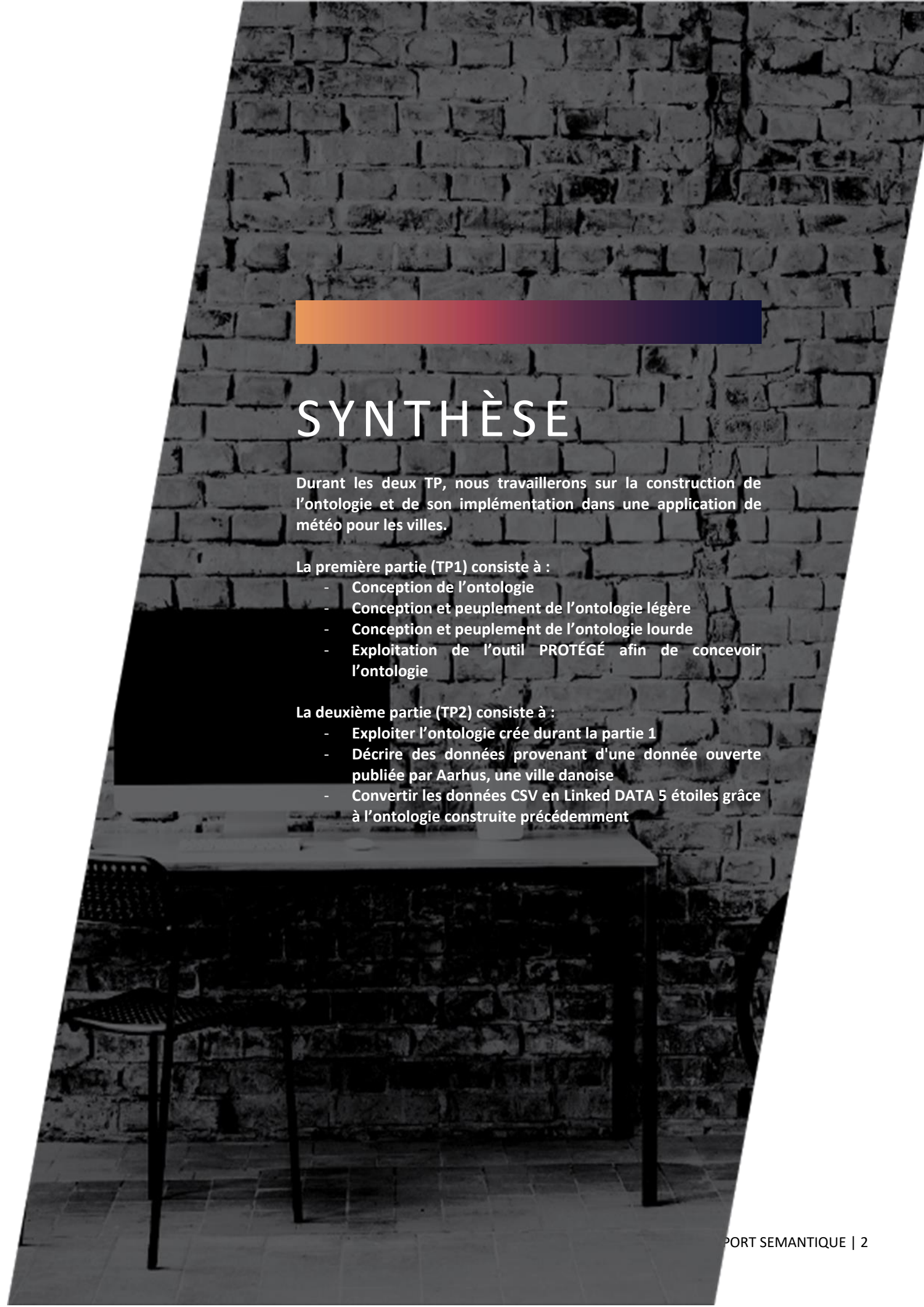
RAPPORT SEMANTIQUE WEB

Florien LEON & Walid KHALED | INSA TOULOUSE 2021/2022

INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES TOULOUSE



SYNTHÈSE

Durant les deux TP, nous travaillerons sur la construction de l'ontologie et de son implémentation dans une application de météo pour les villes.

La première partie (TP1) consiste à :

- **Conception de l'ontologie**
- **Conception et peuplement de l'ontologie légère**
- **Conception et peuplement de l'ontologie lourde**
- **Exploitation de l'outil PROTÉGÉ afin de concevoir l'ontologie**

La deuxième partie (TP2) consiste à :

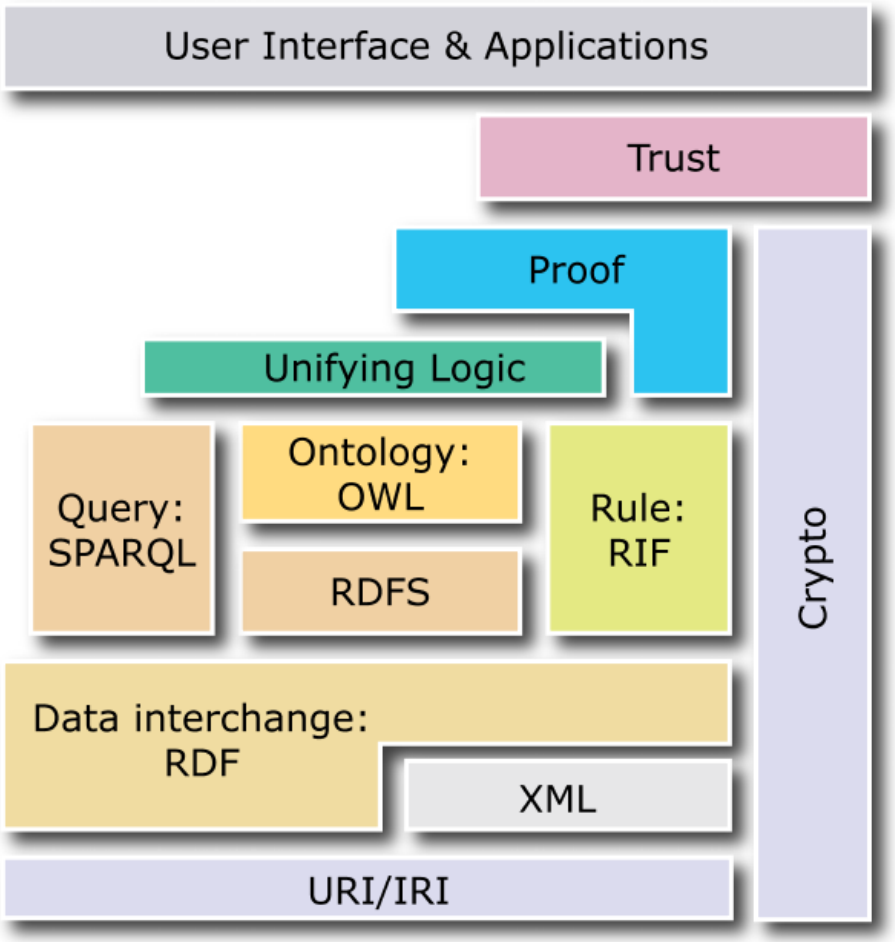
- **Exploiter l'ontologie crée durant la partie 1**
- **Décrire des données provenant d'une donnée ouverte publiée par Aarhus, une ville danoise**
- **Convertir les données CSV en Linked DATA 5 étoiles grâce à l'ontologie construite précédemment**

INTRODUCTION

Le Web sémantique, ou toile sémantique, est une extension du Web standardisée par le World Wide Web Consortium (W3C). Ces standards encouragent l'utilisation de formats de données et de protocoles d'échange normés sur le Web, en s'appuyant sur le modèle Resource Description Framework (RDF).

Le but principal du Web sémantique est d'orienter l'évolution du Web pour permettre aux utilisateurs sans intermédiaires de trouver, partager et combiner l'information plus facilement. Les êtres humains sont capables d'utiliser le Web pour effectuer des tâches telles que trouver le mot Paris pour réserver un livre à la bibliothèque, trouver un plan et réserver son billet de transport. Cependant, les machines ne peuvent pas accomplir toutes ces tâches sans direction humaine, car les pages web sont conçues pour être lues avant tout par des personnes. Le Web sémantique vise à rendre les pages explorables tout aussi bien par l'homme que par la machine.

L'architecture du web sémantique est la suivante :



Une ontologie est une base de connaissance qui décrit un domaine. Une ontologie décrit les concepts généraux d'un domaine et les relations qui peuvent lier ces concepts. Ces concepts et liens sont décrits à l'aide d'un certain nombre de primitives inspirés de la logique de description.

L'outil que nous utiliserons pendant ce TP est le logiciel PROTEGE

Protégé est un système auteur pour la création d'ontologies. Il a été créé à l'université Stanford et est très populaire dans le domaine du Web sémantique et au niveau de la recherche en informatique.

Protégé est développé en Java. Il est gratuit et son code source est publié sous une licence libre (la Mozilla Public License).

Protégé peut lire et sauvegarder des ontologies dans la plupart des formats d'ontologies : RDF, RDFS, OWL, etc

***Source de l'introduction**:* https://fr.wikipedia.org/wiki/Web_sémantique

TP 1

CONCEPTION DE L'ONTOLOGIE

ONTOLOGIE LEGERE

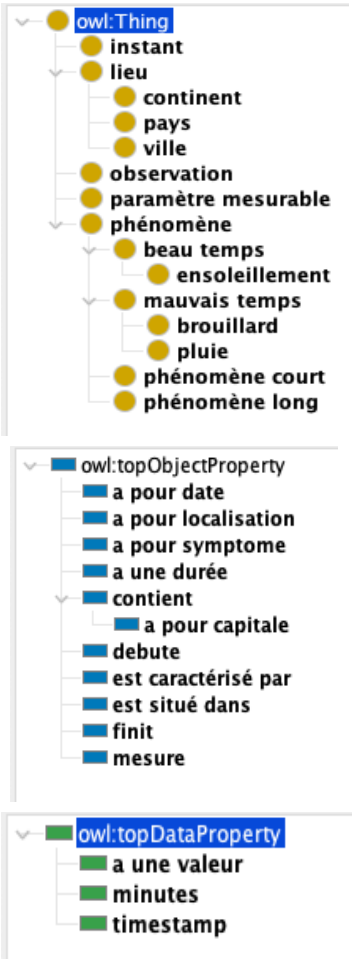
Exprimez les connaissances suivantes en créant les classes appropriées :

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

Ajoutez les propriétés (et éventuellement les concepts) permettant d'exprimer que :

Un phénomène est caractérisé par des paramètres mesurables
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 —> est situé dans
12. Un lieu peut inclure un autre lieu —> contient
13. Un pays a pour capitale une ville

De ces définitions, on en déduit les classes et sous-classes en rouge, les object properties en bleu et les data properties en vert. On a donc les objets suivant :



Détail des propriétés

La plupart du temps pour déclarer une propriété on définit un domaine et un range. Par exemple, la pour la propriété 'a pour date' qui caractérise la règle « Une Observation à pour date un instant » a pour domaine une observation et pour range un instant. Dans un premier temps, les object properties seront remplis uniquement de cette façon. Les détails sont donnés ci-dessous sous la forme <ObjectProperty> : <Domains>, <Ranges>. Ex : a pour date : observation, instant

- a pour date : observation, instant
- a pour localisation : observation, lieu
- A pour symptôme : observation, 'paramètre mesurable'
- A une durée : phénomène, minutes
- Contient : lieu, lieu
- A pour capitale : pays, ville
- Débute : Phénomène, instant
- Est caractérisé par : phénomène, 'paramètre mesurable'
- Est situé dans : lieu, lieu
- Finit : Phénomène, instant
- Mesure : observation, 'paramètre mesurable'

Détail des data properties :

Les données sont décrites de la même manière que précédemment.

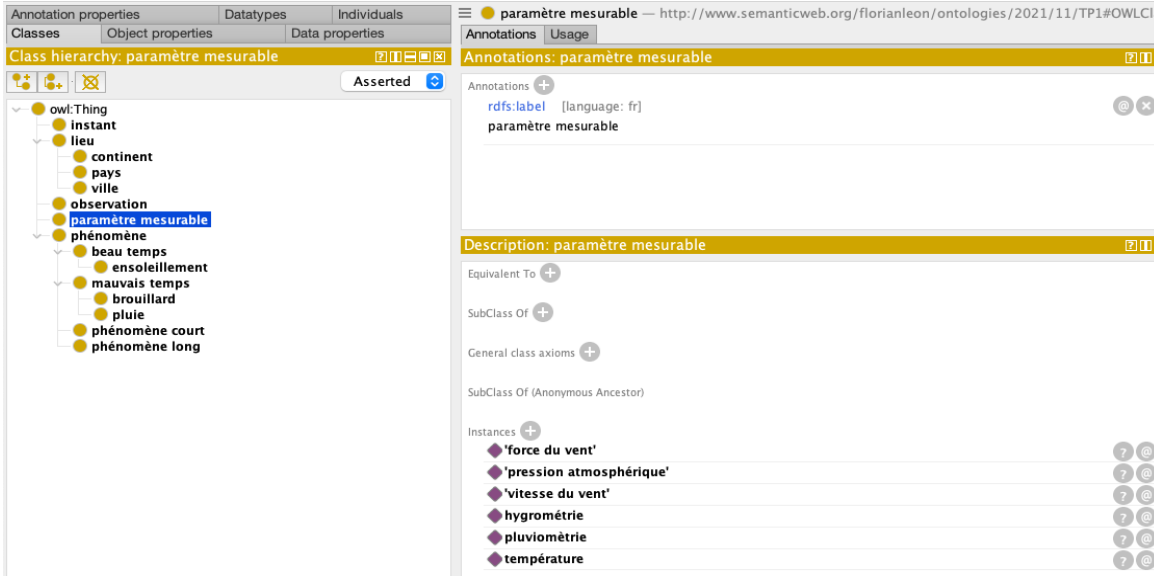
- a une valeur : observation, xsd:float (la valeur n'a donc pas d'unité ici)
- Minutes : <>, some psd:positiveInteger
- Timestamp : instant, xsd :dateTimeStamp

Peuplement de l'ontologie

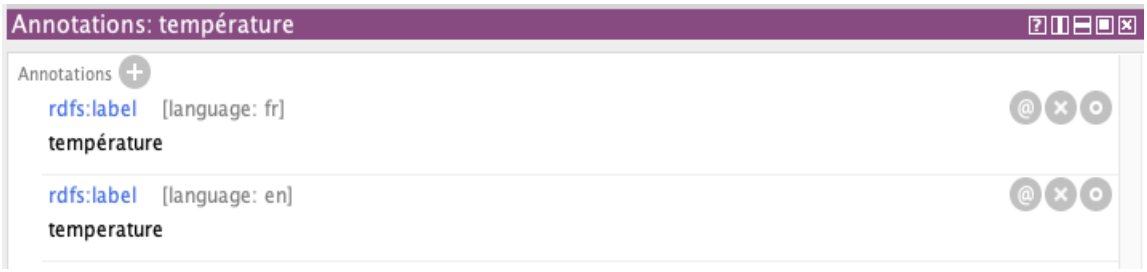
Pour peupler l'ontologie, on rajoute des individus dans l'onglet **Individuals**. Pour certains individus on ne donne pas son type. En effet, le raisonneur d'Hermit va le déduire avec les propriétés qui y sont liés.

1. La température, l'hygrométrie, la pluviométrie, la pression atmosphérique, la vitesse du vent et la force du vent sont des paramètres mesurables.

Les individus sont déclarés comme des instances de paramètre mesurable. Lors de leurs créations, on associe donc un type.

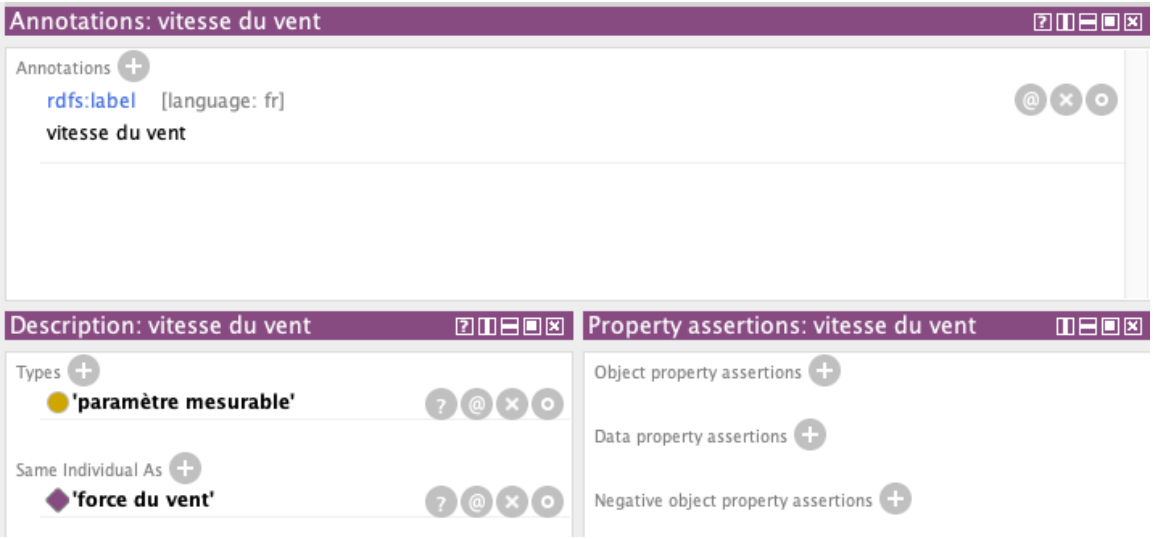


2. Le terme temperature est un synonyme anglais de température.



Pour cette règle, on rajoute un rdfs:label « temperature » en spécifiant la langue (ici, en) en rajoutant une annotation dans l'instance de température.

3. La force du vent est similaire à la vitesse du vent



On a la possibilité d'avoir deux instances qui désigne la même chose. Ici, on déclare 'force du vent' comme étant le même individu que la 'vitesse du vent'. En fixant cette règle ici, la règle symétrique est automatiquement rajoutée dans l'instance 'force du vent'.

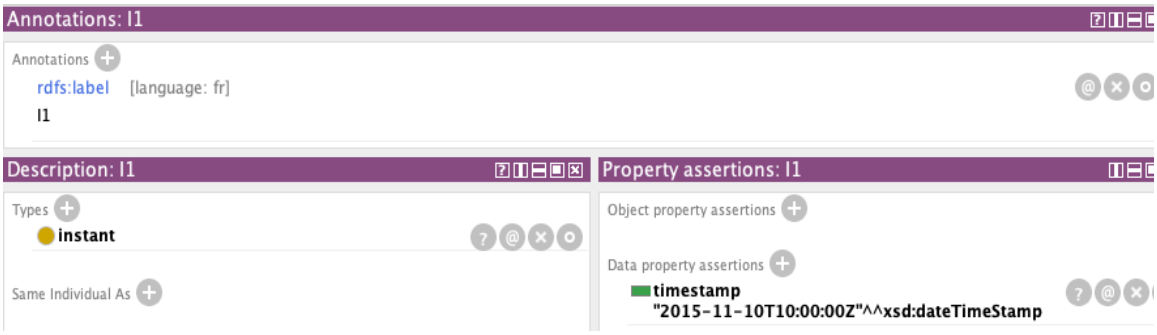
4. Toulouse est située en France et 5. Toulouse est une ville

On rajoute simplement le type ville est la propriété 'est situé dans' France. Par ailleurs, France est créé comme une instance sans type.

6. La France a pour capitale Paris. Ici aussi, Paris est un individu non typé
De la même manière que les propriétés 4 et 5, on ajoute le triplet France ‘a pour capitale’ Paris. L’instance Paris est créée sans type également.

7. Le 10/11/2015 à 10h00 est un instant que l’on appellera I1 (noté 2015-11-10T10 :00 :00Z)

On crée I1 avec le type instant et on lui donne sa valeur avec une data property (en vert sur la capture).



8. P1 est une observation qui a mesure la valeur 3 mm de pluviométrie à Toulouse à l’instant I1

9. A1 a pour symptôme P1

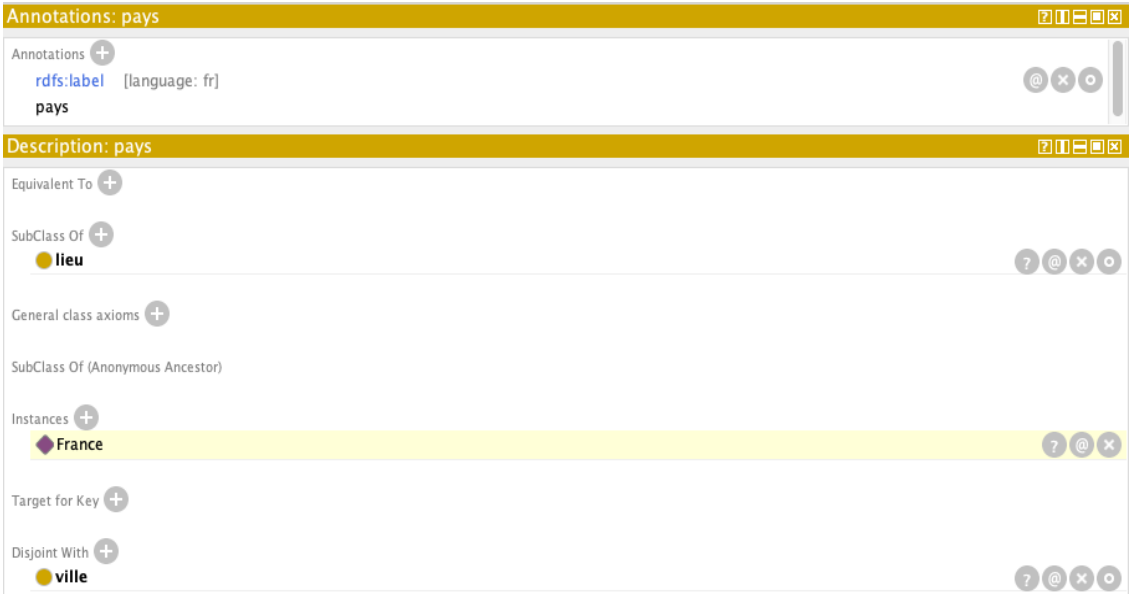
Maintenant que l’ontologie légère est peuplée on peut lancer le raisonneur. Il déduit les propriétés suivantes en plus des propriétés déjà données :

- A1 est une observation
- La France est un pays et elle contient Paris et Toulouse.
- Paris est une ville qui est situé en France.
- P1 est aussi un paramètre mesurable.

ONTOLOGIE LOURDE

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

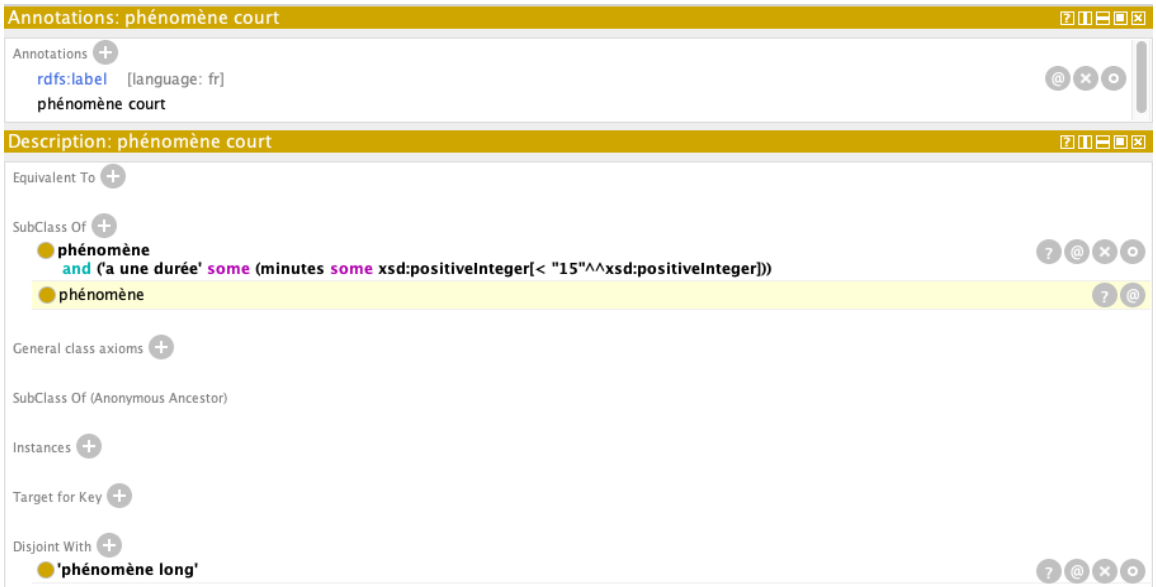
Pour respecter cette règle, on rajoute la condition 'Disjoint With'. Elle est automatiquement mise à jour dans la classe ville.



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

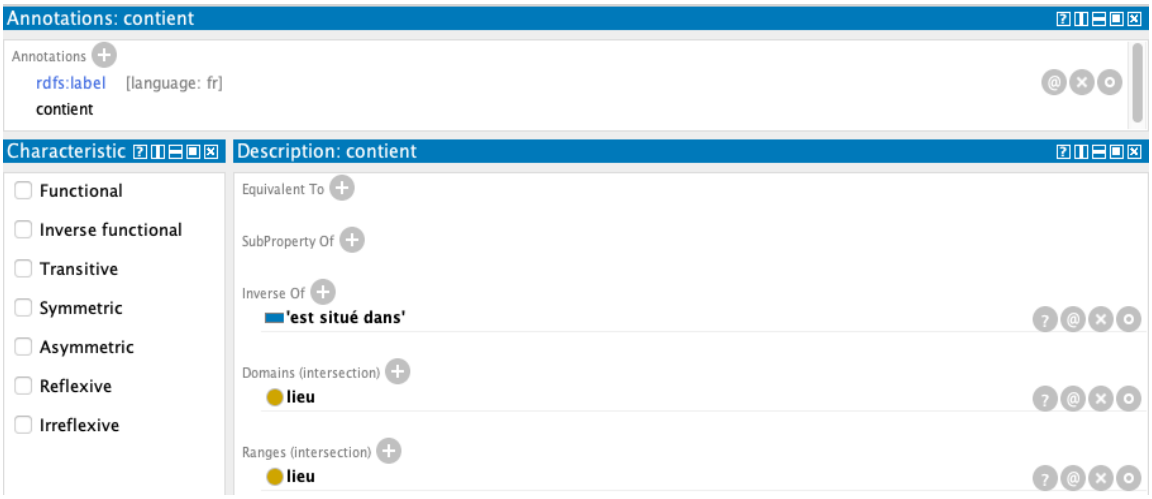
On crée une nouvelle classe phénomène court et phénomène long.

On utilise la syntaxe de Manchester pour décrire le phénomène dans l'onglet sous-classe de phénomène court. Pour qu'un phénomène court soit disjoint avec un phénomène long on rajoute la condition 'Disjoint with'. Pour le phénomène long, la démarche est la même à la seule différence du signe '<' est remplacé par '>='.



5. La propriété indiquant qu'un lieu est inclus dans un autre a pour propriété inverse la propriété

On définit simplement l'une des deux propriétés comme étant l'inverse de l'autre. Ici, 'est situé dans' est l'inverse de contient. Comme d'habitude, la propriété miroir est rajoutée automatiquement dans l'autre relation.



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.

On définit la relation 'est situé dans' comme étant transitif.

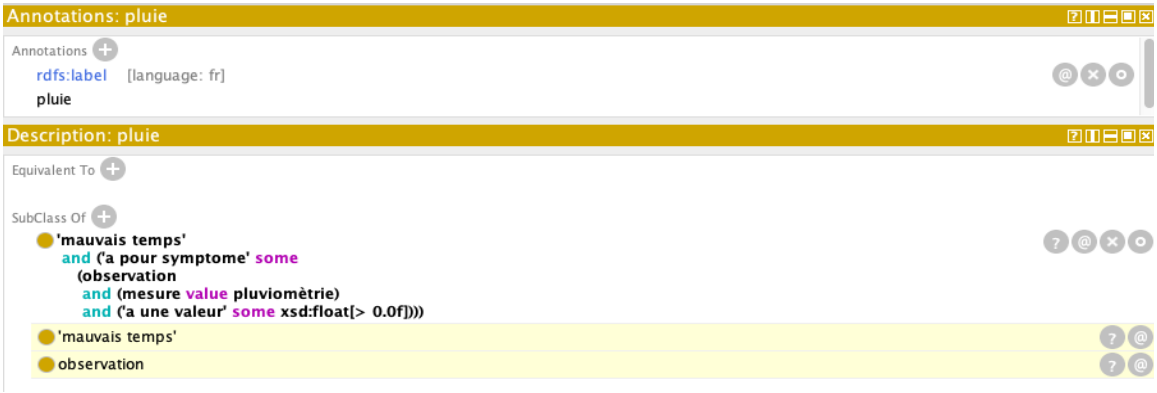
7. A tout pays correspond une et une seule capitale.

On définit la relation 'a pour capitale' comme étant fonctionnelle.

8. Si un pays a pour capitale une ville, alors ce pays contient cette ville.

On définit la relation 'a pour capitale' comme sous propriété de la relation 'contient'.

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



Peuplement de 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

Pour ces propriétés, nous ne détaillerons pas leurs mises à la place car ce serait redondant avec la partie sur l'ontologie légère. Nous dirons juste que Europe et 'La Ville Lumière' sont des individus sans type.

Bien que la propriété 4. est correcte, elle n'est pas valide ici à cause de la règle qui dissocie ville et pays.
Le raisonneur nous rajoute les déductions suivantes :

- Europe est un lieu qui contient France, Paris, Toulouse et ‘La Ville Lumière’
- La France contient ‘La Ville Lumière’
- ‘La Ville Lumière’ est une ville située en France et en Europe et qui est le même individu que Paris.
- A1 est une observation
- Paris est une ville située en France et en Europe et qui est le même individu que ‘La Ville Lumière’
- Si Toulouse est déclarée comme capitale de la France, le raisonneur déduit que Toulouse est le même individu que Paris et ‘La Ville Lumière’.

TP2

EXPLOITATION DE L'ONTOLOGIE

Le but de cette deuxième partie est d'utiliser l'ontologie précédemment créée et d'implémenter des fonctions qui seront ensuite utilisé par cette ontologie. On utilise alors les propriétés définies dans la partie 1 pour représenter certains individus de l'ontologie.

Dans un premier temps, l'objectif est d'implémenter l'interface IModelFunctions à l'aide des fonctions présentes dans IConvenienceInterface. Nous avons donc cinq fonctions à implémenter. Chaque fonction correspond à un jUnit test qu'on doit valider.

• Fonction createPlace :

Cette fonction prend en paramètre le nom d'un Lieu et doit retourner l'URI de l'individu crée. Son but est de créer une instance de la classe Lieu ayant pour nom le paramètre donné en argument de la fonction.

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

Pour cette fonction, on récupère simplement l'URI de la classe Lieu et on l'utilise pour créer et spécifier la nouvelle instance. Enfin, on retourne l'URI de l'instance nouvellement crée. Notez que la méthode getEntityURI(« Lieu ») retourne un tableau de toutes les instances portant ce nom. Dans notre cas, il n'y a qu'une seule (il n'y a pas de doublons dans l'ontologie) et on récupère donc le premier. Ça sera le cas pour les autres appels à cette fonction.

• Fonction createInstant :

L'objectif de cette fonction est la même que createPlace et prend en entrée un TimestampEntity car un instant est caractérisé par son timestamp. Cependant, ce nouvel individu n'est créé uniquement que s'il n'existe pas un autre instant associé au même timestamp.

```java
public String createInstant(TimestampEntity instant) {
    String instantURI = model.getEntityURI("Instant").get(0);
    List<String> instantLabels = model.listLabels(instantURI);
    String value = instant.timestamp;
    for(int i = 0; i < instantLabels.size(); i++) {
        //s'il existe déjà on fait rien
        if(instantLabels.get(i) == value) {
            return null;
        }
    }

    String URI = model.createInstance(value, instantURI);
    //On lui rajoute son attribut
    String timestampURI = model.getEntityURI("a pour timestamp").get(0);
    model.addDataPropertyToIndividual(URI, timestampURI, value);
    return URI;
}
```

On commence par récupérer l'URI de l'entité Instant. On récupère ensuite, tous les labels (donc les individus) ayant pour type Instant. Enfin, on récupère le timestamp passé en paramètre. On parcourt ensuite tous les Instants et on vérifie qu'aucune des instances n'a pour valeur le timestamp donné pour la nouvelle instance. Si c'est le cas on renvoie null sinon on crée une nouvelle instance de Instant ayant pour valeur le timestamp. Finalement, à l'aide de l'URI de la relation 'a pour timestamp' on ajoute une data property à l'instance nouvellement créée et on retourne son URI.

Fonction getInstantURI

Cette fonction retourne l'URI de l'Instant en fonction du timestamp donné en paramètre s'il existe.

```
1 public String getInstantURI(TimestampEntity instant) {
2     //On récupère l'URI de l'instant
3     String instantURI = model.getEntityURI("Instant").get(0);
4     String URI = null;
5     List<String> instantList = model.getInstancesURI(instantURI);
6     for(int i = 0; i < instantList.size(); i++) {
7         //On vérifie s'il existe et on récupere son timestamp s'il existe donc
8         String timestampURI = model.getEntityURI("a pour timestamp").get(0);
9         if(model.hasDataPropertyValue(instantList.get(i), timestampURI, instant.timestamp)) {
10            URI = model.getInstancesURI(instantURI).get(i);
11        }
12    }
13
14    return URI;
15 }
```

Le début de la fonction est identique au début de la précédente sauf que cette fois on ne récupère plus les labels mais la liste de toutes les instances d'Instant. On parcours ensuite cette liste et on vérifie pour chaque Instant de la liste s'il a un triplet qui correspond à la règle suivant : Instant :'a pour timestamp' instant.timestamp. Si c'est le cas on récupère son URI et en le renvoie sinon on renvoie null.

• Fonction getInstantTimestamp

Cette fonction retourne le timestamp associé à l'instance d'Instant passé en paramètre.

```
1 public String getInstantTimestamp(String instantURI)
2 {
3     String timestamp = null;
4     List<List<String>> instantProperties = model.listProperties(instantURI);
5     //On parcours les proprietés de l'URI
6     for(int i = 0; i < instantProperties.size(); i++) {
7         String timestampURI = model.getEntityURI("a pour timestamp").get(0);
8         if(instantProperties.get(i).get(0).equals(timestampURI)) {
9             timestamp = instantProperties.get(i).get(1);
10        }
11    }
12
13    return timestamp;
14 }
```

On commence par récupérer la liste des couples <Propriété, objet> à l'aide de la méthode listProperties. On parcourt ensuite cette liste et on regarde si l'Instant à une propriété 'a pour timestamp'. Si c'est le cas on récupère sa valeur. Si aucune des propriétés ne correspond on renvoie un timestamp null.

• Fonction createObs

Cette fonction permet de créer une observation à partir des différents éléments qui la qui la caractérisent : sa valeur, son "Instant" et le type du paramètre qui est observé. La fonction retourne l'URI de l'observation créée. Une observation étant plus complexe, elle est donc plus longue que la fonction pour créer un Lieu.

```java
public String createObs(String value, String paramURI, String instantURI) {
    String obsURI = null;

    obsURI = model.createInstance(value, model.getEntityURI("Observation").get(0));
    model.addObjectPropertyToIndividual(obsURI, model.getEntityURI("a pour date").get(0), instantURI);
    model.addObservationToSensor(obsURI, model.whichSensorDidIt(getInstantTimestamp(instantURI), paramURI));
    model.addObjectPropertyToIndividual(obsURI, model.getEntityURI("mesure").get(0), paramURI);
    model.addDataPropertyToIndividual(obsURI, model.getEntityURI("a pour valeur").get(0), value);

    return obsURI;
}
```

On commence par créer l'instance de type Observation. On lie l'Instant à l'observation à l'aide de l'URI de 'a pour date'. Ensuite, on a une fonction appelée whichSensorDidIt qui en fonction de l'Instant et de l'URI du paramètre renvoie l'URI du capteur ayant fait la mesure. On lie ensuite l'observation au capteur. Puis, on rajoute le triplet suivant, Observation : mesure paramètre. Finalement, on lie la valeur à l'observation.

Une fois les fonctions codées, on peut les tester à l'aide des JUnit tests qui ne renvoie aucune erreur prouvant que nos fonctions fonctionnent normalement.

On passe donc maintenant à l'implémentation de la fonction de contrôle instantiateObservations.
Cette fonction parse la liste des observations extraites du dataset et les ajoutes à notre base de connaissances. Elle prend en argument la liste des observations et l'URI du phénomène.

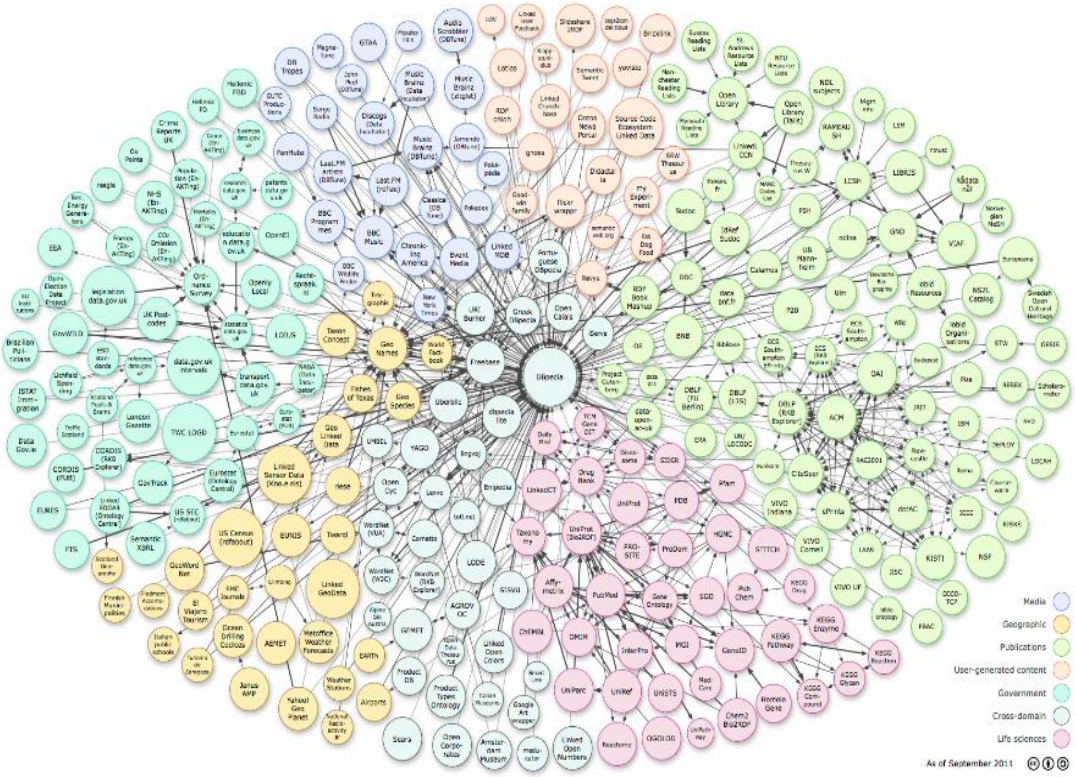
```java
public void instantiateObservations(List<ObservationEntity> obsList, String paramURI) {

    for(int i = 0; i < obsList.size(); i++) {
        ObservationEntity obs = obsList.get(i);
        String value = obs.getValue().toString();
        TimestampEntity timestamp = obs.getTimestamp();
        String instant = this.customModel.createInstant(timestamp);
        this.customModel.createObs(value, paramURI, instant);
    }

}
```

Pour cette fonction, on utilise les méthodes présentes dans la classe ObservationEntity.
On parcourt donc cette liste et pour chaque observation on réalise les opérations suivantes :

- Récupération de la valeur liée à l'observation
- Récupération du timestamp
- Création d'une instance d'Instant avec comme valeur le timestamp précédemment récupéré.
- Création de l'instance d'Observation avec les 3 précédentes opérations.

CONCLUSION

Ce TP nous a permit de voire concrètement comment l'ontologie peut ramener une amélioration dans les fonctionnalités d'une application. Nous avons appris à utiliser le logiciel PROTEGE, nous pourrons l'utiliser dans d'autres applications au futur.



https://commons.wikimedia.org/wiki/File:LOD_Cloud_Diagram_as_of_September_2011.png?uselang=fr