14/06/2010

IRIT UTM
5 allées Antonio Machado
31058 TOULOUSE Cedex 9

IMPLEMENTATION D'UNE INTERFACE D'AIDE A LA FORMULATION DE REQUETE SPARQL

Rapport d'activité du 6 Avril 2010 au 11 Juin 2010

A l'attention de Nathalie Hernandez, Catherine Comparot, Ollivier Haemmerlé et Eric Jacoboni

Guillaume Noisette - Promotion 2009/2010
Licence MIASHS – Université Toulouse 2 – Le Mirail

Guillaume Noisette

Remercîments

Tout d'abord, merci à Luis Fariñas del Cerro de m'avoir accueilli au sein du laboratoire qu'il dirige, l'IRIT.

Je tiens à remercier particulièrement mes trois professeurs et tuteurs : Nathalie Hernandez, Catherine Comparot et Ollivier Haemmerlé pour m'avoir permis d'effectuer ce stage très intéressant sur le thème du web sémantique.

Je voudrai aussi remercier les personnes qui ont bien voulu partager leurs bureaux avec moi à savoir Caroline Thierry et Wafa Karoui.

Sommaire

Introduction

Dans le cadre de la formation IUP MIASHS (Mathématiques et Informatique Appliquées aux Sciences Humaines et Sociales), j'ai réalisé mon stage de fin d'étude dans le laboratoire de l'IRIT (Institut de Recherche en Informatique de Toulouse), où de nombreux enseignants chercheurs, qui m'ont appris à programmer, travaillent. Ce stage qui a duré dix semaines (du 5 avril au 11 juin 2010) a été encadré par trois de mes professeurs de cette année à savoir : Nathalie Hernandez (enseignante de Web), Catherine Comparot (enseignante d'Analyse/Conception de Système d'Information et d'Interface Homme-Machine) et Ollivier Haemmerlé (enseignant de C++ et de Graphe).

Ce stage consistait à réaliser une application sur le web sémantique sur le thème du cinéma. Plus simplement, l'utilisateur saisit des mots clés en rapport avec le septième art, par exemple « Jean Reno » et « festival de Cannes », l'application va ensuite lui proposer une série de phrases dont la concordance avec sa recherche sera la plus grande possible. Ainsi l'utilisateur n'aura plus qu'à choisir celle qui correspond exactement à ce qu'il souhaite pour obtenir les réponses voulues.

Avant de vous donner tous les détails de cette application, je vous présenterai le laboratoire pour lequel j'ai travaillé, puis je vous expliquerai ce qu'est le web sémantique. Par la suite je vous parlerai du projet qui a donné lieu à ce stage, et pour terminer une partie sur la compréhension et la conception de cette application.

I. Présentation du laboratoire

a) Historique

Institut de Recherche en Informatique de Toulouse, IRIT, a été fondé en 1990 et représente un établissement mixte de recherche entre l'Université Paul Sabatier de Toulouse (UPS), le Centre National de Recherche Scientifique (CNRS), l'ENSEEIHT, l'Institut National Polytechnique de Toulouse et l'Université Toulouse 1 Capitole, et depuis janvier 2007, l'Université Toulouse 2 Le Mirail (UTM). Depuis 1998, la direction a été confiée à Luis Fariñas del Cerro.

Il comptait, en novembre 2009, un effectif total de plus de 650 personnes dont 214 enseignants-chercheurs, 29 chercheurs et 234 doctorats. Depuis l'année 2008, l'IRIT a défini quatre axes stratégiques sur des grands défis scientifiques et socio-économiques qui sont :

- Informatique pour la santé : basé sur quatre domaines : l'imagerie du vivant ; la gestion de données biomédicales et infrastructure d'accès et de traitement ; la modélisation et la simulation du vivant ; ainsi que le handicap et l'autonomie
- Masses de données et calcul : cela concerne les différentes facettes du traitement et du calcul de données massives et couvre à la fois les infrastructures et les algorithmes
- Systèmes sociotechniques ambiants : possédant des entités physiques ou des logiciels distribués, qui ont des capacités d'interaction, un comportement autonome et ont la capacité de s'adapter a` la tâche courante de l'être humain et aux ressources numériques et physiques disponibles
- Systèmes embarqués critiques : conçus comme un assemblage de composants communiquant par un ou plusieurs réseaux numériques, dont on veut garantir des propriétés de qualité de service ; on étudie pour cela des environnements de développement certifiés et on développe des plates-formes d'exécution.

Ces axes de recherches ont été disséminés à travers sept thèmes :

- Thème 1 : Analyse et synthèse de l'information
- Thème 2 : Indexation et recherche d'informations
- Thème 3 : Interaction, autonomie, dialogue et coopération
- Thème 4 : Raisonnement et décision
- Thème 5 : Modélisation, algorithmes et calcul haute performance
- Thème 6 : Architecture, systèmes et réseaux

- Thème 7 : Sûreté de développement du logiciel

Chaque thème possède plusieurs équipes. Pour ce stage, j'ai participé à l'avancement du thème 3 « Interaction, autonomie, dialogue et coopération » ; dont les recherches portent sur la modélisation et la conception de systèmes dans lesquels l'interaction, le dialogue et la coopération entre agents sont prépondérants. Cependant j'ai plus particulièrement exécuté mon stage dans l'équipe IC3 (Ingénierie de la Connaissance, de la Cognition et de la Coopération).

b) Equipe IC3

Elle est basée sur un regroupement de deux équipes : Conception de Systèmes Coopératifs (CSC) et Groupe de Recherche en Ingénierie Cognitive (GRIC) ainsi que plusieurs chercheurs du Groupe de Recherche en Informatique et Mathématiques du Mirail (GRIMM) en 2007. Elle compte 11 membres permanents recherche, 2 membres permanents associés et 17 membres non-permanents.

L'objectif principal, au départ, était l'ingénierie des modèles de connaissances et/pour des systèmes coopératifs, mais elle a défini récemment une nouvelle orientation de ses recherches, centrée sur les ontologies statiques et dynamiques, les systèmes d'aides au travail coopératif mettant en œuvre ces ontologies ainsi que leurs usages dans les situations nominales ou dégradées.

L'équipe est partagée selon trois groupes tels que l'ingénierie de la cognition, l'ingénierie de la connaissance et l'ingénierie de la coopération.

1- Ingénierie de la cognition

Cela correspond à l'étude des collectifs de travail pour une meilleure gestion des situations de crise.

2- Ingénierie de la connaissance

Par rapport à l'ingénierie de la connaissance, l'équipe IC3 étudie particulièrement la représentation des connaissances associées à des textes, afin de construire des ontologies à partir de texte ou de représenter la sémantique des contenus textuels selon des approches linguistiques et statistiques. L'équipe s'est spécialisée dans les méthodes ascendantes en exploitant une sémantique textuelle grâce à des outils basés sur les approches par patrons ou encore sur l'étude des points de vue.

3- Ingénierie de la coopération

Cette composante travaille sur le développement des systèmes coopératifs d'aide à la décision en situations nominales. La coopération est définie à deux niveaux :

- Coopération entre plusieurs décideurs
- Coopération homme/machine

c) Place au sein du laboratoire

J'ai effectué ce stage dans les locaux de l'IRIT à l'Université Toulouse 2 Le Mirail avec trois personnes, qui sont des membres permanents de l'équipe IC3 et aussi des professeurs de ma formation, que j'ai considéré comme mes tuteurs de stage et qui sont :

- Catherine Comparot : Maître de Conférences à UTM
- Nathalie Hernandez : Maître de Conférences à UTM
- Ollivier Haemmerlé : Professeur à UTM

II. Présentation du sujet

Mon stage est dans la continuité du travail effectué par mes professeurs. Je devais implémenter toutes les recherches qu'ils ont faites sur le web sémantique puisqu'ils n'en n'avaient jusqu'à ce jour aucune. C'est donc une première application qui est basée sur le cinéma que je devais présenter à la fin de ces dix semaines. Elle est basée sur le principe suivant :

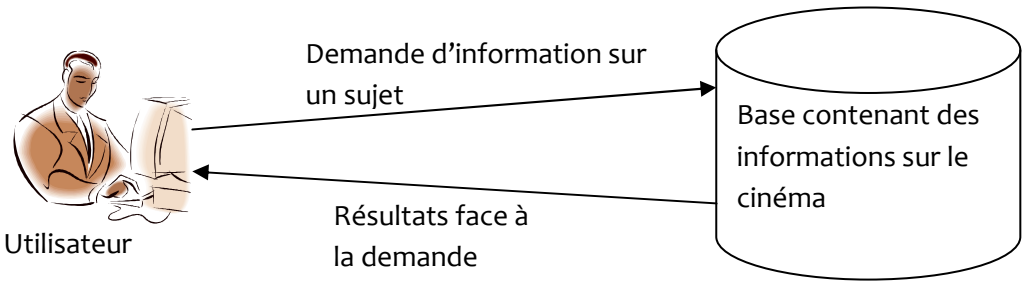

Figure 1 : Schéma basique de l'application

Un utilisateur exprime sa demande sur le domaine du cinéma et l'application doit lui donner tous les résultats qui correspondent à celle-ci.

Je vous expliquerai le lien entre cette application et le web sémantique une fois que nous aurons vu toutes les notions qu'il y a connaître sur ce dernier.

III. Qu'est-ce que le web sémantique ?

Tout d'abord il faut bien comprendre ce qu'est la sémantique. Lorsque l'on parle de sémantique nous parlons des liens qu'il y a entre les mots. Par exemple, nous pouvons dire : « la table court très vite », la phrase est syntaxiquement correcte cependant elle est sémantiquement incorrect car cela présente une action qui est impossible puisqu'une table ne peut courir.

Donc quand nous parlons de web sémantique, on dénomme un web qui vérifie l'aspect sémantique des informations et qui interprètent celles qui sont trouvées par la suite. Cette idée innovante a été introduite par Tim Berners-Lee dans les années quatre-vingt-dix. Ci-dessous, vous verrez l'architecture du web sémantique d'après le W3C (World Wide Web Consortium).

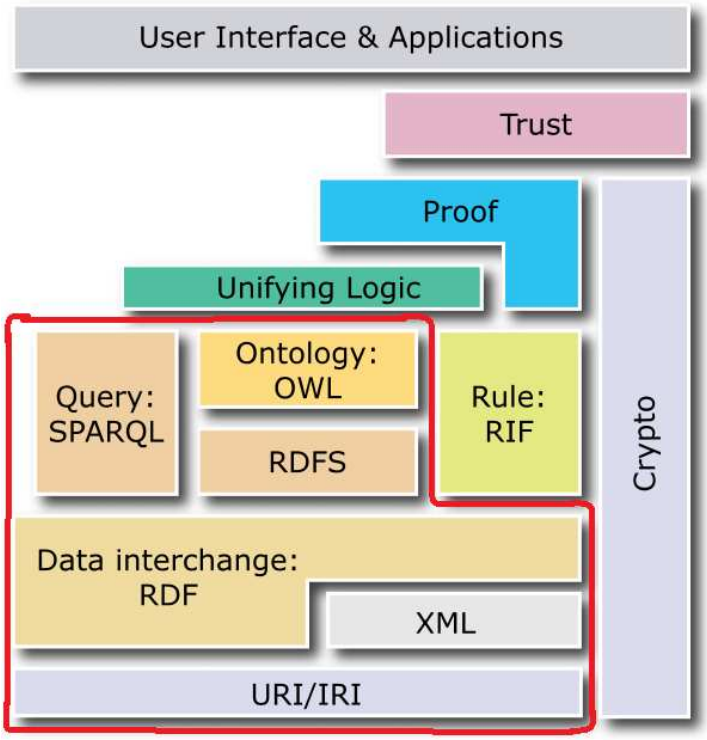

Figure 2 : "layercake" du web sémantique par W3C

Nous remarquons que cette architecture est découpée en plusieurs étages. La partie encadrée en rouge est celle que nous avons manipulée durant ce stage et c'est donc cette partie que je vais vous expliquer.

a) URI

Universal Resource Identifier signifie identifiant uniforme de ressource. Cela signifie qu'une URI correspond à l'identifiant d'une ressource sur un réseau (le plus souvent sur internet), doit correspondre à une norme définie par le W3C. Par exemple, voici l'URI de Jean Reno dans DBpedia : http://dbpedia.org/resource/Jean_Reno. DBpedia est un endpoint, c'est-à-dire une base de connaissance distante via le protocole SPARQL qui a utilisé l'encyclopédie Wikipédia pour créer un milliard de triplet.

b) XML

Extensible Markup Language signifie langage extensible de balisage. C'est un langage informatique qui est basé sur un système de balisage des données. Autrement dit, chaque donnée doit être encadrée par une balise ouvrante et une balise fermante de même nom : <balise>Texte</balise>. C'est un langage qui est très structuré ce qui permet de stocker des données de façon claire sous forme d'arborescence. Afin de gérer le contenu présent dans un fichier XML, on rattache un fichier DTD (Document Type Definition) qui servira de schéma à ce dernier. Dans ce fichier DTD, on indique toutes les balises qui seront présentes dans le fichier XML, ainsi que toute l'arborescence de celui-ci.

c) RDF

Resource Description Framework est un langage du web sémantique qui s'appuie sur le langage XML. Il permet de décrire des ressources sous forme de triplet.

⇨ Qu'est-ce qu'un triplet ?

Un triplet, correspond à un sujet, un prédicat et un objet.

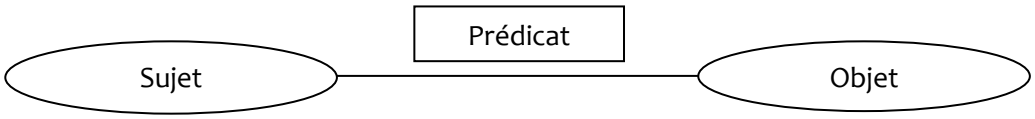

Figure 3 : Triplet

Par exemple dans la phrase suivante : « Jean Reno a joué dans Léon », le terme « Jean Reno » est le sujet du triplet, celui « a joué dans » correspond au prédicat et « Léon » est l'objet de ce triplet. Si l'on veut transformer cette phrase sous forme d'URI l'équivalence donnée serait la suivante :

<http://dbpedia.org/resource/Léon_(film)> <http://dbpedia.org/ontology/starring> <http://dbpedia.org/resource/Jean_Reno>

Ou celle-ci sous la forme RDF:

```
<dbpedia:Film rdf:about="http://dbpedia.org/resource/L%C3%A9on_%28film%29">
  <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Léon
(film)</rdfs:label>
  <http://dbpedia.org/ontology/starring
rdf:resource="http://dbpedia.org/resource/Jean_Reno"/>
</dbpedia:Film>
```

d) RDFS

RDFS correspond à RDF Schema qui est un langage identique à RDF. Il est utile pour la représentation des connaissances. RDFS fournit des éléments de base pour la modélisation d'ontologies ou de vocabulaires destinés à structurer des ressources RDF. RDF Schema détermine le plus petit nombre de notions et de propriétés nécessaires à la définition d'un vocabulaire simple.

e) Ontologie OWL

Les ontologies décrivent en général les classes, les attributs ou les relations qui lient les objets entre eux. OWL est un formalisme pour décrire celles-ci et il s'appuie sur RDF. Par exemple, si nous voulons décrire une classe « Evenement », alors, nous aurions besoin de dire où il a lieu, en quelle année, et bien sûr il ne faut pas oublier de lui donner un nom, cela nous donnerai un résultat comme celui présenté ci-dessous.

```
<owl:Class rdf:about="#Evenement">
  <rdfs:label
rdf:datatype="http://www.w3.org/2001/XMLSchema#string">evenement</rdfs:label>
</owl:Class>
<owl:ObjectProperty rdf:ID="date">
  <rdfs:range rdf:resource="#Annee"/>
  <rdfs:domain rdf:resource="#Evenement"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="lieu">
  <rdfs:range rdf:resource="#Lieu"/>
  <rdfs:domain rdf:resource="#Evenement"/>
</owl:ObjectProperty>
```

Et si par la suite, nous souhaitions rajouter une classe « Festival » qui aurait un attribut supplémentaire « Jury », il nous suffirait alors de faire la modification suivante :

```
<owl:Class rdf:ID="Festival">
    <rdfs:subClassOf>
        <owl:Class rdf:ID="Evenement"/>
    </rdfs:subClassOf>
</owl:Class>
<owl:ObjectProperty rdf:ID="jury">
    <rdfs:range rdf:resource="#Jury"/>
    <rdfs:domain rdf:resource="#Festival"/>
</owl:ObjectProperty>
```

f) SPARQL

C'est un langage de requête qui ressemble à SQL, mais qui a été adapté pour questionner des triplets RDF. Grâce à SPARQL, nous pouvons savoir si un élément est présent dans la base de triplets et/ou nous pouvons également extraire les informations qui nous intéressent à partir des clauses que nous avons ajoutées. Nous pouvons aussi créer une nouvelle base RDF à partir des triplets que l'on a préalablement sélectionnés. Voici un exemple de requête SPARQL :

```
SELECT ?sujet ?predicat ?objet WHERE { ?sujet ?predicat ?objet }
```

Avec cette requête nous sélectionnons tous les triplets de la base RDF.

g) Exemple

Voici un petit exemple final afin de bien comprendre ces différents concepts.

Fichier OWL (Ontologie) :

On définit les classes qui nous serons utiles. Dans ce cas précis, celles qui nous importent sont la classe « Personne » et la classe « Annee » et on indique qu'il y a une liaison « dateNaissance » entre ces dernières.

En voici la concrétisation :

```
<rdf:RDF
   xmlns:owl="http://www.w3.org/2002/07/owl#"
   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">
  <owl:Ontology rdf:about=""/>
  <owl:Class rdf:about="Annee">
   <rdfs:label
rdf:datatype="http://www.w3.org/2001/XMLSchema#string">personne</rdfs:label>
  </owl:Class>
  <owl:Class rdf:about="Personne">
   <rdfs:label
rdf:datatype="http://www.w3.org/2001/XMLSchema#string">personne</rdfs:label>
  </owl:Class>
  <owl:ObjectProperty rdf:ID="dateNaissance">
         <rdfs:domain rdf:resource="#Annee"/>
   <rdfs:range rdf:resource="#Personne"/>
  </owl:ObjectProperty></rdf:RDF>
```

Fichier RDF:

Ici on instancie les classes définie plus haut et on indique « an_1985 » comme une instance de la classe « Annee » et Jean_Spradix comme celle de la classe « Personne ». Pour terminer cette étape on met en place la liaison entre les deux spécialisations des classes.

```
<rdf:RDF
   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
   xmlns:ontoMemo="http://ontoMemo#">
   <ontoMemo:Annee rdf:about="http://Memo#an_1985">
      <rdfs:label
rdf:datatype="http://www.w3.org/2001/XMLSchema#string">2009</rdfs:label>
   </ontoMemo:Annee>
   <ontoMemo:Personne rdf:about="http://Memo#Jean_ Spradix">
         <rdfs:label         rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Jean
Spradix</rdfs:label>
         <ontoMemo:dateNaissance rdf:resource="http://annotFilms#an_1985"/>
  </ontoMemo:Personne>
</rdf:RDF>
```

Requête SPARQL :

Nous souhaitons obtenir l'année de naissance de Jean Spradix, pour cela nous devons lancer la requête qui suit :

```
SELECT ?date WHERE { <http://Memo#Jean_ Spradix> <http://ontoMemo#dateNaissance> ?date }
```

Le résultat de cette requête est l'URI de l'année de naissance de Jean Spradix, ainsi nous obtenons comme réponse : http://annotFilms#an_1985 .

IV. Avant de programmer

a) Complément sur le sujet

Maintenant que nous avons davantage de connaissances sur le web sémantique, je peux approfondir sur ce sujet.

Mon stage consistait à mettre en place cette application sur le web sémantique en m'appuyant sur les diverses recherches effectuées au préalable par mes professeurs. Pour cela, j'ai utilisé deux documents qu'ils ont écrits et qui se nomment respectivement « *Expression de requêtes en graphes conceptuels à partir de mots-clés et de patrons* » et « *Construction of SPARQL queries from keywords and query patterns* », grâce à ces écrits, j'ai mieux compris les attentes qu'ils avaient vis-à-vis de mon travail. Cependant, j'ai tout de même du approfondir en trouvant d'autres informations sur le ce sujet grâce à Internet pour en voir d'autres exemples, que ceux qui m'avaient déjà été présentés à travers les spécifications ou avec les cours de mes enseignants. Une fois cette tâche exécutée, j'ai analysé ce qui m'était demandé et ai décidé de modifier le schéma de l'application.

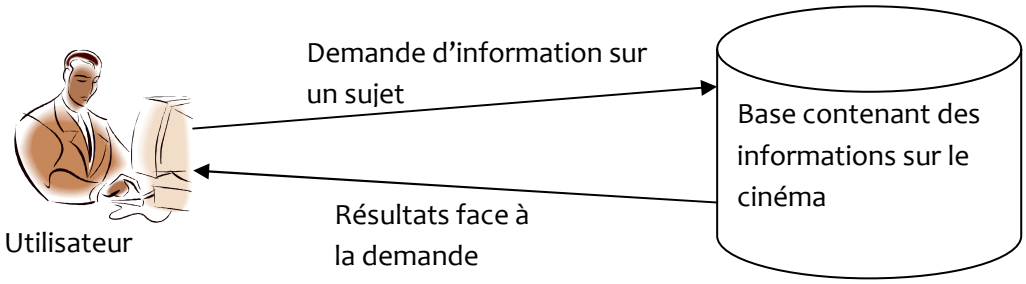

Figure 4 : Schéma basique de l'application

L'utilisateur peut ainsi faire sa demande d'information en ajoutant des mots-clés, qui seront ensuite désambiguïsés par l'application et par celui qui en fait la requête. Nous en verrons le fonctionnement par la suite. Ensuite le programme va produire différentes phrases afin de répondre au mieux à la demande. Ce sera à la personne ayant lancée la demande d'en sélectionner une afin que l'application lui donne finalement tous les résultats correspondant à sa cherche.

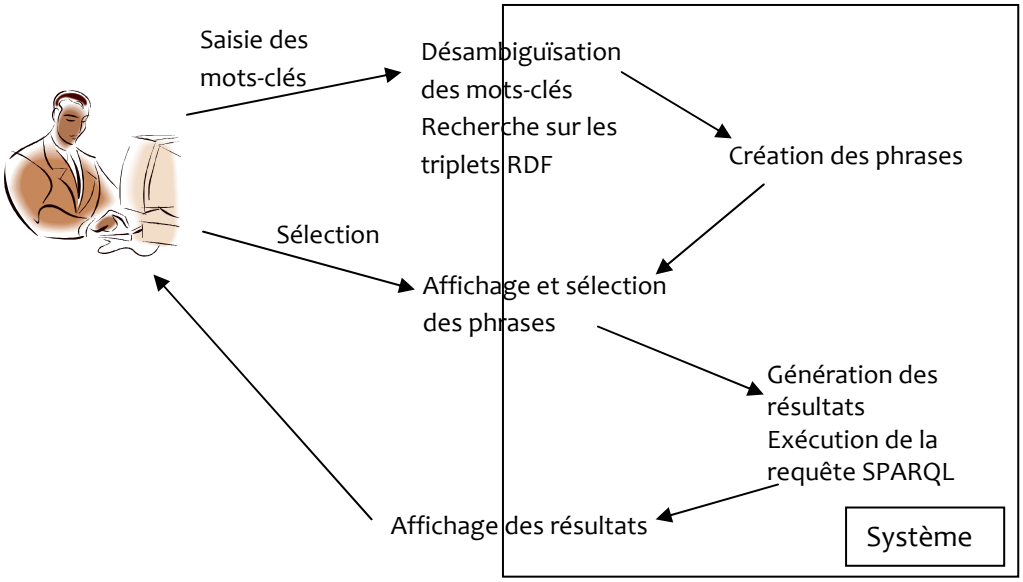

Figure 5 : Schéma de l'application plus poussé

La formation des phrases se fait grâce à des patrons que nous écrivons, pour produire des mappings. Ces deux notions seront approfondies et expliquées dans la suite de ce dossier. Ensuite, pour que la phrase présentée en premier dans les résultats soit la plus cohérente avec la requête présentée, nous effectuons un classement des mappings selon leur degré de pertinence.

En voici la schématisation :

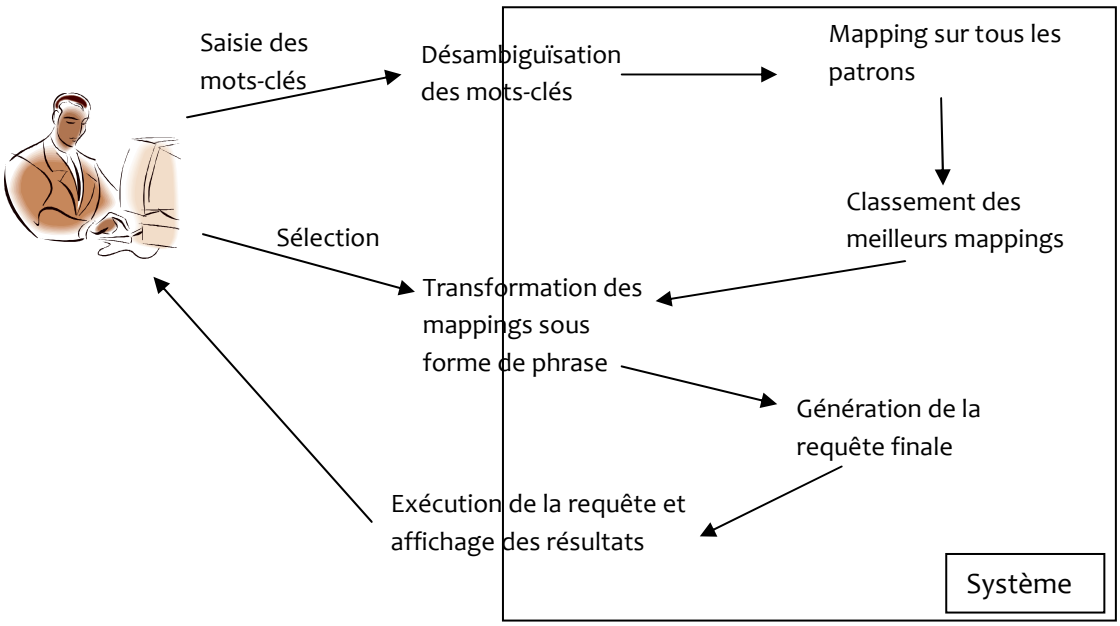

Figure 6 : Schéma de l'application finale

1- Saisie des mots-clés

La saisie des mots-clés doit s'effectuer au lancement de l'application et se fait par l'utilisateur. Toutes les requêtes sont possibles à concrétiser à la condition que les termes entrés pour cette recherche soient basés sur le monde du cinéma.

2- Désambiguïsation des mots-clés

Pour cette étape, on interroge un fichier RDF et OWL, pour savoir si le mot-clé correspond à un label d'une classe OWL ou d'une instance. Dans le cas, où ce terme est présent plusieurs fois, l'utilisateur définit plus précisément sa recherche avec plusieurs réponses présentant des choix différents pour celui-ci. Par exemple, si l'on saisit le mot-clé « Cannes » pour parler de la ville, et que ce dernier, dans la base de connaissance, correspond à la ville ou à un festival, alors on devra sélectionner « Cannes » en tant que ville.

3- Les patrons

Pour commencer il est nécessaire de préciser la notion de concept. En effet c'est un terme que l'on emploiera souvent dans ce dossier. Ainsi il correspond à un nom et une valeur. Comme exemple, on peut avoir [Acteur : Jean_Reno] ou encore [Acteur : *]. Lorsque la valeur correspond au symbole « * », c'est équivaut à prendre tous les résultats correspondant à la classe « Acteur ».

Un patron est défini par plusieurs données présentées dans la liste suivante :

- Une signature qui indique les concepts interrogeables par l'utilisateur.
- Une liste de triplets qui correspond à tous les concepts utilisés par le patron (il est possible que l'on trouve plus de triplets que de concepts présent dans la signature, car on peut y en avoir que l'utilisateur ne peut interroger).
- Une liste de triplets correspondant à toutes les propriétés utilisées par le patron et qui permettent de faire les liens entre les concepts.
- Une phrase qui correspond au sens du patron.

Voici, un exemple de patron afin de visualiser correctement comment cela se représente.

➔ Signature :
Acteur : *, Prix : *, Festival : *, Annee :*, Œuvre_filmee : *

➔ Concepts :

?a	<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>	<http://ontoMemo#Acteur>
?b	<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>	<http://ontoMemo#Prix>
?c	<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>	<http://ontoMemo#Festival>
?d	<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>	<http://ontoMemo#Annee>
?e	<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>	<http://ontoMemo#Œuvre_filmee>

Dans la troisième colonne, nous avons les URI correspondant aux concepts.

➔ Propriétés :

?a	<http://ontoMemo#participe>	?e
?a	<http://ontoMemo#reçoit>	?b
?b	<http://ontoMemo#date>	?d
?b	<http://ontoMemo#cadre>	?c

➔ Phrase :
« Un acteur ayant obtenu un prix lors de un festival en une année pour une œuvre filmée »

Les termes qui ont été soulignés sont ceux qui sont présents dans le patron et qui devront être modifiés en fonction des résultats obtenus.

Grâce à toutes ces informations, nous pouvons construire la requête SPARQL correspondant à ce patron :

```
SELECT ?a ?b ?c ?d ?e WHERE {
  ?a <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://ontoMemo#Acteur> .
  ?b <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://ontoMemo#Prix> .
  ?c <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://ontoMemo#Festival> .
  ?d <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://ontoMemo#Annee> .
  ?e <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
<http://ontoMemo#Œuvre_filmee> .
  ?a <http://ontoMemo#participe> ?e.
  ?a <http://ontoMemo#reçoit> ?b.
  ?b <http://ontoMemo#date> ?d.
  ?b <http://ontoMemo#cadre> ?c.
}
```

4- Mapping sur les patrons

Il s'agit de l'opération la plus difficile à réaliser puisqu'il faut, pour chaque patron, regarder si les mots-clés ont une correspondance avec les concepts de celui-ci. Cela implique que l'on peut avoir plusieurs mappings par patron, car un mot-clé peut correspondre à plusieurs concepts et on peut avoir plusieurs mots-clés pour un concept. Par exemple, si un patron est composé de deux concepts « Acteur » et que l'utilisateur a saisi le mot-clé « Jean Reno », nous aurons un premier mapping dans lequel « Jean Reno » sera le premier acteur et le second dans le deuxième mapping. On remarque donc que ce n'est pas un algorithme aussi simple que nous le pensions au départ, puisqu'il s'agit d'un de ceux que l'on nomme combinatoire. Il est très dur à mettre en place et est très couteux au niveau des ressources. Cela sera expliqué dans la suite de ce dossier et plus précisément dans la partie concernant la programmation.

5- Correspondance Mot-clé / concept du patron

Un exemple vaut mieux qu'un long discours. Imaginons que nous avons le concept du patron qui est « Acteur ». Voici les possibilités de mots-clés que l'utilisateur peut avoir entré dans sa recherche ainsi que la façon de noter leurs correspondances :

- Acteur : nous avons donc ici une égalité de concept

  $\tau^{1}_{=}$([Acteur :*]) = [Acteur :*]

- Jean Reno : Jean Reno est un acteur, donc Jean Reno est une spécialisation du concept Acteur, c'est donc un cas infériorité

  $\tau^{1}_{<}$([Acteur :Jean_Reno]) = [Acteur :*]

- Personne : un acteur est une personne, donc le concept Personne est supérieur au concept Acteur

  $\tau^{1}_{>}$([Personne :*]) = [Acteur :*]

- Film : ici il n'y a aucune correspondance entre ces deux concepts, c'est donc une différence, sans notation

6- Classement des meilleurs mappings

Afin de classer tous les mappings trouvés lors de l'opération précédente, nous avons proposé un degré de pertinence des résultats qui tient compte des correspondances entre les mots-clés désambiguïsés et les concepts des patrons.

$$Pertinence(mapping) = (|C_{=}| + (0.9 * |C_{>}|) + (0.9 * |C_{<}|)) / (1 + (0.9 * |p_{only}|))$$

$|C_{=}|$ correspond au nombre de concepts identiques.

$|C_{<}|$ correspond au nombre de cas inférieurs.

$|C_{>}|$ correspond au nombre de cas supérieurs.

$|p_{only}|$ correspond au nombre de concepts du patron qui n'ont aucune correspondance.

7- Transformation de mappings sous forme de phrases

Je vous ai montré précédemment que les patrons étaient aussi caractérisés par une phrase .A présent que nous avons tous les mappings, nous allons les transformer en phrases afin que l'utilisateur les comprenne et choisisse celui qui correspond le plus à sa recherche.

8- Génération de la requête finale

Une fois que l'utilisateur a fait son choix dans les phrases présentées, nous devons générer la requête finale pour obtenir les résultats attendus par l'usager. Celle-ci est basée sur les informations du patron, où nous avons remplacé les termes qui ont été identifiés précédemment.

9- Exécution de la requête et affichage des résultats

Une fois arrivés à cette opération, nous sommes à la fin de l'application. En effet, nous avons obtenu la requête à exécuter, il ne reste plus qu'à la lancer et à en récupérer les résultats souhaités. Tous seront affichés sous forme de phrases comme pour le choix du mapping.

b) Avantage du web sémantique pour notre application

Le web sémantique permet de représenter des données de manière structurée, cela permet ainsi l'utilisation de ces données par des machines et notamment des ordinateurs. C'est dans le but d'exploiter ces avantages que nous le mettons en place dans cette application.

c) La conception

Avant de commencer à programmer, j'ai choisi de réaliser une étape de conception. En effet, cela m'a permis de mieux comprendre le sujet et de mieux répartir les tâches à effectuer par chacune des parties. Vous pouvez voir ci-dessous la première version de cette conception, qui n'était pas tout à fait conforme à ce qui était attendu.

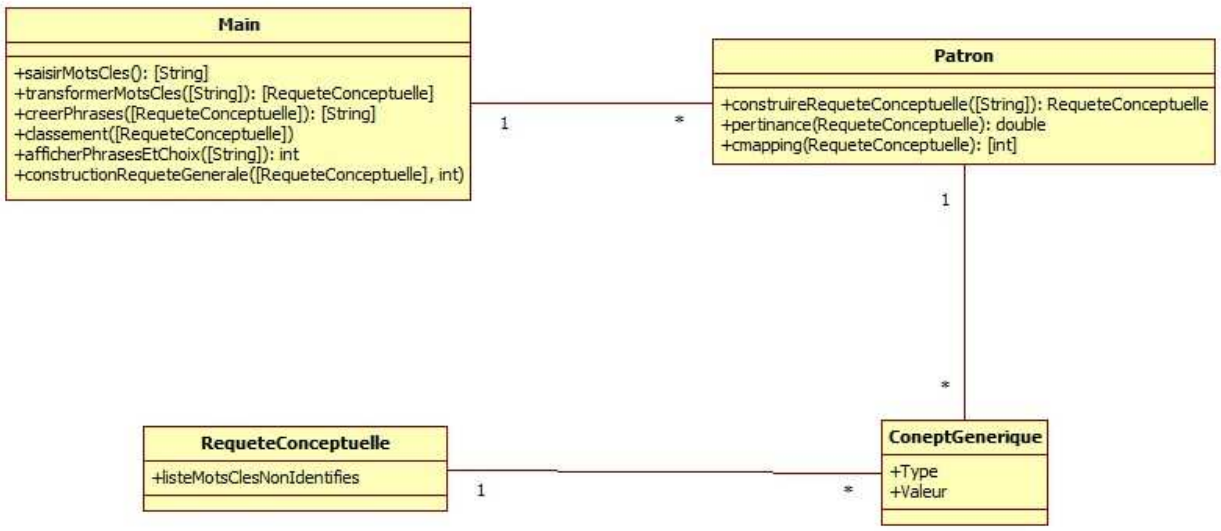

Figure 7 : Première conception

J'ai donc repris ce diagramme des classes afin de le faire correspondre aux attentes de mes professeurs, et qui me permettrait de mieux réaliser ce projet. Pour celui-ci je me suis davantage basé sur l'aspect MVC (Modèle - Vue - Contrôleur) qui est la meilleure façon de réaliser un projet informatique. Cela permet d'être plus modulaire.

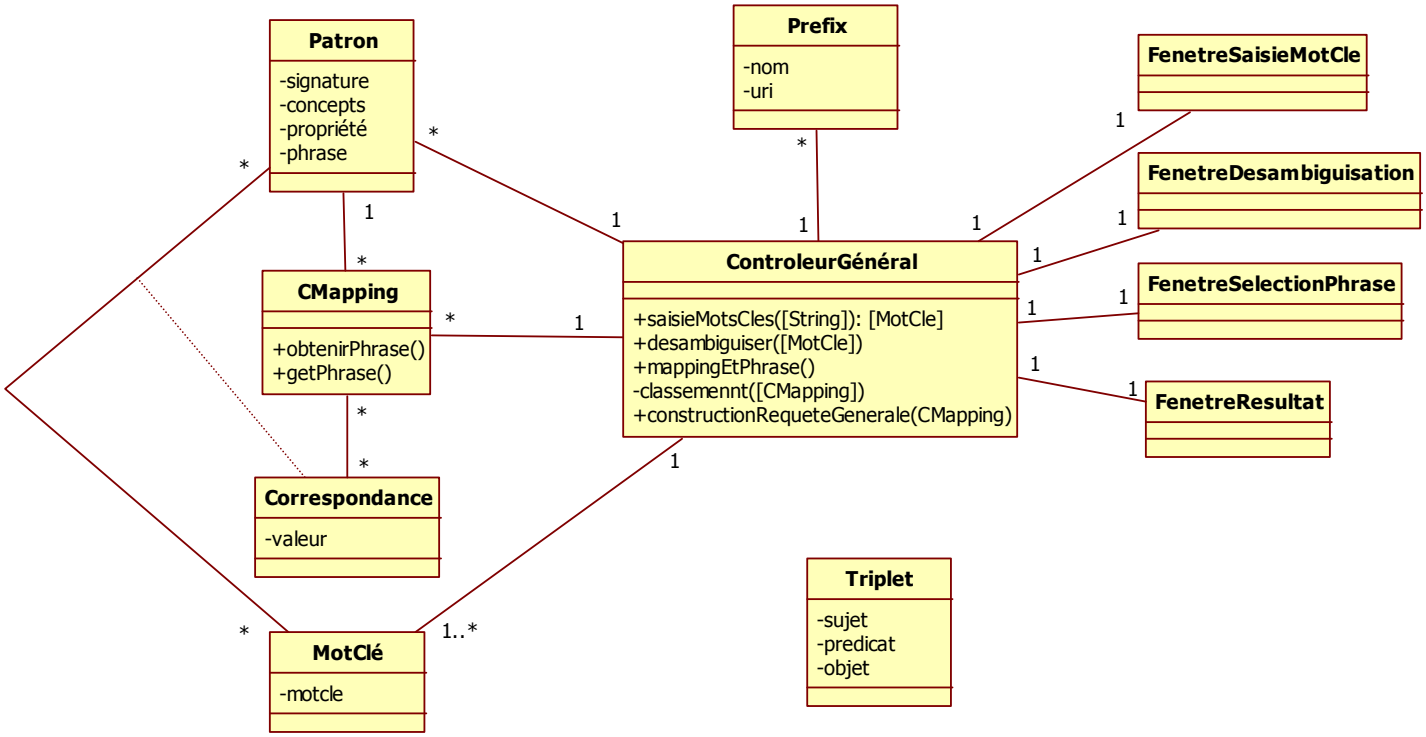

Figure 8 : Seconde conception

d) Les choix

J'ai testé plusieurs API qui implémentent les fonctionnalités du web sémantique, pour déterminer quel langage serait le mieux adapté, entre Java et Ruby, pour développer cette application. Cela aura été mon occupation principale durant une semaine entière.

Pour le langage Ruby, j'ai essayé deux API :

- RDF.rb : C'est une très bonne API, facile d'emploi mais il est impossible de trouver comment lancer une requête SPARQL sur un fichier en local. Cependant elle fonctionne très bien pour les fichiers en ligne ou/et les endpoints.

- RedLand : c'est une API qui est partagée en plusieurs parties. Il fallait donc tout rassembler afin qu'elle fonctionne. Je n'y suis jamais parvenu.

Pour le langage Java, j'ai essayé trois nouvelles API :

- Sesame : Tout était implémenté mais ce n'est pas tout à fait le langage SPARQL qui était utilisé pour interroger les triplets, il s'agissait du SeRQL qui en est très proche.
- OWL API : Il n'y avait pas de prise en charge de SPARQL.
- Jena : C'est le regroupement de plusieurs types de raisonneurs autrement appelé moteur d'inférence, il s'agit d'un programme informatique qui tente de tirer des réponses à partir d'une base de connaissances. Par exemple, cette API possède le raisonneur transitif, le raisonneur OWL ou encore le raisonneur règle générique. Le langage SPARQL est bien implémenté. Nous pouvons aussi bien interroger un endpoint qu'un fichier en local.

C'est donc l'API Jena que j'ai sélectionné car c'est celle qui correspondait le plus à toutes nos recherches.

V. La partie programmation

Afin de partir sur de bonnes bases pour la réalisation de l'application, j'ai demandé un récapitulatif des grandes lignes du projet cela nous a permis de mettre en place un calendrier. Dans un premier temps, j'ai travaillé sur des fichiers RDF et OWL en local. Par la suite j'ai interrogé un endpoint dont je donnerai davantage d'informations dans la continuité de ce dossier

a) Saisie des mots-clés

Cette partie de l'application fut très simple à réaliser ainsi le temps que j'y ai consacré fut bref. En effet il suffisait de réaliser une interface afin que l'utilisateur puisse saisir ou supprimer les mots-clés selon sa convenance.

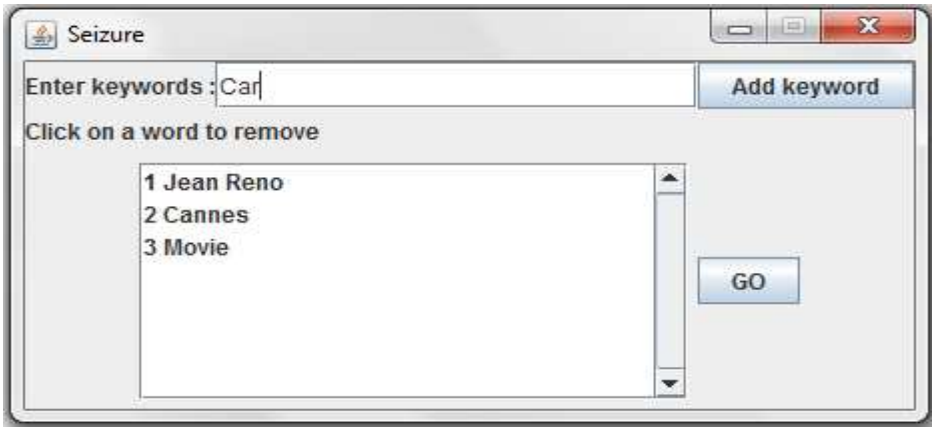

Figure 9 : Fenêtre de saisie des mots-clés

Afin de simplifier la compréhension sur le déroulement de l'application, je vais illustrer chaque étape par le rendu visuel que l'utilisateur perçoit avec comme mots-clés : « Jean Reno », « Cannes », « Film » et « Voiture ». Cela présente le souhait selon lequel nous voulons voir les films de Jean Reno qui ont été récompensés au festival de Cannes. Le mot-clé « Voiture » est rajouté afin de démontrer que l'application ne prend en compte que les termes portant sur le cinéma.

b) Désambiguïsation des mots-clés

Pour mettre en place la désambiguïsation, je devais installer dans un premier lieu l'API Jena pour faire des requêtes SPARQL, car le système de cette étape est basé sur une correspondance entre le mot-clé saisi et le label d'une classe, d'une instance ou d'une propriété.

Grâce aux tests que j'ai effectués lors du choix de l'API, j'ai su comment utiliser celle-ci. Il ne me restait plus qu'à construire la requête qui permettrait de récupérer toutes les correspondances. Cependant, il ne faut pas oublier que l'utilisateur peut saisir des caractères spéciaux au moment où il entre les mots-clés. Il a donc était nécessaire que je trouve le moyen de les supprimer de la saisie. Dans ce but j'ai cherché sur Internet une fonction. Puis j'ai découvert celle d'une personne en ayant développée une qui réalisait exactement ce que je souhaitais. Cependant elle était écrite en Php, j'ai donc du l'adapté à Java. Au final, voici la requête que je lance pour obtenir toutes les URI des classes, des instances et des propriétés qui correspondent au mot-clé :

```
SELECT ?s WHERE { ?s <http://www.w3.org/2000/01/rdf-schema#label> "motCleSaisi" }
```

Une fois toutes les URI stockées, nous allons demander à l'utilisateur de lever les ambiguïtés restantes s'il y en a. Effectivement certains termes peuvent être ignorés si la requête ne leur a retourné aucune correspondance. D'autres peuvent ne pas être ambigus car la requête n'aura renvoyé qu'un seul résultat. Au final de cette étape chaque mot-clé est associé à zéro ou à une URI de la base.

Figure 10 : Fenêtre de désambiguïsation

On remarque que dans l'exemple donné plus haut :

- Le terme « Jean Reno » et celui de « Film » ne sont pas ambigus, donc ils ont bien été trouvés dans la base RDF + OWL.
- Le terme « Cannes » qu'en a lui est ambigu car il correspond à la ville de Cannes comme au festival de Cannes. Nous sélectionnons le premier choix.
- Le terme « Voiture » a été ignoré. Il n'a pas été trouvé puisqu'il ne concerne pas l'environnement du septième art.

c) Représentation des patrons

Avant d'effectuer les mappings sur les patrons, il faut formaliser ces derniers pour que l'application puisse les interpréter. Pour cela, j'ai proposé d'utiliser le langage XML car il est très structuré et très strict lorsqu'il est associé à ce que l'on appelle une DTD.

J'ai donc proposé la DTD suivante :

```
<!ELEMENT patterns (allprefix,pattern*)>
<!ELEMENT allprefix (prefix)*>
<!ELEMENT prefix EMPTY>
<!ATTLIST prefix prefixName CDATA #REQUIRED>
<!ATTLIST prefix uri CDATA #REQUIRED>

<!ELEMENT pattern (sentence,description)>
<!ELEMENT sentence (sentenceConcept|allwords)*>
<!ELEMENT sentenceConcept EMPTY>
<!ATTLIST sentenceConcept id CDATA #REQUIRED>
<!ATTLIST sentenceConcept determinant CDATA #IMPLIED>
<!ATTLIST sentenceConcept conceptName CDATA #REQUIRED>
<!ELEMENT allwords (#PCDATA)>
<!ATTLIST allwords value CDATA #REQUIRED>
<!ELEMENT description (signatures,concepts,properties)>

<!ELEMENT signatures (signature)*>
<!ELEMENT signature EMPTY>
<!ATTLIST signature id CDATA #REQUIRED>
<!ATTLIST signature concept CDATA #REQUIRED>
<!ATTLIST signature instance CDATA #REQUIRED>

<!ELEMENT concepts (concept)*>
<!ELEMENT concept EMPTY>
<!ATTLIST concept subject CDATA #REQUIRED>
<!ATTLIST concept prop CDATA #REQUIRED>
<!ATTLIST concept object CDATA #REQUIRED>

<!ELEMENT properties (property)*>
<!ELEMENT property EMPTY>
<!ATTLIST property subject CDATA #REQUIRED>
<!ATTLIST property prop CDATA #REQUIRED>
<!ATTLIST property object CDATA #REQUIRED>
```

Ci-dessous est visible un exemple quant à l'application de la DTD présentée précédemment.

```
<!DOCTYPE patterns SYSTEM "patterns.dtd">
<patterns>
  <allprefix>
    <prefix prefixName="rdf" uri="http://www.w3.org/1999/02/22-rdf-syntax-ns#" />
    <prefix prefixName="onto" uri="http://ontoTestFilms#" />
  </allprefix>
  <pattern>
    <sentence>
      <sentenceConcept id="?a" determinant="An" conceptName="actor" />
      <allwords value="who has obtained" />
      <sentenceConcept id="?b" determinant="a" conceptName="prize" />
      <allwords value="during" />
      <sentenceConcept id="?c" determinant="a" conceptName="festival" />
      <allwords value="in" />
      <sentenceConcept id="?d" determinant="a" conceptName="year" />
      <allwords value="for" />
      <sentenceConcept id="?e" determinant="a" conceptName="shooted work" />
    </sentence>
    <description>
      <signatures>
        <signature id="?a" concept="Actor" instance="*" />
        <signature id="?b" concept="Prize" instance="*" />
        <signature id="?c" concept="Festival" instance="*" />
        <signature id="?d" concept="Year" instance="*" />
        <signature id="?e" concept="Shooted_work" instance="*" />
      </signatures>
      <concepts>
        <concept subject="?a" prop="rdf:type" object="onto:Actor" />
        <concept subject="?b" prop="rdf:type" object="onto:Prize" />
        <concept subject="?c" prop="rdf:type" object="onto:Festival" />
        <concept subject="?d" prop="rdf:type" object="onto:Year" />
        <concept subject="?e" prop="rdf:type" object="onto:Shooted_work" />
      </concepts>
      <properties>
        <property subject="?a" prop="onto:takes_part_in" object="?e" />
       <property subject="?a" prop="onto:has_received" object="?b" />
       <property subject="?b" prop="onto:obtained_in" object="?d" />
       <property subject="?b" prop="onto:obtained_at" object="?c" />
      </properties>
    </description>
  </pattern>
</patterns>
```

On remarque qu'il y a dans tout le patron un identifiant du type « ?lettre ». Chaque identifiant correspond à un concept à interroger lors de la requête SPARQL finale. De plus, au sein de la balise « phrase » est visible une balise « conceptPhrase », qui correspond à un concept qui peut être interrogé par l'utilisateur et qui doit apparaître dans la phrase du patron. Il est utile de le noter ainsi car au moment de construire les phrases associées au mapping, on va remplacer cette partie de la phrase par le mot-clé saisi s'il y a correspondance. Dans le cas contraire, on affiche le déterminant avec le nom du concept. La signature, les propriétés et les concepts correspondent exactement à la description des patrons vue précédemment dans ce dossier.

Pour lire le document XML contenant tous les patrons, j'ai utilisé l'API SAX qui signifie : Simple API for XML. En effet, elle est moins couteuse en mémoire et donc plus rapide que l'API DOM (Document Object Model). Cette dernière API doit construire un arbre contenant l'intégralité des éléments du document en mémoire. Ainsi si nous avons un document au volume important en entrée, cela prendra tout l'espace mémoire disponible.

d) Etablissement des mappings

Je vais expliquer toute la démarche qui a été mise en place afin de réussir cet algorithme. Par la suite je présenterai tous les tests que j'ai réalisés afin d'évaluer la robustesse de celui-ci.

1- L'algorithme

Cet algorithme est effectué sur chaque patron. Dans un premier temps, il fallait évaluer toutes les correspondances entre tous les concepts de ce dernier, ainsi que tous les mots-clés. Pour faire cela, j'ai programmé une fonction, qui après quelques tests avec des requêtes SPARQL, donnait le type de correspondance entre l'URI d'un concept et l'URI associé à un mot-clé Toutes ces correspondances sont stockées par concept. Pour cela j'ai créé une classe « ConceptMotsClesCorrespondances », qui contient un concept, la liste des mots-clés avec leur correspondance et également un indice de pile initialisé au départ à zéro. Lorsqu'il n'y a aucune correspondance entre le concept et le mot-clef ils ne sont pas pris en compte.

J'ai choisi cette représentation en arbre car on voit plus facilement les différents mappings à obtenir.

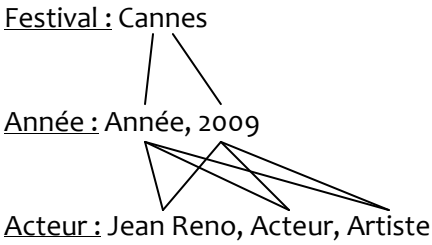

Figure 11 : Représentation des mappings

Une fois ces étapes réalisées, nous arrivons à l'algorithme combinatoire pour la création de tous les mappings. Il faut mettre en place une boucle qui va continuer tant qu'il y a un changement. Dans celle-ci nous allons parcourir tous les « ConceptMotsClesCorrespondances » qui ne sont pas vides. On effectue une nouvelle boucle tant que l'URI du mot-clé à l'indice de pile est présente dans le mapping. Si l'on continue la boucle, on vérifie l'existence d'une URI après celle qui est actuellement utilisée. Dans ce cas on passe l'URI suivante, sinon cela signifie que toutes les URI du concept sont déjà dans le mapping actuel. Il faut donc prévoir une autre structure pour stocker tous les mots-clés du concept actuel, il s'agit en l'occurrence de la classe « MotCleConceptIgnore » qui contient un mot-clé du concept, le concept lui-même et un booléen qui nous permet de savoir si l'on doit le remplacer ou non dans le mapping. Effectivement nous devrons exploiter ces autres possibilités un peu plus tard.

Cependant, avant d'ajouter le mot-clé dans la liste des ceux qui sont ignorés, il est nécessaire de vérifier s'il n'y est pas déjà présent. Si cela est le cas et qu'il est noté pour être remplacé, il faut changer le concept et la correspondance de ce mot dans le mapping.

Une fois la boucle terminée (celle pour obtenir un mot-clé non-présent dans le mapping), nous allons vérifier s'il reste des mots-clés dans le concept. Si c'est le cas, on note celui-ci comme un concept à incrémenter lors de la prochaine boucle sur tous les autres.

Maintenant, on quitte la boucle qui était sur tous les concepts. Avant de continuer, on vérifie que le mapping contient bien le nombre minimal de concept à instancier. Puis on ajoute tous les mots-clés qui sont ignorés et on vérifie que ce n'est pas un mapping vide, c'est-à-dire qu'il n'est pas uniquement composé de ceux qui ne sont pas pris en compte.

Nous sommes maintenant dans la partie qui permet de savoir si l'on perdure la boucle principale. Pour cela on effectue une nouvelle boucle qui continue tant que tous les mots-clés ignorés ne sont pas analysés, et tant que le booléen « fini » donne le résultat faux. Ensuite, nous testons si l'élément en cours de traitement est équivalent au premier

et s'il est à remplacer. Dans ce cas, on supprime le premier élément de la liste. Dans le cas contraire on passe au suivant.

Dans le cas où l'élément est supprimé, il est nécessaire d'effectuer d'autres tests. S'il n'y a plus de mots-clés ignorés et si le concept possède un autre mot-clé alors on augmente l'indice de la pile de celui-ci. On continue la boucle après avoir ajouté le mapping à la liste ceux du patron.

Dans l'autre cas, on vérifie que l'on n'a pas encore fait l'ajout du mapping, que l'on possède encore des mots-clés ignorés et que le premier n'est pas à remplacer.

Dans un autre cas, on continue la boucle après avoir ajouté le mapping à la liste ceux du patron et on indique que le premier mot-clé ignoré est maintenant à remplacer. Sinon, on indique que l'on fait une suppression d'élément. On ferme la boucle sur les mots-clés ignorés.

Avant de continuer, il faut tester si on déjà ajoutait le mapping. Dans ce cas, on ne fait rien et dans le cas contraire on teste si on a supprimé un mot-clé ignoré et on y ajoute le mapping. S'il n'y a plus de mot-clé ignoré alors on ajoute le mapping et on continue la boucle tant que l'on a un concept avec un autre mot-clé. S'il nous reste des mots-clés ignorés, on continue la boucle et on précise que le premier mot-clé ignoré est à remplacer.

2- Les tests

Afin de vérifier la robustesse de l'algorithme réalisé, j'ai effectué plusieurs séries de tests réalisés sur l'application et qui sont présentés ci-dessous. D'autres ont été effectués mais n'ont pas été notés dans le tableau ci-joint.

Concepts présents dans le patron	Mots-clés saisis par l'utilisateur
Lieu, Artiste	Cannes, Jean Reno
Lieu, Artiste	Cannes, Jean Reno, Voiture
Lieu, Acteur, Année	Cannes, Jean Reno, Acteur, Artiste, 2009
Lieu, Acteur, Année	Cannes, Année, 2009, Acteur, Jean Reno
Acteur, Acteur	Jean Reno, Acteur
Acteur, Acteur	Jean Reno, Acteur, Artiste
Lieu, Ville, Artiste, Acteur, Evènement, Festival	Cannes, Jean Reno, Festival de Cannes

Acteur, Acteur, Festival, Festival	Jean Reno, Acteur, Festival de Cannes, Festival
Festival, Evènement, Année, Acteur	Festival de Cannes, Festival de Deauville, Année, 2009, Jean Reno, Acteur, Artiste
Acteur, Acteur, Année	Jean Reno, Acteur, Année, 2009
Acteur, Acteur, Année, Lieu	Jean Reno, Acteur, Artiste, Année, 2009, Cannes
Acteur, Acteur, Festival, Festival	Jean Reno, Festival de Cannes
Acteur, Acteur, Acteur	Jean Reno
Acteur, Acteur, Acteur, Ville, Ville	Jean Reno, Acteur, Cannes
Acteur, Acteur, Acteur, Acteur	Jean Reno
Acteur, Acteur, Acteur, Acteur, Ville	Jean Reno, Acteur, Cannes
Acteur, Acteur, Acteur, Acteur, Ville, Ville	Jean Reno, Cannes
Acteur, Acteur, Acteur, Acteur, Ville, Ville, Ville	Jean Reno, Cannes

Tableau 1 : Différents tests effectués sur l'application

e) Affichage des mappings sous forme de phrases

Avant de pouvoir afficher les phrases, il faut classer les mappings afin que celle qui corresponde le plus à la demande de l'utilisateur soit placée dans les premiers résultats. Ainsi j'ai implémenté une fonction qui calcule la pertinence d'un mapping. Je l'ai utilisé conjointement à un algorithme de tri.

Cette partie est réalisée sur chacun des mappings et cela correspond à une juxtaposition des ensembles de mots et des concepts qui peuvent être remplacés par les mots-clés saisis par l'usager. A chaque fin de phrase, nous allons ajouter tous les mots-clés qui sont ignorés.

On obtient ainsi les résultats suivants pour notre exemple :

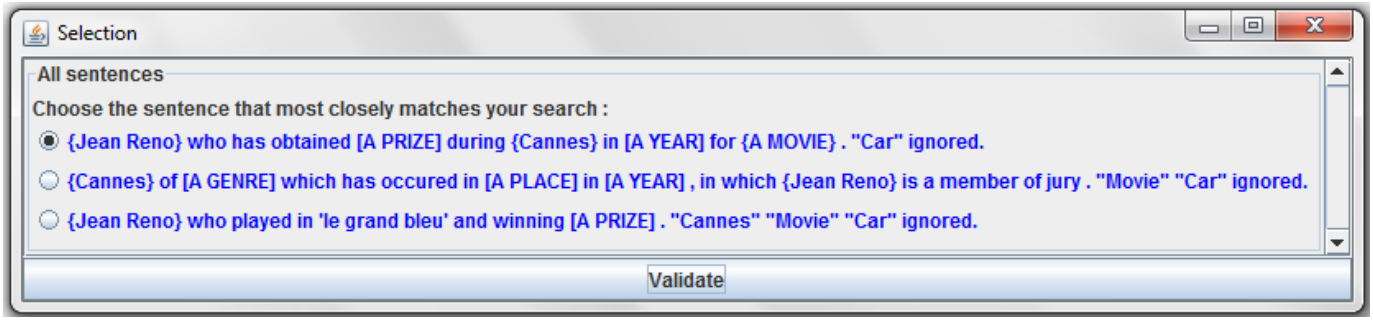

Figure 12 : Affichage des mapping sous forme de phrase

On remarque ici que la première correspond exactement à notre recherche. C'est donc celle que nous allons sélectionner.

f) Construction de la requête SPARQL finale et affichage des résultats

Pour la construction de la requête finale, c'est chaque mapping qui crée la sienne car chacun est associé à un patron. Il récupère donc toutes les informations de ce dernier tel que les concepts non interrogeables, les concepts interrogeables et les propriétés. Il y a deux styles de requête qui peuvent être générées, soit une requête sous forme de SELECT, soit sous forme de ASK.

On choisit la forme ASK lorsque tous les concepts du patron, qui peuvent être interrogés par l'utilisateur, sont instanciés. Dans ce cas la réponse qui sera affichée sera soit « La phrase est vraie » soit « La phrase est fausse ».

Le SELECT est choisi dans tous les autres cas. Une fois la requête générée, les résultats obtenus seront affichés sous forme de phrases comme lors de l'opération précédente.

1- Génération de la requête

Que ce soit pour un ASK ou un SELECT, la requête est générée de la même façon. Dans un premier temps, on va prendre tous les concepts que l'utilisateur ne peut interroger, puis ceux sur lesquels il en a la possibilité et enfin toutes les propriétés. La requête se décompose en deux parties dont celle contenant toutes les variables à interroger et celle possédant les triplets entre accolades.

Pour les concepts non interrogeables, on note les noms des variables qui leurs sont affectés dans la partie variables interrogés, et le triplet en entier dans la partie entre accolades.

Pour les concepts interrogeables, si l'un d'eux est associé à un mot-clé, on garde l'URI de celui-ci et l'identifiant du concept en mémoire, afin de remplacer ce dernier

lorsqu'il est utilisé dans la requête SPARQL. Sinon on effectue la même opération que pour les concepts non interrogeables.

Pour les propriétés, si le sujet ou l'objet du triplet correspond à l'identifiant d'un concept interrogeable qui est associé à un mot-clé, on fait directement le remplacement de celui-ci par l'URI du mot-clé ; sinon on note directement le triplet dans la partie entre accolades.

2- Récupération des résultats d'un SELECT

Les résultats générés par l'API Jena sont équivalents à un tableau à deux dimensions. Nous récupérons d'abord les résultats ligne par ligne. En effet, dans chaque ligne il y a une URI pour chaque variable interrogée.

?a	?b	?c	?d
Valeur1A	Valeur1B	Valeur1C	Valeur1D
Valeur2A	Valeur2B	Valeur2C	Valeur2D
Valeur3A	Valeur3B	Valeur3C	Valeur3D

Tableau 2 : Exemple de résultats d'une requête

Pour bien structurer ces résultats, il est nécessaire de mettre en place une structure « Resultat » qui contient deux listes de même taille, au sein desquelles on associe un identifiant interrogé et l'URI du résultat. Nous aurons donc autant d'instance de la classe « Resultat » que de lignes retournées par la requête.

3- Transformation des résultats en phrase

Nous faisons ici une boucle sur tous les résultats trouvés, et on construit la phrase au fur et à mesure en utilisant également les mots-clés que l'on a saisis précédemment. Cette fonction est semblable à celle qui a été réalisée pour l'affichage des mappings sous forme de phrases.

Voici donc les résultats que l'on a obtenus selon le mapping que l'on a sélectionné.

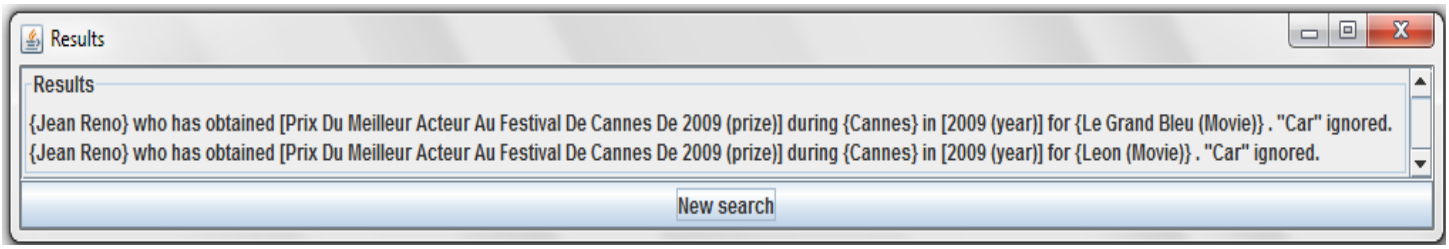

Figure 13 : Affichage des résultats de la requête

On remarque qu'il y a une partie des résultats en français. En effet, à la base, tous les fichiers étaient en français. Puis j'ai progressivement traduis la totalité en anglais. Cependant le fichier contenant les triplets RDF n'avait pas encore été modifié.

g) Optimisation

Pour cette application, j'ai réalisé une importante amélioration au moment de faire la correspondance entre les mots-clés et les concepts des patrons. Sans cette dernière la réalisation de cette étape durait environ une minute en additionnant les temps de chaque patron. Mais grâce à la modification effectuée on obtient un temps d'environ dix à quinze secondes. Cette amélioration consiste à regrouper tous les concepts de tous les patrons et de réaliser par la suite les correspondances sur ces mêmes concepts. On obtient un gain de temps car on effectue seulement une boucle au lieu d'en exécuter autant qu'il y a de patrons. A présent, un seul test de correspondance est effectué lorsqu'il y a plusieurs concepts identiques puisqu'ils ont été regroupés en un seul.

h) Améliorations réalisées

Une des améliorations qui a été apportée à cette application fût la possibilité d'interroger un endpoint, et plus particulièrement celui de DBpedia. Cela permet d'accéder au milliard de triplets qui sont contenus dans cet endpoint. Pour ce faire, je laisse le choix au lancement du programme à l'utilisateur d'interroger les fichiers locaux (RDF et OWL) ou d'interroger l'endpoint qu'il souhaite. Cependant, cette application ne gère pas parfaitement cette dernière possibilité, car elle est plus spécialisée pour DBpedia. En effet, pour certaines requêtes SPARQL je rajoute des filtres qui ne sont valables que pour DBpedia.

L'autre perfectionnement est la transformation de l'application en service web. Maintenant l'utilisateur peut accéder directement au programme sur Internet grâce à l'adresse suivante : http://mass-cara.univ-tlse2.fr/~ntiel3-20/Films/classes/ . Tous les évènements en rapport avec cette transformation se trouve dans la partie qui suit.

i) Transformation en service web

Dans un premier temps, j'ai transformé mon application Java en service web grâce que l'on appelle les servlets. Une servlet est une application Java qui permet de créer dynamiquement des données au sein d'un serveur HTTP. J'ai donc transformé les fenêtres en page web, mais je n'ai touché à aucune des classes car j'ai utilisé le schéma MVC qui permet une vraie décomposition entre les vues, les contrôleurs et les modèles. Pour le serveur HTTP, j'ai utilisé celui qui se nomme Tomcat qui interprète correctement les servlets. Toutes ce manipulations ont eu lieu sur mon ordinateur, j'ai donc souhaitait ensuite transférer tout cela sur un serveur externe. Mais avant d'installer cela, j'ai demandé l'avis de la personne qui s'occupe de ce serveur, il m'a dit que cela été une technologie un peu dépassée, et qu'il y avait maintenant de meilleurs outils pour faire cela.

Après plusieurs recherches sur ce sujet, je me suis rabattu sur un passage de toute l'application en langage PHP, car on peut le mettre très facilement sur un serveur et cela ne demande d'effectuer aucun paramétrage.

Pour faire correctement cette opération, il a de nouveau fallu que je recherche une API qui implémente les diverses fonctionnalités du web sémantique. J'ai trouvé deux API qui correspondaient à ces critères :

- L'API ARC
- L'API RDFAPI

L'API ARC étant trop compliqué à mettre en place, j'ai préféré utiliser l'autre API. Cette API est facile à utiliser, cependant toutes les fonctionnalités n'étaient pas implémentées parfaitement, j'ai dû en effet en reprendre certaines pour ajouter la transitivité au niveau des requêtes SPARQL, et modifier la fonction qui permettait de récupérer les résultats d'une requête sur un endpoint.

Les quelques difficultés restantes étaient liées à la recherche de fonction pour les tableaux en PHP. Par exemple, pour enlever toutes les valeurs du tableau qui valent null il faut utiliser la fonction array_values, cette fonction est utile lorsque l'on supprime un mot-clé qui se trouve en première position alors que nous en avons d'autre après lui.

En plus, de mettre en place ce service web, j'ai ajouté un espace de compte où il y a trois administrateurs à savoir mes trois tuteurs. Chacun d'eux peut modifier ses fichiers : le fichier RDF, le fichier OWL et le fichier contenant tous les patrons, mais il peut aussi le faire pour l'utilisateur qui n'est pas identifié. Il a la possibilité d'interroger ou non un endpoint, et il peut faire de même pour l'utilisateur non identifié.

j) Difficultés

Les plus importantes difficultés pour réaliser cette application ont été l'implémentation de l'algorithme combinatoire pour la création des mapping et les tests qui lui sont associés car il ne fallait oublier aucun cas, mais aussi la modification de l'API PHP que j'ai utilisé car cela est toujours délicat de trouver les parties qui posent problème lorsque ce n'est nous qui l'avons programmé.

h) Améliorations possibles

Les différents perfectionnements que l'on peut réaliser dans ce programme, se situent :

- Au niveau du temps, il faudrait trouver un moyen de réduire le temps au moment d'afficher les mappings et les résultats de la requête finale
- Améliorer les filtres appliqués aux requêtes SPARQL, afin de pouvoir interroger n'importe quel endpoint
- Améliorer les espaces de comptes sur le site PHP, pour que les administrateurs puissent par exemple ajouter plus compte

VI. Autres informations

Durant ce stage, je devais gérer les heures que je faisais comme bon me sembler du moment où je respectais les trente-cinq heures obligatoires. Pour être sûr de réaliser le bon nombre d'heure, je notais toutes celles-ci dans un fichier Excel où je les comptabilisais toutes.

De plus, j'étais pendant la première semaine de mon stage, dans une salle à côté d'une salle serveur, mais vu qu'il y avait beaucoup de bruit lié à la ventilation de cette salle, on m'a dit que j'aillais être changer de salle. Je me suis donc installé dans le bureau de Caroline Thierry et de Wafa Karoui.

VII. Bilans

1- Technique

Pendant ce stage, j'ai appris tous les langages qui étaient en rapport avec le web sémantique, c'est-à-dire le RDF, le OWL et le SPARQL. Pour ce qui est du XML je l'avais déjà étudié durant ma formation.

J'ai pu comprendre la difficulté d'implémenter un algorithme combinatoire, car il faut vraiment penser à tous les cas possibles et imaginables. Et donc, il faut réaliser des tests qui permettront de vérifier la robustesse de cet algorithme.

Il a aussi fallu que j'apprenne à manier les servlets ainsi que serveur HTTP Tomcat, lorsque nous souhaitions transposer l'application en service web.

De plus, j'ai compris toute la difficulté qu'il y a à modifier une API qui préexiste, car il faut bien comprendre comment elle est structurée, afin de faire les ajustements aux bons endroits.

2- Professionnel

Sur un plan personnel, j'ai appris faire les bons choix que ce soit par rapport aux API pour trouver celle qui convient le plus à nos attentes, ou par rapport aux méthodes à utiliser pour obtenir un service web.

Il a aussi fallu bien gérer son temps, pour obtenir des résultats à la hauteur des attentes de mes professeurs vis-à-vis de mon stage. Il n'est pas seulement question de temps face au projet, il est aussi question du temps passé à travailler à savoir les trente-cinq heures hebdomadaire que je devais respecter.

J'ai dû faire correspondre au maximum l'application avec tous les besoins qu'on exprimait mes enseignants. Afin qu'ils soient les plus satisfait possible de mon programme.

Mais il ne faut pas seulement penser au temps présent, car une personne peut très bien reprendre cette application. J'ai donc pensé à cette futur personne en lui laissant des commentaires tout au long de mon programme pour qu'elle puisse bien suivre l'application et savoir où elle pourra faire les modifications.

3- Personnel

Du point de vue personnel, j'ai appris énormément de choses. Tout d'abord, j'ai compris ce que c'était que le travail enseignant-chercheur, celui-ci est de longue à la haine avant de voir une mise en place d'un projet. Mais avant même cette étape beaucoup de recherches doivent être réalisées pour définir correctement le sujet de l'analyse.

De plus, j'ai appris à travailler avec plusieurs tuteurs. Il fallait donc être à l'écoute de chacun d'entre eux et attendre parfois qu'ils se mettent d'accord sur certains points. Afin de faire des comptes rendus ou des mises au point, nous avons fait plusieurs réunion, ce à quoi je n'ai jusqu'à ce stage jamais participé.

Une notion que j'ai surtout appris durant ce stage a été celle du web sémantique car je pense sincèrement au vu son expansion qu'il fera totalement partie du web de demain.

Conclusion

Le web sémantique, les algorithmes combinatoires mais aussi le temps sont des choses que j'ai appris à maitriser. Le web sémantique était le sujet de ce stage et plus particulièrement le développement d'une application basé sur le domaine du cinéma, où l'algorithme combinatoire servant à trouver les différentes recherches possibles d'un utilisateur occupait la place centrale du programme. Toute cette application a été réalisée en Java et Php en suivant l'architecture Modèle-Vue-Contrôleur qui sont deux langages qui j'ai appris durant ma scolarité.

En utilisant toutes les informations et tout le temps qui était mis à ma disposition, j'ai pu réaliser l'application qui correspondait au maximum aux attentes de mes tuteurs de stage, en y incorporant des fonctionnalités supplémentaires.

Le fait que mes responsables soit souvent présents, que le projet soit intéressant, que son avancée se déroulait parfaitement et que tout se déroulait dans une très bonne ambiance, ont transformé ce stage en une très bonne expérience dans le domaine de la recherche. Cependant ce n'est pas tout car pour bien réussir ce stage, j'ai dû faire preuve d'autonomie pour obtenir plus de renseignements sur le web sémantique et pour les recherches d'API implémentant cette technologie.

Table des illustrations

Sources documentaires

Sources Internet

Pour les informations de l'IRIT :

- http://www.irit.fr/

Informations sur le web sémantique :

- http://www.slideshare.net/ivan_herman/tats-des-lieux-du-web-smantique
- http://igm.univ-mlv.fr/~dr/XPOSE2009/Le%20Web%203.0/technologies.html
- http://interstices.info/jcms/c_17672/ontologies-informatiques
- http://www.slideshare.net/fabien_gandon/le-futur-du-web-la-lecture-des-recommandations-du-w3c-presentation
- http://www.inria.fr/valorisation/rencontres/web-semantique/transparents/A_Napoli.pdf
- http://www.youtube.com/watch?v=1kFAzYDbCiI

Pour les différentes API testées :

- http://rdf.rubyforge.org/
- http://librdf.org/
- http://www.openrdf.org/
- http://jena.sourceforge.net/
- Complément pour l'API Jena : http://kill.devc.at/node/84
- http://owlapi.sourceforge.net/
- http://arc.semsol.org/
- http://www.seasr.org/wp-content/plugins/meandre/rdfapi-php/doc/

Documentation sur les servlets :

- http://jmdoudoux.developpez.com/cours/developpons/java/chap-servlets.php

Endpoint DBpedia :

- http://dbpedia.org/sparql