# TP : Web sémantique et internet des objets

## Objectif

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

## Contexte

Le contexte est le suivant : on veut développer une application de météorologie intelligente. Pour ce faire, on va décrire des stations météos pour rendre les données qui en sont issues le plus riches possibles.

## Programme pour les deux séances

Il s'agira dans un premier temps de définir une ontologie de la météo (très simple), contenant une notion “ad-hoc” de capteur. Vous utiliserez pour cela l'éditeur d'ontologie Protégé.[1] Cette **première étape** permettra de mettre en lumière les principaux constituants d'une ontologie.

Ensuite, vous enrichirez cette ontologie à l'aide de SSN, qui est une ontologie générique de description de capteurs et d'observations. Tout au long de ce processus, vous utiliserez le raisonneur Hermit disponible à partir d Protégé pour voir les déductions qu'il est possible de faire à partir de la connaissance que vous avez représentée.

La **deuxième étape** visera à décrire sémantiquement les données d'un dataset issu d'un open data : http://iot.ee.surrey.ac.uk:8080/datasets.html, ouvert par la ville d'Aarhus au Danemark. Ces données correspondant à des observations faites par des capteurs météorologiques vous seront fournies au format CSV. Vous devrez les convertirez en données 5 étoiles à partir de l'ontologie que vous avez construite lors de la première étape.

Enfin, vous serez amenés à vous intéresser de plus près à SSN pour y trouver des éléments qui vous aideront à définir formellement la notion de capteur défectueux. Une fois cette définition établie, les capteurs défectueux seront automatiquement identifiés à partir des données que vous avez enrichies.

# Seconde séance : Enrichissement de données et utilisation du raisonneur en Java

## 1- Démarche générale :

L'idée de ce TP est la suivante : vous allez réutiliser l'ontologie que vous avez fait la dernière fois, dans une version corrigée que vous récupèrerez avec le code du TP.  pour annoter un dataset disponible en open data ici, ouvert par la ville d'Aarhus au Danemark. Vous utiliserez ensuite la

1 http://protege.stanford.edu/

base de connaissance issue de ce processus pour déterminer si on peut avoir confiance dans toutes les mesures.

Pour ce faire, une base de code que vous devrez compléter est mise à votre disposition. Cette base de code ajoute une surcouche à une librairie de manipulation des principes du web sémantique en Java : **Jena**, distribué par Apache. Vous n'aurez cependant pas à vous en préoccuper, tout ce dont vous avez besoin vous est détaillé dans le code fourni.

Vous allez donc télécharger un dataset, écrire un programme pour enrichir et annoter ce dataset sémantiquement, et enfin exploiter cette version enrichie dans protégé afin de vérifier certaines propriétés des capteurs.

## 2- Mise en place de l'environnement :

Il y a deux façons de mettre en place l'environnement : soit via une machine virtuelle, soit en utilisant maven et git. **Choisissez l'une des deux**, sachant que la méthode git est mieux (et du coup l'autre n'est qu'un recours si le git échoue).

### 2.1 : La machine virtuelle

J'ai sur des clés USB une machine virtuelle VirtualBox, découpée en 2 pour faciliter le transfert. Si vous avez un système basé UNIX (Mac ou distrib Linux), vous pouvez récupérer les fichiers, et les remettre ensemble simplement en utilisant la commande cat : « cat vm_* > vm.ova ».

Assurez-vous d'allouer assez de RAM à la VM (2-3Go), pour y faire tourner l'IDE c'est mieux.

### 2.2 : Installation manuelle depuis git

La base de code dont vous aurez besoin pour le TP est disponible sur un git public du laas : https://redmine.laas.fr/laas/users/nseydoux/tp-iss.git. Un petit « git clone https://redmine.laas.fr/laas/users/nseydoux/tp-iss.git » plus tard, vous disposez d'un projet maven que vous pouvez compiler avec la commande « mvn compile » (il faut avoir installé maven), et pour lequel vous pouvez exécuter des tests avec 'mvn test'.

Ce projet contient une base de code que vous devrez compléter en implémentant les interfaces IControlFunctions et IModelFunctions, dans les classes DoItYourselfControl et DoItYourselfModel. Initialement, vous allez avoir plein d' erreurs (des nullPointerException entre autres) parce que rien n'est implémenté.

- Téléchargez ensuite le dataset que vous allez manipuler[2], issu d'une ville danoise : Aarhus. Plus d'informations sont disponibles ici : http://iot.ee.surrey.ac.uk:8080/index.html. C'est sur un sous-ensemble de ce dataset que portera le TP : les mesures de température, et les mesures d'humidité. Dans le dataset original, il n'y a pas d'informations sur les capteurs, celles-ci sont rajoutées via un morceau d'ontologie créé pour l'occasion.

2 http://iot.ee.surrey.ac.uk:8080/datasets/weather/feb_jun_2014/raw_weather_data_aarhus.tar.gz

### 2.3 : Vérification de l’installation

À ce moment-là du TP, quelle que soit la méthode choisie, appelez l’enseignant pour vérifier que le système mise en place est valide.

## 3- Implémentations des interfaces

Les interfaces IControlFunctions et IModelFunctions comportent une série de fonctions que vous devez implémenter, plutôt classées par complexité croissante. Elles sont fournies avec un ensemble de tests unitaires qui valident votre implémentation dans les grandes lignes.

**Commencez par implémenter les fonctions de IModelFunctions**, qui sont liées au modèle sémantique. Elles visent à interagir avec la base de connaissance. Un ensemble de fonctions est fourni dans une autre interface, IConvenienceInterface, qui offrent une surcouche à Jena et font pour vous les requêtes SPARQL sur la base de connaissance. Implémentez les fonctions les unes après les autres, et testez progressivement. Pour tester, deux méthodes s’offrent à vous :

- Soit vous développez sous eclipse, auquel cas vous pouvez faire clic droit sur le projet, Run as... Maven Test, et les résultats des tests s’afficheront dans la console

- Soit vous voulez lancer les tests en terminal, auquel cas utilisez la commande « mvn test »

Dans tous les cas, les tests n’affichent de retour que s’ils ont échoué, maven indiquant le nombre de test exécuté et le nombre de test qui échouent (à vous d’en déduire le nombre de tests qui réussissent). Vous pouvez vous référer au code des tests pour mieux comprendre ce que vous avez à faire.

Une fois ces fonctions implémentées, vous pouvez passer au contrôleur. Celui-ci a pour but d’utiliser les fonctions du modèle pour effectuer l’enrichissement effectif du dataset. Une fois l'interface complétée, vous trouverez le main du programme dans la classe Controler. Prêtez attention aux commentaires comportants des TODO, qui indiquent les morceaux du code à adapter à votre environnement.

L'implémentation étant faite à but pédagogique, vous remarquerez que les performances ne sont pas terribles, mais beaucoup d'optimisations pourraient être faites. De plus, dans notre cas, le graphe de connaissance généré est volatile, et entièrement géré en RAM, donc quand votre programme se termine, la base de connaissance disparaît : pensez à l'exporter si vous voulez la réutiliser par la suite. Pour plus de performances, pensez à limiter le nombre de requêtes sur la base de connaissances : si vous devez accéder plusieurs fois à une information, il peut être rentable de la stocker, comme par exemple certaines URI.

## 4- Exploitation dans Protégé

Importez votre modèle généré dans Protégé. L'objectif est maintenant de vérifier que les capteurs ont été correctement utilisés. SSN comporte un ensemble de classes et de propriétés qui servent à décrire les conditions d'utilisations nominales d'un capteur.

En vous appuyant sur la description de SSN proposée par le W3C ici : http://www.w3.org/2005/Incubator/ssn/XGR-ssn-20110628/#Ontology_structure, identifiez quelles classes permettent de définir les conditions nominales d'utilisation d'un capteur. Une fois ces

classes, ainsi que les relations qui les lient au reste de l'ontologie, identifiées, proposez un graphe qui caractérise un capteur utilisé hors de ses conditions initiales. Ce graphe pourrait servir à décrire une classe définie, mais l'absence de binding de variable en syntaxe de Manchester rend nécessaire l'utilisation d'un autre langage (par exemple, SWRL), et ce n'est pas le propos ici. Les Danois ont-ils judicieusement choisi tous leurs capteurs ?

## 5- Questions d'évaluation des compétences

- Justifiez succintement vos choix concernant l'enrichissement du dataset (choix des classes, des relations…)

- Expliquez en quelques phrases la différence entre object property et data property