

Web sémantique

5 ISS



Présenté par :

*Amour Chadi
Amour Redouane
Bouisson Arnaud*

Présenté à :

Nicolas Seydoux

07/01/2022

Table des matières

Introduction.....	3
TP1 : Conception d'une ontologie	4
I. Conception de l'ontologie légère	4
II. Peuplement de l'ontologie légère	7
III. Ontologie lourde	13
IV. Peuplement de l'ontologie lourde.....	20
TP2 : Manipuler une ontologie à partir d'un code source Java.....	26
I. Implémenter l'interface ImodelFunctions.....	26
A. La méthode createPlace	26
B. La méthode createInstant	28
II. Implémenter l'interface IcontrolFunctions	36
Conclusion.....	41

Introduction

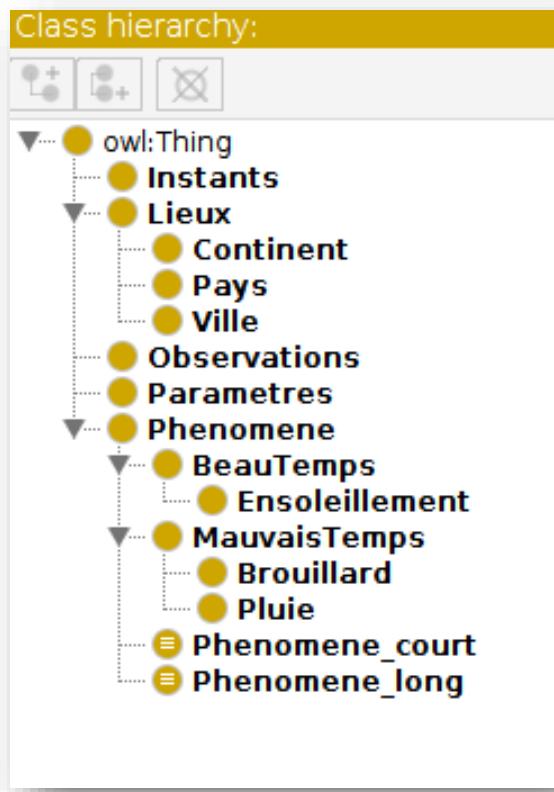
Avec l'augmentation du trafic lié à l'échange massif de données Web, il devient primordial d'organiser et structurer les données de manière à faciliter leur exploitation. On parle de Web Sémantique. Ce dernier constitue une extension du Web standardisé par le W3C. Il a pour objectif final de faciliter l'analyse et la compréhension de la structuration des données d'un site web. D'où la notion de sémantique. Par ailleurs, le domaine des objets connectés est un domaine où l'échange de données est au cœur des problématiques. Dans ce cadre il est très important de structurer la donnée.

L'objectif est justement de comprendre comment fonctionne le Web sémantique et comment il facilite l'interprétation des données par des programmes. Dans cette optique nous présenterons les analyses effectuées lors de Travaux Pratiques. Le premier visera à utiliser le logiciel Protégé pour saisir les concepts clés en comprenant le comportant d'un raisonneur. Le second visera à implémenter les différentes notions abordées lors du premier TP via du code Java sur Eclipse.

TP1 : Conception d'une ontologie

I. Conception de l'ontologie légère

Il s'agit ici de définir une ontologie appliquée à la météorologie



On retrouve ici la même logique qu'en orienté objet. Si on prend l'exemple des phénomènes météorologiques : le beau temps et le mauvais temps. Ce sont tous deux, deux types de phénomènes spécifiques d'où leur héritage de la classe phénomène comme on peut le voir ci-dessus. C'est avec cette logique-là que nous avons ainsi construit la hiérarchie de classe à la manière d'un arbre de parenté.

Il est très important ici de noter que l'on parle de classes, d'où l'appellation de hiérarchie de classes mais que ce n'est pas tout à fait le même raisonnement qu'en orienté objet.

Cependant ces classes ont beaucoup de relations exprimées dans la consigne suivante :

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

Il convient alors de créer des relations entre ces classes objets ou des relations entre ces mêmes classes et des valeurs de données. Ces relations sont appelées :

- propriété objet
- propriété de donnée

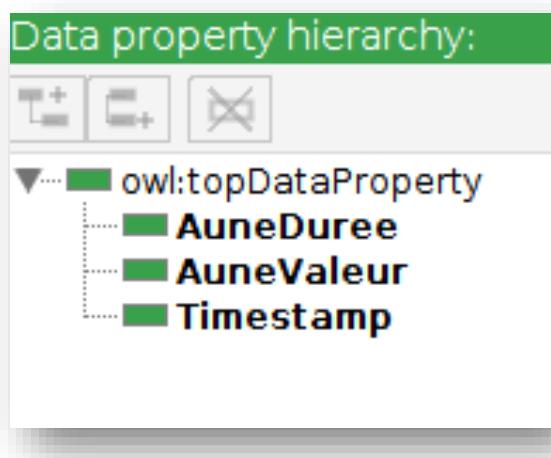
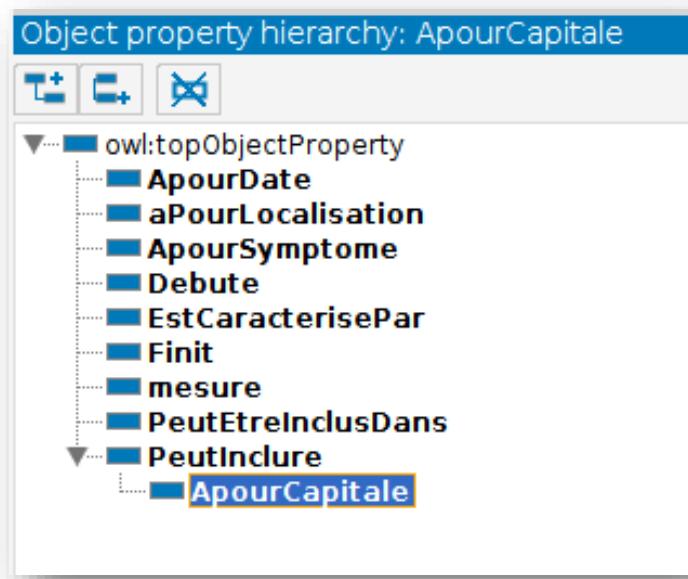
et elles permettent la création de **triplet de données** :

- ✚ sujet – propriété objet – objet (individu d'une autre classe)
- ✚ sujet – propriété de donnée – donnée (individu ayant une valeur littérale comme un entier, une chaîne de caractères etc.)

exemple pour un triplet engageant une propriété de donnée :



De la même manière que la hiérarchie de classe, il faut créer ces propriétés :



II. Peuplement de l'ontologie légère

Vous allez décrire les faits simples suivants à partir de l'ébauche d'ontologie que vous venez de construire. Après avoir représenté **chacun** des faits suivants, lancez le raisonneur Hermit, analysez les déductions faites dans un rapport.

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 à la vitesse du vent
4. 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 ?
5. Toulouse est une ville
6. La France a pour capitale Paris. Ici aussi, Paris est un individu non typé
7. Le 10/11/2015 à 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

1. On crée les instances d'individus :

The screenshot shows the 'Individuals:' list in the Protégé interface. The list contains the following individuals:

- A1
- Europe
- ForceVent
- France
- Hygrometrie
- I1
- LaVilleLumiere
- P1
- Paris
- Pluie
- Pluviometrie
- PressionAtmospherique
- Température
- Toulouse
- VitesseVent

2. Température est un synonyme anglais de température

Annotations: Température

Annotations +

rdfs:label [language: fr]
Température

rdfs:comment [language: en]
Temperature

Description: Température

Types +

Paramètres

Same Individual As +

Different Individuals +

Property assertions: Température

Object property assertions +

Data property assertions +

Negative object property assertions +

Negative data property assertions +

3. Il est indiqué que la vitesse du vent est similaire à la force du vent. Nous ajoutons cette information à l'individu « vitesse du vent » avec la mention « Same Individual As ». Lorsque l'on lance ensuite le raisonneur, on s'aperçoit que la force du vent est similaire à la vitesse avec la mention “Same Individual As”

Annotations: VitesseVent

Annotations +

rdfs:label [language: fr]
VitesseVent

Le raisonneur l'a déduit seul

Description: VitesseVent

Types +

Paramètres

Same Individual As +

ForceVent

Different Individuals +

Property assertions: VitesseVent

Object property assertions +

Data property assertions +

Negative object property assertions +

Negative data property assertions +

Ceci est très intéressant: sans même avoir mentionné quoi que ce soit sur l'individu « ForceVent », le raisonneur a déduit que cet individu est aussi pareil que l'individu « VitesseVent »

4. En définissant Toulouse et France comme des instances sans types, et qu'on définit leur relation par "Toulouse PeutEtreInclusDans France", le raisonneur parvient automatiquement à identifier leur type. Toulouse est identifié comme un **lieu** et la France aussi puisque la propriété d'objets 'PeutEtreInclusDans' relie **par définition** deux **lieux**. Ainsi une propriété d'objets liant deux instances suffit à déduire les types manipulés de ces instances. Ceci nous montre d'ores et déjà la notion de logique/sémantique.

The screenshot shows a semantic web interface with three main sections:

- Annotations: Toulouse**: Shows an annotation for `rdfs:label` with value `Toulouse` and language `fr`.
- Description: Toulouse**: Shows the individual is of type **Lieux**. It also lists **Same Individual As** and **Different Individuals**.
- Property assertions: Toulouse**: Shows an object property assertion `PeutEtreInclusDans` with value `France`.

5. Toulouse est une ville : Comme “ville” est une sous-classe de lieux, on constate que la propriété “PeutEtreInclusDans” qui joint deux lieux, est toujours valable.

The screenshot shows a user interface for managing RDF triples. At the top, there's a purple header bar labeled "Annotations: Toulouse". Below it, a sidebar titled "Annotations" lists an "rdfs:label" entry with the value "Toulouse" and its language "fr".

The main area is divided into three panels:

- Description: Toulouse**: This panel shows "Types" with "Lieux" highlighted (indicated by a red arrow). It also includes sections for "Same Individual As" and "Different Individuals".
- Property assertions: Toulouse**: This panel shows "Object property assertions" with a single entry: "PeutEtreInclusDans France" (also indicated by a red arrow).
- Data property assertions**: This panel is currently empty.

6. En définissant Paris comme étant la capitale de la France avec la relation "ApourCapitale" qui relie initialement un pays et une ville, le raisonneur déduit automatiquement que **Paris est une ville** et la **France un pays** :

Annotations: France

Annotations +
 rdfs:label [language: fr]
 France

Description: France

Types +
Pays

Same Individual As +
 Different Individuals +

Property assertions: France

Object property assertions +
ApourCapitale Paris

Data property assertions +
 Negative object property assertions +
 Negative data property assertions +

Annotations: Paris

Annotations +
 rdfs:label [language: fr]
 Paris

Description: Paris

Types +
Ville

Same Individual As +
 Different Individuals +

Property assertions: Paris

Object property assertions +
 Data property assertions +
 Negative object property assertions +
 Negative data property assertions +

 The screenshot displays three main panels:
 - Top-left: Annotations for 'France'. It shows an annotation for 'rdfs:label' with the value 'France' in French ('[language: fr]').
 - Middle-left: Description for 'France'. It lists 'Pays' as a type. Below it are buttons for 'Same Individual As' and 'Different Individuals'.
 - Right: Property assertions for 'France'. It shows an object property assertion 'ApourCapitale' with the target 'Paris'.
 - Bottom-left: Annotations for 'Paris'. It shows an annotation for 'rdfs:label' with the value 'Paris' in French ('[language: fr]').
 - Middle-right: Description for 'Paris'. It lists 'Ville' as a type. Below it are buttons for 'Same Individual As' and 'Different Individuals'.
 - Far-right: Property assertions for 'Paris'. It shows object and data property assertions, along with negative versions of these assertions.
 A red arrow points to the 'Ville' entry in the Paris description panel.

7. En spécifiant la relation « a pour symptôme » entre A1 et P1 le raisonneur déduit bien que A1 est un phénomène puisque cette relation a pour domaine les **phénomènes** et pour ranges les **observations sachant que P1 est une observation** :

Annotations: P1

Annotations +
 rdfs:label [language: fr]
 P1

Description: P1

Types + **Observations** (highlighted with a red arrow)

Same Individual As +
 Different Individuals +

Property assertions: P1

- Object property assertions +
 - aPourLocalisation Toulouse
 - mesure Pluviometrie
 - ApourDate I1
- Data property assertions +
 - AuneValeur "3"^^xsd:int
- Negative object property assertions +
- Negative data property assertions +

Question 8

Annotations: A1

Annotations +
 rdfs:label [language: fr]
 A1

Description: A1

Types + **Phenomene** (highlighted with a yellow background)

Same Individual As +
 Different Individuals +

Property assertions: A1

- Object property assertions +
 - ApourSymptome P1
- Data property assertions +
- Negative object property assertions +
- Negative data property assertions +

Asserted in: <http://www.semanticweb.org/amour/ontologies/2021/11/untitled-ontology-2>

III. Ontologie lourde

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. À 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]))

Il s'agit de créer des assertions entre classes venant complexifier les relations entre classes tout en apportant plus de logique permettant au raisonneur de déduire de nombreuses informations non explicites: c'est là l'avantage de la sémantique.

1. Toute instance de ville n'est pas un pays :

Description: Ville

Equivalent To +

SubClass Of +

 ● Lieux

General class axioms +

SubClass Of (Anonymous Ancestor)

Instances +

 ● Paris

 ● Toulouse

Target for Key +

Disjoint With + 

 ● Pays

2. Un phénomène court est un phénomène dont la durée est inférieure à 15

Description: Phenomene_court

Equivalent To +

 ● Phenomene that AuneDuree some xsd:float [< 15] 

SubClass Of +

 ● Phenomene

General class axioms +

SubClass Of (Anonymous Ancestor)

3. Un phénomène long est un phénomène dont la durée est supérieure ou égale à 15

Description: Phenomene_long

Equivalent To +

- Phénomene that AuneDuree some xsd:float [>= 15]

SubClass Of +

- Phénomene

General class axioms +

SubClass Of (Anonymous Ancestor)

4. Un phénomène long ne peut être un phénomène court : ces propriétés sont donc disjointes → on utilise la relation « disjoint with »

5. La propriété « PeutEtreDans » est l'inverse de « PeutInclure » :

Characteristics: PeutEtreDans Description: PeutEtreInclusDans

<input type="checkbox"/> Functional <input type="checkbox"/> Inverse functional <input type="checkbox"/> Transitive <input type="checkbox"/> Symmetric <input type="checkbox"/> Asymmetric <input type="checkbox"/> Reflexive <input type="checkbox"/> Irreflexive	Equivalent To + SubProperty Of + Inverse Of +  <ul style="list-style-type: none"> PeutInclure Domains (intersection) + <ul style="list-style-type: none"> Lieux Ranges (intersection) + <ul style="list-style-type: none"> Lieux Disjoint With + SuperProperty Of (Chain) +
--	--

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) : il s'agit ici d'une relation transitive

Characteristics: PeutIn

Description: PeutInclure

<input type="checkbox"/> Functional	Equivalent To
<input type="checkbox"/> Inverse functional	SubProperty Of
<input checked="" type="checkbox"/> Transitive	Inverse Of PeutEtreInclusDans
<input type="checkbox"/> Symmetric	Domains (intersection) Lieux
<input type="checkbox"/> Asymmetric	Ranges (intersection) Lieux
<input type="checkbox"/> Reflexive	Disjoint With
<input type="checkbox"/> Irreflexive	SuperProperty Of (Chain)

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

The screenshot shows the Protégé ontology editor interface. At the top, there's a navigation bar with tabs for 'Annotations' and 'Usage'. Below it, a blue header bar says 'Annotations: ApourCapitale'. The main content area has two sections: 'Annotations' and 'Characteristics'.

Annotations: Shows an annotation 'rdfs:label' with value '[language: fr] ApourCapitale'.

Characteristics: Shows the following properties for the 'ApourCapitale' class:

- Functional** (checkbox checked, highlighted with a red arrow)
- Inverse functional**
- Transitive**
- Symmetric**
- Asymmetric**
- Reflexive**
- Irreflexive**

On the right side of the characteristics section, there are several associated properties:

- Equivalent To**: +
- SubProperty Of**: +
↳ **PeutInclude**
- Inverse Of**: +
- Domains (intersection)**: +
↳ **Pays**
- Ranges (intersection)**: +
↳ **Ville**
- Disjoint With**: +
- SuperProperty Of (Chain)**: +

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

The screenshot shows the Protege ontology editor interface. At the top, the title bar displays "ApourCapitale — http://www.semanticweb.org/amaro/ontologies/2021/11/untitled-ontology-2#OWLOB". Below the title bar, there are two tabs: "Annotations" (selected) and "Usage".

Annotations: ApourCapitale

Annotations +
rdfs:label [language: fr]
ApourCapitale

Characteristics: ApourCapitale

Description: ApourCapitale

Functional (checked)
Inverse functional
Transitive
Symmetric
Asymmetric
Reflexive
Irreflexive

Equivalent To +
SubProperty Of + ← **PeutInclure**

Inverse Of +
Domains (intersection) + **Pays**

Ranges (intersection) + **Ville**

Disjoint With +
SuperProperty Of (Chain) +

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

Description: Pluie

Types +

Phénomène that ApourSymptome some (Observations that (mesure value Pluviometrie) and (AuneValeur some xsd:int [>0]))

Same Individual As +

Different Individuals +

Help ? Email @ Cancel X Save S

IV. Peuplement de l'ontologie lourde

1. La France est située en Europe :

→ On définit la France comme étant située en Europe avec la relation "PeutEtreInclusDans"

Et on observe bien que le raisonneur en déduit que l'Europe est de type "lieux". Le raisonneur va même plus loin puisque dans la partie précédente, on a défini Toulouse comme étant une ville de France, ce qui veut dire que Toulouse est inclus en France. Mais comme la France est inclus en Europe, alors Toulouse est inclus en Europe

The screenshot shows the Amour ontology editor interface. At the top, there is a navigation bar with tabs for Annotations (selected), Usage, and Annotations: Europe. Below this, the main content area displays the individual 'Europe' with its annotations. One annotation is shown: rdfs:label [language: fr] Europe. In the bottom half of the screen, there are two panels: 'Description: Europe' on the left and 'Property assertions: Europe' on the right. The 'Description' panel shows 'Types' with 'Lieux' selected. The 'Property assertions' panel shows 'Object property assertions' with four entries: PeutInclure France, PeutInclure Toulouse, PeutInclure LaVilleLumiere, and PeutInclure Paris. Two red arrows point to the entries 'PeutInclure Toulouse' and 'PeutInclure France' in the list.

2. Paris est la capitale de la France

→ Comme la relation “ApourCapitale” est une sous propriété de “PeutInclure”, ceci veut dire que Paris est inclus en France et donc que Paris est inclus en Europe par transitivité. Ce qui veut dire que l’Europe inclut Paris et on voit bien ceci :

The screenshot shows a semantic web interface for the resource `Europe` (http://www.semanticweb.org/amour/ontologies/2021/11/untitled-ontology-2#OWLNamedIndividual_ff298201_18f4_4582). The interface includes tabs for Annotations and Usage, and a sidebar for Annotations.

Annotations: Europe

Description: Europe

Object property assertions: Europe

Types: Lieux

Same Individual As:

Different Individuals:

Object property assertions:

- PeutInclure France
- PeutInclure Toulouse
- PeutInclure LaVilleLumiere
- PeutInclure Paris ←

Data property assertions:

Negative object property assertions:

Negative data property assertions:

Si on observe l'instance Paris elle-même, on voit que le raisonneur a déduit que Paris était une ville puisque la relation "ApourCapitale" met en relation un pays et une ville

The screenshot shows the Amour ontology editor interface. At the top, there is a navigation bar with tabs for 'Annotations' and 'Usage'. Below this, a purple header bar displays the URL http://www.semanticweb.org/amour/ontologies/2021/11/united-ontology-2#OWLNamedIndividual_0c82b6c6_5a58_4feb_ and the title 'Annotations: Paris'. The main content area is divided into several sections:

- Annotations:** Shows a single entry: `rdfs:label` [language: fr] `Paris`.
- Description: Paris**:
 - Types**: `+ Ville` (highlighted with a red arrow)
 - Same Individual As**: `+ LaVilleLumiere`
 - Different Individuals**: `+`
- Property assertions: Paris**:
 - Object property assertions**:
 - `PeutEtreInclusDans France`
 - `PeutEtreInclusDans Europe`
 - Data property assertions**: `+`
 - Negative object property assertions**: `+`
 - Negative data property assertions**: `+`

3. La Ville Lumière est la capitale de la France

→ Comme la relation “ApourCapitale” est fonctionnelle, ceci veut dire qu'il existe une unique capitale de la France et donc qu'il ne peut y avoir deux capitales. Or nous avons déjà défini Paris comme étant la capitale de la France, ce qui implique que la ville lumière et Paris sont identiques :

The screenshot shows the Amour ontology editor interface. At the top, there is a navigation bar with tabs for "Annotations" and "Usage". Below this, the main area is titled "Annotations: LaVilleLumiere". Under "Annotations", there is a single entry: "rdfs:label" (language: fr) "LaVilleLumiere".

On the left side, there is a sidebar with sections for "Description: LaVilleLumiere" and "Property assertions: LaVilleLumiere".

- Description: LaVilleLumiere:** This section contains:
 - "Types": "Ville" (highlighted with a yellow background and a red arrow pointing to it).
 - "Same Individual As": "Paris" (highlighted with a yellow background and a red arrow pointing to it).
 - "Different Individuals":
- Property assertions: LaVilleLumiere:** This section contains:
 - "Object property assertions":
 - "PeutEtreInclusDans France"
 - "PeutEtreInclusDans Europe"(both highlighted with yellow backgrounds and red arrows pointing to them).
 - "Data property assertions":
 - "Negative object property assertions":
 - "Negative data property assertions":

Ceci implique donc que « LaVilleLumière » est inclus en Europe :

Le raisonneur en déduit également que l'instance « LaVilleLumière » est inclus en France et en Europe.

The screenshot shows the Amour ontology editor interface. On the left, the 'Annotations' tab is selected for the 'Europe' class. It contains an annotation `rdfs:label` with the value 'Europe'. Below this, under 'Description: Europe', there is a 'Types' section with a single item 'Lieux' highlighted in yellow. To the right, the 'Property assertions: Europe' panel is shown. It lists object property assertions: 'PeutInclure France', 'PeutInclure Toulouse', 'PeutInclure LaVilleLumiere' (which has a red arrow pointing to it), and 'PeutInclure Paris'. There are also sections for data property assertions, negative object property assertions, and negative data property assertions.

4. Singapour est une ville et un pays

→ Ici le raisonneur va relever une inconsistance ce qui est normal puisque nous avons défini qu'une ville ne peut être un pays avec la disjointure :

The screenshot shows the 'Inconsistent ontology explanation' dialog in the Amour editor. It includes options to show regular or laconic justifications, and a limit of 2 justifications. Below this, 'Explanation 1' is displayed, with a checkbox for 'Display laconic explanation' which is unchecked. The explanation text reads: 'Explanation for: owl:Thing SubClassOf owl:Nothing'. It lists three statements: 'Singapour Type Pays', 'Ville DisjointWith Pays', and 'Singapour Type Ville'. The 'DisjointWith' statement is highlighted in blue.

Constatez la réaction du raisonneur si Toulouse est déclarée comme la capitale de la France

→ “ApourCapitale” est une relation fonctionnelle ce qui veut dire qu'il a une unique capitale par pays et donc par déduction, toutes les instances capitales d'un même pays sont identiques. On observe bien dans la figure suivante que le raisonneur déduit de “la France a pour capitale Toulouse” que Toulouse est donc identique aux instances “Paris” et “LaVilleLumiere”

→ Par ailleurs, comme la relation “ApourCapitale” est une sous-propriété de “PeutInclure”, alors le raisonneur en déduit que Toulouse est inclus en France et même en Europe par transitivité.

The screenshot shows the Amour ontology editor interface with the following details:

- Annotations Tab:** Shows annotations for 'Toulouse'. One annotation is present: `rdfs:label [language: fr]` with value `Toulouse`.
- Description Tab:** Shows the type `Ville` and same individual as `Paris, LaVilleLumiere`.
- Property assertions Tab:** Shows object property assertions:
 - `PeutEtreInclusDans France` (highlighted with a yellow background and red arrows)
 - `PeutEtreInclusDans Europe` (highlighted with a yellow background and red arrows)

Ce TP1 nous a permis de manipuler les concepts de classe, de propriété objets et de donnée mais également de saisir la logique du raisonneur en peuplant l'ontologie. Tout ceci s'est fait via le logiciel protégé. Dans le prochain TP nous apprendrons à peupler une ontologie via du code source.

TP2 : Manipuler une ontologie à partir d'un code source Java

Implementing the interfaces

The codebase that you cloned contains **Java interfaces** (functions specifications), namely `IControlFunctions` and `IModelFunctions`. These interfaces are implemented by the stubbed classes `DoItYourselfModel` and `DoItYourselfControl`. The functions to implement are of increasing complexity in order: the first one is the easiest, and as you progress you can reuse the previous elements you developed.

Implementing `IModelFunction`

Start by implementing `IModelFunction` in `DoItYourselfModel`. These functions provide knowledge-base related operations. To help you, you will find functions in the `IConvenienceInterface` that wrap lower-level Jena functions and SPARQL queries.

After having written each functions, run the tests. You can read the code for the tests if you are unsure of what you should do.

Implementing `IControlFunctions`

The controller uses functions from the model, and uses them to enrich the dataset. Once you complete the interface implementation, go to the main function in the Controller class. You must edit some code snippets depending on your environment.

I. Implémenter l'interface `IModelFunctions`

Code source : <https://github.com/camour/WebSemantic.git>

A. La méthode `createPlace`

Nous devons commencer par écrire le corps des méthodes. La première méthode est “`createPlace`” :

```
@Override
public String createPlace(String name) {
    // TODO Auto-generated method stub
    return null;
}

@Override
```

Si on se rend sur l'interface java qui contient la description de la méthode, on lit :

```
/**  
 * Creates an instance of the class "Place" of your ontology  
 * @param name  
 * @return the URI of the instance  
 */  
public String createPlace(String name);
```

Bilan : la méthode “createPlace” doit renvoyer l’URI de l’instance “Lieu” avec en paramètre le nom de ce lieu (le label). Donc si on fait “createPlace(“Paris”), on doit pouvoir créer l’instance “Paris” qui aura pour classe “Lieu”

Si on se rend sur l’interface “IConvenience” on remarque que cette interface permet de générer une URI d’une instance dont on passe le type (l’URI du type) avec la méthode “createInstance” :

```
/**  
 * Creates an instance of the provided type, with the provided label.  
 * @param label  
 * @param the URI of the type  
 * @return the URI of the created individual  
 */  
public String createInstance(String label, String type);
```

Le label est connu puisqu’il est passé en paramètre de “createPlace” mais il reste cependant l’URI de la classe “Lieu”. D’ailleurs dans le fichier « tp-iss.ttl », on peut lire ce nom de classe :

```
### http://homepages.laas.fr/nseydoux/ontologies/tp-iss#Place  
tp-iss:Place rdf:type owl:Class ;  
               rdfs:label "Lieu"@fr, "Place"@en .
```

On voit que l'interface "IConvenience" possède une méthode "getEntityURI" que nous allons donc utiliser pour obtenir automatiquement l'URI de la classe (du type). Ce qui donne finalement pour l'ensemble de la méthode "createPlace" :

The screenshot shows a Java code editor with the following code:

```

7  public DoItYourselfModel(IConvenienceInterface m) {
8      this.model = m;
9  }
10
11 @Override
12 public String createPlace(String name) {
13     return this.model.createInstance(name, this.model.getEntityURI("Lieu").get(0));
14 }
15

```

A red arrow points from the line "return this.model.createInstance(name, this.model.getEntityURI("Lieu").get(0));" up towards the interface reference "this.model". Below the code editor is a JUnit test runner interface. A red box highlights the "Test" button, and a red arrow points from it to the "testPlaceCreation [Runner: JUnit 4] (0,555 s)" entry in the list.

Markers Properties Servers Data Source Explorer Snippets Console JUnit Finished after 0,564 seconds

Runs: 1/1 Errors: 0 Failures: 0

testPlaceCreation [Runner: JUnit 4] (0,555 s) ← Test Failure Tra

On notera que le test lié à cette méthode "createPlace" est validé !

B. La méthode createInstant

```

/**
 * Creates an instance of the "Instant" class of your ontology. You'll have to link it to
 * a data property that represents the timestamp, serialized as it is in the original data file.
 * Only one instance should be created for each actual timestamp.
 * @param instant
 * @return the URI of the created instant, null if it already existed
 */
public String createInstant(TimestampEntity instant);

```

On doit ici retourner une instance (l'URI de l'instance) de type "Instant", on va donc utiliser encore une fois la méthode "createInstance" de l'interface "IConvenience" :

```

@Override
public String createInstant(TimestampEntity instant) {
    String instance = this.model.createInstance("instant", this.model.getEntityURI("Instant").get(0));
}

```

A red arrow points from the line "String instance = this.model.createInstance("instant", this.model.getEntityURI("Instant").get(0));" up towards the interface reference "this.model".

Cependant cette instance a pour valeur “un timestamp”, on doit donc lier/attribuer cette valeur timestamp à l’instance. En fait, le timestamp est une data. On doit donc utiliser la propriété permettant de lier une instance de type « Instant » à une donnée de type « timestamp ». Si on se rend dans le fichier « tp-iss.ttl », on observe :

```
### http://homepages.laas.fr/nseydoux/ontologies/tp-iss#hasTimestamp
tp-iss:hasTimestamp rdf:type owl:DatatypeProperty ;
    rdfs:domain tp-iss:Instant ;
    rdfs:range xsd:dateTimeStamp ;
    rdfs:label "a pour timestamp"@fr ;
    rdfs:label "has timestamp"@en .
```

La dataProperty “has timestamp” permet ainsi de lier le timestamp à l’instance “instant”. De plus, en analysant l’interface “IConvenience”, on remarque que la méthode “addDataProperty” est exposée, on va donc l’utiliser pour lier l’instant timestamp à l’instance “instant”:

```
@Override
public String createInstant(TimestampEntity instant) {
    String instance = this.model.createInstance("instant", this.model.getEntityURI("Instant").get(0));
    this.model.addDataPropertyToIndividual(instance, this.model.getEntityURI("has timestamp").get(0), instant.getTimestamp());
    return instance;
}
```

Création d’un triplet « individu – propriété - valeur »

Pour aller plus loin, il est mentionné que pour un timestamp donné, il ne devrait exister qu’une seule instance de type « Instant ». Le code de la fonction createInstant serait finalement le suivant :

```
@Override
public String createInstant(TimestampEntity instant) {

    for(String instanceURI : this.model.getInstancesURI(this.model.getEntityURI("Instant").get(0))) {
        if(this.model.hasDataPropertyValue(instanceURI, this.model.getEntityURI("has timestamp").get(0), instant.getTimestamp())) {
            return null;
        }
    }
    String instance = this.model.createInstance("instant", this.model.getEntityURI("Instant").get(0));
    this.model.addDataPropertyToIndividual(instance, this.model.getEntityURI("has timestamp").get(0), instant.getTimestamp());
    return instance;
}
```

On doit en effet vérifier que parmi toutes les instances de type « Instant » existante, aucune n’est déjà associée au timestamp qu’on « instant.getTimestamp() ».

On vérifie ensuite via Junit, que la fonction “createInstant” est bien implémentée :

The screenshot shows an IDE interface. The top part displays a portion of a Java class with code related to timestamp entities. The bottom part shows a JUnit test results window with the following details:

- JUnit X
- Finished after 2,833 seconds
- Runs: 1/1
- Errors: 0
- Failures: 0
- A green progress bar is fully filled.
- Test name: testInstantCreation [Runner: JUnit 4] (2,783 s)
- Failure Trace button

C. La méthode getInstantURI

```
/**  
 * Returns the instant with the provided timestamp if it exists.  
 * @param instant  
 * @return the URI of the representation of the instant, null otherwise.  
 */  
public String getInstantURI(TimestampEntity instant);
```

Dans la fonction précédente, on créait l’instance de type “Instant” et ensuite on lui assignait une valeur. Ici, c’est un processus un peu inverse : à partir d’une valeur de timestamp donnée, on veut savoir si cette valeur est associée à une instance. Le raisonnement serait donc le suivant : pour chaque instance de type “Instant” existante dans l’ontologie, il faut regarder si l’une de ces instances est liée (par la propriété “has timestamp”) à la valeur “instant” donnée en paramètre de la méthode getInstantURI.

En analysant l'interface "IConvenienceInterface" on constate qu'il existe une méthode :

```
/*
 * @param subjectURI
 * @param dataPropertyURI
 * @param dataValue
 * @return if provided subject has a certain value for provided property
 */
public boolean hasDataPropertyValue(String subjectURI, String dataPropertyURI, String dataValue);
```

On peut alors utiliser cette méthode pour vérifier si l'URI d'une instance de type "Instant" est liée (notion de triplet), par la propriété "has timestamp", à la valeur de timestamp recherchée :

```
@Override
public String getInstantURI(TimestampEntity instant) {
    String instanceURIFound = null;
    for(String instanceURI : this.model.getInstancesURI(this.model.getEntityURI("Instant").get(0))) {
        if(this.model.hasDataPropertyValue(instanceURI, this.model.getEntityURI("has timestamp").get(0), instant.getTimestamp()))
            instanceURIFound = instanceURI;
    }
    return instanceURIFound;
}
```

Nous stockons donc l'URI, vérifiant la condition (dans le if), dans "instanceURIFound" sachant qu'il ne peut normalement y avoir qu'une seule instance pour un timestamp donné.

On n'oubliera pas d'initialiser le résultat contenu dans "instanceURIFound" à null, au début, "au cas où aucune URI ne vérifierait la condition placée dans le "if".

De nouveau on effectue un test (qui valide bien notre fonction):

```
25     @Override
26     public String getInstantURI(TimestampEntity instant) {
27         String instanceURIFound = null;
28         for(String instanceURI : this.model.getInstancesURI(this.mo
29             if(this.model.hasDataPropertyValue(instanceURI, this.mo
30                 instanceURIFound = instanceURI;
31             }
32         }
33         return instanceURIFound;
34     }
```

JUnit X
Finished after 3,34 seconds
Runs: 1/1 Errors: 0 Failures: 0
testInstantRetrieval [Runner: JUnit 4] (3,295 s) ←

D. La méthode getInstantTimestamp

```
/**  
 * @param instantURI  
 * @return the value of the timestamp associated to the instant individual, null if the individual doesn't exist  
 */  
public String getInstantTimestamp(String instantURI);
```

Ici il s'agit de vérifier si pour un instant donné, il existe un timestamp associé.

Tout d'abord il convient de vérifier que l'instant "instantURI" existe bien :

```
public String getInstantTimestamp(String instantURI)
{
    String timeStamp = null;
    boolean keepSearching = false;

    for(String instanceURI : this.model.getInstancesURI(this.model.getEntityURI("Instant").get(0))) {
        if(instanceURI.equals(instantURI) && (!keepSearching)) {
            keepSearching = true;
        }
    }
}
```

Remarque : En vue d'éviter l'imbrication d'une boucle for dans une boucle for nous préférons utiliser un booléen "keepSearching" qui indique si, après avoir recherché l'existence de "instantURI", nous devons poursuivre le programme pour trouver le timestamp associé à "instantURI" justement.

Si "instantURI" est bien existant, nous devons alors vérifier s'il y a bien un timestamp associé. Dans l'interface IConvenient, nous constatons qu'il existe une méthode "listProperties" qui retourne la liste des couples "property, object" d'une instance :

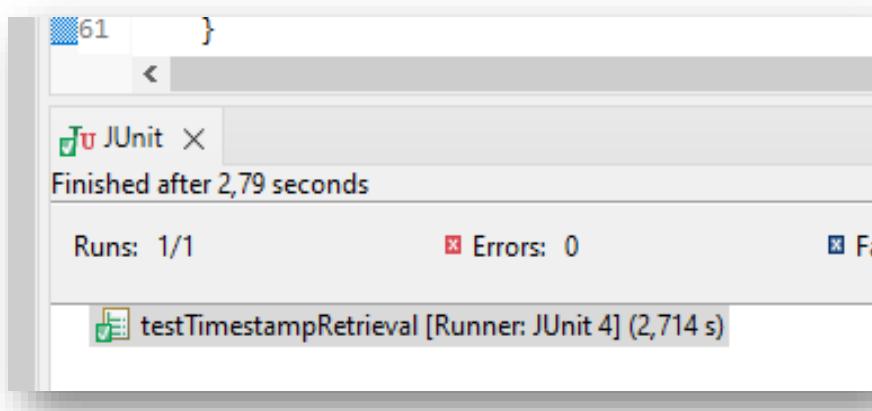
```
/**  
 * @param entityURI  
 * @return A list of couples <property, object>  
 */  
public List<List<String>> listProperties(String entityURI);
```

Donc pour notre instance, nous devons utiliser cette liste de couples et chercher le couple "has timestamp, valeur_timestamp" :

```
if(keepSearching) {  
    for(List<String> liste: this.model.listProperties(instantURI)) {  
        if(liste.get(0).equals(this.model.getEntityURI("has timestamp").get(0))) {  
            if(this.model.hasDataPropertyValue(instantURI, liste.get(0), liste.get(1))){  
                timeStamp = liste.get(1);  
                System.out.println(liste);  
            }  
        }  
    }  
}
```



Il nous reste ensuite à tester notre programme :



E. La méthode createObs

```
36
37 /**
38 * Creates an Observation of the provided value for the provided parameter
39 * at the provided time. It uses both object and data properties from the ontology
40 * to link the observation to its value, instant, and parameter.
41 * @param value
42 * @param param
43 * @param instantURI
44 * @return the URI of the created observation
45 */
46 public String createObs(String value, String paramURI, String instantURI);
47 }
48
```

Une observation est une classe de concept qui **mesure** un paramètre mesurable. D'après le fichier « tp-iss.ttl », nous observons que la data propriété est nommée “measures” :

```
@Override
public String createObs(String value, String paramURI, String instantURI) {
    String observationURI = this.model.createInstance("observation", this.model.getEntityURI("Observation").get(0));
    this.model.addObjectPropertyToIndividual(observationURI, this.model.getEntityURI("measures").get(0), paramURI);
```



Nous devons ainsi créer une instance de type “Observation” et la lier au paramètre à travers la propriété objet “measures” en utilisant “addObjectPropertyToIndividual”:

```
### http://homepages.laas.fr/nseydoux/ontologies/tp-iss#measures
tp-iss:measures rdf:type owl:ObjectProperty ;
    rdfs:domain tp-iss:WeatherObservation ;
    rdfs:range tp-iss:Parameters ;
    rdfs:label "mesure"@fr ;
    rdfs:label "measures"@en .
```

Lions ensuite l'instant à cette observation. Pour ce faire on se rend dans le fichier « tp-iss.ttl » et on voit que pour lier une observation à un instant:

```
### http://homepages.laas.fr/nseydoux/ontologies/tp-iss#hasDate
tp-iss:hasDate rdf:type owl:ObjectProperty ;
    rdfs:domain tp-iss:WeatherObservation ;
    rdfs:range tp-iss:Instant ;
    rdfs:label "a pour date"@fr;
    rdfs:label "has date"@en .
```

Il faut donc lier cette observation avec l'instant en utilisant la méthode “addObjectPropertyToIndividual”

```
if(this.getInstantTimestamp(instantURI) == null) {
    this.model.addDataPropertyToIndividual(instantURI, this.model.getEntityURI("has timestamp").get(0), this.getInstantTimestamp(instantURI));
}
this.model.addObjectPropertyToIndividual(observationURI, this.model.getEntityURI("has date").get(0), instantURI);
```

Nous devons ensuite lier 'observation avec la valeur mesurée justement :

```
### http://homepages.laas.fr/nseydoux/ontologies/tp-iss#hasValue
tp-iss:hasValue rdf:type owl:DatatypeProperty ;
    rdfs:domain tp-iss:WeatherObservation ;
    rdfs:range xsd:float ;
    rdfs:label "a pour valeur"@fr ;
    rdfs:label "has value"@en .
```

```
this.model.addDataPropertyToIndividual(observationURI, this.model.getEntityURI("has value").get(0), value);
```

Dernière étape pour terminer la fonction : une observation est faite par un capteur, il faut donc enregistrer dans la base de connaissance, le capteur ayant effectué l'observation :

The screenshot shows an IDE interface with a code editor and a test results window. The code editor displays a Java class with an overridden method `createObs`. A red arrow points to the line `this.model.addObservationToSensor(observationURI, sensorURI);`. The test results window below shows a green bar indicating success: "JUnit X" with "Runs: 1/1", "Errors: 0", and "Failures: 0". A red arrow points to the status bar which says "testObservationCreation [Runner: JUnit 4] (3,936 s)".

```
69  @Override
70  public String createObs(String value, String paramURI, String instantURI) {
71      String observationURI = this.model.createInstance("observation", this.model.getEntityURI("Observation"));
72      this.model.addObjectPropertyToIndividual(observationURI, this.model.getEntityURI("measures").get(0));
73
74
75
76
77
78
79      this.model.addObjectPropertyToIndividual(observationURI, this.model.getEntityURI("has date").get(0));
80
81      this.model.addDataPropertyToIndividual(observationURI, this.model.getEntityURI("has value").get(0));
82
83      String sensorURI = this.model.whichSensorDidIt(this.getInstantTimestamp(instantURI), paramURI);
84      this.model.addObservationToSensor(observationURI, sensorURI);
85
86      return observationURI;
87  }
```

II. Implémenter l'interface IcontrolFunctions

Nous devons écrire le corps de la fonction suivante :

```
 /**
 * This function parses the list of observations extracted from the dataset,
 * and instanciates them in the knowledge base.
 * @param obsList
 * @param phenomenonURI
 */
public void instantiateObservations(List<ObservationEntity> obsList, String phenomenonURI);
```

Nous allons nous servir des fonctions développées dans la partie précédente et notamment de la fonction “createObs” qui permet de créer une observation :

```
@Override  
public void instantiateObservations(List<ObservationEntity> obsList,  
    String paramURI) {  
    for(ObservationEntity observation : obsList) {  
        this.cusotmModel.createObs(observation.getValue().toString(), paramURI, this.cusotmModel.getInstantURI(observation.getTimestamp()));  
    }  
}
```



La fonction “createObs” prend en premier paramètre, la valeur mesurée. Or si on s'intéresse de plus près à la classe “ObservationEntity” on voit qu'elle dispose de la méthode “getValue”

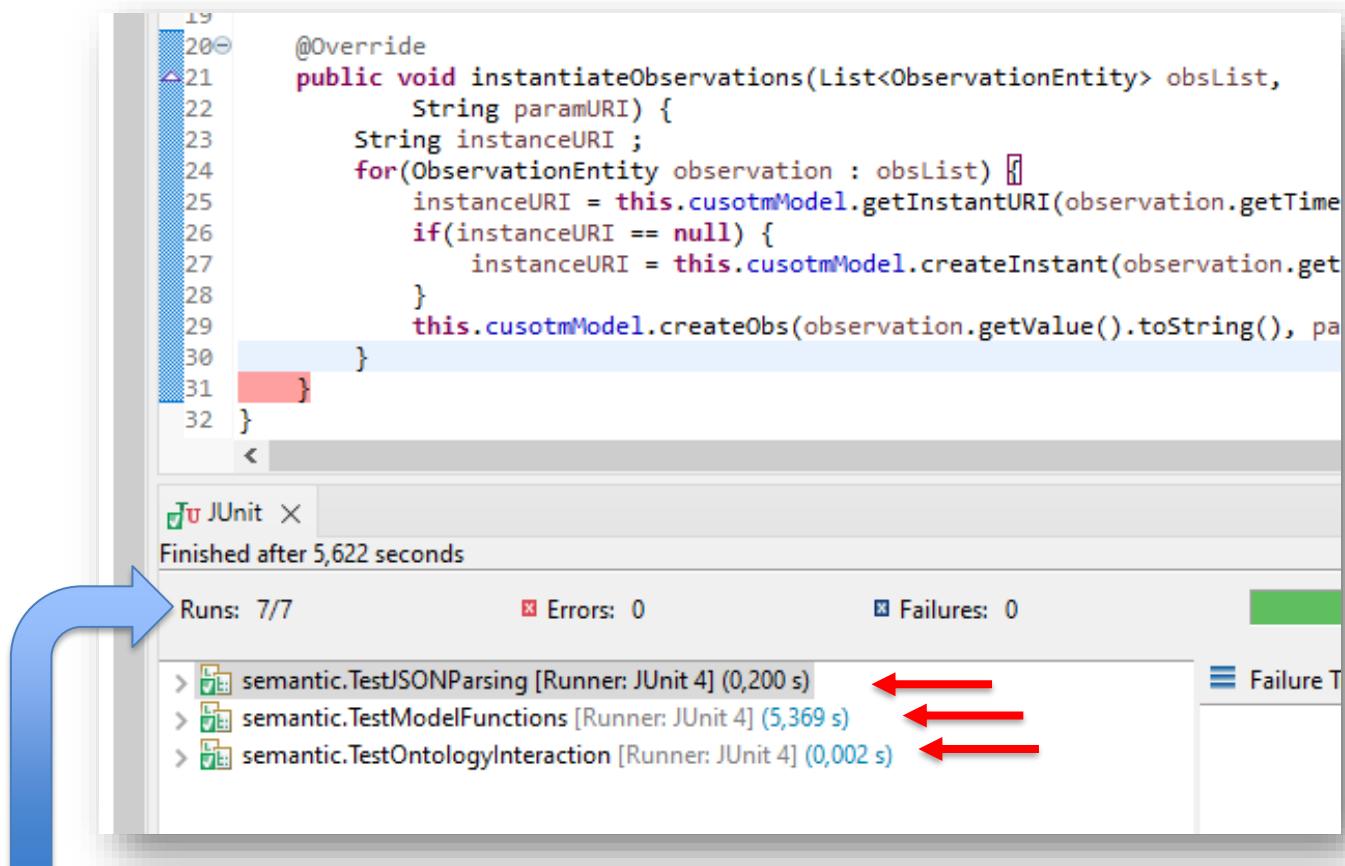
```
2  
3 public class ObservationEntity  
4 {  
5     private Float value;  
6     private TimestampEntity timestamp;  
7  
8     public ObservationEntity(Float value, TimestampEntity timestamp)  
9     {  
10         this.value = value;  
11         this.timestamp = timestamp;  
12     }  
13  
14     public Float getValue() ←  
15     {  
16         return value;  
17     }
```



Le second paramètre de la méthode “createObs” nous est déjà donné en paramètre de notre fonction “instantiateObservations”. Le dernier paramètre de la méthode « createObs », lui, correspond à l'URI de l'instant lié à l'observation. Normalement aucun instant correspondant à cette observation n'a encore été créé. Pour retrouver si cet instant l'a été , nous nous servons de la partie précédente. A savoir, nous avons développé une fonction permettant de retrouver l'instant à partir du timestamp ET nous savons que la classe “ObservationEntity” dispose d'un timestamp que nous obtenons avec la méthode “getTimestamp” :

```
public TimestampEntity getTimestamp()
{
    return timestamp;
}
```

Tests validés :



The screenshot shows an IDE interface with two main panes. The top pane displays Java code, and the bottom pane shows the JUnit test results.

Code (Top Pane):

```
19
20     @Override
21     public void instantiateObservations(List<ObservationEntity> obsList,
22                                         String paramURI) {
23         String instanceURI ;
24         for(ObservationEntity observation : obsList) {
25             instanceURI = this.cusotmModel.getInstantURI(observation.getTime
26             if(instanceURI == null) {
27                 instanceURI = this.cusotmModel.createInstant(observation.get
28             }
29             this.cusotmModel.createObs(observation.getValue().toString(), pa
30         }
31     }
32 }
```

Test Results (Bottom Pane):

JUnit X

Finished after 5,622 seconds

Runs:	Errors:	Failures:
7/7	0	0

Failure Tree

- > semantic.TestJSONParsing [Runner: JUnit 4] (0,200 s)
- > semantic.TestModelFunctions [Runner: JUnit 4] (5,369 s)
- > semantic.TestOntologyInteraction [Runner: JUnit 4] (0,002 s)

Three red arrows point from the failure tree list to the three test cases listed.

Les 7 tests sont réussis

Par ailleurs nous pouvons analyser notre fichier « export.ttl » généré :

```
<http://homepages.laas.fr/nseydoux/ontologies/tp-iss#measures>
    a      <http://www.w3.org/2002/07/owl#ObjectProperty> ;
    <http://www.w3.org/2000/01/rdf-schema#domain>
        <http://homepages.laas.fr/nseydoux/ontologies/tp-iss#WeatherObservation> ;
    <http://www.w3.org/2000/01/rdf-schema#label>
        "mesure@fr" , "measures@en" ;
    <http://www.w3.org/2000/01/rdf-schema#range>
        <http://homepages.laas.fr/nseydoux/ontologies/tp-iss#Parameters> .
```

Sur l'image ci-dessus, on observe que :

- Les données sont accessibles via le Web avec la présence d'URL « http... »
- Les données sont structurées (présence de balises) et de « ; »
- Les données sont exprimées en utilisant le langage RDF (avec présence notamment de triplets). Selon le modèle OSI, le langage RDF est situé au-dessus de la couche http, assurant ainsi une interopérabilité.
- Les données sont liées, notamment ici, on voit qu'une observation météo est reliée à la propriété « mesure » et à l'objet « paramètre » formant un triplet. Pour chacune des trois composantes de ce triplet, on a l'URI correspondante.
- Les données suivent les spécifications W3C

On peut voir encore cette notion de liaison de données :

```
<http://homepages.laas.fr/nseydoux/ontologies/tp-iss#Country>
    a      <http://www.w3.org/2002/07/owl#Class> ;
    <http://www.w3.org/2000/01/rdf-schema#label>
        "Pays@fr" , "Country@en" ;
    <http://www.w3.org/2000/01/rdf-schema#subClassOf>
        <http://homepages.laas.fr/nseydoux/ontologies/tp-iss#Place> ;
    <http://www.w3.org/2002/07/owl#disjointWith>
        <http://homepages.laas.fr/nseydoux/ontologies/tp-iss#City> .
```

L'image ci-dessus exprime le fait que la classe « Country » est une sous-classe de la classe « Place » et que cette même classe « Country » est disjointe avec la classe « City ». En effet un pays ne peut-être en même temps une ville .

Import sur Protege :

Ontology metrics:	
Metrics	
Axiom	863
Logical axiom count	223
Declaration axioms count	182
Class count	69
Object property count	67
Data property count	5
Individual count	30
Annotation Property count	18
Class axioms	
SubClassOf	97
EquivalentClasses	2
DisjointClasses	2
GCI count	0
Hidden GCI Count	3
Object property axioms	
SubObjectPropertyOf	31
EquivalentObjectProperties	0
InverseObjectProperties	7
DisjointObjectProperties	0
FunctionalObjectProperty	0
InverseFunctionalObjectProperty	0
TransitiveObjectProperty	1
SymmetricObjectProperty	0

Individuals:



The individuals list shows 30 entries, each represented by a purple diamond icon followed by a label. The labels include:
 A1@en
 Aarhus
 Common temperatures@en
 France@en
 France@fr
 Hygromtrie@en
 Hygromtrie@fr
 I1@en
 La ville Lumire@fr
 Output T2@en
 Output T1@en
 Output T3@en
 P1@en
 Paris@fr
 Platform1@en
 Pluviomtrie@en
 Pression atmosphrique@en
 Range A@en
 Range B@en
 Sonde T1@en
 Sonde_P1@en
 Temperature@en
 TemperatureSensor_1@en
 TemperatureSensor_2@en
 TemperatureSensor_3@en
 Tempature
 Tempatures positives
 Toulouse@en
 Vitesse du vent@en
 Vitesse du vent@en

Conclusion

Ce TP a été une réelle opportunité de comprendre les bases même du Web sémantique en introduisant la notion de triplets de données avec :

- Les propriétés d'objets
- Les propriétés de donnée

On peut d'ailleurs définir la différence entre propriété objet et propriété de donnée : La différence réside notamment dans le fait qu'une propriété objet relie un individu à un autre tandis tout autant complexe (notion de classe) tandis que la propriété de donnée vient lier un individu à un littéral simple (string, Integer etc.).

Une autre notion au cœur même des deux TPs correspondait au concept d'URI. Finalement toute ressource détient une URI, qu'elle soit une donnée, un type ou même une propriété.

Enfin, ces deux TPs nous ont fait comprendre que plus les données sont liées entre elles (de manière organisée bien sûr) plus nous donnons l'opportunités à des programmes raisonneurs de rapidement analyser et déduire des informations tout en économisant des requêtes Web. Ce qui permet non seulement une optimisation du traffic de données Web mais aussi une interopérabilité des systèmes. A savoir que le sémantique Web est une extension du Web et que par conséquent il repose sur le protocole http, permettant son utilisation quelque soit la technologie utilisée : ce qui est très important pour l'internet des objets.

Nous tenons enfin à remercier notre professeur encadrant pour nous avoir donné l'opportunité de saisir les enjeux du Web sémantique ainsi que de l'importance de bien structurer et lier les données tout en les rendant disponibles et interopérables.