VIETNAM NATIONAL UNIVERSITY (VNU)
INSTITUT FRANCOPHONE INTERNATIONAL (IFI)

Option : Systèmes Intelligents et Multimédia (SIM)

Promotion : XXI

Projet sur l'Ontologie et Web Sémantique

« Simulation d'un réseaux sociaux »

Auteurs :

MALLE Zoumana

MEDOU Daniel Magloire

RAKOTOARIVELO Anjara Nobby

Encadrant :

Monsieur TA Tuan Anh

année académique 2017-2018

TABLE DES MATIÈRES

LISTE DES FIGURES

LISTE DES TABLEAUX

LISTE DES ALGORITHMES

1 Introduction

Au terme de l'unité d'enseignement intitulée : **ONTOLOGIE ET WEB SEMANTIQUE**, il a été soumis à notre étude un projet donc la réalisation dans notre cas devra être effectuée en trinôme. L'objectif général de ce dernier étant de simuler un réseau social tout en s'inspirant de l'une des technologies du web sémantique suivante RDF, OWL et XML. Dans le cadre de ce projet et de notre groupe en particulier, nous avons choisi d'implémenter une application de simulation d'un réseau social en utilisant l'approche du modèle RDF (Resource Description Framework). Il est également à noter l'utilisation de la librairie python **Rdflib** pour la construction du RDF et **Jena Serveur** pour le déploiement du fichier RDF. A la suite de ce rapport, il vous sera présenté de façon détaillée les différentes parties que constituent notre projet et programme et une partie expérimentations.

1.1 Contexte

L'influence grandissante des nouvelles technologies de l'information et de la communication à l'air des réseaux sociaux, ont littéralement changer notre vision sociale ou les barrières des liens sociaux peuvent être déterminer à travers d'algorithme structurer de données. L'explosion des fonctionnalités sociales au sein des applications du Web a favorisé le déploiement d'un panorama de médias sociaux permettant aux utilisateurs de librement contribuer, de se regrouper et d'interagir entre eux. La combinaison de divers moyens de publication et de socialisation permet de rapidement partager, recommander et propager l'information dans son réseau social, ainsi que de solliciter des réactions et de nouvelles contributions. Ces espaces partagés ont favorisé la création et le développement de communautés d'intérêts qui publient, filtrent et organisent de vastes répertoires de références dans leurs domaines, avec une impressionnante réactivité aux changements. Afin de reproduire les succès du Web dans la gestion d'information, de plus en plus de plates-formes sociales sont déployées dans des intranets d'entreprise. Cependant, l'avantage de ces plates-formes est fortement atténué lorsque le réseau social devient si grand que les informations pertinentes sont noyées dans des flux continus de notifications. Organiser cette énorme quantité d'informations est l'un des défis majeurs du Web 2.0 afin de tirer pleinement partie des bénéfices de l'Entreprise 2.0, à savoir, l'utilisation des technologies du Web 2.0, tel que les blogs et les wikis, dans un intranet.

1.2 Problématique

L'analyse des réseaux sociaux propose des algorithmes de graphes complexes pour caractériser la structure d'un réseau social et ses positions stratégiques. Les technologies du Web Sémantique permettent de représenter et d'échanger les connaissances entre des applications distribuées sur le Web avec un modèle de graphes richement typés (RDF), un langage de requête (SPARQL) et des langages de description de modèles (RDFS et OWL).Le web change et, avec lui, changent les enjeux de l'éditorialisation. Ainsi, nous sommes passés du web statique des origines (1.0) au web participatif (2.0), puis au web sémantique (3.0) qui permet aux machines de comprendre la signification des données et de mieux les exploiter. Nous considérons que le passage au web sémantique

est un enjeu majeur. Dans ce domaine, ce sont nos choix qui détermineront la structuration des connaissances dans le futur. Ce passage n'est pas neutre et comporte une série de questionnements politiques, philosophiques, économiques, sociaux et techniques.

1.3 Objectifs et problème a résoudre

Les interactions des utilisateurs au travers des usages du web 2.0 amènent la communauté scientifique à réfléchir sur les moyens de capter ces usages pour y appliquer les techniques d'analyse des réseaux sociaux. Les applications bien connues à l'origine de l'émergence du web 2.0 sont les blogs, les wikis (ex : wikipedia), les services de social bookmarking (ex : del.ico.us), les sites de partages de médias (ex : youtube, flickr) et bien sûr les sites de réseaux sociaux (ex : facebook, LinkedIn). Ces applications ont considérablement accru la participation, les interactions et le partage entre les utilisateurs du web. L'analyse et la compréhension de tels réseaux sociaux suscitent de vifs intérêts au sein de plusieurs communautés scientifiques. Le web sémantique fournit des formalismes pour la représentation sémantique des personnes et de leurs usages sur le web.Ainsi l'objectif du-dit programme est fondé essentiellement sur l'orientation d'informations principalement afin d'effectuer une simulation à travers une base de connaissance de déterminer les potentiels liens sociales dérivant de l'approche des réseaux sociaux. le système permet d'aider un utilisateur entre autres :

- - Retracer l'arbre généalogique sociale d'une personne donnée
- - Détermination des liens de parentés directe ou indirecte de n'importe quelle personne contenue dans la base de connaissance sociale pré-établie.
- -Web social plus structuré et organisé, inter-opérable grâce aux ontologies et RDF
- - faciliter la description, l'échange et la portabilité des données
- -Donner plus de sens à l'activité des utilisateurs dans les réseaux sociaux

1.4 Plan de travail

2 Analyse du sujet

Il existe plusieurs moyens pour résoudre notre sujet, mais comme déjà citer ci-haut dans l'introduction, le meilleur moyens pour concevoir notre système est d'utiliser l'approche sémantique et ontologique. L'arbre ontologique et l'engagement sémantique qu'il explicite mettent à disposition des primitives. Il est alors possible de représenter formellement des connaissances puisque la modélisation ontologique a dégagé ce que la formalisation des connaissances présuppose comme pré requis. Il devient possible de définir une sémantique formelle pour les concepts et de retrouver les propriétés auxquelles on est habitué en représentation des connaissances.Il devient possible de définir un terme par une sémantique référentielle, c'est-à-dire une sémantique formelle.les concepts formels vérifient les relations d'identité unissant les concepts sémantiques. L'engagement ontologique permet ainsi de définir ce que nous appelons une ontologie formelle ou ontologie référentielle. Une ontologie référentielle est constituée de prédicats formels pourvus d'une sémantique référentielle [Charlet 2002].

2.1 Description du projet

Le projet est constitué de 7 entités qui sont :

Personne : une personne dans le réseaux sociaux

Homme : les personnes de sexe masculin

Femme : les personnes de sexe féminin

Ville : la ville où habite une personne

Pays : le pays où se situe la ville

École : l'école où les étudiants étudient

Entreprise : l'entreprise où les personnes travaillent

2.2 Diagramme des cas d'utilisations

Le Diagramme va nous permettre d'énumérer les futures fonctionnalités de l'application que nous allons développer, représenter par la (FIGURE 1)

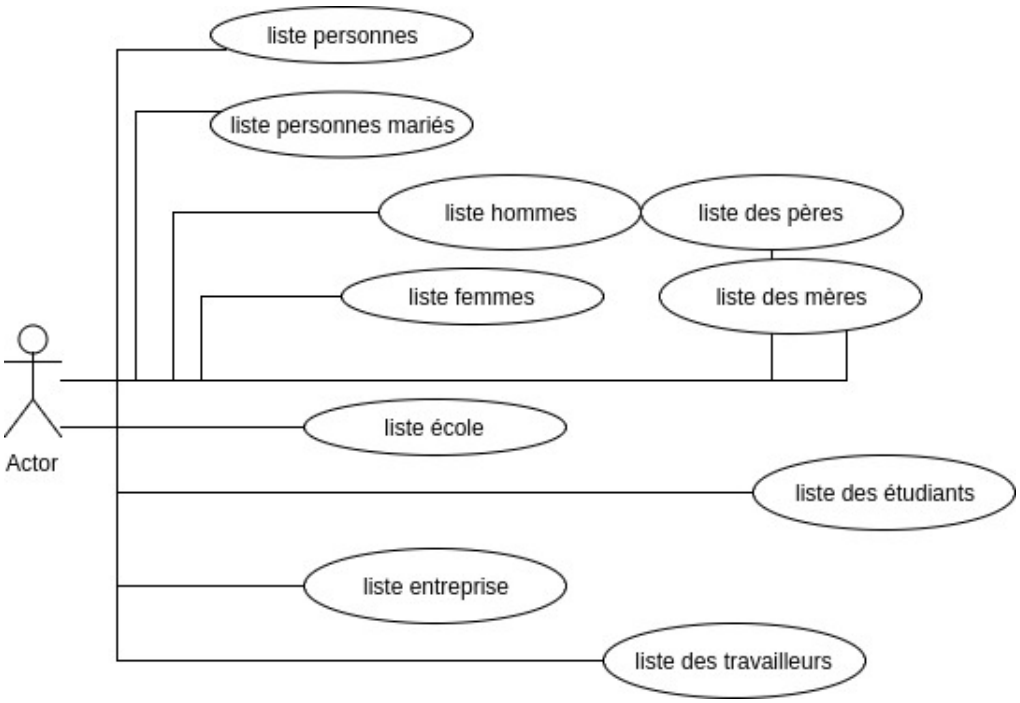

FIGURE 1 – Diagramme des cas d'utilisations

2.3 Description des cas d'utilisations

(TABLEAU 1)

Numéro	Nomination	Description
Cas 1	Lister personnes	Lister tous les personnes enregistrer dans la base de connaissance.
Cas 2	Lister hommes	Lister tous les hommes enregistrer dans la base de connaissance.
Cas 3	Lister femmes	Lister tous les femmes enregistrer dans la base de connaissance.
Cas 4	Lister hommes mariés	Lister tous les hommes mariés enregistrer dans la base de connaissance.
Cas 5	Lister femmes mariés	Lister tous les femmes mariés enregistrer dans la base de connaissance.
Cas 6	Lister écoles	Lister tous les écoles enregistrer dans la base de connaissance.
Cas 7	Lister entreprises	Lister tous les entreprises enregistrer dans la base de connaissance.
Cas 8	Lister pères	Lister tous les pères enregistrer dans la base de connaissance.
Cas 9	Lister mères	Lister tous les mères enregistrer dans la base de connaissance.
Cas 10	Lister étudiants	Lister tous les étudiants enregistrer dans la base de connaissance.
Cas 11	Lister travailleurs	Lister tous les travailleurs enregistrer dans la base de connaissance.

TABLE 1 – Description des cas utilisations

2.4 Termes techniques

Ontologie : A l'origine domaine philosophique de la « science de l'être en tant qu'être », une ontologie est, selon l'entendement du Web sémantique, un ensemble structuré de savoirs. Une ontologie définit les termes employés pour décrire et représenter un domaine de connaissance.

Métadonnée : Une métadonnée est une information permettant de décrire une autre information, quels qu'en soient la nature et le support.

Jena Framework : pour le web sémantique en Java, Jena fournit un environnement permettant de travailler avec RDF, RDFS et OWL.

HTML : HyperText Markup language

RDF : Ressource Description Framework, permet de présenter des données et des métadonnées.

RDFS : RDF Schema, permet de définir des vocabulaires RDF.

XLM : eXtensible Markup Language.

Littéral : une chaîne de caractères qui peut être la valeur d'une propriété.

Objet : la partie d'un triplet qui est la valeur de la déclaration.

Prédicat : la partie de la propriété dans un triplet.

Propriété : une propriété est un attribut d'une ressource.

Ressource : certaines entités. Elle pourrait être une ressource Web comme une page Web, ou une chose physique concrète comme un arbre ou une voiture. Elle pourrait être une idée abstraite comme les échecs ou le football. Les ressources sont nommées par une URI.

Déclaration : un arc dans un modèle RDF, normalement interprété comme un fait.

Sujet : la ressource qui est la source d'un arc dans un modèle RDF.

Triplet : une structure contenant un sujet, un prédicat et un objet. Un autre terme pour une déclaration.

2.5 Approches choisis

Dans le cadre de notre projet, l'approche donc nous avons jetté notre dévolu est le RDF. Ayant comme définition de son sigle en anglais qui est Resource Description Framework qui trouve sa définition en français comme étant un standard (techniquement une recommandation du W3C) pour la description de ressources. RDF est, un modèle nous permettant de décrire des métadonnées pour les ressources. Ce pendant dans la phase théorique c'est à dire cours du module, Nous nous concentrons sur trois langages d'ontologie qui ont été proposées pour décrire les ressources Web. Ces derniers sont RDF, RDFS et OWL. Nous considérons d'abord le langage RDF, un langage pour exprimer les faits (se concentrant principalement sur la base de données). Les deux autres langages permettent RDF dans les faits limitant les domaines d'application particuliers : RDFS est assez simple, alors que OWL est beaucoup plus riche. RDF (Resource Description Framework) fournit un langage simple pour décrire les annotations sur les ressources Web identifiées par URIs. Ce sont des faits. Les contraintes sur ces faits dans des domaines particuliers seront indiqués dans RDFS ou OWL. Au vue de cette importance simpliste qu'est RDF, et son importance par rapport aux deux autres (RDFS et OWL), nous avons choisi de par sont apport au autres langages de mettre en pratique notre projet avec le langage RDF.

3 Conception

3.1 Diagramme de classe système

Le diagramme de classe système de notre application est représenté par la (FIGURE 2)

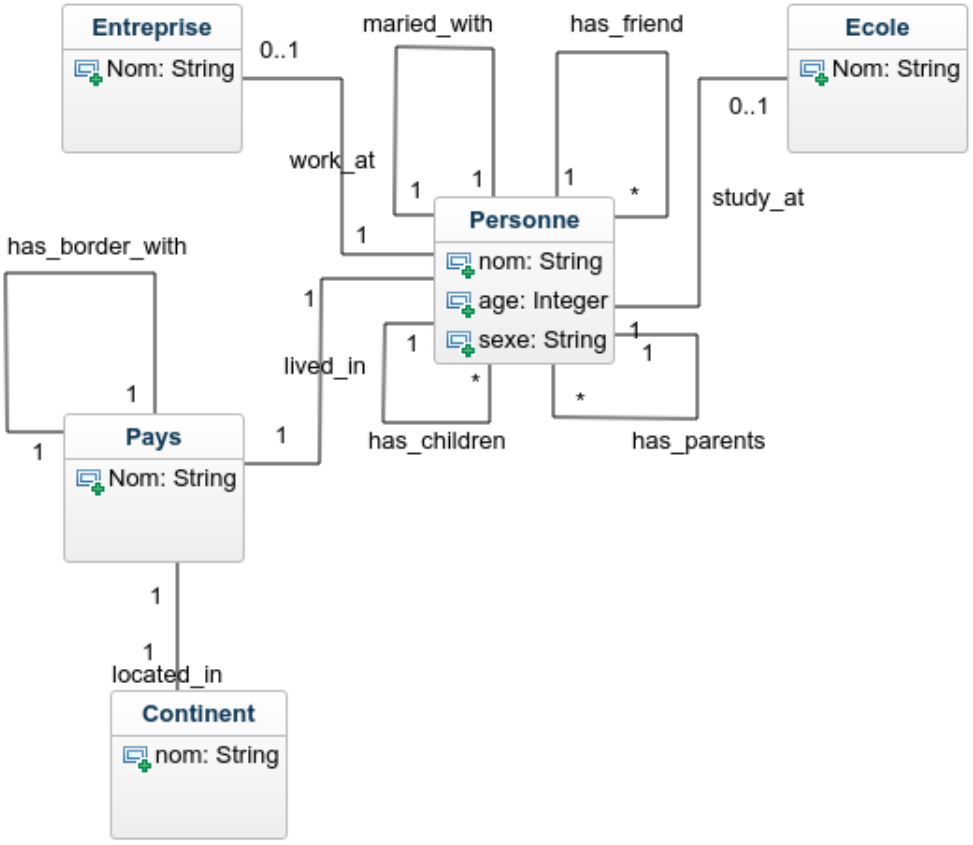

FIGURE 2 – Diagramme de classe système

3.2 Ontologie du système

(FIGURE 3)

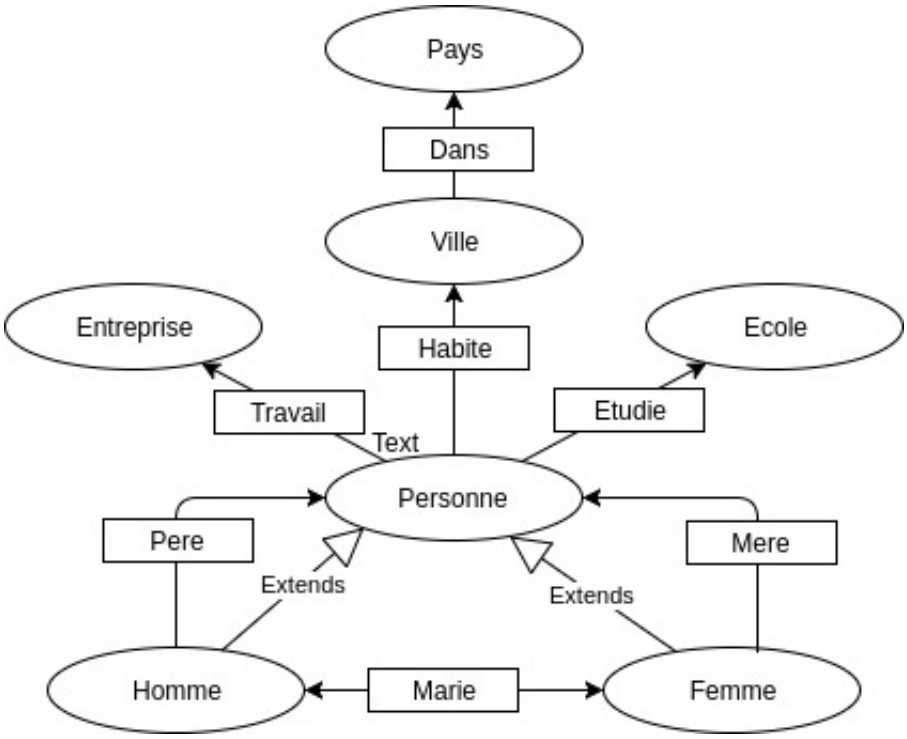

FIGURE 3 – ontologie du système

4 Implémentions

4.1 Représentation des triplet

Toute information en RDF est représentée par un triplet (sujet, prédicat et objet). Dans le cas de notre projet, il existe 9 triplets qui sont :

4.2 Environnement matériel

Pour la conception et les expérimentations de la simulation, nous avons utilisé un ordinateur portable « ASUS » avec les caractéristiques suivantes :

Processeur : Intel(R) CoreT M i5-M370 @2.4 GHZ

RAM : 4.00 Go

OS : UBUNTU 16.4

4.3 Environnement logiciel

Pour la réalisation de notre projet, les outils et frameworks cité ci-dessous ont été d'une importance capitale. Nous avons entre autre :

Éditeur d'ontologies RDFLib : Bibliothèque python pour la création de la base d'une onthologie RDF.

Framework JENA : Jena est un framework Java pour la construction d'applications Web sémantiques. Il fournit un environnement programmatique pour RDF, RDFS et OWL, SPARQL et inclut un moteur d'inférence basé sur des règles. En effet, Jena est une API Java qui peut est utilisé pour créer et manipuler des graphes RDF. Dans le cadre de notre projet, il nous a permit de réaliser le processus de récupération des informations contenues dans le fichier OWL généré par Protégé.

Framework JENA Fuseki Serveur : Apache Jena Fuseki est un serveur SPARQL. Ce dernier peut fonctionner en tant que service du système d'exploitation par exemple une application Java Web (fichier War), et en tant que serveur autonome. Il assure la sécurité (en utilisant Apache Shiro) et dispose d' une interface utilisateur pour la surveillance et l' administration serveur. Il peut être utilisé pour fournir le moteur de protocole pour d' autres systèmes de requête et de stockage RDF.

RDF (Ressource Description Framework) : Quand à lui n'est pas à proprement parler un langage. Il s'agit plutôt d'un modèle de données pour décrire des ressources sur le web. Ici, nous entendons par ressource, tout entité que nous voulons décrire sur le web mais qui n'est pas nécessairement accessible sur le web.

XML : Le XML ou eXtensible Markup Language est un langage informatique de balisage générique. En d'autre terme, XML est un langage qui permet de décrire des données à l'aide de balises et de règles que l'on peut personnaliser.

Langage de programmation : JAVA pour la partie cliente et Python pour la partie création de la base et la aussi la partie cliente.

Environnement de développement (IDE) : éclipse pour JAVA et sublime texte pour python.

4.4 Démarche adopter

Le processus de développement de notre application se subdivise en 3 parties représente par la (FIGURE 4).

(FIGURE 4)

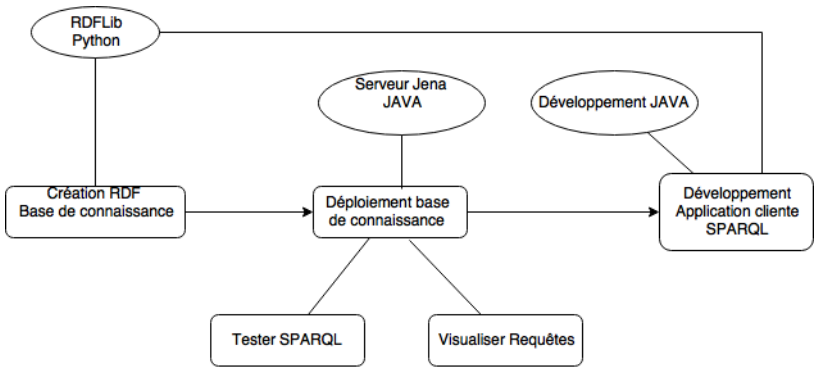

FIGURE 4 – Processus de développement

4.5 Construction de la base de connaissance

Comme déjà mentionner dans la partie (4.3 Environnement logiciel), nous avons utilisé l'éditeur Protégé pour la modélisation et conception de notre base de connaissance.

4.5.1 Présentation « RDFLib »

Rdflib est un bibliothèque Python pour travailler avec RDF. C'est un outils simple mais puissant pour représenter les informations. La bibliothèque contient un analyseur RDF / XML / sérialiseur qui est conforme à la spécification Syntaxe RDF / XML (révisée). La bibliothèque contient aussi à la fois en mémoire et persistants Graphique backends. Il est développé par un certain nombre de collaborateurs et a été créé par Daniel Krech qui continue de le maintenir.

Quelques commande à retenir :

La commande d'installation du bibliothèque :

```
\$sudo apt-get install python-rdflib python-bsddb3
```

La commande de lancement du script RDFLib :

```
\$python nomFichier.py
```

Après le lancement du script, le fichier « graph.rdf » est créer dans le même répertoire que le projet.

4.5.2 Implémentation

Le code suivant est la version simplifier pour générer une de connaissance. D'abord la ligne 4 à la ligne 11 permet de créer les prédicats, ensuite la ligne 13 jusqu'à la ligne 49 pour la construction des sujets et des objets. Enfin, pour ajouter les triplets, il faut se référer sur la ligne du code numéro 51 à 53. La dernière partie du code la ligne 53 à 57 permet la génération du fichier RDF dans le répertoire courant du projet.

Le code complet de notre projet est présentée dans la partie annexe du rapport.

```
import rdflib
g = Graph()
has_border_with = rdflib.URIRef('http://example.org/has_border_with
    ')
located_in = rdflib.URIRef('http://example.org/located_in')
work_at = rdflib.URIRef('http://example.org/work_at')
study_at = rdflib.URIRef('http://example.org/study_at')
lived_in = rdflib.URIRef('http://example.org/lived_in')
maried_with = rdflib.URIRef('http://example.org/maried_with')
has_children = rdflib.URIRef('http://example.org/has_children')
has_parents = rdflib.URIRef('http://example.org/has_parents')
has_friend = rdflib.URIRef('http://example.org/has_friend')

personne1 = URIRef("http://example.org/Person1")
name = Literal('Bob') # passing a string
age = Literal(24)
sexe = Literal('M')
g.add( (personne1, RDF.type, FOAF.Person) )
g.add( (personne1, FOAF.name, name) )
g.add( (personne1, FOAF.age, age) )
g.add( (personne1, FOAF.sexe, sexe) )
```

```
21
22 personne2 = URIRef("http://example.org/Person2")
23 name = Literal('Jean') # passing a string
24 age = Literal(64)
25 sexe = Literal('M')
26 g.add( (personne2, RDF.type, FOAF.Person) )
27 g.add( (personne2, FOAF.name, name) )
28 g.add( (personne2, FOAF.age, age) )
29 g.add( (personne2, FOAF.sexe, sexe) )
30
31 country1 = URIRef("http://example.org/Country1")
32 name = Literal('France') # passing a string
33 g.add( (country1, RDF.type, FOAF.Country) )
34 g.add( (country1, FOAF.name, name) )
35
36 continent1 = URIRef("http://example.org/Continent1")
37 name = Literal('Europe') # passing a string
38 g.add( (continent1, RDF.type, FOAF.Continent) )
39 g.add( (continent1, FOAF.name, name) )
40
41 entreprise1 = URIRef("http://example.org/Entreprise1")
42 name = Literal('Orange') # passing a string
43 g.add( (entreprise1, RDF.type, FOAF.Entreprise) )
44 g.add( (entreprise1, FOAF.name, name) )
45
46 ecole1 = URIRef("http://example.org/Ecole1")
47 name = Literal('La risiere') # passing a string
48 g.add( (ecole1, RDF.type, FOAF.Ecole) )
49 g.add( (ecole1, FOAF.name, name) )
50
51 g.add((personne1,lived_in,country1))
52 g.add((personne1,has_parents,personne2))
53 g.add((personne1,work_at,entreprise1))
54
55 g.serialize("monRDF.rdf")
56 g1 = rdflib.Graph()
57 g1.parse("graph.rdf", format="xml")
```

ALGORITHME 1 – Génération du fichier RDF avec RDFLib

Après avoir conçue l'ontologie, on obtient le fichier RDF. Le code suivant montre le fichier generer par le script du programme.

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <rdf:RDF
3    xmlns:ns1="http://xmlns.com/foaf/0.1/"
4    xmlns:ns2="http://example.org/"
5    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
6 >
7   <rdf:Description rdf:about="http://example.org/Person1">
8     <ns1:name>Bob</ns1:name>
9     <ns2:lived_in rdf:resource="http://example.org/Country1"/>
10    <rdf:type rdf:resource="http://xmlns.com/foaf/0.1/Person"/>
11    <ns2:work_at rdf:resource="http://example.org/Entreprise1"/>
12    <ns2:has_parents rdf:resource="http://example.org/Person2"/>
13    <ns1:sexe>M</ns1:sexe>
14    <ns1:age rdf:datatype="http://www.w3.org/2001/XMLSchema#integer
      ">24</ns1:age>
15  </rdf:Description>
16  <rdf:Description rdf:about="http://example.org/Entreprise1">
17    <ns1:name>Orange</ns1:name>
18    <rdf:type rdf:resource="http://xmlns.com/foaf/0.1/Entreprise"/>
19  </rdf:Description>
```

```
20   <rdf:Description rdf:about="http://example.org/Country1">
21     <rdf:type rdf:resource="http://xmlns.com/foaf/0.1/Country"/>
22     <ns1:name>France</ns1:name>
23   </rdf:Description>
24   <rdf:Description rdf:about="http://example.org/Continent1">
25     <rdf:type rdf:resource="http://xmlns.com/foaf/0.1/Continent"/>
26     <ns1:name>Europe</ns1:name>
27   </rdf:Description>
28   <rdf:Description rdf:about="http://example.org/Person2">
29     <rdf:type rdf:resource="http://xmlns.com/foaf/0.1/Person"/>
30     <ns1:age rdf:datatype="http://www.w3.org/2001/XMLSchema#integer
       ">64</ns1:age>
31     <ns1:name>Jean</ns1:name>
32     <ns1:sexe>M</ns1:sexe>
33   </rdf:Description>
34   <rdf:Description rdf:about="http://example.org/Ecole1">
35     <rdf:type rdf:resource="http://xmlns.com/foaf/0.1/Ecole"/>
36     <ns1:name>La risiere</ns1:name>
37   </rdf:Description>
38 </rdf:RDF>
```

ALGORITHME 2 – RDF Générer par le script RDFLib

4.6 Déploiement de la base de connaissance

Après voir mis en place la base de connaissance, il faut déployer cette base. Ainsi nous avons utilise le serveur **JENA FUSEKI**.

Processus de déploiement du **JENA FUSEKI** :

1. Entrer dans le dossier contenant le serveur,
2. Démarrage du serveur avec la commande suivante :

```
1   \$./fuseki-server --update --mem /ds
2
```

Le serveur démarre sur le port 3030 ensuite, on obtient la (FIGURE 5) suivante.

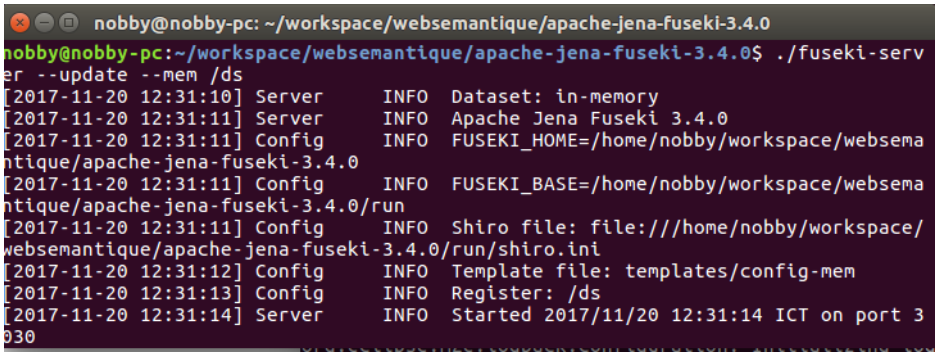

FIGURE 5 – Démarrage serveur JENA FUSEKI

3. connexion sur JENA FUSEKI : la connexion sur le serveur s'effectue en allant sur le l'url http ://localhost :3030/ (FIGURE 6).

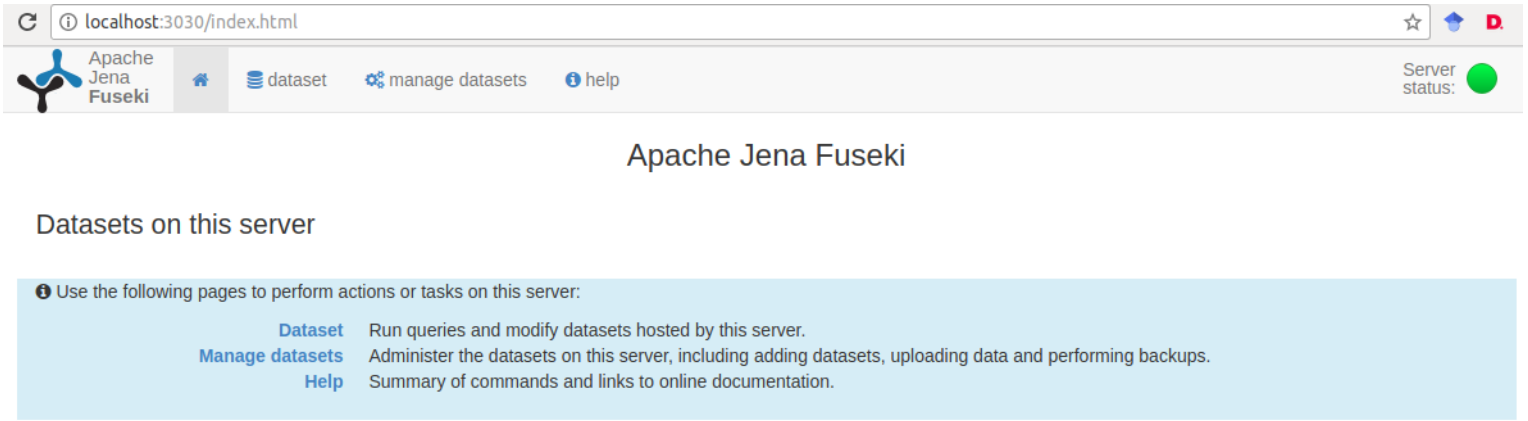

FIGURE 6 – Présentation de l'interface de JENA FUSEKI

4. déploiement de la base de connaissance : Pour déployer la base, il faut tous simplement aller dans le menu dataset ensuite upload file (la base à ajouter) (FIGURE 7).

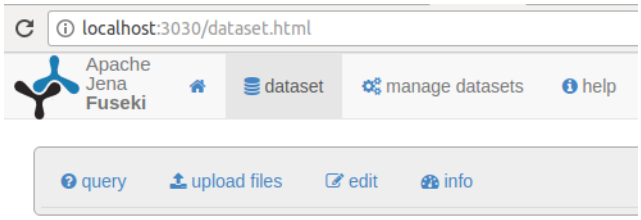

FIGURE 7 – Démarrage serveur JENA FUSEKI

5. tester et visualiser les requêtes SPARQL : la dernière étape consiste a construire et exécuté les requêtes SPARQL (FIGURE 8).

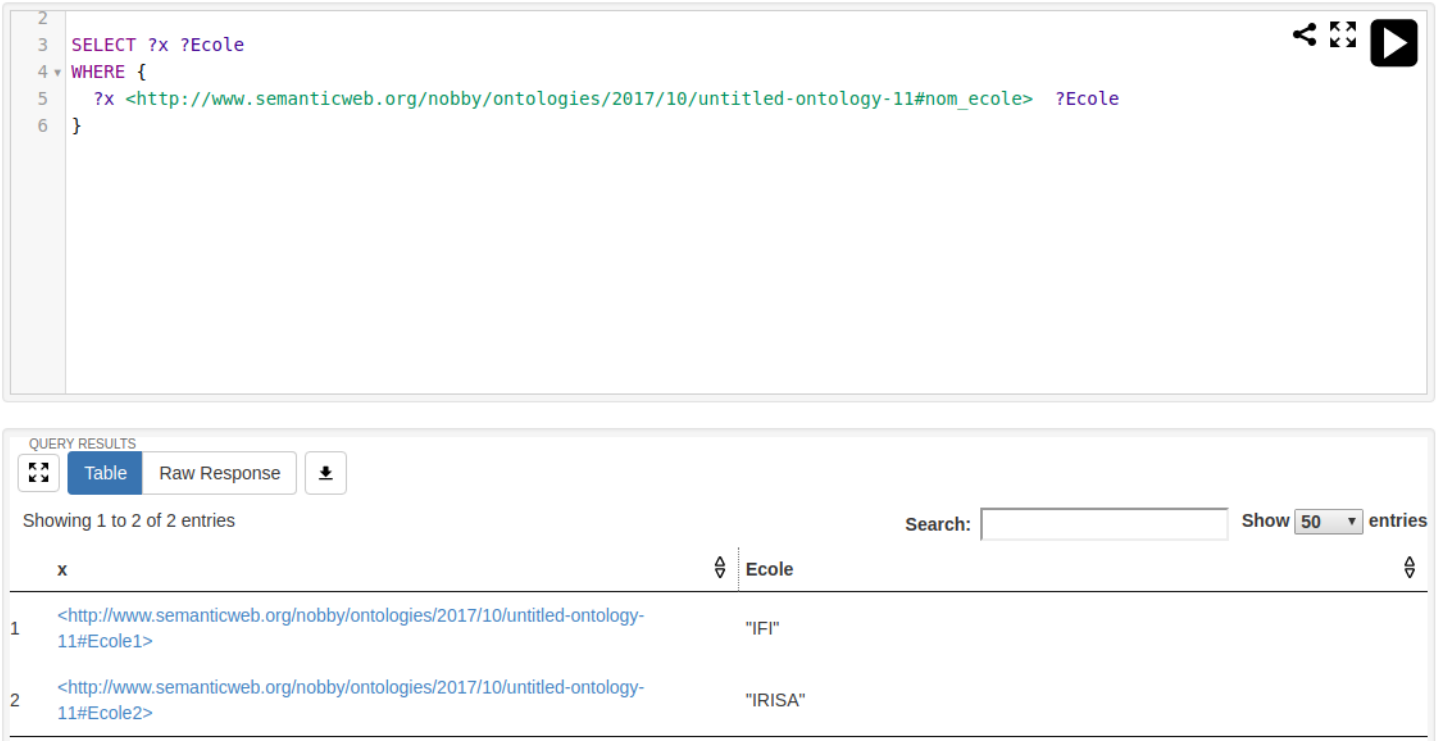

FIGURE 8 – Résultat d'exécution d'un requête SPARQL

4.7 Développement de l'application cliente

Nous avons conçue deux applications cliente qui utilise la base de connaissance qui est déjà présenté dans la partie démarche adopter.

— Une application JAVA qui se connecte sur le serveur JENA

— Une application python qui utilise le fichier RDF générer. Pour le cas cette approche, l'architecture de l'application n'est pas client serveur.

4.7.1 Application cliente avec JAVA

L'algorithme principale du programme est représenté par (ALGORITHME 4.7.1). La ligne 2 du code utilise la variable q qui prend en paramètre l'url du serveur JENA et la requête SPARQL qui est exécute sur le même ligne. Ensuite, la ligne 7 permet d'afficher les résultats.

```
1 public static void execSelectAndProcess(String serviceURI, String
    query) {
2     QueryExecution q = QueryExecutionFactory.sparqlService(
    serviceURI,
3         query);
4     ResultSet results = q.execSelect();
5     while (results.hasNext()) {
6       QuerySolution soln = results.nextSolution();
7       RDFNode x = soln.get("x");
8       System.out.println(x);
```

```
    }
  }
```

4.7.2 Application cliente avec Python

Comme l'application cliente avec Python n'est pas client servuer, alors il faut initialiser le fichier avant de lancer l'application. Dans le cas actuel, « g » (la viable qui stocker le graphe ou la base de connaissance) est initialiser lors de la génération du fichier RDF.

Ensuite, on crée les requêtes SPARQL entre la ligne 8 et la ligne 20 ; qui vont être exécuter sur la ligne 25.

```
1  print "LISTE AMIS CHOIX = 1"
2  print "LISTE PERSONNE QUI TRAVAILLE DANS ENTREPRISE CHOIX = 2"
3  print "LISTE PERSONNE QUI ETUDIE DANS ECOLE CHOIX = 3"
4
5  choix = raw_input("CHOISIR CHOIX : ")
6  if choix == str(1):
7    print "LISTE AMIS"
8    str1 = "select ?person where { ?person <http://example.org/
       has_friend> <http://example.org/"
9    str2 = "> }"
10   req = raw_input("Saisir Personne :")
11 elif choix == str(2):
12   print "LISTE PERSONNE QUI TRAVAILLE"
13   str1 = "select ?person where { ?person <http://example.org/
       work_at> <http://example.org/"
14   str2 = "> }"
15   req = raw_input("Saisir Entreprise :")
16 elif choix == str(3):
17   print "LISTE PERSONNE QUI ETUDIE"
18   str1 = "select ?person where { ?person <http://example.org/
       study_at> <http://example.org/"
19   str2 = "> }"
20   req = raw_input("Saisir Ecole :")
21 else:
22   print "BYE"
23   sys.exit()
24 q = str1 + req + str2
25 x = g.query(q)
26 print list(x)
```

ALGORITHME 3 – Application cliente avec python

5 Présentation de l'application

5.1 Présentation de l'application JAVA

La (FIGURE 9) nous donne un aperçu générale de notre application.

```
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/home/nobby/workspace/workspace_oxygene/TP_ONTHO/
SLF4J: Found binding in [jar:file:/home/nobby/workspace/workspace_oxygene/TP_ONTHO/
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
log4j:WARN No appenders could be found for logger (Jena).
log4j:WARN Please initialize the log4j system properly.
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more info.
-----------
| Ecole   |
===========
| "IFI"   |
| "IRISA" |
-----------
```

FIGURE 9 – Présentation de l'application

5.2 Présentation de l'application python

La (FIGURE 10) nous donne un aperçu générale de notre application. Nous pouvons voir qu'au démarrage de l'application, on doit définir le choix c'est-a-dire :

1 : liste des amis d'une personne ;

2 : liste des personnes qui travaille dans une entreprise donné ;

3 : liste des personne qui étudient dans une école ;

4 : liste des personne qui habitent dans un pays ;

5 : identifier l'épouse d'une personne ;

6 : liste enfants d'une personne ;

0 : Quitter l'application

```
nobby@nobby-pc: ~/workspace/websemantique/python
nobby@nobby-pc:~/workspace/websemantique/python$ python test33.py
LISTE AMIS CHOIX = 1
LISTE PERSONNE QUI TRAVAILLE DANS ENTREPRISE CHOIX = 2
LISTE PERSONNE QUI ETUDIE DANS ECOLE CHOIX = 3
LISTE PERSONNE QUI HABITE DANS PAYS = 4
IDENTIFIER EPOUSE CHOIX = 5
LISTE ENFANT D'UNE PERSONNE = 6
CHOISIR CHOIX : 1
LISTE AMIS
Saisir Personne :Person1
[(rdflib.term.URIRef(u'http://example.org/Person4'),), (rdflib.term.URIRef(u'htt
p://example.org/Person3'),), (rdflib.term.URIRef(u'http://example.org/Person2'),
)]
nobby@nobby-pc:~/workspace/websemantique/python$
```

FIGURE 10 – Présentation de l'application

6 Conclusion générale

Arrivée au terme de notre projet donc l'objectif était lié au développement et la création d'une ontologie sur la simulation d'un réseau social, ceci dans le but de mettre en pratique la théorie de l'unité d'enseignement intitulée Ontologie et Web Sémantique. La mise en oeuvre de ce projet a amené l'équipe à effectuer un regroupement de nombreux concepts ainsi que les liens entre ces concepts

(classes et sous classes), des relations entre ces concepts, leurs propriétés et des instances. Dans la réalisation du projet qui a été d'un grande importance voir même dune importance capitale, nous avons développé un outil/application pour la gestion de notre ontologie, qui structure sous forme de hiérarchique les différentes relations, entre les classes et les sous classe, et servira aux utilisateurs comme outil de recherche et de traitement de l'information. Le choix du langage d'implémentation joue un rôle important dans la construction d'une application donnée. Une liste d'outils de construction d'ontologie est présentée. L'état de l'art que nous avons présenté concerne les notions liées aux ontologies ainsi que les langages et techniques qui permettent de représenter les ontologies utilisés dans le cadre d'un système d'informations. En utilisant le langage JAVA et la bibliothèque Jena, nous avons développé un outil de simulation d'un réseau social.

ANNEXE 1 : Code Python pour générer le RDF

```python
import sys
from rdflib import URIRef, BNode, Literal
from rdflib import Namespace
from rdflib.namespace import RDF, FOAF
from rdflib import Graph
import rdflib
g = Graph()
has_border_with = rdflib.URIRef('http://example.org/has_border_with
    ')
located_in = rdflib.URIRef('http://example.org/located_in')
work_at = rdflib.URIRef('http://example.org/work_at')
study_at = rdflib.URIRef('http://example.org/study_at')
lived_in = rdflib.URIRef('http://example.org/lived_in')
maried_with = rdflib.URIRef('http://example.org/maried_with')
has_children = rdflib.URIRef('http://example.org/has_children')
has_parents = rdflib.URIRef('http://example.org/has_parents')
has_friend = rdflib.URIRef('http://example.org/has_friend')

personne1 = URIRef("http://example.org/Person1")
name = Literal('Bob') # passing a string
age = Literal(24)
sexe = Literal('M')
g.add( (personne1, RDF.type, FOAF.Person) )
g.add( (personne1, FOAF.name, name) )
g.add( (personne1, FOAF.age, age) )
g.add( (personne1, FOAF.sexe, sexe) )

personne2 = URIRef("http://example.org/Person2")
name = Literal('Jean') # passing a string
age = Literal(26)
sexe = Literal('M')
g.add( (personne2, RDF.type, FOAF.Person) )
g.add( (personne2, FOAF.name, name) )
g.add( (personne2, FOAF.age, age) )
g.add( (personne2, FOAF.sexe, sexe) )

personne3 = URIRef("http://example.org/Person3")
name = Literal('Marie') # passing a string
age = Literal(21)
sexe = Literal('F')
g.add( (personne3, RDF.type, FOAF.Person) )
g.add( (personne3, FOAF.name, name) )
g.add( (personne3, FOAF.age, age) )
g.add( (personne3, FOAF.sexe, sexe) )

personne4 = URIRef("http://example.org/Person4")
name = Literal('Julie') # passing a string
age = Literal(23)
sexe = Literal('F')
g.add( (personne4, RDF.type, FOAF.Person) )
g.add( (personne4, FOAF.name, name) )
g.add( (personne4, FOAF.age, age) )
g.add( (personne4, FOAF.sexe, sexe) )

personne5 = URIRef("http://example.org/Person5")
name = Literal('Paul') # passing a string
age = Literal(18)
sexe = Literal('M')
g.add( (personne5, RDF.type, FOAF.Person) )
g.add( (personne5, FOAF.name, name) )
```

```
g.add( (personne5, FOAF.age, age) )
g.add( (personne5, FOAF.sexe, sexe) )

personne6 = URIRef("http://example.org/Person6")
name = Literal('Piere') # passing a string
age = Literal(4)
sexe = Literal('M')
g.add( (personne6, RDF.type, FOAF.Person) )
g.add( (personne6, FOAF.name, name) )
g.add( (personne6, FOAF.age, age) )
g.add( (personne6, FOAF.sexe, sexe) )

personne7 = URIRef("http://example.org/Person7")
name = Literal('Victoire') # passing a string
age = Literal(2)
sexe = Literal('F')
g.add( (personne7, RDF.type, FOAF.Person) )
g.add( (personne7, FOAF.name, name) )
g.add( (personne7, FOAF.age, age) )
g.add( (personne7, FOAF.sexe, sexe) )

country1 = URIRef("http://example.org/Country1")
name = Literal('France') # passing a string
g.add( (country1, RDF.type, FOAF.Country) )
g.add( (country1, FOAF.name, name) )

country2 = URIRef("http://example.org/Country2")
name = Literal('Belgique') # passing a string
g.add( (country2, RDF.type, FOAF.Country) )
g.add( (country2, FOAF.name, name) )

country3 = URIRef("http://example.org/Country3")
name = Literal('Allemagne') # passing a string
g.add( (country3, RDF.type, FOAF.Country) )
g.add( (country3, FOAF.name, name) )

country4 = URIRef("http://example.org/Country4")
name = Literal('Chine') # passing a string
g.add( (country4, RDF.type, FOAF.Country) )
g.add( (country4, FOAF.name, name) )

continent1 = URIRef("http://example.org/Continent1")
name = Literal('Europe') # passing a string
g.add( (continent1, RDF.type, FOAF.Continent) )
g.add( (continent1, FOAF.name, name) )

continent2 = URIRef("http://example.org/Continent2")
name = Literal('Asie') # passing a string
g.add( (continent2, RDF.type, FOAF.Continent) )
g.add( (continent2, FOAF.name, name) )

entreprise1 = URIRef("http://example.org/Entreprise1")
name = Literal('Orange') # passing a string
g.add( (entreprise1, RDF.type, FOAF.Entreprise) )
g.add( (entreprise1, FOAF.name, name) )

entreprise2 = URIRef("http://example.org/Entreprise2")
name = Literal('Vietel') # passing a string
g.add( (entreprise2, RDF.type, FOAF.Entreprise) )
g.add( (entreprise2, FOAF.name, name) )

entreprise3 = URIRef("http://example.org/Entreprise3")
```

```
name = Literal('Mobiphone') # passing a string
g.add( (entreprise3, RDF.type, FOAF.Entreprise) )
g.add( (entreprise3, FOAF.name, name) )

ecole1 = URIRef("http://example.org/Ecole1")
name = Literal('La risiere') # passing a string
g.add( (ecole1, RDF.type, FOAF.Ecole) )
g.add( (ecole1, FOAF.name, name) )

ecole2 = URIRef("http://example.org/Ecole2")
name = Literal('Le victoire') # passing a string
g.add( (ecole2, RDF.type, FOAF.Ecole) )
g.add( (ecole2, FOAF.name, name) )

ecole3 = URIRef("http://example.org/Ecole3")
name = Literal('IFI') # passing a string
g.add( (ecole3, RDF.type, FOAF.Ecole) )
g.add( (ecole3, FOAF.name, name) )

g.add((country1,has_border_with,country2))
g.add((country1,has_border_with,country3))

g.add((country1,located_in,continent1))
g.add((country2,located_in,continent1))
g.add((country3,located_in,continent1))

g.add((country4,located_in,continent2))

g.add((personne1,lived_in,country1))
g.add((personne2,lived_in,country1))
g.add((personne3,lived_in,country2))
g.add((personne4,lived_in,country4))

g.add((personne6,has_parents,personne2))
g.add((personne7,has_parents,personne2))

g.add((personne1,maried_with,personne3))
g.add((personne3,maried_with,personne1))
g.add((personne2,maried_with,personne4))
g.add((personne4,maried_with,personne2))

g.add((personne1,work_at,entreprise1))
g.add((personne2,work_at,entreprise1))
g.add((personne3,work_at,entreprise2))
g.add((personne4,work_at,entreprise3))

g.add((personne2,has_friend,personne1))
g.add((personne3,has_friend,personne1))
g.add((personne4,has_friend,personne1))

g.add((personne6,has_friend,personne7))
g.add((personne7,has_friend,personne6))

g.add((personne6,study_at,ecole1))
g.add((personne7,study_at,ecole2))
g.add((personne1,study_at,ecole3))
g.add((personne2,study_at,ecole3))
g.add((personne3,study_at,ecole3))

str1 = ""
str2 = ""
req = ""
```

```
print "LISTE AMIS CHOIX = 1"
print "LISTE PERSONNE QUI TRAVAILLE DANS ENTREPRISE CHOIX = 2"
print "LISTE PERSONNE QUI ETUDIE DANS ECOLE CHOIX = 3"
print "LISTE PERSONNE QUI HABITE DANS PAYS = 4"
print "IDENTIFIER EPOUSE CHOIX = 5"
print "LISTE ENFANT D'UNE PERSONNE = 6"

choix = raw_input("CHOISIR CHOIX : ")
if choix == str(1):
  print "LISTE AMIS"
  str1 = "select ?person where { ?person <http://example.org/
    has_friend> <http://example.org/"
  str2 = "> }"
  req = raw_input("Saisir Personne :")
elif choix == str(2):
  print "LISTE PERSONNE QUI TRAVAILLE"
  str1 = "select ?person where { ?person <http://example.org/
    work_at> <http://example.org/"
  str2 = "> }"
  req = raw_input("Saisir Entreprise :")
elif choix == str(3):
  print "LISTE PERSONNE QUI ETUDIE"
  str1 = "select ?person where { ?person <http://example.org/
    study_at> <http://example.org/"
  str2 = "> }"
  req = raw_input("Saisir Ecole :")
elif choix == str(4):
  print "LISTE PERSONNE QUI HABITE"
  str1 = "select ?person where { ?person <http://example.org/
    lived_in> <http://example.org/"
  str2 = "> }"
  req = raw_input("Saisir Pays :")
elif choix == str(5):
  print "IDENTIFIER EPOUSE"
  str1 = "select ?person where { ?person <http://example.org/
    maried_with> <http://example.org/"
  str2 = "> }"
  req = raw_input("Saisir Personne :")
elif choix == str(6):
  print "LISTE ENFANT D'UNE PERSONNE"
  str1 = "select ?person where { ?person <http://example.org/
    has_parents> <http://example.org/"
  str2 = "> }"
  req = raw_input("Saisir Personne :")
else:
  print "BYE"
  sys.exit()
q = str1 + req + str2
x = g.query(q)
#print list(x)
# write graph to file, re-read it and query the newly created graph
g.serialize("graph.rdf")
g1 = rdflib.Graph()
g1.parse("graph.rdf", format="xml")
x1 = g1.query(q)
print list(x1)
```

ALGORITHME 4 – scripte code pour générer RDF

ANNEXE 2 : RDF

```
<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF
    xmlns:ns1="http://example.org/"
    xmlns:ns2="http://xmlns.com/foaf/0.1/"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
>
  <rdf:Description rdf:about="http://example.org/Country4">
    <ns1:located_in rdf:resource="http://example.org/Continent2"/>
    <rdf:type rdf:resource="http://xmlns.com/foaf/0.1/Country"/>
    <ns2:name>Chine</ns2:name>
  </rdf:Description>
  <rdf:Description rdf:about="http://example.org/Person5">
    <ns2:age rdf:datatype="http://www.w3.org/2001/XMLSchema#integer
    ">18</ns2:age>
    <rdf:type rdf:resource="http://xmlns.com/foaf/0.1/Person"/>
    <ns2:sexe>M</ns2:sexe>
    <ns2:name>Paul</ns2:name>
  </rdf:Description>
  <rdf:Description rdf:about="http://example.org/Ecole3">
    <ns2:name>IFI</ns2:name>
    <rdf:type rdf:resource="http://xmlns.com/foaf/0.1/Ecole"/>
  </rdf:Description>
  <rdf:Description rdf:about="http://example.org/Person2">
    <ns1:study_at rdf:resource="http://example.org/Ecole3"/>
    <ns1:work_at rdf:resource="http://example.org/Entreprise1"/>
    <ns2:name>Jean</ns2:name>
    <ns1:maried_with rdf:resource="http://example.org/Person4"/>
    <rdf:type rdf:resource="http://xmlns.com/foaf/0.1/Person"/>
    <ns2:age rdf:datatype="http://www.w3.org/2001/XMLSchema#integer
    ">26</ns2:age>
    <ns2:sexe>M</ns2:sexe>
    <ns1:lived_in rdf:resource="http://example.org/Country1"/>
    <ns1:has_friend rdf:resource="http://example.org/Person1"/>
  </rdf:Description>
  <rdf:Description rdf:about="http://example.org/Person1">
    <ns2:sexe>M</ns2:sexe>
    <ns2:name>Bob</ns2:name>
    <ns1:lived_in rdf:resource="http://example.org/Country1"/>
    <rdf:type rdf:resource="http://xmlns.com/foaf/0.1/Person"/>
    <ns1:study_at rdf:resource="http://example.org/Ecole3"/>
    <ns1:work_at rdf:resource="http://example.org/Entreprise1"/>
    <ns1:maried_with rdf:resource="http://example.org/Person3"/>
    <ns2:age rdf:datatype="http://www.w3.org/2001/XMLSchema#integer
    ">24</ns2:age>
  </rdf:Description>
  <rdf:Description rdf:about="http://example.org/Person6">
    <ns1:has_parents rdf:resource="http://example.org/Person2"/>
    <ns1:has_friend rdf:resource="http://example.org/Person7"/>
    <ns2:sexe>M</ns2:sexe>
    <rdf:type rdf:resource="http://xmlns.com/foaf/0.1/Person"/>
    <ns1:study_at rdf:resource="http://example.org/Ecole1"/>
    <ns2:age rdf:datatype="http://www.w3.org/2001/XMLSchema#integer
    ">4</ns2:age>
    <ns2:name>Piere</ns2:name>
  </rdf:Description>
  <rdf:Description rdf:about="http://example.org/Country1">
    <ns1:has_border_with rdf:resource="http://example.org/Country2"
    />
    <rdf:type rdf:resource="http://xmlns.com/foaf/0.1/Country"/>
    <ns2:name>France</ns2:name>
```

```
    <ns1:located_in rdf:resource="http://example.org/Continent1"/>
    <ns1:has_border_with rdf:resource="http://example.org/Country3"
    />
  </rdf:Description>
  <rdf:Description rdf:about="http://example.org/Person3">
    <ns1:has_friend rdf:resource="http://example.org/Person1"/>
    <ns2:sexe>F</ns2:sexe>
    <ns2:name>Marie</ns2:name>
    <ns1:study_at rdf:resource="http://example.org/Ecole3"/>
    <ns1:maried_with rdf:resource="http://example.org/Person1"/>
    <ns2:age rdf:datatype="http://www.w3.org/2001/XMLSchema#integer
    ">21</ns2:age>
    <ns1:lived_in rdf:resource="http://example.org/Country2"/>
    <rdf:type rdf:resource="http://xmlns.com/foaf/0.1/Person"/>
    <ns1:work_at rdf:resource="http://example.org/Entreprise2"/>
  </rdf:Description>
  <rdf:Description rdf:about="http://example.org/Person4">
    <ns2:sexe>F</ns2:sexe>
    <ns1:has_friend rdf:resource="http://example.org/Person1"/>
    <ns2:name>Julie</ns2:name>
    <ns2:age rdf:datatype="http://www.w3.org/2001/XMLSchema#integer
    ">23</ns2:age>
    <ns1:work_at rdf:resource="http://example.org/Entreprise3"/>
    <rdf:type rdf:resource="http://xmlns.com/foaf/0.1/Person"/>
    <ns1:maried_with rdf:resource="http://example.org/Person2"/>
    <ns1:lived_in rdf:resource="http://example.org/Country4"/>
  </rdf:Description>
  <rdf:Description rdf:about="http://example.org/Country2">
    <ns1:located_in rdf:resource="http://example.org/Continent1"/>
    <rdf:type rdf:resource="http://xmlns.com/foaf/0.1/Country"/>
    <ns2:name>Belgique</ns2:name>
  </rdf:Description>
  <rdf:Description rdf:about="http://example.org/Entreprise1">
    <ns2:name>Orange</ns2:name>
    <rdf:type rdf:resource="http://xmlns.com/foaf/0.1/Entreprise"/>
  </rdf:Description>
  <rdf:Description rdf:about="http://example.org/Country3">
    <rdf:type rdf:resource="http://xmlns.com/foaf/0.1/Country"/>
    <ns1:located_in rdf:resource="http://example.org/Continent1"/>
    <ns2:name>Allemagne</ns2:name>
  </rdf:Description>
  <rdf:Description rdf:about="http://example.org/Continent2">
    <ns2:name>Asie</ns2:name>
    <rdf:type rdf:resource="http://xmlns.com/foaf/0.1/Continent"/>
  </rdf:Description>
  <rdf:Description rdf:about="http://example.org/Entreprise3">
    <rdf:type rdf:resource="http://xmlns.com/foaf/0.1/Entreprise"/>
    <ns2:name>Mobiphone</ns2:name>
  </rdf:Description>
  <rdf:Description rdf:about="http://example.org/Continent1">
    <ns2:name>Europe</ns2:name>
    <rdf:type rdf:resource="http://xmlns.com/foaf/0.1/Continent"/>
  </rdf:Description>
  <rdf:Description rdf:about="http://example.org/Person7">
    <rdf:type rdf:resource="http://xmlns.com/foaf/0.1/Person"/>
    <ns1:has_friend rdf:resource="http://example.org/Person6"/>
    <ns2:name>Victoire</ns2:name>
    <ns2:age rdf:datatype="http://www.w3.org/2001/XMLSchema#integer
    ">2</ns2:age>
    <ns1:has_parents rdf:resource="http://example.org/Person2"/>
    <ns2:sexe>F</ns2:sexe>
    <ns1:study_at rdf:resource="http://example.org/Ecole2"/>
```

```
  </rdf:Description>
  <rdf:Description rdf:about="http://example.org/Ecole2">
    <rdf:type rdf:resource="http://xmlns.com/foaf/0.1/Ecole"/>
    <ns2:name>Le victoire</ns2:name>
  </rdf:Description>
  <rdf:Description rdf:about="http://example.org/Entreprise2">
    <ns2:name>Vietel</ns2:name>
    <rdf:type rdf:resource="http://xmlns.com/foaf/0.1/Entreprise"/>
  </rdf:Description>
  <rdf:Description rdf:about="http://example.org/Ecole1">
    <rdf:type rdf:resource="http://xmlns.com/foaf/0.1/Ecole"/>
    <ns2:name>La risiere</ns2:name>
  </rdf:Description>
</rdf:RDF>
```

ALGORITHME 5 – RDF générer

ANNEXE 3 : Client Java

```java
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.util.Scanner;

import org.apache.jena.query.DatasetAccessor;
import org.apache.jena.query.DatasetAccessorFactory;
import org.apache.jena.query.QueryExecution;
import org.apache.jena.query.QueryExecutionFactory;
import org.apache.jena.query.QuerySolution;
import org.apache.jena.query.ResultSet;
import org.apache.jena.query.ResultSetFormatter;
import org.apache.jena.rdf.model.Model;
import org.apache.jena.rdf.model.ModelFactory;
import org.apache.jena.rdf.model.RDFNode;




public class Ontho {

  public static void uploadRDF(File rdf, String serviceURI)
      throws IOException {

    // parse the file
    Model m = ModelFactory.createDefaultModel();
    try (FileInputStream in = new FileInputStream(rdf)) {
      m.read(in, null, "RDF/XML");
    }

    // upload the resulting model
    DatasetAccessor accessor = DatasetAccessorFactory
        .createHTTP(serviceURI);
    accessor.putModel(m);
  }

  public static void execSelectAndPrint(String serviceURI, String
    query) {
    QueryExecution q = QueryExecutionFactory.sparqlService(
    serviceURI,
        query);
    ResultSet results = q.execSelect();

    ResultSetFormatter.out(System.out, results);

    while (results.hasNext()) {
      QuerySolution soln = results.nextSolution();
      RDFNode x = soln.get("x");
      System.out.println(x);
    }
  }

  public static void execSelectAndProcess(String serviceURI, String
     query) {
    QueryExecution q = QueryExecutionFactory.sparqlService(
    serviceURI,
        query);
    ResultSet results = q.execSelect();

    while (results.hasNext()) {
```

```
57        QuerySolution soln = results.nextSolution();
58        // assumes that you have an "?x" in your query
59        RDFNode x = soln.get("x");
60        System.out.println(x);
61      }
62    }
63
64    public static void main(String[] args) throws IOException {
65      // TODO Auto-generated method stub
66      //uploadRDF(new File("test.rdf"), );
67      String query = null;
68      String serivceUri = "http://localhost:3030/ds";
69
70      String queryEcole = "SELECT ?Ecole " +
71          " WHERE {?x  <http://www.semanticweb.org/nobby/ontologies
      /2017/10/untitled-ontology-11#nom_ecole>  ?Ecole}";
72
73      String queryEntreprise = "SELECT ?Entreprise " +
74          " WHERE {?x  <http://www.semanticweb.org/nobby/ontologies
      /2017/10/untitled-ontology-11#nom_entreprise>  ?Entreprise}";
75
76      String queryFemme = "SELECT ?Femme " +
77          " WHERE {?x  <http://www.semanticweb.org/nobby/ontologies
      /2017/10/untitled-ontology-11#nom>  ?Femme}";
78
79      String queryVille = "SELECT ?Ville " +
80          " WHERE {?x  <http://www.semanticweb.org/nobby/ontologies
      /2017/10/untitled-ontology-11#nom_ville>  ?Ville}";
81
82      String queryTravaille = "SELECT ?Personne ?nom" +
83          " WHERE {?x  <http://www.semanticweb.org/nobby/ontologies
      /2017/10/untitled-ontology-11#travaille>  ?Personne ?nom}";
84
85      Scanner sc = new Scanner(System.in);
86      System.out.println("Ecrire le choix");
87      int choix = sc.nextInt();
88      if (choix == 1) {
89        query = queryEcole;
90      }
91      if (choix == 2) {
92        query = queryEntreprise;
93      }
94      if (choix == 3) {
95        query = queryVille;
96      }
97      if (choix == 4) {
98        query = queryTravaille;
99      }
100     if (choix == 5) {
101       System.out.println("Recherche par nom ANJARA");
102       //String nom = sc.next();
103       query = "SELECT ?x ?Nom ?Prenom" +
104           " WHERE {?x  <http://www.semanticweb.org/nobby/ontologies
      /2017/10/untitled-ontology-11#nom>  'Anjara' . " +
105           "        ?x  <http://www.semanticweb.org/nobby/ontologies
      /2017/10/untitled-ontology-11#nom>  ?Nom . " +
106           "        ?x  <http://www.semanticweb.org/nobby/
      ontologies/2017/10/untitled-ontology-11#prenom>  ?Prenom .}";
107     }
108     //String nameFile = "personne2.owl";
109
110
```

```
111     //uploadRDF(new File(nameFile), serivceUri);
112     execSelectAndPrint(serivceUri, query);
113
114     //execSelectAndProcess(serivceUri, query);
115   }
116 }
```

ALGORITHME 6 – Client Java

Références

Charlet, Jean (2002). “L’ingénierie des connaissances : développements, résultats et perspectives pour la gestion des connaissances médicales”. In :

Articles only

Charlet, Jean (2002). “L’ingénierie des connaissances : développements, résultats et perspectives pour la gestion des connaissances médicales”. In :