.print(" of ");
    System.out.println(total);

    }

}
```

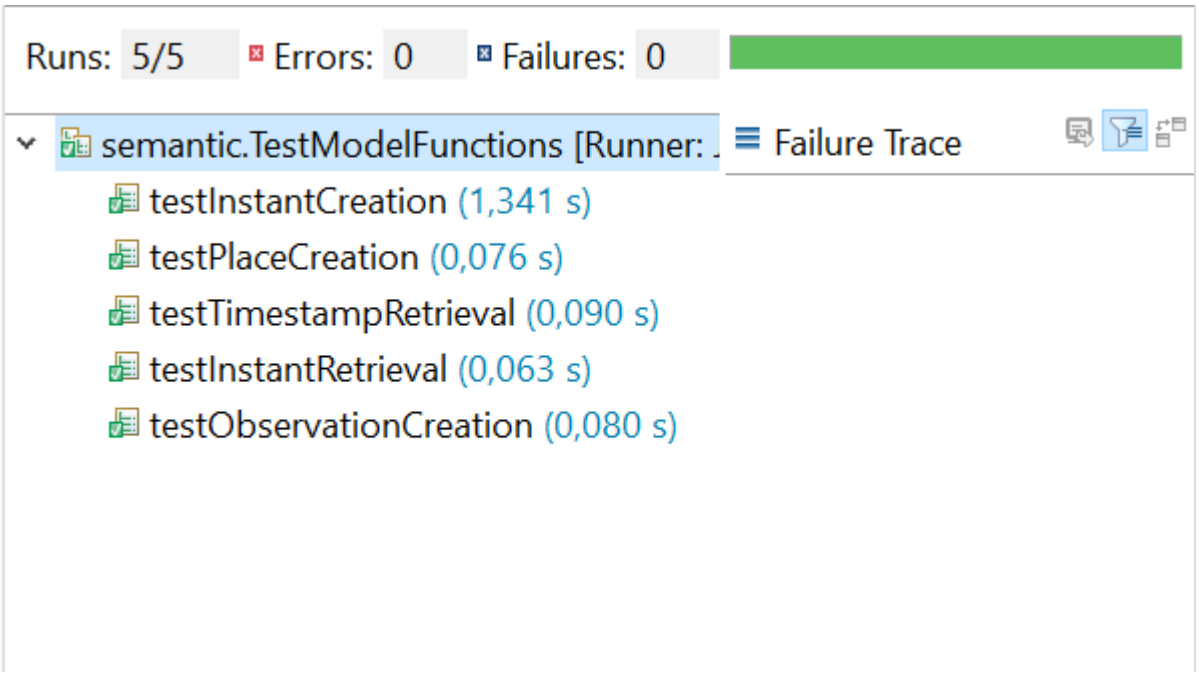
Tests

i) Classe controller

A l'aide des méthodes *parseObservations,* nous parcourons les fichiers CSV. Si le test est bon, cette méthode nous retourne une liste d'objet de la classe "*ObservationEntity*".

La classe "*ObservationEntity*" contient les méthodes qui nous permettent d'extraire les informations nécessaires pour instancier une *Observations* dans notre ontologie.

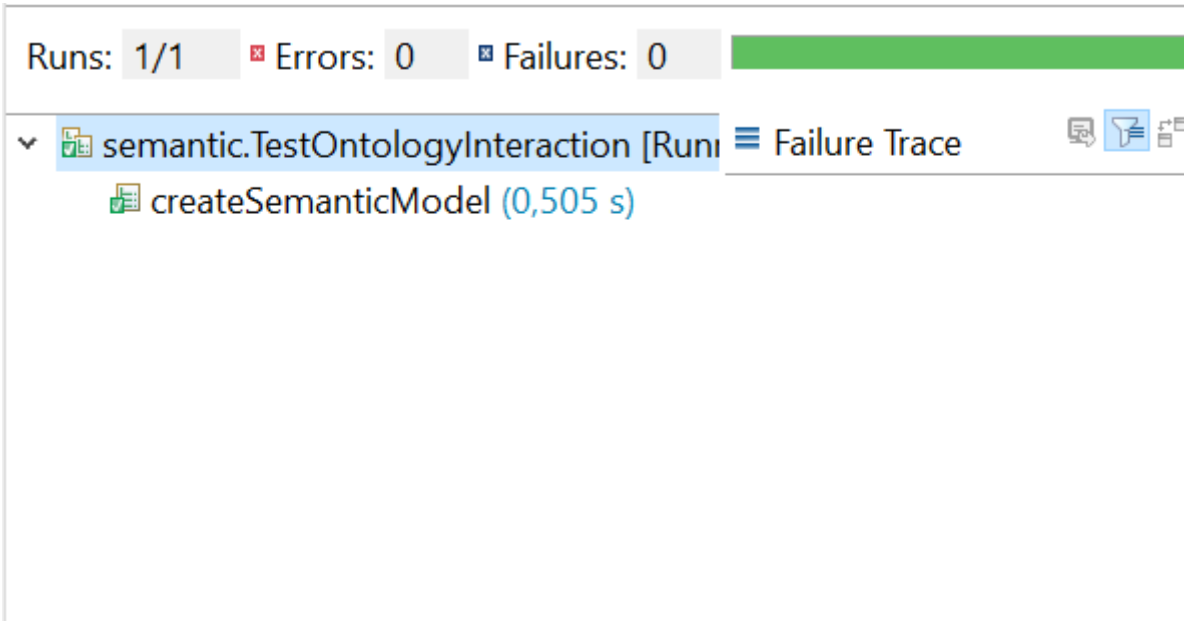
En effet, l'exécution du programme prend beaucoup de temps et peut être optimisée.



Voici, les tests unitaires liés à cette classe validant notre implémentation :

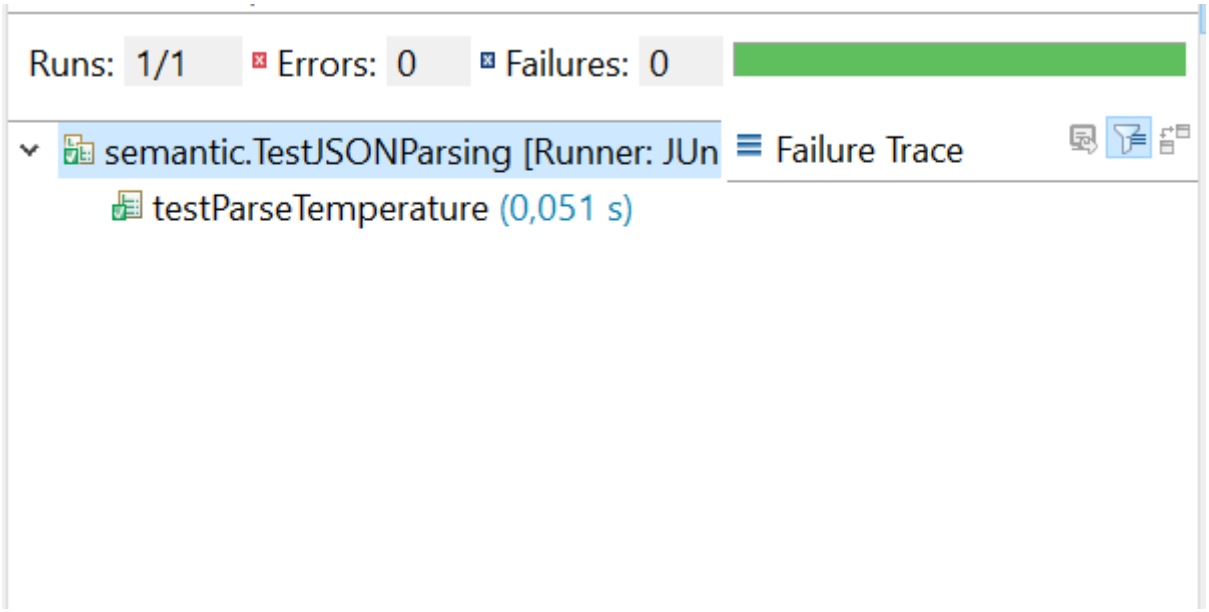
ii) Compatibilité avec Protégé

Ces tests ci-dessous ont pour but de valider la compatibilité de notre implémentation pour



pouvoir l'exporter sur Protégé :

De même, le test ci-dessus permet de valider que notre implémentation est conforme :



CONCLUSION

Ainsi ces travaux pratiques nous ont permis de nous approprier les concepts de Web sémantique.

Le Web sémantique fournit un ensemble de fonctions qui offrent une structure commune à toutes les données du web. Il permet un traitement automatique traitement automatique. Le web sémantique vise à convertir le web actuel, avec ses documents non structurés et semi-structurés, en un "web de données".

De plus, le deuxième travail pratique nous a permis de mettre en application notre background « développement » en Java en implémentant des méthodes dans un code qui nous est fourni au préalable afin de créer notre propre base de connaissance.