INNOVATIVE SMART SYSTEMS

INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES TOULOUSE

Compte-rendu de TP

Web sémantique & IoT

Elie Taillardat
Jonathan Malique

PTP ISS
23/11/2018

SOMMAIRE

INTRODUCTION

Dans le cadre de notre formation de dernière année à l'INSA de Toulouse, nous faisons partie d'un parcours pluridisciplinaire centré sur l'innovation et les objets connectés. Une partie de cette formation concerne donc les données sémantiques.

À la suite du cours dispensé, nous avions deux TPs à effectuer, permettant d'introduire la **manipulation d'une ontologie** (étapes de développement). Le premier TP concernait une ontologie simple de météo à l'aide du logiciel Protégé, pour découvrir les principaux éléments de celle-ci. Le deuxième portera sur la description d'un *dataset* ouvert par une ville du Danemark, puis sur la définition d'un capteur défectueux.

Ce rapport résume donc les différentes étapes effectuées lors de chaque TP pour arriver à notre objectif final et maîtriser le « Web sémantique » pour l'appliquer aux objets connectés.

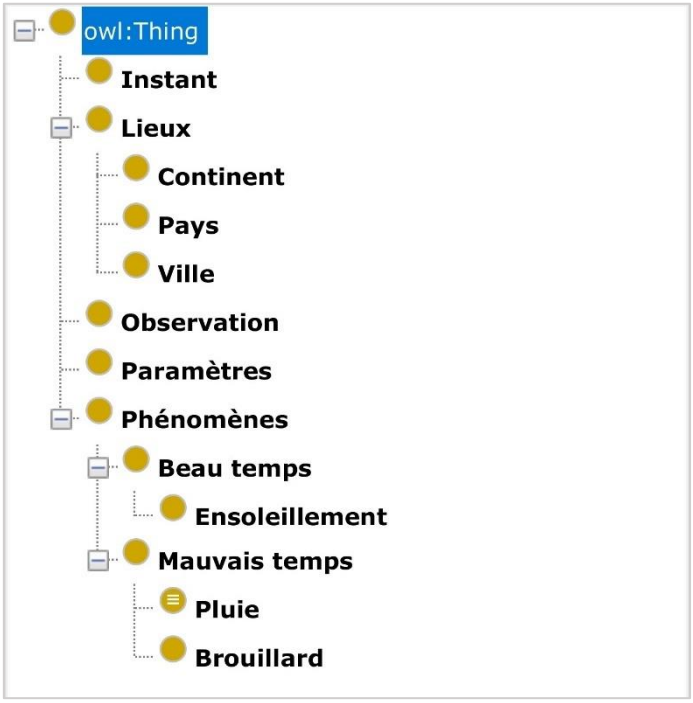
TP N°1 : CREATION D'UNE ONTOLOGIE AVEC PROTEGE

Pour commencer, nous avons installé le logiciel Protégé sur nos machines personnelles et pris connaissance du sujet. Ce premier TP porte sur une ontologie de **météo** (définir les différents événements climatiques).

A. CONCEPTION DE L'ONTOLOGIE LEGERE

1. CREATION DES CLASSES

La première étape de la création d'une ontologie est la **définition des classes**. En partant des connaissances du sujet nous avons créé différentes classes :



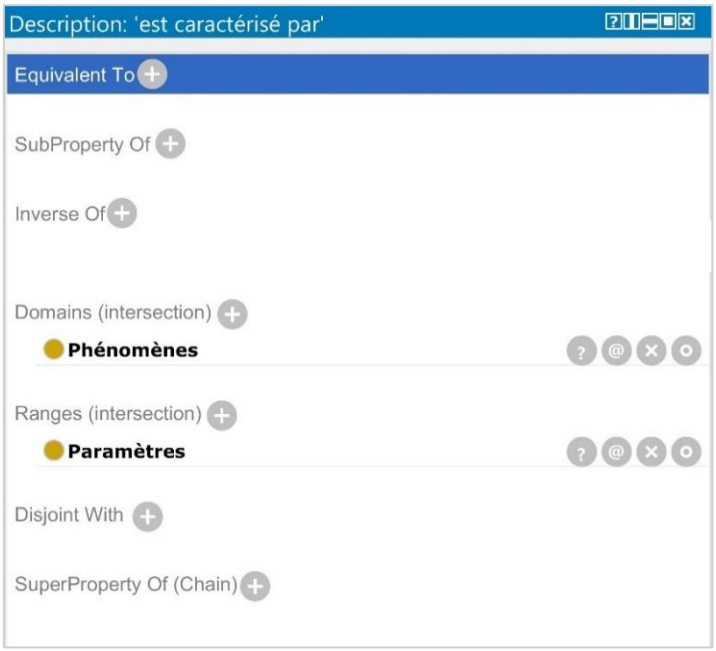
Nous avons donc décidé de créer **5** grandes classes caractérisant une météo : l'heure, l'endroit, l'observation, les paramètres mesurés et le phénomène. Le beau temps et le mauvais temps sont des sous-classes des phénomènes, comme le continent, le pays et la ville le sont pour les lieux (nous les avons définis disjoints les uns des autres, car un continent ne peut pas être une ville, ni un pays également sauf cas rare). On peut en effet définir des contraintes grâce aux **axiomes logiques** qui permettent d'établir des relations entre les classes.

2. AJOUT DES PROPRIETES ET CONCEPTS

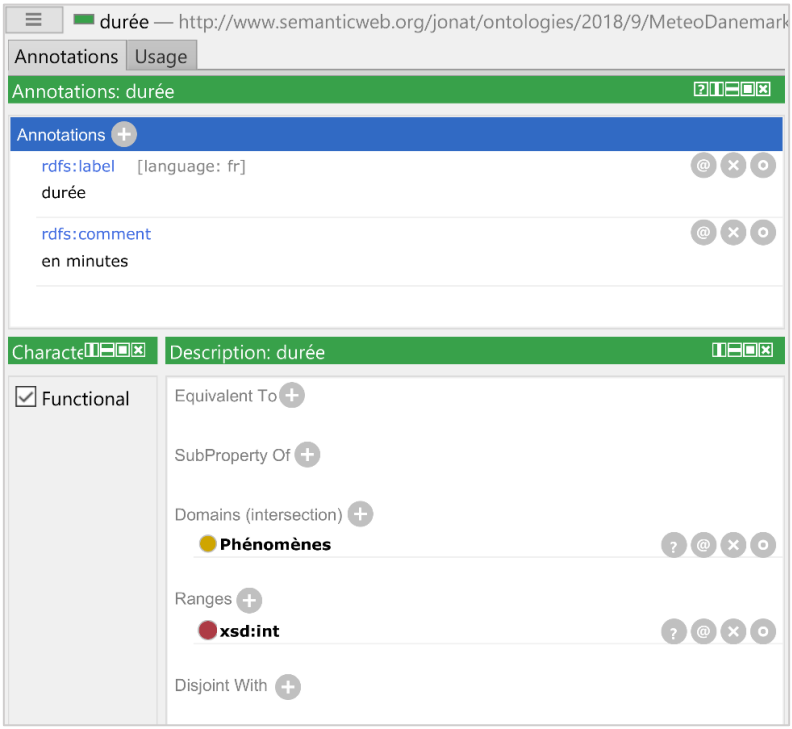
Les propriétés permettent d'associer des **classes entre elles** ou une **variable à une classe**.

- Pour la première propriété, *« Un phénomène est caractérisé par des paramètres mesurables »*, elle concerne deux classes, on se place donc dans l'onglet *Object properties* et on définit la

propriété '*est caractérisé par*'. On donne ensuite la description *Domains* (la première classe, Phénomènes) puis *Ranges* (la seconde classe, Paramètres) :

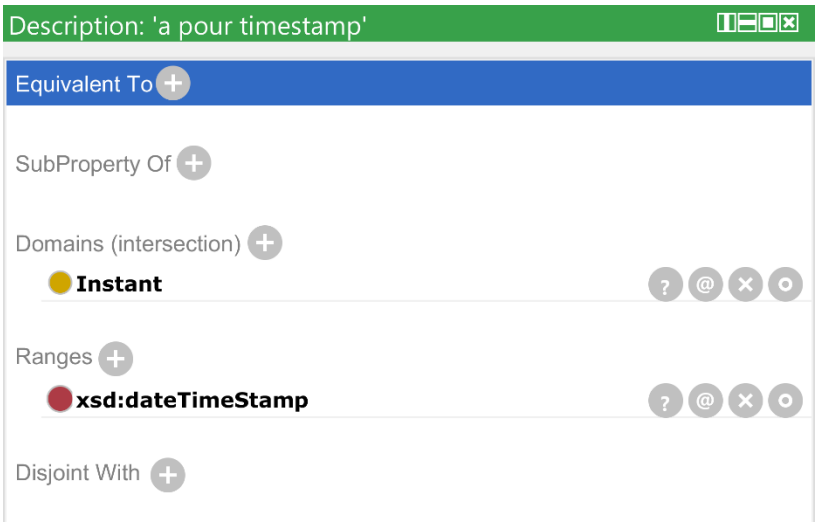


- La seconde propriété, *« Un phénomène a une durée en minutes »,* nécessite l'association d'une variable à une classe. Pour cela, on se place dans l'onglet *Data properties* pour ajouter la propriété/variable '*Durée*'. Nous avons décidé de lui donner un commentaire pour connaître son unité (en minutes) et un *Ranges* pour définir le type de variable (chiffre entier Integer) :



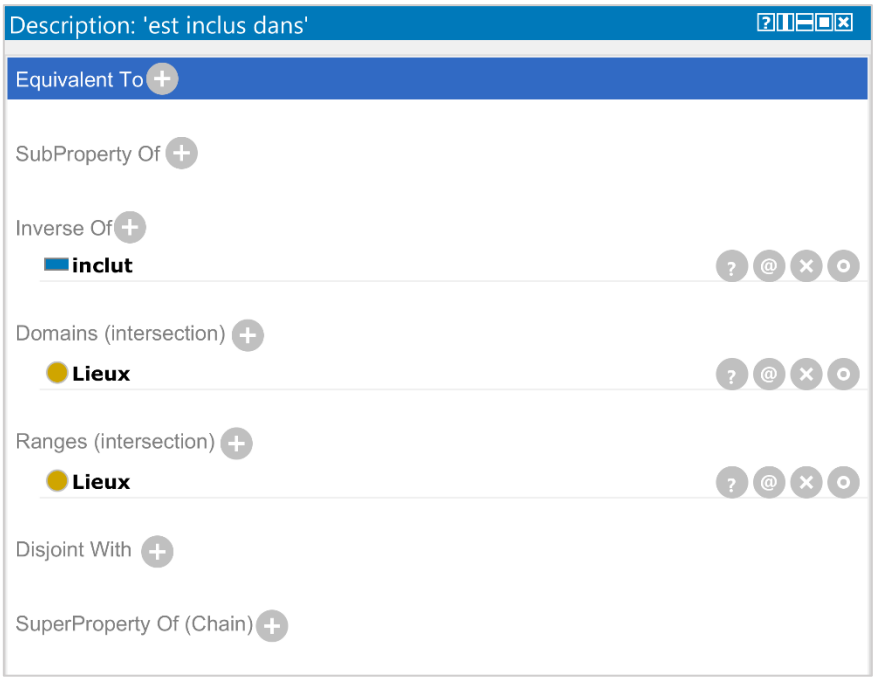
Nous avons également donné la caractéristique ***Functional*** car un phénomène ne peut avoir qu'une seule durée.

- Les trois propriétés suivantes concernent l'instant du phénomène observé. Nous avons créé 2 propriétés d'objet pour lier la classe phénomène et la classe instant : '*débute à*' et '*finit à*'. Ensuite, nous avons défini une propriété de donnée '*a pour timestamp*' qui permet de dire qu'un instant correspond à une date et heure du calendrier, et qui a donc pour type *dateTimeStamp* :

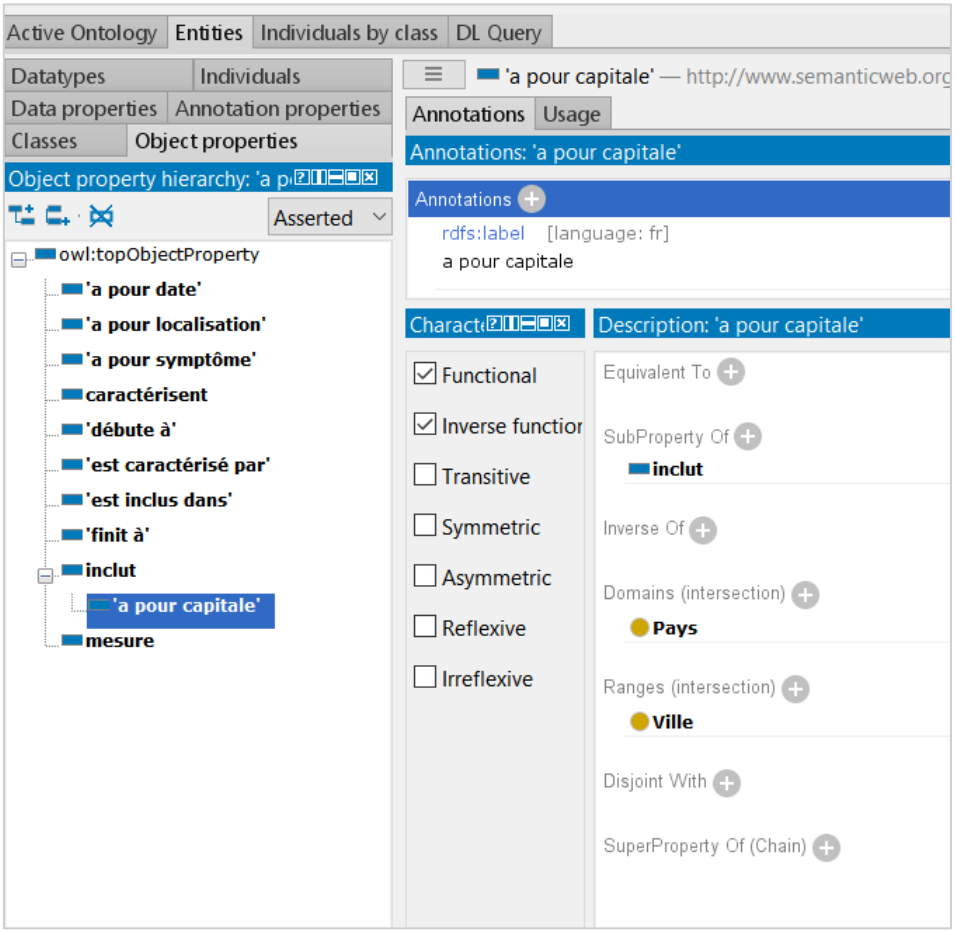


- Par la suite, nous avons fini de définir les propriétés d'objet basique, '*a pour symptôme*', '*mesure*', '*a pour localisation*' et '*a pour date*', ainsi que la *Data property* '*valeur*' qui n'a pas d'unité et est liée à la classe *Observation*.

- Concernant les propriétés *d'un lieu inclut dans un autre lieu* ou qui peut inclure un autre lieu, nous avons défini les propriétés d'objet *'est inclus dans'* (caractéristique **transitive**) et *'inclut'* (entre la classe *Lieux* et elle-même) qui sont des propriétés inverses, définies comme cela, dans *Inverse Of* :



- Enfin, *« un pays a pour capitale une ville »* est une sous-propriété **d'inclut**, en effet un pays, qui est un lieu, inclut une ville (qui est aussi un lieu). Aussi, cette propriété est *Functional* et *Inverse Functional* (car un pays n'a qu'une seule capitale et une ville ne peut être incluse que dans un seul pays). Finalement, nous avons donc cet ensemble de propriétés objet et de données :

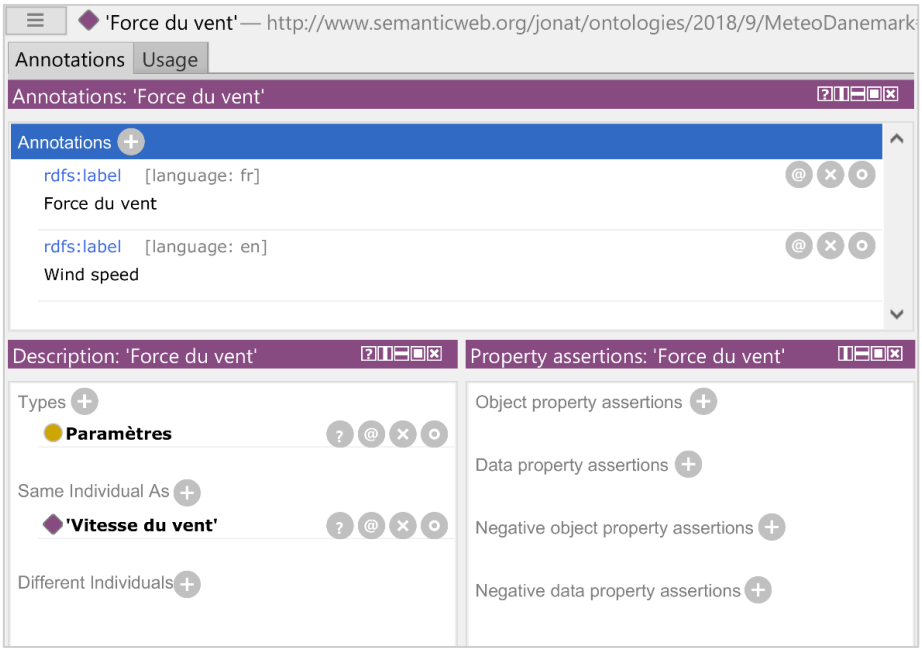


B. PEUPLEMENT DE L'ONTOLOGIE LEGERE

À partir de l'ontologie que nous venons de construire, nous pouvons maintenant représenter des événements, des instances. On ajoute tout d'abord les instances de la classe *paramètre* :

Nous voulons que *température* soit un synonyme de *temperature* en anglais. Pour cela, il faut se rendre dans les annotations de l'instance *Température* (dans l'onglet *Individuals*) et ajouter autant de labels que de langues souhaitées.

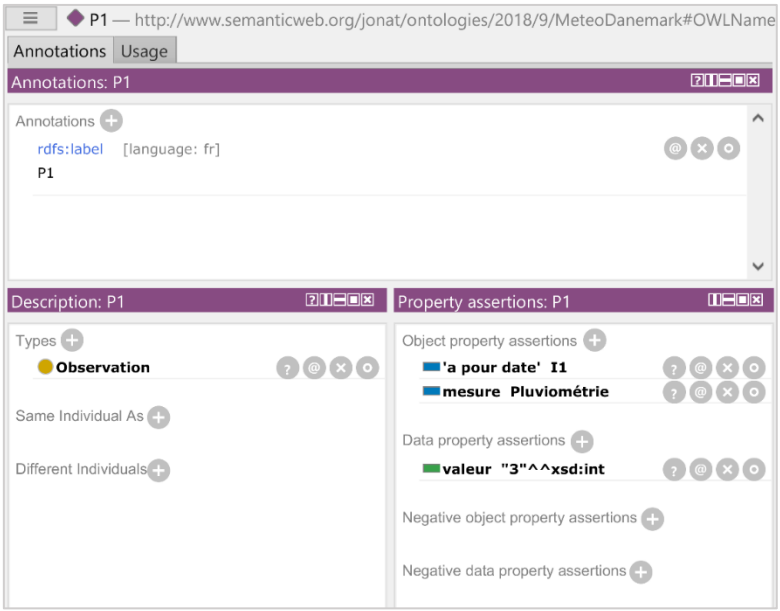
Toujours dans *Individuals* et l'onglet *Annotations* se trouve une partie description où l'on peut rajouter un élément à *« Same Individual As »* pour signifier que deux instances sont similaires. Par exemple, pour la *force du vent* similaire à la *vitesse du vent* (et *Wind speed* en anglais) on obtient :



Par la suite, en créant *Toulouse* et *France* comme deux instances globales et en définissant *Toulouse* comme incluse en *France*, le raisonneur Hermit lancé déduit de lui-même que *Toulouse* et *France* sont des lieux.

En créant *Paris* non typé et en associant le lien *a pour capitale* entre la *France* et *Paris*, le raisonneur déduit que *Paris* est inclus dans la *France* mais aussi que *Paris* est une ville (car le lien *'a pour capitale'* utilise les classes *Pays* et *Ville*. Il donne également la classe *Pays* pour la *France* grâce à cette propriété.

Pour le fait *« P1 est une observation qui a mesuré la valeur 3 mm de pluviométrie à Toulouse à l'instant I1 »*, nous avons défini une instance P1 et lui avons donné les propriétés suivantes :



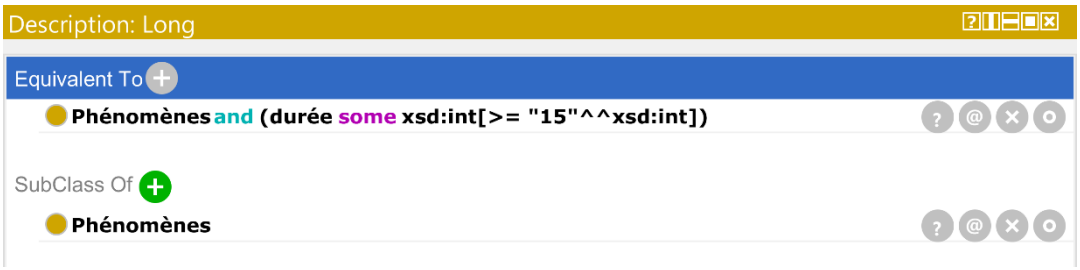
A1 qui est une instance non typée et qui a pour symptôme *P1* est automatiquement catégorisée dans la classe des *Phénomènes* par le raisonneur car la propriété *'a pour symptôme'* se fait uniquement entre un phénomène et une observation, le raisonneur peut donc en déduire la classe de *A1*.

C. CONCEPTION DE L'ONTOLOGIE LOURDE

Nous avions déjà commencé quelques parties de l'ontologie lourde dans l'ontologie légère. Par exemple, nous avions déjà utilisé la propriété *Disjoint With* entre les classes *Pays*, *Ville* et *Continent*. Nous avons aussi déjà utilisé les caractéristiques inverses entre les lieux inclus les uns dans les autres et la caractéristique *Transitive* pour l'imbrication des lieux.

- Ensuite, nous avons défini deux nouvelles sous-classes de phénomènes : *Long* et *Court* qui sont disjoints pour ne pas avoir un phénomène court et long à la fois. On a ensuite codé en Manchester leurs propriétés :

  - Un phénomène long doit durer au moins 15 minutes :



  - Un phénomène court si il est de moins de 15 minutes : **Phénomènes and (durée some xsd:int[< "15"^^xsd:int])**

- La dernière propriété de l'ontologie lourde, « La pluie est un phénomène ayant pour symptôme une observation de pluviométrie dont la valeur est supérieure à 0 » s'écrit :

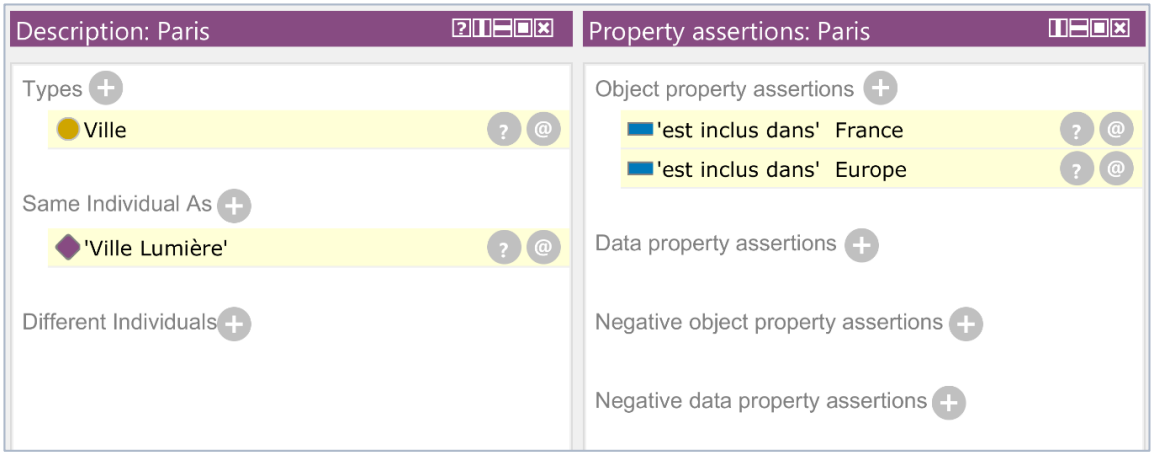
'Mauvais temps' and ('a pour symptôme' some (Observation and (mesure value Pluviométrie) and (valeur some xsd:int[> "0"^^xsd:int])))

D. PEUPLEMENT DE L'ONTOLOGIE LOURDE

Pour le peuplement de l'ontologie lourde on doit créer les instances et les liens permettant de confirmer que :

- La France est en Europe, le raisonneur en déduit que l'Europe est un lieu (grâce à la propriété '*est inclus dans*'
- Paris est la capitale de la France

- La Ville Lumière est la capitale de la France :



  Le raisonneur sait donc que *Paris* est la même instance que *Ville Lumière* car la propriété '*a pour capitale*' est une propriété ***Functiona****l*, il ne peut donc y a voir qu'une capitale par pays.

- Singapour est une ville et un pays : problème, nous avons créé des classes disjointes *Pays* et *Villes*, la même instance ne peut donc pas être à la fois un pays et une ville, le raisonneur affiche une erreur.
  Pour régler le problème on peut soit supprimer le fait qu'un pays ne peut pas être une ville, en enlevant la partie *Disjoint With*, soit créer deux instances avec le même nom Singapour en donnant à l'une le type *Pays* et l'autre le type *Ville*.

Si l'on définit *Toulouse* comme la capitale de la *France*, alors pour le raisonneur *Toulouse* est la même instance que *Paris* et *Ville Lumière*, sous un nom différent.

* * *

On peut conclure qu'en utilisant les *domain/range* d'un *object property*, le raisonneur peut trouver le type d'une instance si **l'autre instance a déjà un type** ou si la *property* utilise **deux classes de même type**.

Un ***object property*** crée une relation qualitative entre deux classes alors qu'une ***data property*** crée une donnée quantitative associée à une classe, le raisonneur associe donc une variable à une classe.

Enfin, la syntaxe **Manchester** permet de décrire plus facilement les ontologies, on peut ajouter plusieurs informations à une classe, plusieurs propriétés de manière rapide.

TP N°2 : ENRICHIR LES DATA ET UTILISER LE RAISONNEUR JAVA

Dans ce TP nous avons utilisé l'ontologie créée au premier TP pour annoter et enrichir un ***dataset*** d'une ville au Danemark. L'objectif étant ensuite d'exploiter cette version enrichie pour vérifier les propriétés des capteurs (savoir si l'on peut faire confiance).

Nous avons dû tout d'abord installer l'environnement en utilisant **git** avant de télécharger le *dataset*. Nous sommes partis d'une base de code implémentant une librairie de manipulation pour le web sémantique en Java - **Jena** – et nous avons complété les parties manquantes.

A. IMPLEMENTER LES INTERFACES

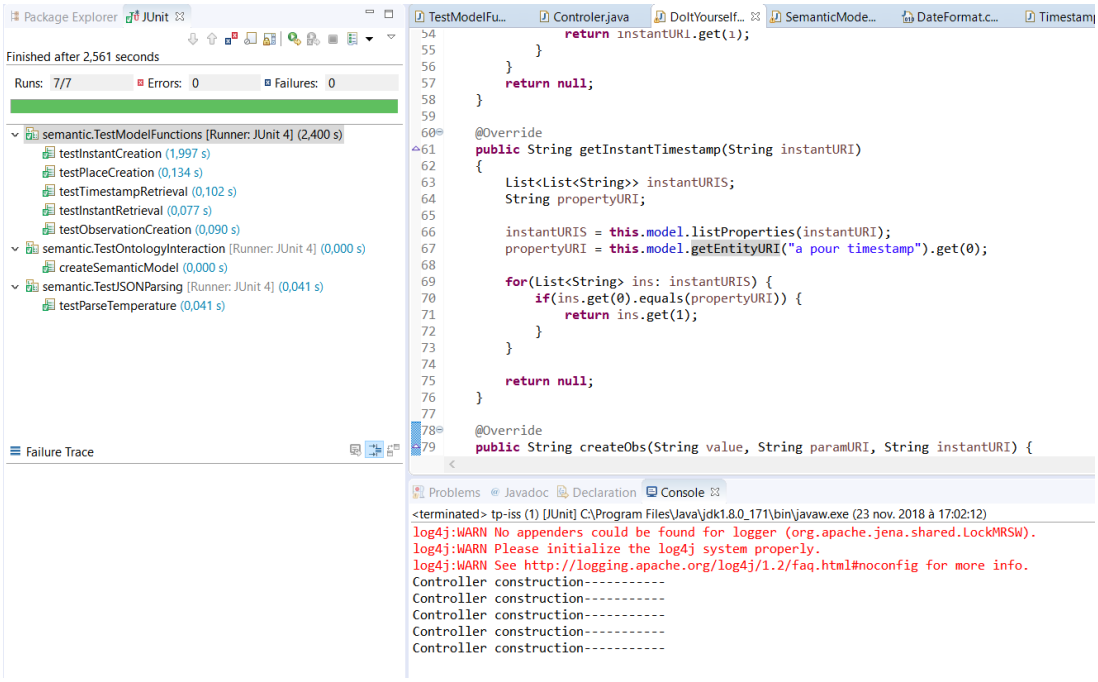
Nous avons d'abord implémenté les interfaces *IModelFunctions* liées au modèle sémantique, les fonctions définies dans *IConvenianceInterface* permettant d'interroger la base de données par des requêtes SPARQL.

- ***IModelFunctions***

```
public class DoItYourselfModel implements IModelFunctions
{

	IConvenienceInterface model;
	List<String> cache= new ArrayList();

	public DoItYourselfModel(IConvenienceInterface m) {
		this.model = m;
	}


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


	@Override
	public String createInstant(TimestampEntity instant) {

		String instantURI,propertyURI,result;
		result = null;
		boolean exi = false;

		for(String ts : cache) {
			if(ts.equals(instant.getTimeStamp())) {
				System.out.println("Instant already exists");
				exi = true;
			}
		}
```

```
		if(!(exi)) {
			cache.add(instant.getTimeStamp());
			instantURI = model.getEntityURI("Instant").get(0);
			result = model.createInstance("time", instantURI);
			propertyURI = model.getEntityURI("a pour timestamp").get(0);
			model.addDataPropertyToIndividual(result, propertyURI,
instant.getTimeStamp());

		}
		return result;
	}

	@Override
	public String getInstantURI(TimestampEntity instant) {
		List<String> instantURI, propertyURI;
		instantURI =
this.model.getInstancesURI(this.model.getEntityURI("Instant").get(0));
		propertyURI = this.model.getEntityURI("a pour timestamp");

		for(int i=0; i<instantURI.size(); i++) {
			if(this.model.hasDataPropertyValue(instantURI.get(i),
propertyURI.get(0), instant.timestamp)) {
				return instantURI.get(i);
			}
		}
		return null;
	}

	@Override
	public String getInstantTimestamp(String instantURI)
	{
		List<List<String>> instantURIS;
		String propertyURI;

		instantURIS = this.model.listProperties(instantURI);
		propertyURI = this.model.getEntityURI("a pour timestamp").get(0);

		for(List<String> ins: instantURIS) {
			if(ins.get(0).equals(propertyURI)) {
				return ins.get(1);
			}
		}
		return null;
	}


	@Override
	public String createObs(String value, String paramURI, String instantURI) {
		String obsURI, dataproperty, objectproperty, sensorURI;
		String timestamp = this.getInstantTimestamp(instantURI);
		obsURI = this.model.getEntityURI("Observation").get(0);
		String obs = this.model.createInstance("Observation", obsURI);
		objectproperty = this.model.getEntityURI("a pour date").get(0);
		this.model.addObjectPropertyToIndividual(obs, objectproperty, instantURI);
		dataproperty = this.model.getEntityURI("a pour valeur").get(0);
		this.model.addDataPropertyToIndividual(obs, dataproperty, value);
		sensorURI = this.model.whichSensorDidIt(timestamp, paramURI);
		this.model.addObservationToSensor(obs, sensorURI);
		return obs;
	}
}
```

Chaque fonction peut être testée grâce à des modèles de **test unitaire** qui permettent de vérifier si l'exécution du code retourne le résultat attendu.



- ***IControlFunctions***

Une fois les fonctions du modèle validées, nous avons commencé à coder les parties « **TODO** » du contrôleur, ayant pour objectif l'enrichissement effectif du *dataset.* Cependant, par manque de temps et de compréhension de la question une fois le TP repris en dehors du cours, nous avons préféré utiliser l'élément de correction donné pour vérifier le fonctionnement et analyser la rédaction de ce dernier. Ci-dessous le code retenu (servant de trace) :

```
public class DoItYourselfControl implements IControlFunctions
{
	IConvenienceInterface model;
	IModelFunctions customModel;

	public DoItYourselfControl(IConvenienceInterface model, IModelFunctions
customModel)
	{
		this.model = model;
		this.customModel = customModel;
	}

	@Override
	public void instantiateObservations(List<ObservationEntity> obsList,
			String paramURI) {
		// The key of this map is the timestamp of instants individuals, to avoid
creating
		// multiple individuals for the same instant
		Map<String, String> instantsMap = new HashMap<String, String>();
		// For each element of the list, create its representation in the KB
		int i = 0;
```

```
            for(ObservationEntity oe : obsList)
            {
                  i++;
                  System.out.println("Instantiating observation "+i);
                  String instantURI;
                  if(!instantsMap.containsKey(oe.getTimestamp().getTimeStamp()))
                  {
                        instantURI =
this.customModel.createInstant(oe.getTimestamp());
                        instantsMap.put(oe.getTimestamp().getTimeStamp(),
instantURI);
                  }
                  else
                  {
                        instantURI =
instantsMap.get(oe.getTimestamp().getTimeStamp());
                  }
                  this.customModel.createObs(oe.getValue().toString(), paramURI,
instantURI);
            }

      }
}
```

CONCLUSION

Pour conclure, les deux séances de TP étaient très **constructives** et bien structurées, et permettaient de cerner rapidement les concepts de **l'ontologie WEB**. Cela nous permet de même d'appréhender différents outils et de se familiariser avec ceux-ci.

La conception d'une ontologie demande de l'organisation et un certain **sens logique**. Le raisonneur permet de prendre compte ses éventuelles erreurs et de vérifier si l'on a bien défini tous les concepts que l'on souhaitait.

Enfin, nous avons pu avoir **quelques difficultés** pour comprendre quels éléments devaient être des classes ou instances au tout début, mais cela n'a pas été handicapant et nous avons réussi à bien avancer lors des deux TPs, malgré le manque de temps pour le second TP.