Hes·so VALAIS WALLIS
Haute Ecole Spécialisée de Suisse occidentale
Fachhochschule Westschweiz
University of Applied Sciences Western Switzerland

Travail de Bachelor 2012

Filière Informatique de Gestion

Web 3.0 des Widgets Sémantiques

Étudiant : Yacine Aghzaf

Professeur : Anne Le Calvé

Préface

Ce travail de bachelor s'inscrit dans le cadre du projet OntoManag, projet de l'institut informatique de la gestion de la Haute École Spécialisé de la Suisse Occidentale HES-SO Valais.

OntoManag a pour but d'aider à explorer de façon appliquée quatre champs de recherche et de pouvoir tester et évaluer leur réelle mise en application au travers des quatre modules suivants (Calvé, 2011):

Module 1 Gestion des métadonnées : récupération, taggage et découverte des métadonnées sur les objets concernés

Module 2 Classification des objets concernés : association à une ou plusieurs ontologies.

Module 3 Organisation et classification des ontologies.

Module 4 Recherche et suggestion d'information.

SemWidgets, objet principal de ce travail de bachelor, est une application Web développée dans le but de servir de preuve de concept dans le cadre du projet OntoManag.

Le développement de l'application SemWidgets consiste à mettre en place une infrastructure intégrant divers Widgets afin de pouvoir enrichir leur découverte et utilisation grâce à la gestion d'ontologies (via l'outil SemText), permettant ainsi la découverte thématique et la classification de ces objets.

Actuellement, l'outil SemText est un prototype de classification d'informations textuelles, développé par l'institut informatique de la gestion de la HES-SO, qui gère des ontologies. La version actuelle de cet outil est utilisée uniquement pour répondre aux besoins d'un seul projet (SoftCust), et de ce fait, l'élaboration de SemWidget a dû relever deux défis:

- La recherche et le test de différentes versions de librairies devant permettre de gérer les problèmes de compatibilité :
  - Entre l'intégration dans une application en Java Entreprise Edition
  - Et le paramétrage et l'utilisation des différentes fonctionnalités
- Les requêtes prédéfinies de l'actuelle version SemText ne permettant pas d'exploiter tout le potentiel des recherches du Web Sémantique, la méthode pour générer des requêtes SPARQL a été améliorée, offrant ainsi la possibilité de recherche par synonymes.

  De plus un système de *ranking* a été créé dans le but de perfectionner la classification des résultats de recherche en fonction de leur pertinence.

Remerciements

Je tiens à adresser mes remerciements à toutes les personnes qui ont participé à la réalisation de ce projet, tout particulièrement:

Madame **Anne le Calvé,** mon professeur responsable, pour son encadrement et ses judicieux conseils.

Monsieur **Fabian Creton** et Monsieur **Zhan Liu**, assistants à l'institut informatique de la gestion de l'HES-SO Valais, pour m'avoir consacré de leur temps et pour avoir répondu présents à l'ensemble de mes sollicitations.

Monsieur **David Russo**, professeur à l'HES-SO Valais pour ses précieux conseils techniques.

Ma famille qui m'a soutenu tout le long de ce projet.

Structure du document

Ci-dessous les parties principales de ce rapport:

1. **Introduction** décrit le contexte général et les objectifs de ce travail (à partir de la page 1).

2. **Web sémantique** introduit différentes notions du web sémantique et leur application dans le cadre du projet. (à partir de la page 7)

3. **Widgets** explique le fonctionnement et les standards qui les régissent. (à partir de la page 21)

4. **L'application SemWidgets** décrit l'architecture, les outils et technologies utilisés dans le développement de l'application. (à partir de la page 29)

5. **Le processus global** décrit l'ensemble des étapes de fonctionnement de l'application SemWidget. (à partir de la page 42)

6. **Installation et configuration** montre comment installer l'application sur un environnement neutre ou encore son mode de fonctionnement sur une machine virtuelle . (à partir de la page 64).

7. **Conclusion** résume les principaux résultats, les obstacles rencontrés et expose les possibles améliorations futures (à partir de la page 72).

Résumé

Deux modules principaux ont été développés dans le cadre de ce travail de Bachelor: un **moteur de recherche** basé sur un système de gestion d'ontologies permettant de chercher des Widgets référencés dans une base de connaissances et la mise en place d'un **système d'extraction de Widgets** à partir d'une plateforme externe tel que Netvibes ou Google gadgets. (Fig.1)

Le système d'extraction permet de:

- Chercher des Widgets par mots-clés et définir le nombre maximum des Widgets que l'on veut importer à partir des annuaires de Widgets présents sur Internet comme Google Gadgets ou Netvibes ;
- Extraire les métadonnées relatives à chaque Widget avant de les persister dans la base de données et créer un fichier texte contenant ces données ;
- Importer les fichiers TXT générés dans l'ontologie.

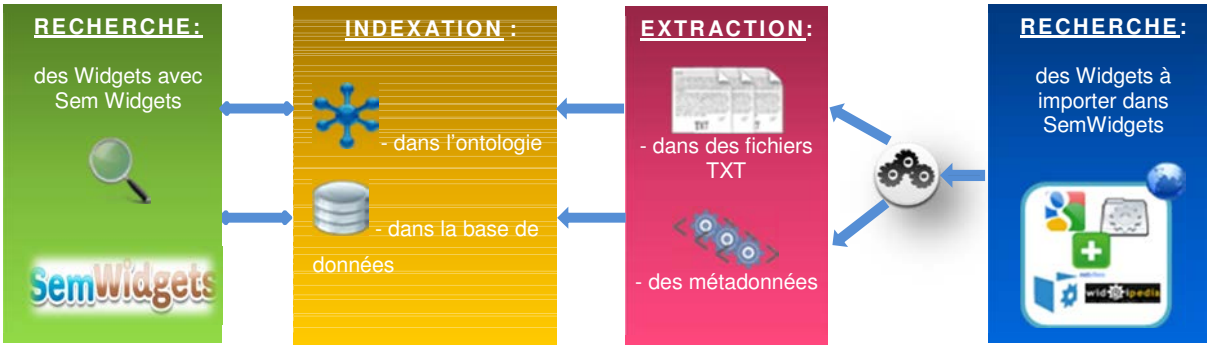


Figure 1- Mise en place du moteur de recherche

Après l'extraction et l'indexation des Widgets dans la base de données et dans l'ontologie, l'application **SemWidgets** permet de faire différentes recherches par catégories ou mots-clés avec un classement des résultats par divers critères.

Une recherche de **Widget** s'effectue en interrogeant le **module de gestion d'ontologie SemText**, développé par l'institut informatique de la gestion de la HES-SO. Le résultat de la recherche est constitué des identifiants des Widgets correspondants à la recherche.

Ces identifiants permettent de trouver les Widgets dans la base de données relationnelle pour ensuite les lister dans une page Web similaire à celles des moteurs de recherche traditionnels.

L'utilisateur peut par la suite visionner les détails du Widget qui l'intéresse et l'exporter sur une interface externe de son choix: iGoogle, Opera, Netvibes, Blogger et d'autres.

Table des Matières

1- Introduction 1

1.1 Contexte du travail de diplôme 1

1.1.1 Web sémantique 1

1.1.2 OntoManag 1

1.1.3 Problématique 2

1.2 Travail à effectuer 3

1.2.1 L'extraction automatique des métadonnées des Widgets: 3

1.2.2 Création de l'interface Web 3

1.3 État de l'art 4

2- Introduction au Web sémantique 7

2.1 Définition 7

2.2 RDF 8

2.3 Protocol and RDF Query Language (SPARQL) 11

2.4 Ontologies /OWL 13

2.4.1 OWL 15

2.4.2 WordNet 15

2.5 SemText 19

3- Widgets 20

3.1 Définition 20

3.2 Agent Utilisateur de Widgets 22

3.3 Standards 24

3.3.1 Standards Publiques 24

3.3.2 Définition des technologies à standardiser 25

3.3.3 Synthèse 27

4- SemWidgets: Technologies et Outils 28

4.1 Technologies utilisées 28

4.1.1 Plateforme Java 28

4.2 Outils et librairies 30

4.2.1 Eclipse IDE 30

4.2.2 MySQL 30

4.2.3 Hibernate 31

4.2.4 Maven 31

4.2.5 JBoss Application Server 32

4.2.6 ApacheTomcat 32

4.2.7 Streaming API for XML 32

5- SemWidgets : Architecture 34

5.1 Architecture 34

5.2 Structure des fichiers 37

6- Processus Global 39

6.1 Extraction 39

6.1.1 Métadonnées 39
6.1.2 Méthodes 40
6.1.3 Résultats 46
6.2 Stockage 47
6.3 Recherche 48
6.3.1 Méthodes 48
6.3.2 Résultats 49
6.4 Scénarios 51
6.4.1 Import 52
6.4.3 Recherche 56
7- Installation et configuration 59
7.1 Base de données 59
7.2 JBoss 59
7.3 SemText 59
7.3.1 Gate 60
7.3.2 OWLIM 60
7.3.3 Les fichiers de l'ontologie 61
7.3.4 Arguments JVM 62
7.4 Déploiement de l'application SemWidgets 62
8- Déroulement du projet 64
8.1 Planning et réalisation 64

1ère Itération: Analyse Widgets 64
2ème Itération: Choix et développement 64
3ème Itération: Intégration....64
4ème Itération: Finalisation 64
8.2 Décompte des heures....65
9- Conclusion....67
9.1 Bilan technique 67
9.1.1 Métadonnées....67
9.1.2 Recherche 68
9.2 Difficultés rencontrées 69
9.2.1 Nouvelles technologies....69
9.2.2 Création d'une application JEE from scratch....69
9.2.3 Intégration SemText 70
9.3 Les améliorations futures....70
9.3.1 Web sémantique....71
9.3.2 Ergonomie 71
9.3.3 Application mobile 71
9.3.4 Widget 71
9.4 Conclusion: avis personnel 71
10- Bibliographie....73
10.1 Référence....73

11- Tableau du décompte des heures.....75

12- Table des illustrations.....76

12.1 Table des figures:.....76

12.2 Table des tableaux.....78

12.3 Liste des annexes.....78

1- Introduction

Cette section définira dans un premier temps le contexte général du projet SemWidgets pour ensuite développer les différents objectifs poursuivis dans le cadre de ce travail de bachelor.

1.1 Contexte du travail de diplôme

1.1.1 Web sémantique

Le Web 3.0 (Web sémantique) permet le partage et la réutilisation des données non seulement entre utilisateurs mais aussi entre machines à travers diverses applications basées sur le Modèle RDF (Resource Description Framework) qui décrit les ressources Web et leurs métadonnées.

Les célèbres moteurs de recherche Google et Yahoo ont déjà commencé à s'orienter vers le Web sémantique par la reconnaissance des tags RDFa[1] dans les pages analysées. En parallèle, Google et Yahoo ont fait l'annonce d'une initiative conjointe avec Microsoft et Yandex dans laquelle ils proposent aux webmasters l'intégration des micro-données[2] dans leurs balises HTML. L'objectif est de pouvoir ajouter des marqueurs sémantiques au contenu le rendant ainsi compréhensible par les moteurs de recherches.

1.1.2 OntoManag

OntoManag est un projet de la Haute École Spécialisée de la Suisse Occidentale (HES-SO) qui a pour objectif de mettre en place une application permettant la découverte de données sémantiques relatives à un domaine précis. Sur la base des données classiques extraites (métadonnées dans notre cas), des ontologies seront développées, classifiées et exploitées.

Le projet OntoManag se compose de quatre modules:

[1] RDFa: *Resource Description Framework – in – attributes*: une syntaxe qui permet de décrire des données structurées dans une page web
[2] Informations: http://schema.org/

Tableau 1 - Modules OntoManag

Module 1	Gestion des métadonnées : récupération, taggage et découverte des métadonnées sur les objets concernés.
Module 2	Classification des objets concernés : association à une ou plusieurs ontologies.
Module 3	Organisation et classification des ontologies.
Module 4	Recherche et suggestion d'information

(source: 11_05_23_RCSO 29450_OntoManag.doc (Annexe 2)

1.1.3 Problématique

Le projet OntoManag arrive à un stade où une preuve de concept doit être développée. Il convient donc d'éprouver l'outil de classification d'informations textuelles dans des ontologies au travers d'une application concrète, et de démontrer la possibilité de l'intégrer dans un système fonctionnel. Cet outil de classification se nomme SemText.

Ce travail de diplôme devra répondre à certaines questions:

- Le système d'OntoManag SemText basé sur des ontologies, est-il efficient dans le cadre test d'une application Web basé sur des Widgets?
- Comment est-il possible d'extraire des métadonnées à partir de composants Web? Dans ce travail, les composants en question seront des Widgets (des micros applications intégrables dans des sites Web ou des blogs par exemple)
- Comment classifier ses propres Widgets face à toutes ces informations ?
- Comment aider l'utilisateur à trouver les [nouvelles applications] nouveaux objets (entendons ici Widgets) qui pourraient lui être utiles ?

Partant de l'hypothèse de l'existence d'une organisation sous-jacente et des regroupements de centres d'intérêts dans l'utilisation et l'agencement des Widgets par l'utilisateur (pédagogique, recherche…), certaines de ces informations existent déjà dans la description même des Widgets, par exemple au niveau des métadonnées (spécification W3C) ou encore au niveau de l'ontologie s'ils ont une description sémantique. D'autres sont à découvrir.

Le travail consiste à mettre en place une infrastructure intégrant des Widgets divers afin de pouvoir enrichir la découverte et l'utilisation de ces derniers. Ceci en s'appuyant sur la

gestion d'ontologies (via l'outil d'OntoManag) qui permettra ainsi la découverte thématique et la classification de ces objets.

1.2 Travail à effectuer

Le but de ce travail de diplôme est la création d'une interface Web (nommée SemWidgets) mettant à la disposition de l'utilisateur des Widgets. Cette interface communiquera avec l'outil SemText d'OntoManag qui permet de répertorier et classifier des informations textuelles – ici les métadonnées des Widgets - dans une organisation d'ontologies.

Par le biais de cette application Web, l'utilisateur pourra chercher des Widgets appropriés à ses besoins. La classification des métadonnées des Widgets dans une ou plusieurs ontologies grâce à SemText permettra ainsi à l'application de mieux comprendre la véritable signification de chaque Widget et rendra donc plus pertinente sa recherche.

Cette démarche a pour finalité de fournir une preuve de concept de SemText dans le cadre du projet OntoManag.

Ce travail traitera de deux domaines principaux:

1.2.1 L'extraction automatique des métadonnées des Widgets:

- Une analyse des différents standards et protocoles qui régissent le développement et la structuration des Widgets ;
- Proposer différents processus d'extraction des métadonnées nécessaires pour la classification dans des ontologies ainsi que dans la base de données relationnelle ;
- Développement du module d'extraction.

1.2.2 Création de l'interface Web

- Analyse des technologies à utiliser ;
- Développement de l'application Web qui communiquera avec l'outil SemText pour effectuer des recherches de Widget ;
- Implémentation de l'outil SemText dans une application Java Entreprise Edition.

1.3 État de l'art

Depuis sa création en 2001, le Web sémantique ne cesse de se développer et d'éveiller l'intérêt des chercheurs, scientifiques, entreprises, praticiens, développeurs et autre acteurs du Web. Les grandes enseignes comme Facebook, Google, Microsoft ou Yahoo s'intéressent de plus en plus au potentiel de cette technologie qui vise à faciliter l'exploitation des données structurées du web et leur exploitation par les machines.

Néanmoins, cette technologie n'a pas encore atteint un stade de maturité et nécessite de grands efforts de standardisation ainsi que d'importants investissements pour créer les bases de connaissances de divers domaines.

Afin d'obtenir un Web répondant de manière pertinente aux différents besoins de ses utilisateurs, des solutions restent encore à trouver. Les solutions actuelles existantes sont restreintes ou encore dans une phase de prototype comme le *Google Knowledge Graph* qui est une version "sémantisée" du célèbre moteur de recherche Google.

Google Knowledge Graph propose une multitude de résultats similaires ou en relation avec l'objet de la recherche de l'utilisateur, ces propositions seront visibles sur la barre latérale droite de la page des résultats (Fig. 2).

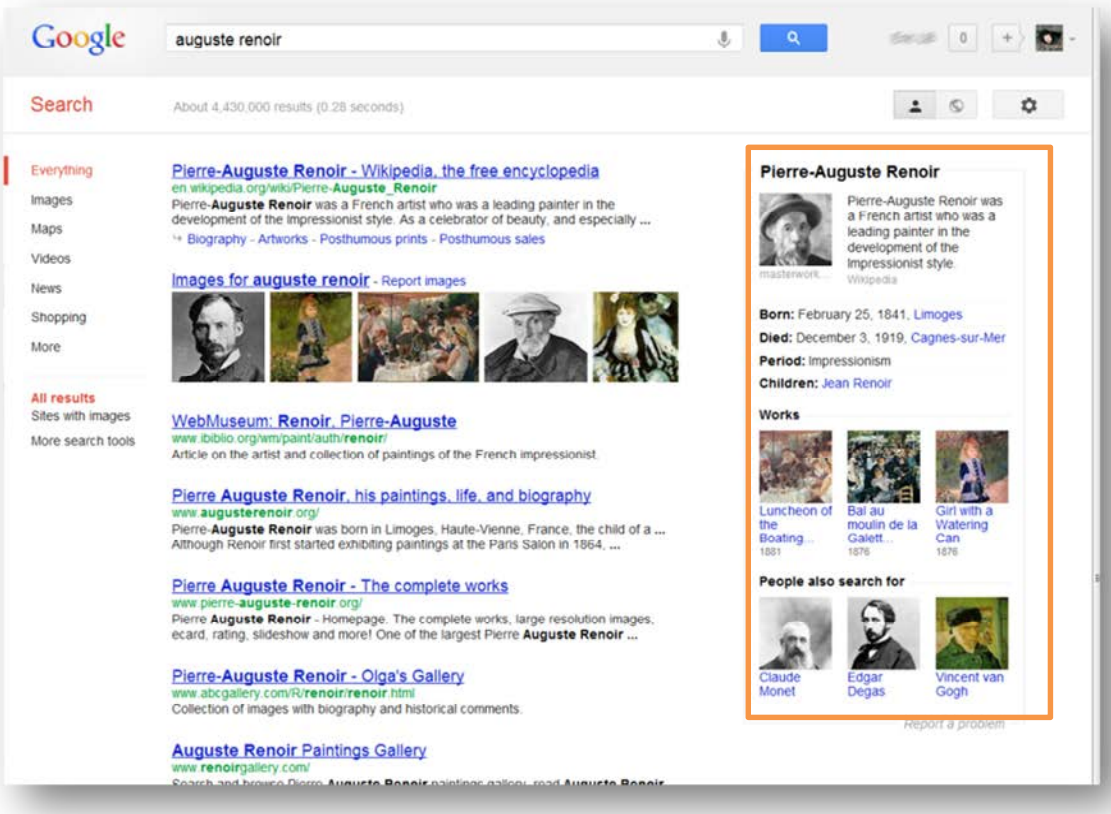


Figure 2 - Google Knowledge Graph[3]

DBpedia est un autre grand projet basé sur des technologies du Web sémantique. " DBpedia s'inscrit dans l'effort d'internationalisation de DBpedia dont le but est de maintenir des données structurées extraites de différents chapitres de Wikipedia."[4]

Des éléments clés du Web sémantique sont expliqués dans la section: Introduction au web sémantique de ce document.

En parallèle au développement du Web sémantique, de nouvelles applications assimilées à des micro-sites commencent à prendre une grande place dans le paysage Internet de nos jours. Ces micro-sites sont appelés communément des Widgets.

Touchant des thèmes très diversifiés, les Widgets séduisent un nombre considérable d'internautes ce qui représente en soit un canal potentiel de commercialisation. De ce fait,

[3] http://www.01net.com/editorial/566341/knowledge-graph-google-fait-un-pas-vers-le-web-semantique/
[4] http://wimmics.inria.fr/projects/dbpedia/doc/index.php/Accueil

plusieurs moteurs de recherches de Widgets ont vu le jour comme Google Gadgets, WidgetBox, NetVibes, Yahoo et bien d'autres. Ces moteurs de recherches offrent un large éventail de Widgets de tout genre et avec différentes thématiques.

Dans leurs algorithmes de recherche, ces moteurs ne mettent pas à profit toutes les technologies du Web Sémantique, ce dernier étant encore en phase d'élaboration. Le chapitre Widgets développera les concepts liés à ces applications plus amplement.

SemWidgets est une ambition d'expérimenter le Web sémantique pour fournir aux utilisateurs une recherche plus intelligente et plus pertinente des Widgets que celle offerte par le Web classique.

2- Introduction au Web sémantique

SemWidgets est une partie d'OntoManag qui utilise différentes technologies du Web Sémantique (Fig.3). Dans cette partie seront traitées brièvement les technologies utilisées de manière indirecte dans SemWidgets: RDF, SPARQL et Ontologie/OWL.

Des parties à ce sujet ont été retranscrites afin de garantir l'authenticité de l'information à l'égard de ces concepts de base très précieux.

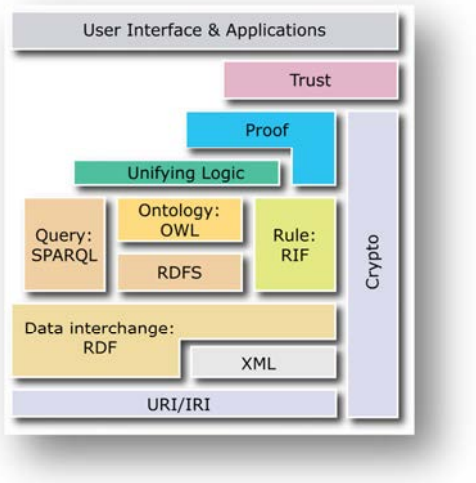


Figure 3 - Architecture du Web sémantique

2.1 Définition

Depuis sa création, le Web n'a cessé d'évoluer. Il est passé de pages statiques reliées entre elles par le biais de liens hypertextes (Web 1.0) à des interfaces simples d'utilisation accessibles aux internautes n'ayant pas forcément des connaissances techniques poussées.

Le Web 2.0 est orienté *crowdsourcing*[5], réseaux sociaux, blogs, wiki, faisant d'internet une plateforme collaborative dans laquelle l'utilisateur ne joue pas uniquement le rôle de consommateur, mais il peut également contribuer en apportant l'information et en la partageant.

[5] *Croudsourcing*: Traduit par "approvisionnement de la foule", consiste à mettre les internautes à contribution pour créer le contenu d'un site, répondre aux questions d'autre visiteurs voir participer à la conception du site (http://www.journaldunet.com/diaporama/0610-dicoweb2/1.shtml)

Actuellement, on parle de Web 3.0 ou Web sémantique, cette évolution est autant orientée machine qu'utilisateur, elle vise à rendre accessible les différentes ressources de la toile non seulement par les internautes mais également par des outils automatisés.

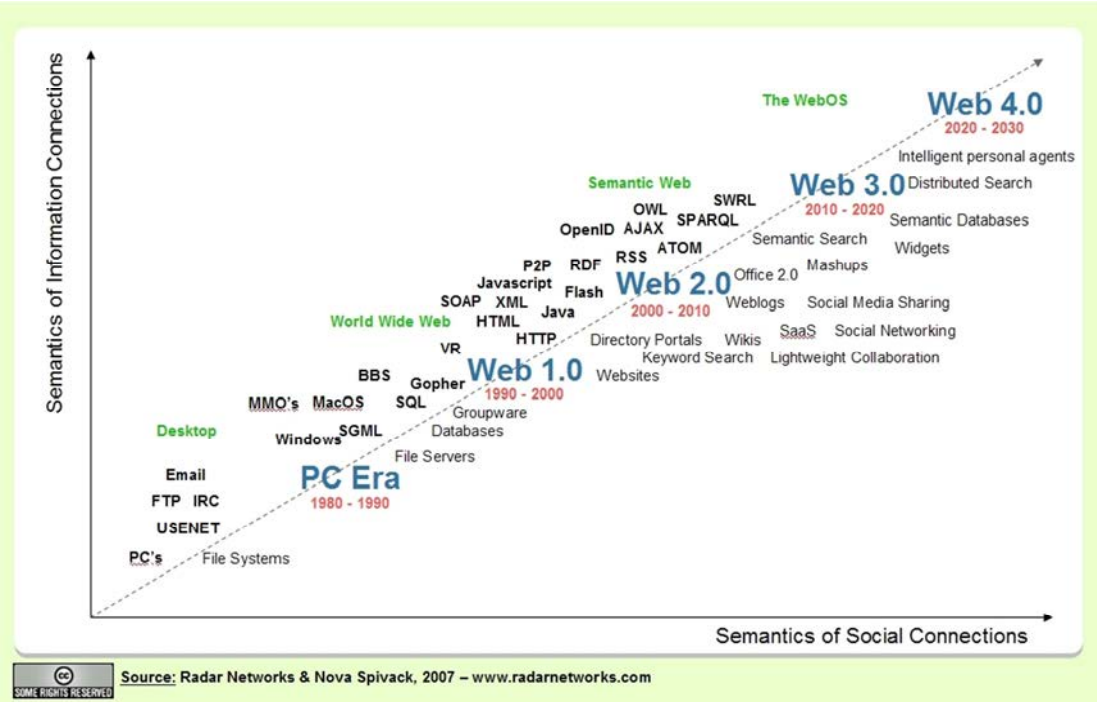


Figure 4 - Evolution du web[6]

2.2 RDF

" Les initiales RDF correspondent à "*Resource Description Framework*", ou cadre de description de ressources en français. Une ressource est simplement une chose. Une personne, un livre, un clavier, un article de blog, un aquarium, une idée, toute chose qui peut être décrite. RDF est un cadre d'applications utilisant l'architecture du Web pour décrire une ressource. Tel HTML qui permet de relier des documents à d'autres documents sur le Web, RDF permet de relier une ressource à d'autres ressources sur le Web. " (Alexander, 2007)[7]. Ce qui représente la caractéristique majeure du Web Sémantique du moment où l'on se sert de ces données RDF.

[6] http://www.slideshare.net/srvwiz/explaining-the-semantic-web-1455176
[7] http://www.yoyodesign.org/doc/digital-web/rdf-for-the-rest-of-us/

RDF a été conçu initialement par W3C dans le but de structurer l'information accessible sur le Web et de l'indexer efficacement. Ainsi, RDF permet à une communauté d'utilisateurs de partager les mêmes métadonnées pour des ressources partagées.[8]

Les caractéristiques qui rendent l'usage du RDF ergonomique sont :

1- **Toutes les données RDF ont la même forme**, ceci ne signifie pas que les données RDF sont toutes similaires. Car le RDF est loin d'être qu'un format, mais plutôt un modèle de données pouvant être publiées en différents formats tels HTML, XML, JavaScript Object Notation ou en texte ordinaire. L'avantage est que ces formats reflètent tous le même schéma de base qui est le triplet. " RDF consiste en déclarations simples appelées triplets. Un triplet se compose d'un sujet, d'un prédicat et d'un objet. "

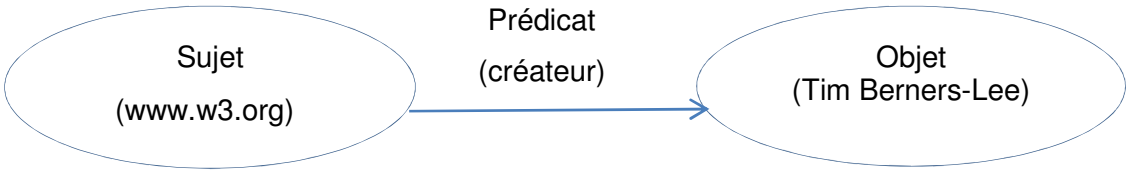


Figure 5 - Exemple de graphe RDF

La direction de l'arc est significative: il pointe toujours vers l'objet. Les nœuds d'un graphe RDF sont ses sujets et objets. Donc, RDF est simplement une structure de données constituée de nœuds et organisée en graphe.

L'exemple suivant illustre la figure 5 : La page Web www.w3.org a un créateur qui s'appelle Tim Berners-Lee.

- Le sujet représente la *chose* (**ressource**) sur laquelle porte la déclaration, et c'est toujours un indicateur de ressource uniforme, (ou adresse URI). Le site Web " **www.w3.org "** en l'occurrence ;
- Le prédicat est le nom d'une **propriété** de la ressource (tel que le nom d'un champ d'enregistrement de base de données), et c'est toujours une adresse URI. Ceci correspond au "**créateur**" ;

[8] http://xmlfr.org/documentations/tutoriels/041015-0001

- L'objet est la **valeur** de cette propriété, qui peut être une adresse URI ou un littéral (texte, nombre, date, etc.) qui est le nom du créateur, c'est à dire " **Tim Berners-Lee** ".

  Les propriétés de ce sujet peuvent varier et porter d'autres valeurs. Par exemple la page Web **www.w3.org** (le sujet) est écrite en anglais. Dans ce cas, la propriété est la langue et la valeur de cette propriété est l'anglais.

2- RDF utilise des adresses URI : Compte tenu du fait que les données utilisent la même structure de base, ceci ne suffit pas à les rendre interopérables. C'est pourquoi RDF utilise des adresses URI[9] qui sont les clés primaires d'un système beaucoup plus vaste, le Web.
Les documents XML utilisent couramment des attributs ID[10] et les bases de données relationnelles des clés primaires comme identificateurs uniques. Mais hors de leurs systèmes respectifs, ces identificateurs n'ont aucune signification.

Grâce à ces clés primaires un moyen démocratique et décentralisé pour identifier une chose de façon unique devient possible. Quiconque dispose d'un espace Web peut créer une adresse URI afin de représenter quelque chose (une ressource RDF) qui soit directement compréhensible à travers le Web.

Il n'y aura pas forcément une page Web à votre adresse URI, et il est conseillé d'avoir une description lisible par toute personne et/ou machine de ce que l'adresse représente à l'autre bout. N'importe qui peut dire tout et n'importe quoi sur tout, c'est la nature du Web, et aucun contrôle n'est donc possible sur ce que d'autres diront de d'une adresse URI.
La description de cette adresse URI, permettra de définir en lieu finale sa réelle signification.
En donnant la même forme à toutes les données et en rendant les choses qu'elles décrivent identifiables de façon unique à travers le Web, RDF accroît la portabilité, l'interopérabilité et l'utilité du code et des données.

[9] URI : *Uniform Ressource Identifier* : identifiant universel de ressource
[10] ID : identifiant

2.3 Protocol and RDF Query Language (SPARQL)

C'est un langage de requête qui permet de rechercher, ajouter, modifier supprimer des données RDF.

> "SPARQL peut être utilisé pour exprimer des interrogations à travers diverses sources de données, que les données soient stockées nativement comme RDF ou vues comme du RDF via un logiciel médiateur (middleware). SPARQL est capable de rechercher des motifs de graphe (graph patterns) obligatoires et optionnels ainsi que leurs conjonctions et leurs disjonctions SPARQL gère également le test extensible des valeurs et la contrainte des interrogations par un graphe RDF source. Les résultats des interrogations SPARQL peuvent être des ensembles de résultats ou des graphes RDF"[11]

Afin de mieux comprendre le principe du langage SPARQL voici un exemple de requête qui retourne la liste de photos, nom et description à partir d'une base RDF utilisant le vocabulaire Friend Of A Friend (FOAF[12])

Base RDF:

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:foaf="http://xmlns.com/foaf/0.1/"
  xmlns:rss="http://purl.org/rss/1.0/"
  xmlns:dc="http://purl.org/dc/elements/1.1/">
        <foaf:Person rdf:about="http://example.net/Usain_Bolt">
                <foaf:name>Usain Bolt</foaf:name>
                <foaf:img rdf:resource="http://example.net/Usain_Bolt.jpg"/>
                <foaf:knows rdf:resource="http://example.net/Michael_Phelps"/>
        </foaf:Person>
        <foaf:Person rdf:about="http://example.net/Michael_Phelps">
                <foaf:name>Michael Phelps</foaf:name>
                <foaf:img rdf:resource="http://example.net/Michael_Phelps.jpg"/>
        </foaf:Person>
        <foaf:Image rdf:about="http://example.net/Usain_Bolt.jpg">
                <dc:description>Photo de Usain Bolt</dc:description>
        </foaf:Image>
        <foaf:Image rdf:about="http://example.net/Michael_Phelps.jpg">
                <dc:description>Photo de Michael Phelps</dc:description>
        </foaf:Image>
</rdf:RDF>
```

La requête SPARQL:

[11] http://www.yoyodesign.org/doc/w3c/rdf-sparql-query/
[12] FOAF: *friend of a friend* (ami d'un ami) projet du web utilisant RDF pour décrire des personnes et les relations qu'elle entretient entre elles il est dédié aux êtres humains et au réseaux sociaux http://www.foaf-project.org/

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX dc: <http://purl.org/dc/elements/1.1/>
SELECT DISTINCT ?nom ?image ?description
WHERE {
        ?personne rdf:type foaf:Person.
        ?personne foaf:name ?nom.
        ?image rdf:type foaf:Image.
        ?personne foaf:img ?image.
        ?image dc:description ?description
}
```

- Les PREFIX définissent des raccourcis vers les espaces de nom "*namespaces*" (espace abstrait contenant des objets appartenant à la même famille)
- La clause "SELECT" comme dans SQL permet de sélectionner les valeurs à retourner (des *tuples*), des variables sont déclarées pour contenir les contraintes de la clause WHERE.
- La clause WHERE contient les contraintes en fonction desquelles les *tuples* seront sélectionnés. Dans cet exemple les propriétés nom, image et description des "Personnes" au sens du vocabulaire FOAF seront extraites comme résultats.

Le résultat de la requête est le suivant: (ici en format XML, la représentation graphique pour être changée pour afficher les résultats dans un tableau en utilisant une feuille de style XSL[13])

[13] *eXtensible Stylesheet Language* : langage de description de feuilles de style

```
<sparql xmlns="http://www.w3.org/2005/sparql-results#"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://www.w3.org/2001/sw/DataAccess/rf1/result2.xsd">
 <head>
  <variable name="nom"/>
  <variable name="image"/>
  <variable name="description"/>
 </head>
    <results ordered="false" distinct="true">
        <result>
                <binding name="nom">
                        <literal>Usain Bolt</literal>
                </binding>
                <binding name="image">
                        <uri>http://example.net/Usain_Bolt.jpg</uri>
                </binding>
                <binding name="description">
                        <literal>Photo de Usain Bolt</literal>
                </binding>
        </result>
        <result>
                <binding name="nom">
                        <literal>Michael Phelps</literal>
                </binding>
                <binding name="image">
                        <uri>http://example.net/Michael_Phelps.jpg</uri>
                </binding>
                <binding name="description">
                        <literal>Photo Michael Phelps</literal>
                </binding>
        </result>
    </results>
</sparql>
```

2.4 Ontologies /OWL

Les ontologies représentent l'une des technologies phares qui distinguent le Web sémantique du Web classique. (Fig.6)

"Une ontologie définit les termes servant à décrire et représenter un champ de connaissance. Les ontologies sont utilisées par les personnes, les bases de données et les applications qui ont besoin de partager des informations de domaine (la médecine, la fabrication d'outils, l'immobilier, etc.) [14]"

[14] http://www.yoyodesign.org/doc/w3c/webont-req-20040210/index.html

Il existe différents types de relations entre les concepts d'une ontologie il s'agit de:

- Relation de spécification: la catégorie des chats est une spécification de la catégorie des félidés.
- Relation propre à un domaine: A se trouve dans B, A est le titre de B ou bien A est le créateur de B

Ces relations sont généralement vérifiées entre des catégories d'un domaine précis. Si cette relation de spécification est établie entre deux concepts par une ontologie, cette information peut alors être utilisée par un moteur de recherche sémantique.

Exemple: si l'utilisateur envoi la requête suivante: donne-moi des images de félidés, le moteur de recherche sera capable d'inclure l'image de chat dans les résultats retournés.

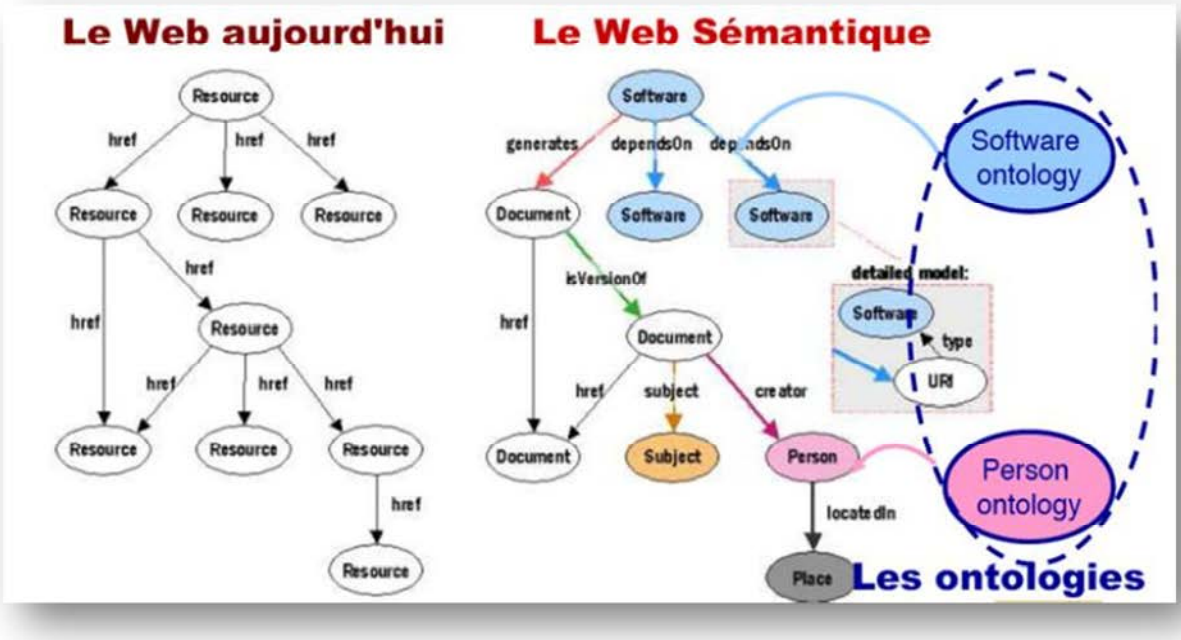


Figure 6 - Architecture Web sémantique Vs. Web classique

(source: http://www.w3.org/2001/12/semweb-fin/w3csw)

2.4.1 OWL

Web Ontology Language, est un langage basé sur le modèle RDF, il apporte aux ontologies les fonctionnalités suivantes:

- Distributivité sur plusieurs systèmes ;
- Transparence et extensibilité ;
- Compatibilité avec les besoin du Web et ses standards ce qui augmente l'accessibilité au niveau international.[15]

Il permet de structurer les données dans des ontologies en ajoutant des contraintes qualifiantes comme les propriétés de classe, la similitude ou la différence de deux ressources, transitivité, symétrie, contraire, etc.
OWL est un langage puissant constituant avec XML[16] et RDF les trois couches principales du Web Sémantique. XML est le support de sérialisation qui sert RDF et OWL dans la structuration et la définition des données et des concepts complexes remplaçant ainsi les traditionnelles bases de données relationnelles.

2.4.2 WordNet

SemText utilise la version anglaise de l'ontologie WordNet. Cette dernière représente la pièce maîtresse dans la recherche sémantique sur laquelle repose l'application SemWidgets. Par conséquent, le développement de cette partie a été traduit du site Web officiel de WordNet : www. wordnet.princeton.edu/.

2.4.2.1 Définition

Développé par le laboratoire des sciences cognitives de l'université Princeton, WordNet® représente une large base de données lexicale de la langue anglaise. Noms, verbes, adjectifs et adverbes sont regroupés en ensembles de synonymes cognitifs (*synsets*: syn pour synonyme et set pour ensemble), chacun exprimant un concept distinct. Les *synsets* sont reliés entre eux par des relations conceptuelles-sémantiques et lexicales.

Le réseau résultant de mots et concepts, reliés de manière significative, peut être parcouru à l'aide d'un navigateur. WordNet est aussi gratuitement et publiquement disponible au

[15] http://www.w3.org/2004/OWL/
[16] *xtensible Markup Language* : Langage de balisage extensible

téléchargement. Sa structure fait de lui un outil utile pour la linguistique computationnelle ainsi que pour le traitement du langage normal.

WordNet ressemble superficiellement à thesaurus[17], dans le sens où il regroupe les mots en se basant sur leur signification. Toutefois, il existe de très importantes distinctions.

Tout d'abord, WordNet relie non seulement la forme des mots (des chaînes de lettres) mais aussi les sens spécifiques des mots. Par conséquent, les mots qui se trouvent dans le réseau approximativement proches les uns des autres sont sémantiquement désambigüisés. Ensuite, WordNet labélise les relations sémantiques parmi les mots (Fellbaum, 1998), tandis que les regroupements de mots dans thesaurus ne suivent aucun critère explicite autre que la similitude des significations.[18]

2.4.2.2 Structure

La relation principale entre les mots dans WordNet est la synonymie, telle celle entre les mots : {*shut*} (fermer) et {*close*} (fermer).ou bien {*car*} (voiture) et {*automobile*} (automobile). Les synonymes – mots indiquant le même concept et qui sont interchangeables dans plusieurs contextes - sont regroupés dans des ensembles désordonnés (*synsets*). Chacun des 117000 *synsets* de WordNet sont liés à d'autres *synsets* par le biais d'un petit nombre de " relations conceptuelles ". En addition, un *synset* contient une brève définition (*gloss*) et, dans la plupart des cas, une ou plusieurs phrases courtes illustrant l'usage des éléments du *synset*. Les formes de mots ayant plusieurs significations distinctes sont représentées dans plusieurs *synsets* distincts. Ainsi, chaque paire de forme-signification est unique dans WordNet.

2.4.2.3 Relations

La relation la plus encodée parmi les *synsets* est la relation super-subordonnée (aussi appelée hyperonymie, hyponymie ou relation ISA). Elle relie de manière croissante les

[17] Thesaurus en latin signifie recueil, répertoire qui a donné naissance au dictionnaire "thesaurus linguae latinae" de Robert Estienne. Thesaurus est aussi un langage contrôlé utilisé pour l'indexation et la recherche de ressources documentaires dans des applications informatiques spécialisées.
Source: http://fr.wikipedia.org/wiki/Th%C3%A9saurus
[18] http://wordnet.princeton.edu/wordnet/

synsets généraux comme {*furniture*, *piece*_of_furniture} (mobilier, meubles) à d'autres plus spécifiques comme {*bed*} (lit) et {*bunkbed*} (lit superposé).

Ainsi, WordNet indique que la catégorie meubles inclue lit, qui à son tour incluse lit superposé ; inversement, les concepts comme lit et lit superposé constitue la catégorie meubles. Toutes les hiérarchies de noms remontent finalement le nœud de la racine {*entity*} (entité).

La relation d'hyponymie est transitive "*if an armchair is a kind of chair, and if a chair is a kind of furniture, then an armchair is a kind of furniture*." Ce qui signifie en français: si le fauteuil est une sorte de chaise, et si la chaise est un meuble, alors le fauteuil est un meuble. WordNet distingue parmi les Types (noms communs) et Instances (exemples) (personne spécifiques, pays et entités géographiques). Par conséquent, le fauteuil est une sorte de chaise, Mozart est un exemple de compositeur. Instances (exemples) sont toujours des nœuds de feuille (terminal) dans leur arbre hiérarchique.

Quant à la " méronymie ", la relation entre toutes les parties réside entre les *synsets* comme {*chair*} (chaise) et {*back, backrest*} (dos, dossier), {*seat*} (siège) et {*leg*} (pied).Les parties sont héritées de leurs subordonnées : "*if a chair has legs, then an armchair has legs as well* " : si une chaise a des pieds, alors le fauteuil en a aussi. Les parties ne sont pas héritées " en amont " comme elles pourraient être une caractéristique pour un seul type spécifique de choses plutôt qu'une classe en entier : les chaises et des types de chaises qui ont des pieds, mais pas tous les types de meubles ont des pieds.

Les *synsets* de verbes sont arrangés en hiérarchies également, les verbes vers le fond des arbres (troponyms) expriment progressivement des manières spécifiques qui caractérisent un évènement, comme dans {*communicate*} (communiquer) {*talk*} (parler) {*whisper*} (chuchoter).

La manière spécifique exprimée dépend du champ sémantique ; le volume (comme dans l'exemple ci-dessus) est juste l'une des dimensions à travers laquelle un verbe peut être élaboré. D'autres sont la vitesse (*move-jog-run*) (bouger-trottiner-courir) ou l'intensité (*like-love-idealize*) (aimer bien-aimer-idéaliser). Les verbes décrivent des évènements qui impliquent nécessairement et uni-directionnellement le fait qu'ils soient reliés les uns les

autres : {*buy*}-{*pay*} (acheter-payer), {*succed*}-{*try*} (réussir-essayer), {*show*}-{*see*} (montrer-regarder), etc.[19]

Les adjectifs sont organisés en termes d'antonymie. Des pairs d'antonymes " directs " comme mouillé-sec et jeune-vieux reflètent un contrat sémantique fort de leurs membres. Chacun de ce pôle d'adjectifs est lié à son tour à un nombre d'autres pôles " sémantiquement similaires " : sec est lié à desséché, très sec, humide, engorgé d'eau, etc.

Les adjectifs sémantiquement similaires sont des " antonymes indirects " du membre central du pôle opposé. Les adjectifs relationnels (" *pertainyms* ") pointent vers les noms desquels ils sont dérivés (*criminal-crime*).

Il y a seulement quelques adverbes dans WordNet (*hardly, mostly, really*, etc.) (à peine, la plupart du temps, vraiment). Comme la majorité des adverbes anglais, les adverbes dans WordNet sont forcément dérivés des adjectifs par le biais d'une apposition morphologique (*surprisingly, strangely*, etc.) (étonnamment, étrangement).

2.4.2.4 Relations Cross-POS

La plupart des relations de WordNet relient les mots de la même partie d'un discours (Part Of Speech, POS). Ainsi, WordNet se constitue en réalité de quatre sous-réseaux, un pour chacun des noms, verbes, adjectifs et adverbes, avec peu de pointeurs de Cross-POS (croisement des parties d'un discours).

Les relations Cross-POS incluent les liens " morphosémantiques " qui détiennent parmi les mots similaires sémantiquement ceux qui partagent une lignée avec la même signification : *observ* (verbe), *observant* (adjectif), *observation*, *observatory* (noms).

Dans plusieurs pairs de nom-verb, le rôle sémantique du nom par respect au verbe a été spécifié : {*sleeper*, *sleeping_car*} (dormeur, wagon-lit) est le LIEU pour {*sleep*} (dormir) et {*painter*} (peintre) est l'AGENT de {*paint*} (peindre), quand {*painting*, *picture*} (peinture, image) est le RESULTAT.[20]

[19] http://wordnet.princeton.edu/wordnet/
[20] http://wordnet.princeton.edu/wordnet/

2.5 SemText

Développé dans le cadre du projet OntoManag, SemText est un module qui permet de répertorier et classifier des informations textuelles dans une organisation d'ontologies. Dans le cas du projet SemWidgets, ces informations textuelles sont des Widgets.

Les technologies précitées dans ce chapitre sont utilisées par SemText :

- La classification des informations dans les ontologies : SemText passe par la couche de stockage et d'inférence qui utilise le langage OWL ;
- La recherche : SemText utilise Wordnet dans la recherche par sens ou par synonymes des mots-clés saisis dans la barre de recherche ;
- L'interrogation : SemText interroge les ontologies par le biais de requêtes SPARQL.

Voici l'exemple d'une requête SPARQL générée par SemText permettant de trouver les Widgets reliés au mot "radio":

```
PREFIX nlp:<http://www.websemantiquech/onto/ontoManageNLP#>
PREFIX xsd:<http://www.w3.org/2001/XMLSchema#>
SELECT ?text ?dbKey (sum(?point) as ?score)
WHERE {{select distinct ?text ?point dbKey{
                          ?text nlp:hasDBKey ?dbKey.
                          ?text nlp:hasTextType nlp:tt_userDesc.
                          ?text nlp:hasLookupURI ?lookup0.
FILTER(?lookup0 IN (<http://purl.org/vocabularies/rinceton/wn30/wordsense-radio-
noun-1>,
 <http://purl.org/vocabularies/princeton/wn30/wordsense-radio-noun-3>,
<http://purl.org/vocabularies/princeton/wn3/wordsense-radio-noun-2>
)).BIND("1"^^xsd:integer as ?point).}}}
group by ?text ?dbKey order by desc(?score) ?text
```

3- Widgets

Le but de cette partie est de donner un aperçu sur les Widgets, leur types et usage.

3.1 Définition

De manière globale, les Widgets sont des mini-logiciels qui peuvent être intégrés dans divers environnements: blog, site Web ou installés sur le système d'exploitation d'un ordinateur, tablette ou Smartphone. Ces petites applications proposent des informations (cours de change, infos, horaires de train, météo, etc.) ou encore des divertissements (mini jeux vidéo, radios, etc.).

Dans le cadre du processus de standardisation de Widgets établit dans un *Working Draft*[21] par le consortium du World Wide Web, W3C, un Widget est définit comme une conceptualisation de l'utilisateur final d'une application interactive à but unique servant à afficher et/ou mettre à jour des données locales ou des données existant sur le Web. Ces données sont rassemblées de différentes sources et " mixées " d'une manière à ce qu'elles permettent un seul téléchargement ou une seule installation sur une machine ou un périphérique mobile. Ainsi, elles sont présentées à l'utilisateur de manière utile et intéressante.

Un Widget peut fonctionner comme une application autonome (Fig.7) " *stand alone* " (qui fonctionne en dehors du navigateur Web) ou comme une application intégrée dans un document Web. Dans ce document, l'environnement d'exécution, dans lequel un Widget est exécuté, est considéré comme un agent utilisateur de Widget. Tandis que le Widget exécuté est appelé Widget instancié " *instantiated* ". Avant l'instanciation, un Widget existe en tant qu'une ressource Widget " *Widget resource* ". (Fig.8)

[21] http://www.w3.org/TR/2008/WD-widgets-land-20080414/

Figure 7 - Ensemble de Widgets pour Windows et Dashboard

Figure 8 - Fonctionnement d'un Widget instancié

Un Web Widget peut avoir différentes caractéristiques, entre autre :

Authentifié : où l'accès à un système d'information requiert une authentification sécurisée. Par exemple, pour ajouter des pièces de musique dans sa liste sur www.deezer.com,

l'utilisateur doit être authentifié grâce à un identifiant et un mot de passe pour réaliser sa requête.

Agrégateur : qui rassemble les informations de différentes sources de données afin de les présenter sur une seule interface à l'utilisateur. Ceci peut correspondre à la recherche de vidéo sur Youtube, ou à des actualités sur Yahoo Actualités ou bien des images sur Google, etc.

Intégrateur et collaboratif : ces deux caractéristiques peuvent être associés dans certains cas. En intégrant des éléments (commentaires ou images par exemple), l'utilisateur collabore à la réalisation d'une page Web. Les réseaux sociaux illustrent bien ces deux caractéristiques, car chaque utilisateur peut changer le contenu du Widget, dit collaboratif, en intégrant ses propres données, comme le permet la fonction " *post it* " que l'on trouve sur Facebook.

Un Widget peut être **intégrateur** sans être collaboratif dans le cas d'un album photo en ligne tel Picasa Albums Web qui permet d'afficher des photos par thème (Voyage à Bali) et modifier les résultats de cette recherche en rajoutant des photos, en les corrigeant grâce à l'éditeur ou en les supprimant.

3.2 Agent Utilisateur de Widgets

A l'instar de l'expansion d'usage des ordinateurs, depuis une décennie, et récemment des périphériques mobiles utilisant le Web comme les tablettes ou Smartphone, un nouveau genre d'application s'est proliféré de manière significative. Les développeurs appellent ce type d'application : Moteur de recherche Widgets. Il s'agit d'une partie d'un logiciel capable d'exécuter d'autres petites applications appelées Widgets ou Gadgets.

Allant de simples Widgets (comme l'horloge, des notes *post it* ou jeux) à des applications plus complexes nécessitant l'usage du Web (comme les actualités, les prévisions météo et convertisseurs de monnaie) ces Widgets ont démontré leur côté pratique et utile. Ils ont ainsi progressivement remplacé les applications traditionnelles à but unique.

Désormais, l'utilisateur n'a plus besoin d'aller sur le Web pour regarder la météo, ou lire des informations, c'est plutôt le Web qui vient jusqu'à son bureau pour le tenir informé grâce au

Widgets. Cette fonction a rendu l'usage des périphériques mobiles plus attractif. Toutefois, cette même fonction représente une vulnérabilité à l'égard de la sécurité car elle fait parvenir l'information jusqu'au bureau de l'utilisateur et pourrait changer des données dans son système. De ce fait, une porte s'ouvre aux hackers et crackers pour accéder, copier ou modifier les données personnelles d'un utilisateur de Desktop Widgets. Ainsi, Yahoo et Microsoft ont suspendu les téléchargements des Desktops Widgets car elles compromettent la sécurité et " l'intimité " de l'utilisateur.[22] En attendant un renforcement de cette dernière, des applications " *Fix it* " ont été mises à disposition pour les Desktop Widgets préexistants.

Du côté des développeurs et des fournisseurs, la plupart des Widgets sont beaucoup plus faciles à créer que les applications avec des langages de programmation de base comme Java ou C#. Car la création des Widgets utilise les mêmes technologies qu'un navigateur Web. Dans le sens où un moteur de recherche Widgets imite le comportement d'un moteur de recherche classique. Il y a même une montée dans la construction directe de Widgets au-dessus du navigateur Web permettant ainsi l'affichage des pages Web.

Cette montée n'est pas sans enjeux pour les parties prenantes (utilisateurs, développeurs, fournisseurs actuels ou futurs). Comme le montre la figure 9, la création des Widgets utilise une pile de technologies différentes d'un agent utilisateur de Widgets à un autre, remplissant différents rôles (comme le packaging et le déploiement). Par conséquent plusieurs fragmentations dans l'espace des Widgets sont constatées dans une étude menée par W3C. Ce dernier conseille une standardisation formelle de ces technologies (celles portant un astérisque Fig. 9).[23] À noter que cette pile de technologies est destinée pour servir de guide et ne représente pas la pile d'un agent utilisateur particulier.

[22] Microsoft : http://technet.microsoft.com/en-us/security/advisory/2719662
Yahoo : http://widgets.yahoo.net/blog/?p=26
[23] http://www.w3.org/TR/widgets-land/

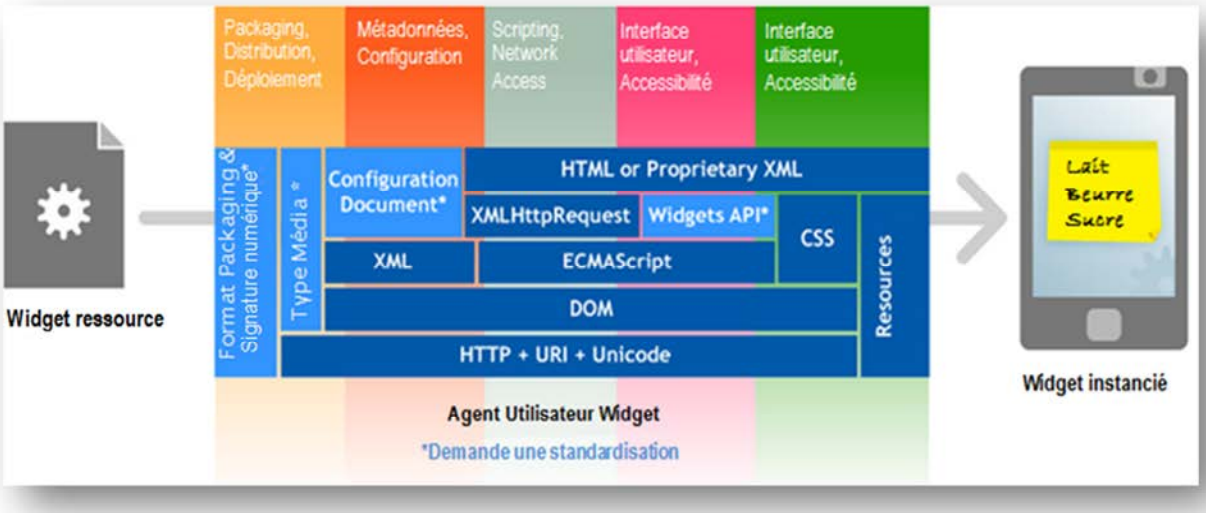


Figure 9 - Une pile typique de technologies et aspects standardisés.

(source : http://www.w3.org/TR/widgets-land//)

3.3 Standards

Cette partie présente les deux standards courants concernant la création de Widgets existant à l'heure actuelle et donne une brève explication (extraite du Working Draft de W3C)[24] de ces technologies à standardiser relatées dans la section précédente.

3.3.1 Standards Publiques

3.3.1.1 W3C Widgets 1.0

W3C (World Wide Web) Consortium est un organisme international qui, selon son fondateur Tim Berners-Lee, a pour mission de mener le Web à son plein potentiel.

Figure 10. Logo W3C

[24] http://www.w3.org/TR/widgets-land/

Pour remplir cette mission, le W3C veille à développer les standards du Web dans le but de normaliser les technologies utilisées dans ses différents développements pour favoriser une certaine interopérabilité entre ces derniers[25].

Devant l'ascension fulgurante qu'ont connue le développement et la propagation des Widgets au niveau international, le W3C a lancé une étude mentionnée précédemment dans le but de standardiser divers aspects touchant aux Widgets tel que le design, packaging, signature digital, sécurité et déploiement. À l'issu de cette étude, la première spécification a été publiée il s'agit de la " W3C Widgets 1.0 ".

Cette spécification concerne exclusivement les Widgets de type desktop ou encore les Widgets installables sur mobile et ne traite pas des Web Widgets (12041).

3.3.1.2 Open Ajax Alliance Task Force / Gadget

Open Ajax Alliance est une organisation composée de fournisseurs, projets open source et entreprises utilisant Ajax. Son objectif est de faciliter l'implémentation des technologies basées sur Ajax tout en promouvant l'interopérabilité de ces dernières. (12042)

Figure 11. Logo Open Ajax Alliance

Cette alliance comporte un nombre considérable de grandes firmes actives dans le domaine du développement Web : Google, Adobe, IBM, Oracle, Zend.

Les recommandations proposées par cet organisme ne sont pas systématiquement appliquées dans le développement des Widgets, ce qui rend l'automatisation de l'extraction des métadonnées plus difficile à réaliser dans le cadre du projet SemWidgets.

3.3.2 Définition des technologies à standardiser

3.3.2.1 Packaging format:

C'est format physique de données utilisé dans la création d'un Widget Ressource. Par exemple : un format de fichier décrit dans la référence Konfabulator de Yahoo, ou un format de fichier zip pris en charge par Opera Widgets et la Vista Sidebar de Microsoft.

[25] http://www.w3.org/Consortium/mission.html

3.3.2.2 Signature Numérique

C'est une spécification de conformité qui doit remplir deux critères :

- définir un moyen pour vérifier l'authenticité et l'intégrité des données de toutes les ressources dans un paquet de Widgets, avec l'exception de toute ressource exclue explicitement par la spécification (par exemple le fichier même de la signature numérique).
- fournir ces capacités en spécifiant ou en recommandant un modèle de traitement pour la génération et la vérification d'une signature numérique associée à un paquet de Widgets. Le schéma de la signature numérique doit être compatible avec Public Key Infrastructures (ICP), particulièrement X. 509v3.

3.3.2.3 Type Media

Le type média est formellement associé à un Widget dit Ressource appartenant à un agent utilisateur Widget. Par exemple, les moteurs de Widgets Joost exigent que les Widgets soient fournis via HTTP avec une application avec un type `application/vnd.joost.joda-archive`.

3.3.2.4 Document de Configuration

Le document de configuration est une ressource distinguée dans laquelle les auteurs peuvent déclarer des métadonnées et/ou des paramètres de configuration pour un Widget. Un agent utilisateur de Widget utilise un document de configuration pour paramétrer un Widget lors de l'instanciation. Le document de configuration pourrait définir également la relation entre les ressources dans un Widget ressource. Ce document de configuration prend généralement la forme d'un fichier XML. Par exemple, la ressource `config.xml` regroupée avec un Widget Opera.

3.3.2.5 Widget API:

L'*Application Programming Interface*, est un ensemble d'interfaces de programmation fournissant une fonctionnalité spécifique à un Widget instancié. La gamme APIs actuelle fournit l'auteur de Widgets en termes de fonctionnalités étendues. Exemple: API Microsoft permet d'accéder au système d'exploitation à travers à la barre latérale (Side Bar).

3.3.3 Synthèse

À travers les concepts du Web sémantique et des Widgets expliqués au paravent, une analyse a été faite dans but de chercher l'éventuelle existence d'un standard officiel appliqué dans le développement des Web Widgets. Car un standard officiel garantirait une structure stable des packagings des Widgets et des métadonnées en particulier. Facilitant de ce fait leur extraction.

Ces métadonnées permettent de sémantiser, de classer des Web Widgets dans des ontologies, et permettent l'exploitation des possibilités offertes par le Web sémantique. Ainsi offrent-elles une recherche plus ciblée et plus performante des Widgets.

Toutefois, il n'existe pas de standard officiel qui normalise le développement et le packaging de ces Web Widgets. A défaut, les annuaires les plus populaires du Web deviennent le seul recours pour analyser la structure des Web Widgets qu'ils proposent.

Les fournisseurs de Web Widgets Google Gadgets, Netvibes et WidgetBox ont été retenus sur le critère de l'importance de la communauté qui les utilise en vue du développement d'un processus d'extraction des métadonnées. (Yahoo ayant arrêté de fournir de Widgets, il a été exclu de cette étude).

4- SemWidgets: Technologies et Outils

Cette section expose les différentes technologies et outils utilisés pour la réalisation de ce projet. Elle donne une brève description et les motivations du choix de tel ou tel technologie.

4.1 Technologies utilisées

Pour les besoins de ce projet toutes les technologies utilisées sont "Open-source" provenant plus particulièrement de la plateforme Java.

4.1.1 Plateforme Java

La plateforme Java est une suite de logiciels permettant de développer et déployer différents types d'applications en langage Java: Sites web, applications mobiles, applications pour les industries et les entreprises et bien d'autres.

Il existe plusieurs distributions Java:

- Java Standard Édition (Java SE) conçue pour les ordinateurs de bureau ;
- Java Enterprise Edition (Java EE) destiné aux serveurs Web ;
- Java Micro Edition (Java ME), idéal pour les appareils portables comme les Smartphones.

La distribution Java EE a été utilisée dans le cadre de ce projet pour les raisons suivantes:

4.1.1.1 Langage Java

De par sa robustesse, la richesse de ses librairies et sa portabilité le choix du langage Java pour la réalisation de ce projet a été presque une évidence pour plusieurs raisons:

- Le Module SemText devant être intégré dans le projet SemWidgets est développé en Java. Donc, Il est judicieux de développer toute l'application en utilisant ce langage dans un souci de création d'une synergie permettant ainsi un gain en temps et en performance ;

- Vue l'importance de la communauté Java (1er langage utilisé par 9 millions de développeurs jusqu'à début 2012 selon l'index TIOBE[26] de juillet 2012) il est plus facile de trouver de la documentation et des solutions sur internet ;
- La mise à profit de l'expérience acquise en langage Java durant la formation HES ;
- Possibilité d'intégrer différents modules rendant l'application extensible et multiplateforme.

4.1.1.2 Entreprise Java Beans

Entreprise Java Beans ou "EJB" constituent la pièce maitresse d'une application Java EE. Ceux sont des composants côtés serveur qui encapsulent la logique métier (le code répondant à l'objectif de l'application) en exposant différents services. Ils permettent également de gérer facilement les transactions de la base de données et les autorisations de sécurité.[27]

L'utilisation des EJBs permet une séparation des couches qui rend l'exécution coté client plus rapide car toute la logique métier et les différentes règles de gestion se trouvent du côté serveur, allégeant de ce fait le client qui ne traite que la logique d'affichage.

L'utilisation des EJBs dans la réalisation de l'application SemWidgets la rend plus performante et extensible notamment pour un futur interfaçage avec une application mobile ou encore une application Client Java.

4.1.1.3 Java Server Faces

Java Server Faces ou JSF est un standard JEE, il s'agit d'un *framework*[28]pour la création d'application Web selon le design pattern MVC (Model, Vue, Contrôleur) respectant une rigoureuse séparation entre la couche Vue et les autres couches de l'application rendant la maintenance plus facile. Il facilite la conception des composants graphiques disponibles par défaut en JSP (Java Server Page) en gérant les évènements entre client et serveur. Ces qualités permettent ainsi de:

[26] Index TIOBE: http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html
[27] Documentation Oracle: http://docs.oracle.com/cd/E19798-01/821-1841/gipmb/index.html
[28] Un f*ramework* est un espace de travail modulaire. C'est un ensemble de bibliothèques et de conventions permettant le développement rapide d'applications. Il fournit suffisamment de briques logicielles et impose suffisamment de rigueur pour pouvoir produire une application aboutie et facile à maintenir. (http://www.techno-science.net/?onglet=glossaire&definition=1471)

- Déposez les composants sur une page en ajoutant des balises de composants ;
- Relier les composant-événements générés au code de l'application côté serveur ;
- Liaison des composants de l'interface utilisateur sur une page aux données côté serveur ;
- Construire une interface avec des composants réutilisables et extensibles ;
- Enregistrer et restaurer l'état d'interface utilisateur au-delà de la durée de vie des requêtes au serveur.

(source: http://docs.oracle.com/javaee/5/tutorial/doc/bnaph.html)

4.2 Outils et librairies

Dans cette partie sont décrits brièvement les différents outils ainsi que les principales librairies utilisés dans le développement de l'application SemWidgets.

4.2.1 Eclipse IDE

Eclipse est l'environnement de développement intégré (IDE[29]), libre et extensible, il est utilisé pour la construction des applications Java (avec différents plugins il supporte également d'autres langages comme C/C++, AspectJ, PHP COBOL).

Figure 12 - Logo Eclipse

Site web: http://www.eclipse.org/[30]
Lisence: EPL (Eclipse Public License)

4.2.2 MySQL

Il s'agit d'un serveur spécialisé dans la gestion des bases données relationnelles SQL. Il a été choisi pour ce projet utilisé pour sa performance et également parce qu'il est Open Source.

Figure 13 - Logo MySQL

Site web: http://www.mysql.com/[31]
Licence: Licence publique générale GNU

[29] IDE: *Integrated Development Environment*
[30] Source du logo: http://www.eclipse.org/
[31] Source du logo: http://www.mysql.com/

4.2.3 Hibernate

Hibernate est un *framework* qui gère la persistance des objets Java dans la base de données. Open source est parfaitement compatible avec MySQL, il permet le mapping objet / relationnel et facilite la persistance et la recherche des données dans une base données. Il a son propre langage de requête orienté objet, le Hibernate Query Language ou HQL qui lui permet d'être compatible avec différents types de bases de données, cependant cela n'empêche pas d'utiliser un autre langage comme JPQL[32] ou créer des requêtes natives SQL[33].

Figure 14 - Logo Hibernate

Site web: http://www.hibernate.org/
Licence: GNU LGPL

4.2.4 Maven

Maven est un outil logiciel libre pour la gestion et l'automatisation de production des projets logiciels Java en général et Java EE en particulier. L'objectif recherché est comparable au système Make sous Unix : produire un logiciel à partir de ses sources, en optimisant les tâches réalisées à cette fin et en garantissant le bon ordre de fabrication." [34]

Figure 15 - Logo Maven

Maven a été utilisé dans le cadre de ce projet pour gérer les dépendances des librairies ainsi que la construction du fichier EAR[35] (format du fichier utilisé par JEE) qui englobe les différents modules du projet pour le déployer sur un le serveur JBoss grâce à des fichiers POM (Project Object Model) qui contiennent les références des différentes librairies et modules externe et décrivent dans quel ordre les différents composant doivent être compilés et exécutés.

Site web: maven.apache.org[36]
Licence: Apache 2.0 licence
Version: 3.0.3

[32] *Java Persistence Query Language* (langage orienté objet)
[33] *Structured Query Language*
[34] http://fr.wikipedia.org/wiki/Apache_Maven (visité le 28.05.2012)
[35] EAR Enterprise Application ARchive
[36] Source Logo: maven.apache.org

4.2.5 JBoss Application Server

Serveur d'applications JEE, totalement écrit en Java il s'exécute sur n'importe quel system d'exploitation supportant le langage Java (fournissant une JVM).

Figure 16 - Logo JBoss

Il a été utilisé pour héberger l'application SemWidgets sous forme de site web. Afin de profiter de la technologie qu'offre la plateforme JEE[37] et plus particulièrement de Entreprise Java Beans (EJB) et JSF[38] il existe deux serveurs d'applications connus JBoss et Glassfish, JBoss a été choisi parce qu'il a été éprouvé lors d'autres projets précédents et a répondu parfaitement aux besoins des applications JEE.

Site web: http://www.jboss.org/[39]
Licence: GNU LGPL
Version: JBoss 6

4.2.6 ApacheTomcat

Appelé simplement Tomcat, est un conteneur de servlet Java et Java Server Pages. Il offre moins de possibilité qu'un serveur JBoss, il a été utilisé dans le cadre de ce projet pour héberger Sesam framework (décrit plus loin dans ce document) l'un des prérequis pour le fonctionnement du module SemText.

Figure 17 - Logo Tomcat

Site web: http://tomcat.apache.org/[40]
Licence: Apache License
Version: Tomcat 6

4.2.7 Streaming API for XML

StaX est une API[41] Java Open Source développée par Codehaus qui sert à analyser et à générer des fichiers XML.

Figure 18 - Logo Codehaus

[37] JEE: Java Entreprise Edition
[38] JSF: Java Server Faces
[39] *Source logo: http://www.jboss.org/*
[40] *Source logo: http://tomcat.apache.org/*
[41] *Application Programming Interface*

Elle a été utilisée dans la création du parseur XML en vue de l'extraction des métadonnées.

D'autres APIs ont été analysées comme SAX ou DOM. StAX a été retenue pour sa rapidité de traitement surtout dans le cas d'un flux entrant à partir d'un site web.

Site: http://stax.codehaus.org/[42]
License: business-friendly license[43]

[42] Source logo: http://stax.codehaus.org/
[43] Prend en charge l'utilisation complète d'un projet dans une entreprise commerciale

5- SemWidgets : Architecture

Cette section donne une vue globale sur l'architecture de l'application SemWidgets, elle décrit le rôle et les composants de chaque couche et comment ils interagissent avec le reste de l'application.

Afin simplifier l'explication des interactions entre les différents composants de l'application SemWidgets, les fonctionnalités les plus importantes de l'application sont décrites sous forme de scénarios.

5.1 Architecture

Avant de commencer à construire l'application SemWidgets, une réflexion sur l'architecture à adopter est primordiale.

En analysant les besoins du *Product Owner*, il en ressort que l'application doit intégrer plusieurs modules distincts qui doivent fonctionner d'une manière fluide tout en garantissant une haute performance de l'ensemble de l'application. Donc l'architecture doit prendre en compte l'importance d'une séparation stricte des couches qui assurera une certaine flexibilité et facilité de maintenance (factorisation du code) tout en supportant la montée en charge (traitement d'un grand volume de données; éventuelle mise en ligne).

Traitant d'un concept nouveau, l'évolutivité et l'extensibilité de l'application sont des qualités qui influenceront le choix de l'architecture car elles faciliteront l'intégration de nouveaux modules ou encore un future interfaçage avec une version mobile par exemple.

Afin de répondre aux besoins de l'application SemWidgets une architecture multicouches semblait la plus adéquate, le schéma suivant résume les principales couches de l'application:

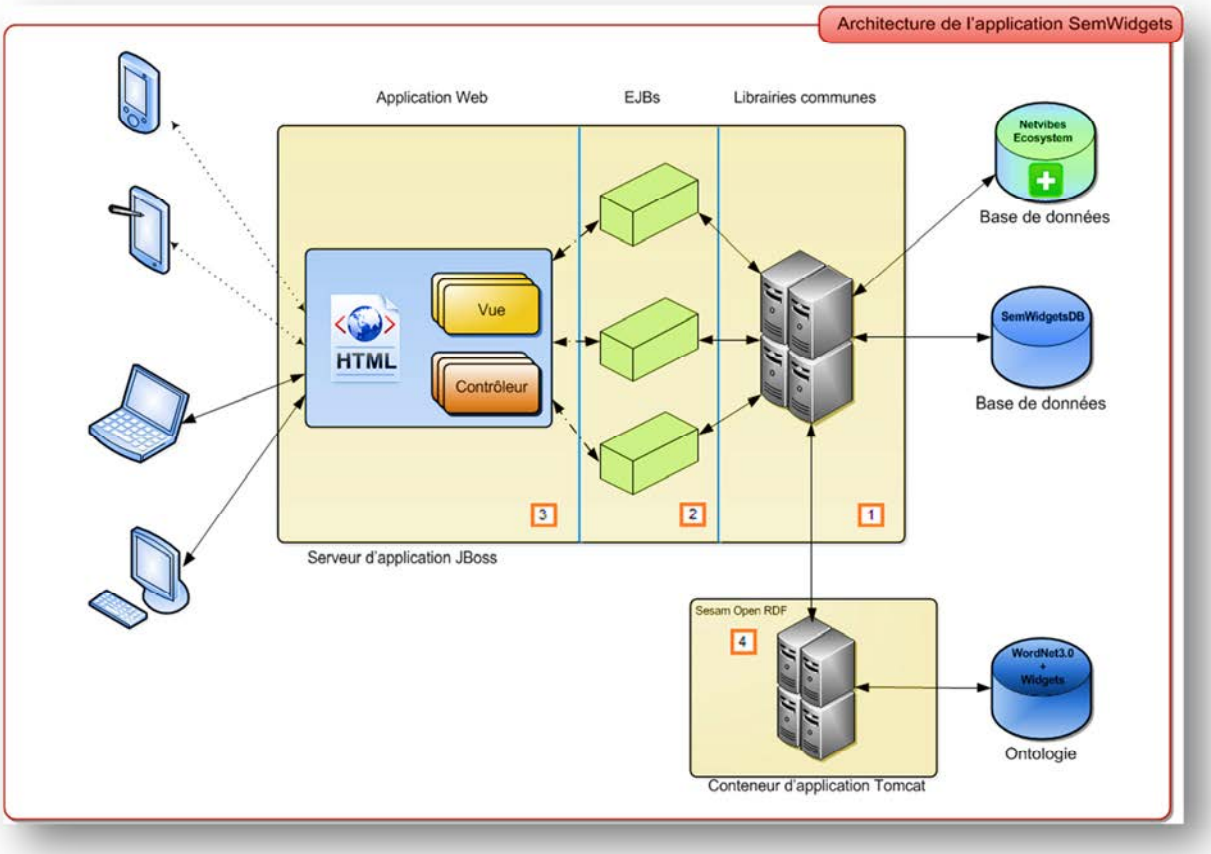


Figure 19 - Architecture de l'application SemWidgets

L'application est hébergée dans un serveur d'application JBoss qui constitue la pièce maitresse de cette architecture car il englobe les couches les plus importantes à savoir:

- **Couche métier** représente l'implémentation de la "logique", les modèles d'objets métiers (par exemple une Widget, une catégorie, un utilisateur) et les différentes règles qui régissent les cycles de vie de ces objets et les interactions qu'ils puissent avoir entre eux. Dans ce schéma cette couche est composée de librairies communes (n° 1) qui contiennent les différents modèles et la partie EJBs ou Entreprise Java Beans (n°2) qui exposent sous forme de services les différentes méthodes (fonctions) pour manipuler ces modèles (objets métier). Ces EJBs permettent également de gérer les transactions avec la base de données ainsi que les règles de sécurité (permettre l'accès à une ressource en fonction du rôle de l'utilisateur).

- **Couche d'accès aux données** est chargée de l'interfaçage des objets métiers avec les données "physiques" qui entrent ou sortent de l'application et provenant de la base de données ou d'un autre système externe. SemWidgets dispose de trois sources de données distinctes:
  - Base de données de l'application, il s'agit de SemWidgetsDB (MySQL)
  - Le serveur Sesam Open RDF qui fournit les données provenant de l'ontologie.
  - Base de données Netvibes Ecosystem qui contient les Widgets exposées par l'annuaire Netvibes.

  La couche d'accès aux données permet de s'affranchir du type de la source de données, ce niveau d'abstraction répond au principe de la modularité voulu pour ce genre d'application. En standard il est assuré par la couche JPA (Java Persistence API) mais il est possible d'implémenter des framework qui facilitent cet accès aux données d'une manière transparente pour l'application. Tel est le cas pour Hibernate, le framework utilisé dans ce projet et décrit précédemment dans la section outils et librairies.

- **Couche application** joue le rôle de médiateur entre la couche métier et la couche de présentation (Vue), dans le respect d'un pattern MVC chaque vue dispose d'un contrôleur qui gère les requêtes utilisateur, les redirections et l'appel des EJBs pour l'interaction avec le modèle.
- **Couche de présentation** représente la partie visible de l'application, elle est formée des différentes pages Web avec lesquelles l'utilisateur interagit avec l'application.

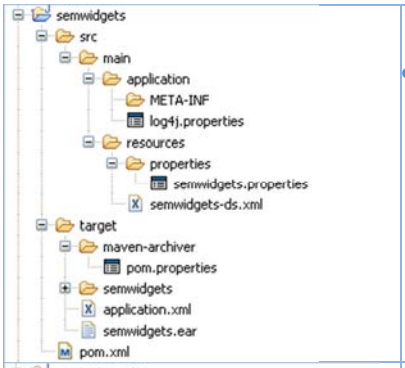
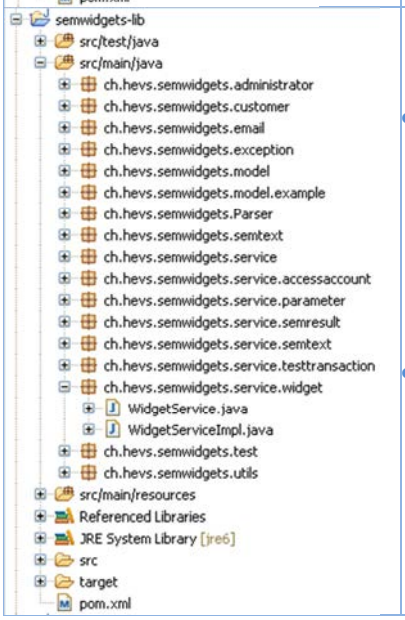
5.2 Structure des fichiers

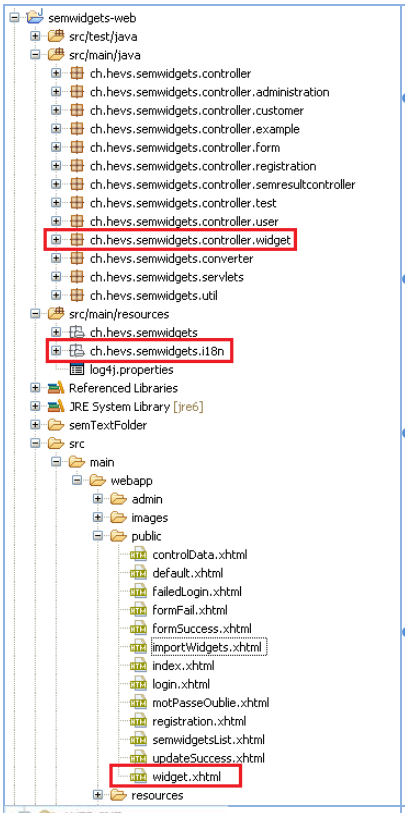
Ce paragraphe explique la structure des fichiers dans l'application

L'application se divise en trois projets Java

Tous les projets contiennent un fichier de configuration appelé POM (Project Object Model), ce fichier XML indique à Maven (décrit dans la section outils et librairies de ce document) les dépendances aux librairies qu'il doit charger, et l'ordre dans lequel les projets doivent être compilés.

Tableau 2 – Structure des fichiers dans l'environnement de développement Eclipse

	• Le projet "SemWidgets" contient les fichiers de configuration contenant les paramètres globaux de l'application.
	• Le projet "SemWidgets-lib" contient tous les objets métier dans les package ainsi que les différent EJB qui expose les différentes méthodes pour les manipuler sous forme de service • SemText a été intégré comme un module et un service pour exposer ses fonctionnalités. Il a été mis en place pour être interrogé par le contrôleur.



- Le projet "SemWidgets-web" contient l'application Web (Contrôleur + Vue)
- Pour chaque vue il y a un contrôleur qui interroge l'objet métier via l'EJB
- Le répertoire ressources contient les fichiers de traduction des différentes pages web
- Le répertoire WebApp contient toutes les pages web ainsi que les feuilles de style

- Toujours dans le projet "SemWidgets-web", le répertoire "WEB-INF" contient deux fichiers très importants pour le fonctionnement de l'application:
  - Faces-config.xml: contient les règles de navigation ainsi que différent paramètre de langue
  - Web.xml: Contient les caractéristiques et paramètres de l'application

6- Processus Global

Ce volet est consacré à l'explication globale de l'application SemWidgets et de l'extraction des métadonnées des Widgets à l'utilisation de la fonctionnalité de recherche.

L'élaboration de ce processus constitue une partie majeure de ce travail de Bachelor,

6.1 Extraction

Cette section traite de l'extraction des Widgets utilisés dans l'application et plus particulièrement des métadonnées employées dans le référencement des Widgets dans l'ontologie.

Tout d'abord des précisions sur les métadonnées à importer suivies des explications des différentes solutions proposées ainsi que les motivations du choix de l'une d'elle avant de conclure avec le résultat.

6.1.1 Métadonnées

Bien que les standards ne sont pas systématiquement respectés, la plupart des Widgets disponibles sur différentes plateformes comme Google Gadgets, Netvibes, Widgipedia sont accompagnés d'informations décrivant chaque Widget, comme le titre, la catégorie ou encore la description.

Ces métadonnées sont utiles pour le référencement et la classification par thème des Widgets qui seront mis à disposition dans l'application SemWidgets.

Pour ce faire, il est nécessaire de:

- Trouver des Widgets en grande quantité.
- Automatiser l'extraction des métadonnées indispensables au référencement des Widgets à la fois dans l'ontologie et dans la base de données relationnelle.

Les métadonnées suivantes doivent être extraites pour chaque Widget:

Tableau 3 - Métadonnées et support de stockage

Métadonnée	Stockage
Titre	Ontologie + Base de données
Catégorie	Ontologie + Base de données
Description	Ontologie + Base de données
URL	Base de données
Lien vers l'aperçu (Thumbnail)	Base de données
Nombre d'installations (facultatif)	Base de données

6.1.2 Méthodes

Pour trouver les Widgets à importer en grande quantité, plusieurs annuaires ont été consultés, les plus connus sont retenus pour une étude plus minutieuse. Il s'agit de Google Gadgets, Netvibes et WidgetBox.

Plusieurs possibilités ont été examinées afin d'extraire les métadonnées des Widgets à partir des différents annuaires sélectionnés :

1. création d'un parseur HTML pour chaque fournisseur de Widgets ;
2. recherche des API's pour accéder aux métadonnées ;
3. création d'un parseur XML pour l'extraction.

6.1.2.1 Parseur HTML

En faisant une simple recherche sur Google Gadgets, les résultats affichés contiennent certaines informations parmi celles recherchées comme le titre et la description (Fig.20).

Dans le code source de la page (Fig. 21), les métadonnées apparaissent entourées de balises html. Ces Tags peuvent servir à identifier et cibler les éléments à importer.

Une première idée consistait à développer un parseur HTML qui va:

1. Se connecter à la page principale de l'annuaire Google Gadgets.
2. Exécuter une requête avec le mot clé recherché.
3. Parser la page des résultats et extraire les métadonnées en se basant sur les tags pour différencier les attributs des Widgets à importer.

Dans l'exemple suivant une simple recherche de Widgets pour le thème "citations":

Figure 20. Recherche sur Google Gadgets

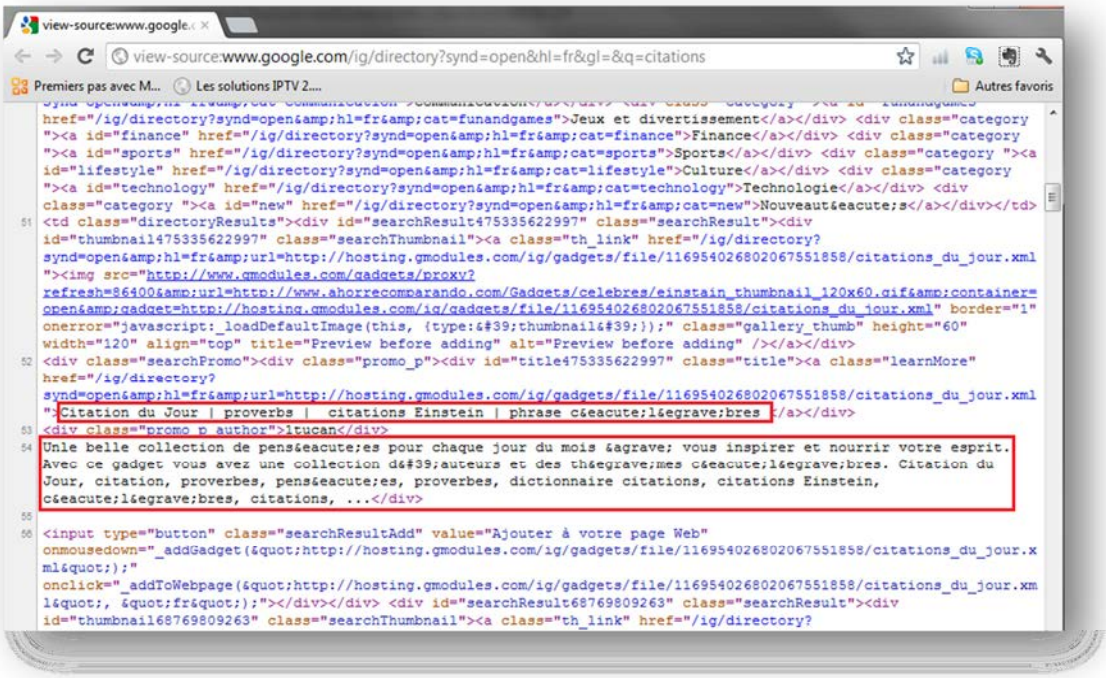


Figure 21 - Google Gadgets - Code source des résultats de la recherche

Toutes les métadonnées recherchées ne figurent pas systématiquement sur la page des résultats, et sont souvent stockées dans des fichiers externes qui n'ont pas tous la même structure. Ce qui complique le processus d'extraction automatisée.

Le schéma BPM (*Business Process Management*) suivant résume le processus d'extraction des métadonnées à partir de Google Gadgets en interrogeant la page dédiée à chaque gadget.

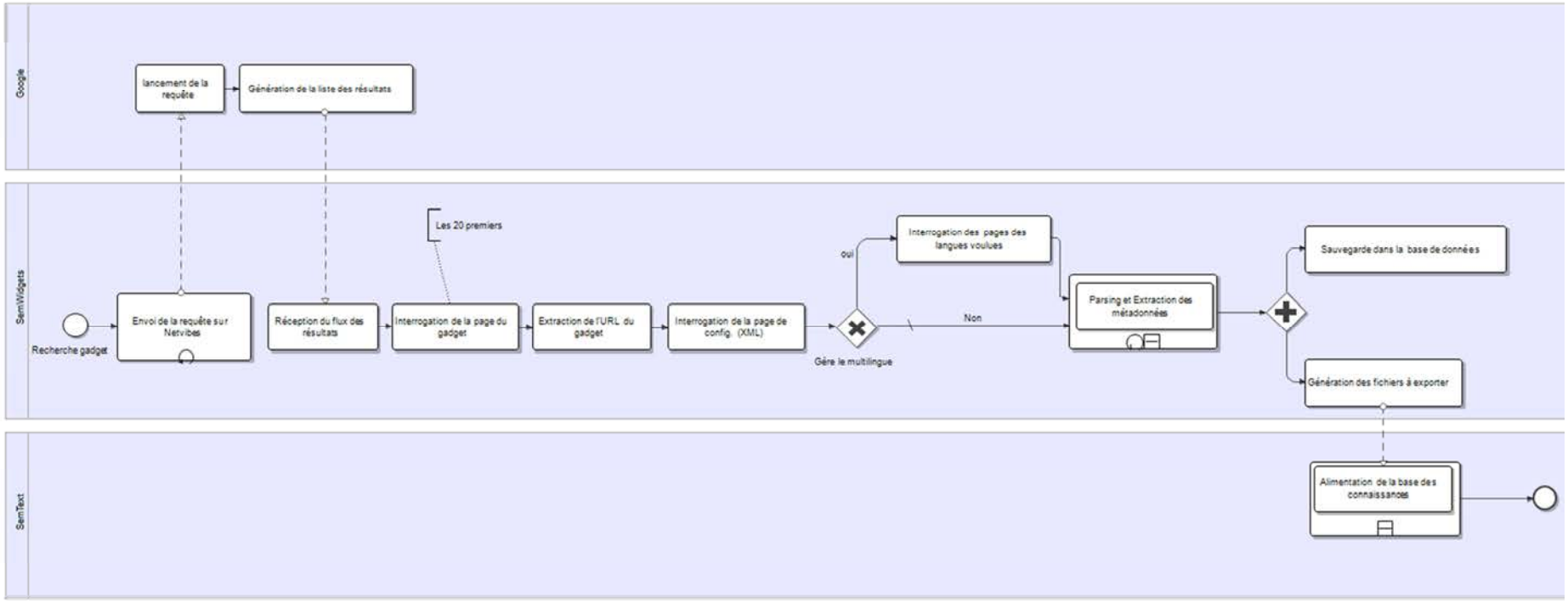


Figure 22 - Schéma BPM: Extraction des Widgets à partir de Google Gadget

Pour les autres annuaires comme Netvibes ou WidgetBox, la tâche s'avère plus simple car toutes les métadonnées recherchées apparaissent dans le code source de la page des résultats de la recherche. En plus, les tags utilisés sont facilement identifiables (Fig. 23). Cela signifie que l'automatisation de leur extraction sera plus simple. Ce qui n'est pas le cas pour Google Gadgets.

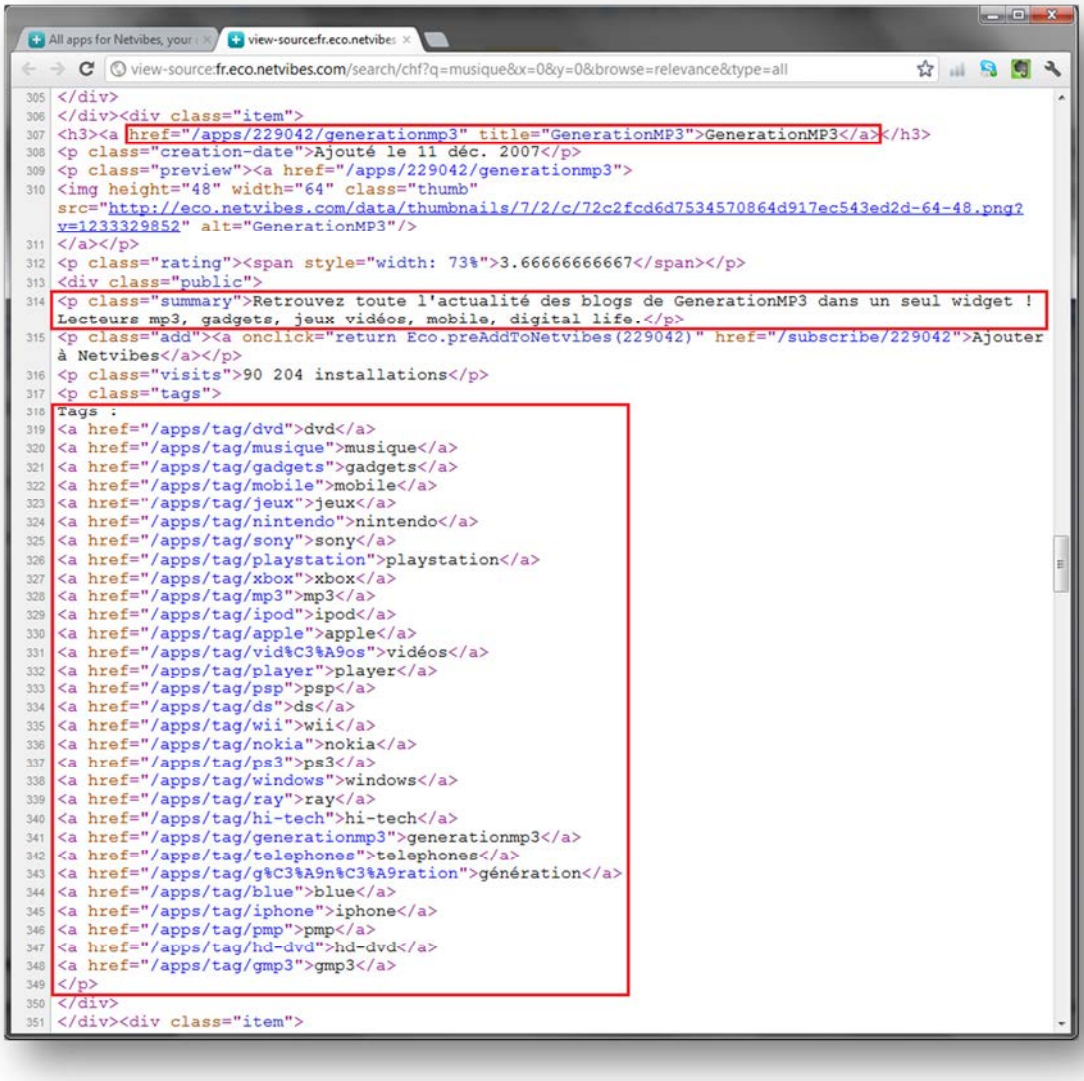


Figure 23. Code source d'une page Netvibes

Le schéma BPM suivant résume le processus d'extraction des Widgets à partir de Netvibes (le principe est le même pour WidgetBox)

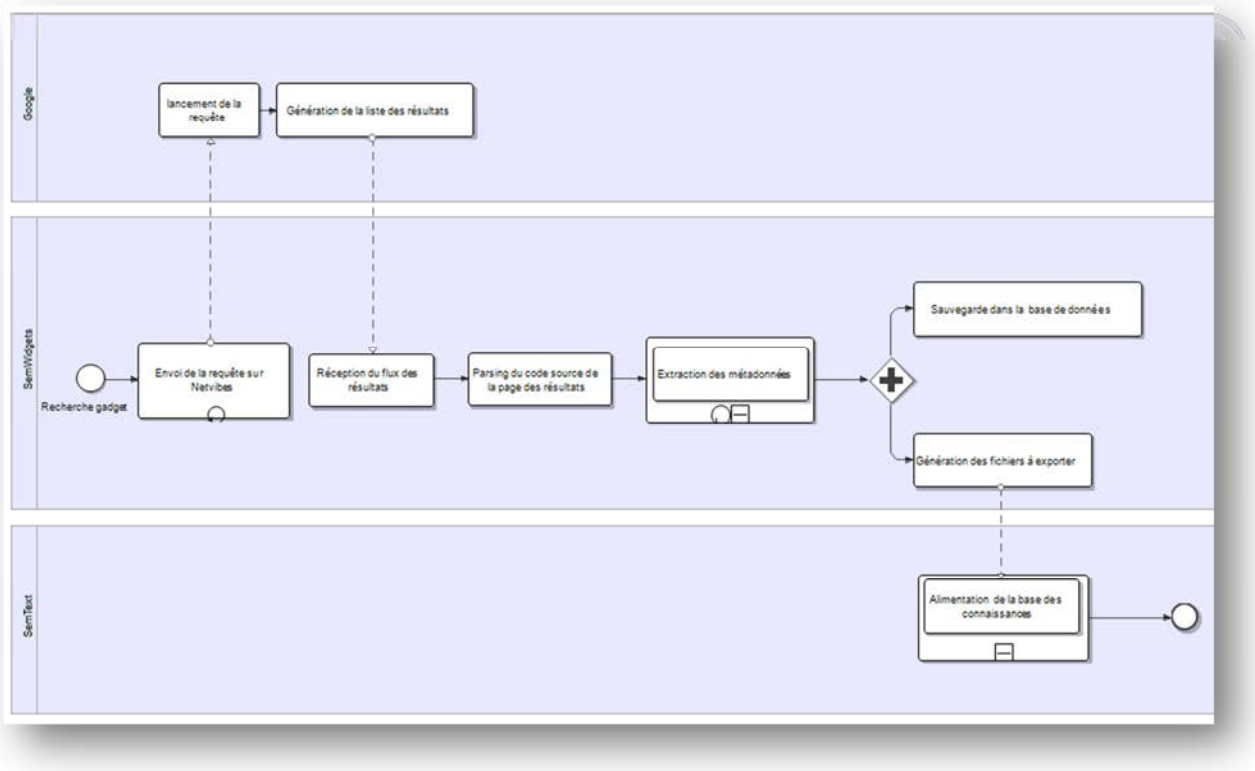


Figure 24- Schéma BPM: Extraction des Widgets à partir de Netvibes

6.1.2.2 Netvibes Ecosystem API

Netvibes a ouvert sa plateforme aux développeurs pour les faire profiter de sa technologie et son riche contenu. Il a mis à disposition des développeurs l'interface *Universal Widget API* [44] (UWA), qui est une interface de programmation permettant de développer des Widgets intégrables dans multiples plateformes comme iGoogle, Blogger, Opera, Desktop Windows et bien d'autres.

Une deuxième API a été mise en ligne spécialement pour permettre la recherche et la récupération de données publiques du site, il s'agit de la Netvibes Ecosystem API. Cette API permet d'interroger le service qui expose l'ensemble des Widgets du site. Le résultat est retourné sous forme de flux XML.

44 http://dev.netvibes.com/

Un parseur XML permet d'extraire les métadonnées à partir du flux XML reçu en réponse à une simple requête.

Le tableau suivant liste les différents paramètres utilisables dans une requête.

Tableau 4 - Paramètre d'une requête Ecosystem Netvibes

Paramètre	Valeur par défaut	Description
query		La requête dans une chaine de caractères (équivalente à une recherche sur le site Ecosystem Netvibes)
page	1	Numéro de la page
limit	20	Le nombre maximum d'éléments retournés par la requête
thumbwidth	160	Largeur des vignettes
thumbheight	120	Hauteur des vignettes
sort	*popular*	Critère de tri (*popular; recent*)
category	0	Id de la catégorie
region	*all*	The region code
type	*all*	Type des element retourné
format	xml	xml ou json

(source: documentation de l'API sur http://dev.netvibes.com/doc/api/eco/search)

Voici un exemple de requête, elle retourne comme résultat deux Widgets qui ont pour thème "voyage": **http://api.eco.netvibes.com/search/?query=voyage&limit=2**

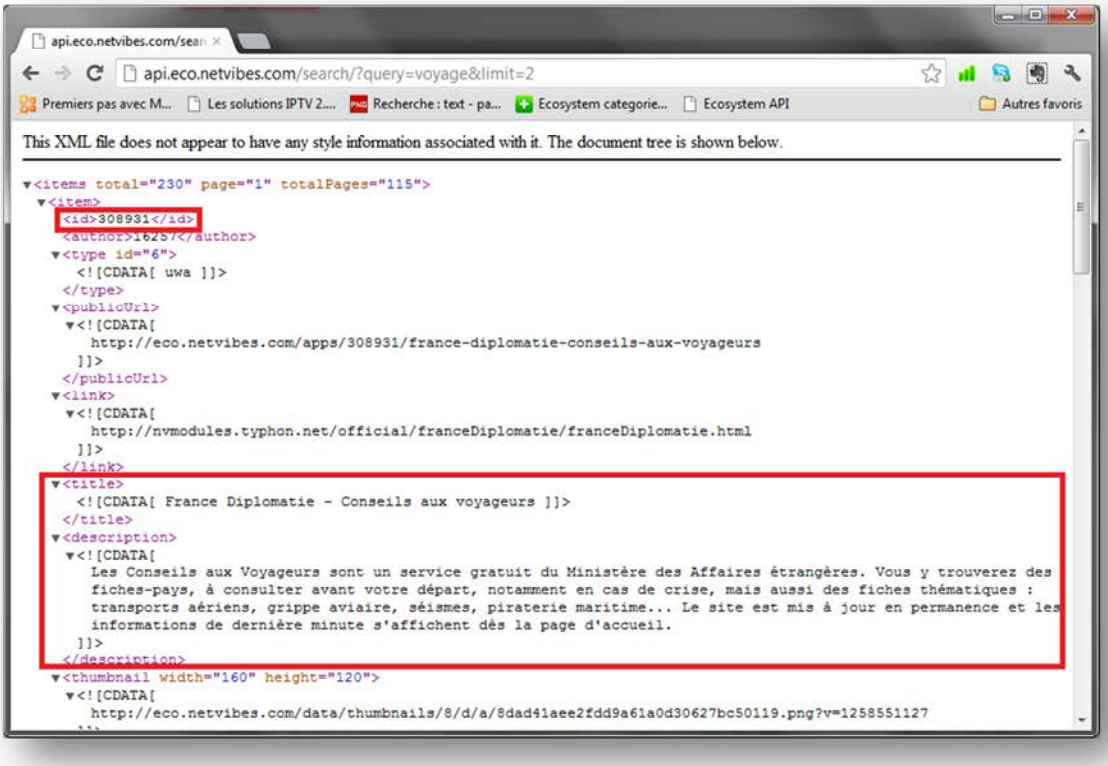


Figure 25 - Résultat de la requête Netvibes Ecosystem

6.1.3 Résultats

6.1.3.1 Parseur HTML

Un parseur HTML a été développé pour l'extraction des métadonnées. Cependant cette méthode a quelques inconvénients:

- Le développement est lié à la structure de la page HTML traitée. Cette structure changeant fréquemment empêche une réutilisation ultérieure ;
- La nécessité d'écriture d'un parseur spécifique pour chaque site;
- Posés par des utilisateurs et sans contrôle rigoureux des packages uploadés, les Google Gadgets ne contiennent pas toujours les métadonnées recherchées, notamment pour les descriptions des Widgets ;
- Une gestion supplémentaire s'impose afin de garantir l'unicité et la cohérence des IDs des Widgets.

6.1.3.2 Netvibes Ecosystem API

Quoique découverte après la réalisation du parseur HTML, l'API Netvibes Ecosystem a été retenue comme solution à intégrer dans l'application SemWidgets pour les raisons suivantes:

- Disponibilité de toutes les métadonnées recherchées ;
- Structure XML stable garantissant la réutilisabilité du parseur ;
- Possibilité d'ajout des Widgets extraites dans différentes plateformes ;
- Dispense de gestion des IDs, leur unicité étant garantie par Netvibes Ecosystem ;
- Facilité d'intégration dans une application Java.

6.2 Stockage

Le parseur XML permet d'analyser le flux XML retourné en réponse à la requête Netvibes. En lisant les données reçues, les opérations suivantes sont exécutées pour chaque Widget:

- Un objet `widget` est créé ;
- Un fichier TXT est généré, il a comme titre l'ID l'objet Widget et il contient à la fois son titre, sa catégorie et sa description ;
- L'objet est persisté dans la base. (Fig.26)
- Les fichiers textes sont ensuite exportés vers l'ontologie en utilisant la méthode `semGraph` de SemText. (Fig. 27)

```
for (int i = 0; i <= nbrOfPages; i++) {
// Read XML stream and store results in a list
   List<Widget> readWidgets = read.readConfig(keywords,widgetsPerPage, i);
   for (Widget widget : readWidgets) {
     try {
          //Create text file
           SimpleFileWrite  file  =  new      SimpleFileWrite(widget.getId().toString(),
                                    directoryPathForExport,widget.getContent());
          //Persist the widget object in database
          getWidgetService().saveWidget(widget);

     } catch (DataException e) {
          System.out.println("!!!Widget "+ widget.getId() + " not saved!!!");
          e.printStackTrace();
     }
   count++;
   }
}
```

Figure 26. Persistance des objets Widgets dans la base et création des fichiers textes

```
semText.semGraph(pathInTexts, null,
                  pathUserDescription + "\\en\\", Lang.lngEn,
```

```
                    SemGraph.textURIPrefix_defaultValue,
                    SemGraph.toSingleFile_YES,
                  SemGraph.loadToStore_YES,
                  "file://userDescriptions_en",
                    SemGraph.oneContextPerFile_NO,
                    SemGraph.addDBKeyFromFileName_YES, "tt_userDesc");
```

Figure 27. Importer les métadonnées des Widgets dans l'ontologie

6.3 Recherche

À cette étape du processus, la base de données contient les métadonnées des Widgets (titre, URL, description, catégories, etc.) qui sont dès lors référencés dans l'ontologie.

Donc, il est possible d'effectuer des recherches par mots clés.

À noter qu'actuellement le module SemText ne supporte qu'une recherche de mots-clés séparés par des virgules.

6.3.1 Méthodes

Quand une recherche est lancée, une instance SemText est créée et permet de:

- Établir une connexion avec le *repository* (référentiel) OntoManag qui contient les Ontologies WordNet 3.0 et SemWidgets par le biais des pipelines *Gate*[45] ;
- Générer les requêtes SPARQL permettant la recherche des résultats dans l'ontologie ;
- Retourner les résultats de la requête sous forme de chaines de caractères (*text, score, dbKey*). Le tableau 5 explique la signification de chacune de ces chaines de caractères.
- Créer un service pour traiter les résultats retournés par SemText. Ce service permet de créer des objets `SemResult` en utilisant les chaines de caractères reçues de SemText. Un objet `SemResult` a un attribut `idWidget_fk` qui contient l'identifiant de la Widget trouvée par la requête.
  Au niveau relationnel cela équivaut à la clé étrangère qui permettra de trouver le Widget dans la table "tbl_widget" pour ensuite soit l'afficher dans la page des

[45] L'outil *Gate* est décrit dans la section Installation et configuration de ce document

résultats ou alors afficher la page des détails, au cas où ce Widget est sélectionné par l'utilisateur.

Tableau 5 - Les résultats d'une requête SPARQL et leurs significations

Chaine de caractères	Signification
Text	*Uniform Ressource Identifier* (URI) du Widget trouvé.
dbKey	L'identifiant du Widget trouvé.
Score	Utile pour la recherche avec plusieurs mots. Si, par exemple, la recherche contient deux mots, et que la requête renvoi les deux mots dans les résultats, dans ce cas, le score sera de 2. (Le score sera de 2 si les 2 mots sont trouvés 1 pour un mot et 0 si aucun mot n'est trouvé)

6.3.2 Résultats

Le module SemText a été implémenté dans l'application SemWidgets, des services ont été créés pour exposer ses fonctionnalités et gérer les résultats retournés.

La requête standard générée par SemText ne permettait de sortir que les Widgets qui contiennent dans leur titre, catégorie ou description le ou les mots-clés recherchés.

```
semText.semQuery("radio", unknownKeyWords1, true,
                    textTypesResultHandlers1, Lang.lngEn);
```

Figure 28 – Exemple d'appel de la méthode SemQuery pour générer une requête SPARQL avec le mot "radio"

La requête générée est la suivante:

```
PREFIX nlp:<http://www.websemantiquech/onto/ontoManageNLP#>
PREFIX xsd:<http://www.w3.org/2001/XMLSchema#>
SELECT ?text ?dbKey (sum(?point) as ?score)
WHERE {{select distinct ?text ?point dbKey{
                          ?text nlp:hasDBKey ?dbKey.
                          ?text nlp:hasTextType nlp:tt_userDesc.
                          ?text nlp:hasLookupURI ?lookup0.
FILTER(?lookup0 IN (<http://purl.org/vocabularies/rinceton/wn30/wordsense-radio-
noun-1>,
 <http://purl.org/vocabularies/princeton/wn30/wordsense-radio-noun-3>,
<http://purl.org/vocabularies/princeton/wn3/wordsense-radio-noun-2>
)).BIND("1"^^xsd:integer as ?point).}}}
group by ?text ?dbKey order by desc(?score) ?text
```

Les résultats extraits dans la liste ne facilitaient pas la tâche pour l'utilisateur qui doit visionner toute la liste pour trouver ce qu'il cherche. Si le mot clé recherché est le même que celui d'une catégorie, tous les Widgets de cette catégorie sont affichés sans aucun filtre (popularité ou intérêt).

Un système de *ranking* a donc été mis en place pour trier les résultats et afficher les plus pertinents en haut de la liste. Il est basé sur un système de points attribués selon le nombre d'occurrences du ou des mots recherchés:

- 100 points si le mot est dans **le titre** * facteur d'importance ;
- 10 points pour chaque occurrence du mot dans **la description** * facteur d'importance ;
- 1 point si le mot clé est dans **la catégorie** * facteur d'importance.

Le facteur d'importance est relatif à la position du mot. À titre d'exemple, si l'utilisateur saisie les mots-clés suivant : "*trip, hotel, weather*" [46] le facteur d'importance sera de 3 pour le mot "*trip*", 2 pour le mot "*hotel*" et de 1 pour le mot "*weather*".

Les résultats sont ensuite triés par rapport au nombre de points, et pour départager les résultats ayant le même nombre de points ils sont triés par rapport au nombre d'installations. Les plus populaires seront mieux placés dans la liste.

[46] Les mots sont en anglais parce que les fichiers WordNet3.0 chargés dans l'ontologie sont en anglais

Une amélioration apportée à la requête SPARQL standard de SemText a permis d'améliorer la pertinence des résultats retournés en faisant recours à une recherche par synonyme à travers la requête suivante:

```
PREFIX nlp:<http://www.websemantique.ch/onto/ontoManageNLP#>
PREFIX xsd:<http://www.w3.org/2001/XMLSchema#>
PREFIX wn30:<http://purl.org/vocabularies/princeton/wn30/>
PREFIX wn20schema:<http://www.w3.org/2006/03/wn/wn20/schema/>

SELECT ?text ?dbKey (sum(?point) as ?score)

WHERE {{select distinct ?text ?point ?dbKey{?text nlp:hasDBKey ?dbKey.
             ?text nlp:hasTextType nlp:tt_userDesc.
             ?text nlp:hasLookupURI ?lookupText0.
             ?synset0 wn20schema:containsWordSense ?lookupText0 , ?lookupQuery0.
FILTER(?lookupQuery0 IN (<http://purl.org/vocabularies/princeton/wn30/wordsense-
Song-noun-6>,
<http://purl.org/vocabularies/princeton/wn30/wordsense-song-noun-5>,
<http://purl.org/vocabularies/princeton/wn30/wordsense-song-noun-4>,
<http://purl.org/vocabularies/princeton/wn30/wordsense-song-noun-3>,
<http://purl.org/vocabularies/princeton/wn30/wordsense-song-noun-2>,
<http://purl.org/vocabularies/princeton/wn30/wordsense-song-noun-1>)).
BIND("1"^^xsd:integer as ?point).}}} group by ?text ?dbKey order by desc(?score)
?text
```

6.4 Scénarios

Afin d'illustrer les résultats d'extraction et de recherche précédemment expliqués, cette partie vient les décrire sous forme de scénarios d'utilisation de l'application SemWidgets. Deux scénarios sont décrits: Import et Recherche.

L'URL pour lancer l'application: **http://localhost:8080/semwidgets-web/public/default.xhtml**

Dans le browser Google Chrome de la machine virtuelle l'application est accessible via la barre des favoris.

Figure 29 - Lancement de l'application à partir Google Chrome

6.4.1 Import

La fonctionnalité de l'import n'est autorisée que pour l'administrateur. Elle est accessible à partir du menu latéral de l'application ou encore via l'adresse URL: **http://localhost:8080/semwidgets-web/public/importWidgets.xhtml**

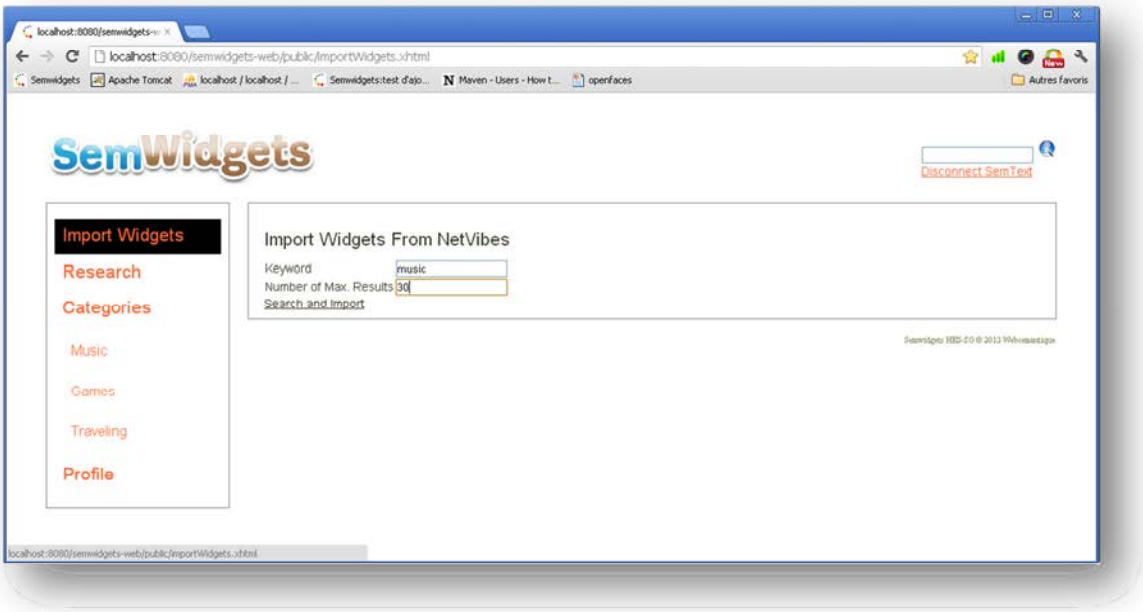


Figure 30 - Page Web pour importer des Widgets

En cliquant sur "*Import Widgets*" l'application se connectera à l'annuaire Netvibes et importera les Widgets correspondants aux mots-clés recherchés par l'utilisateur.

Sur console on peut voir le déroulement de l'opération :

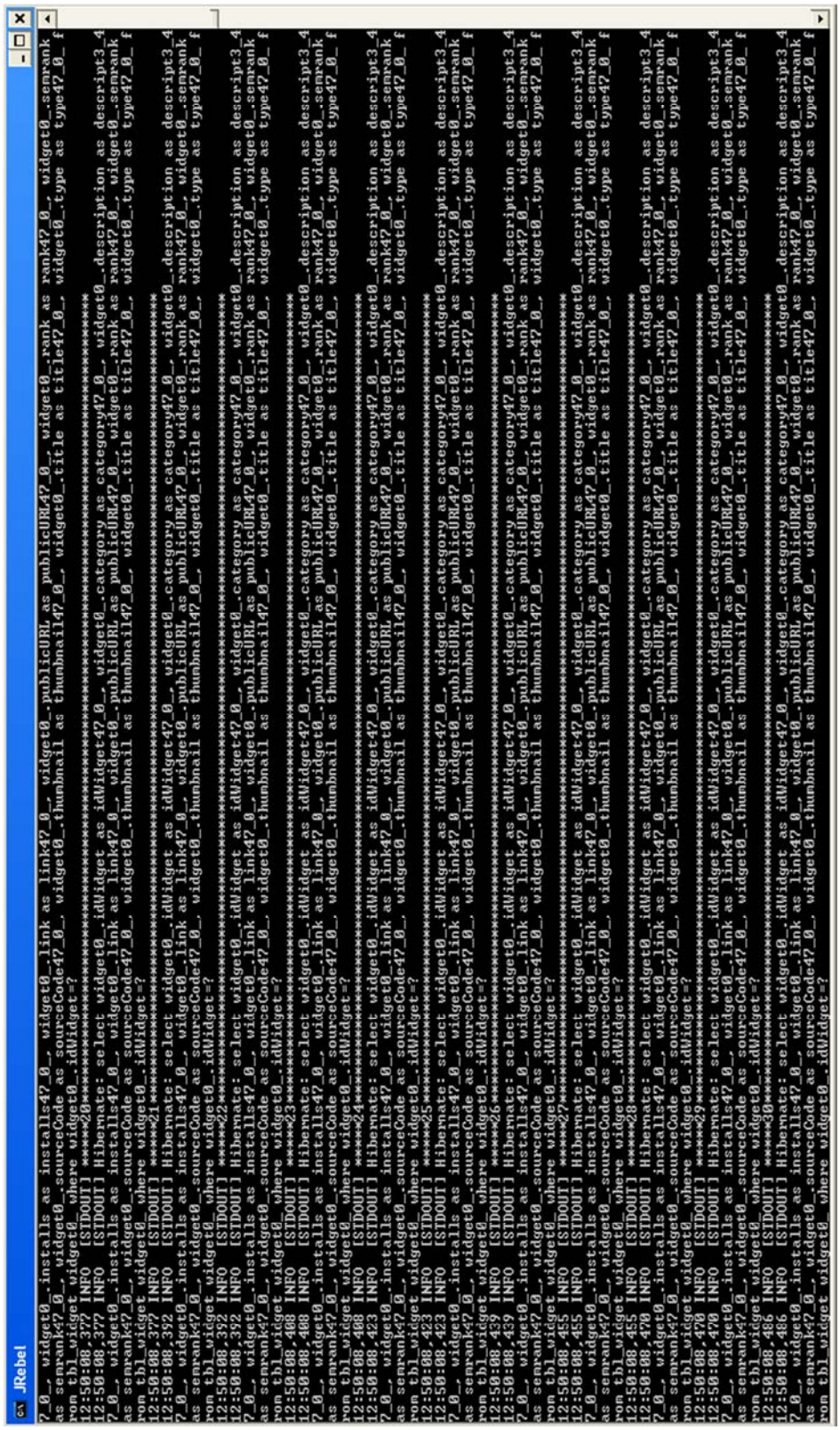


Figure 31 - Déroulement de l'import (sur console)

La figure suivante montre les objets Widgets persistés dans la base de données:

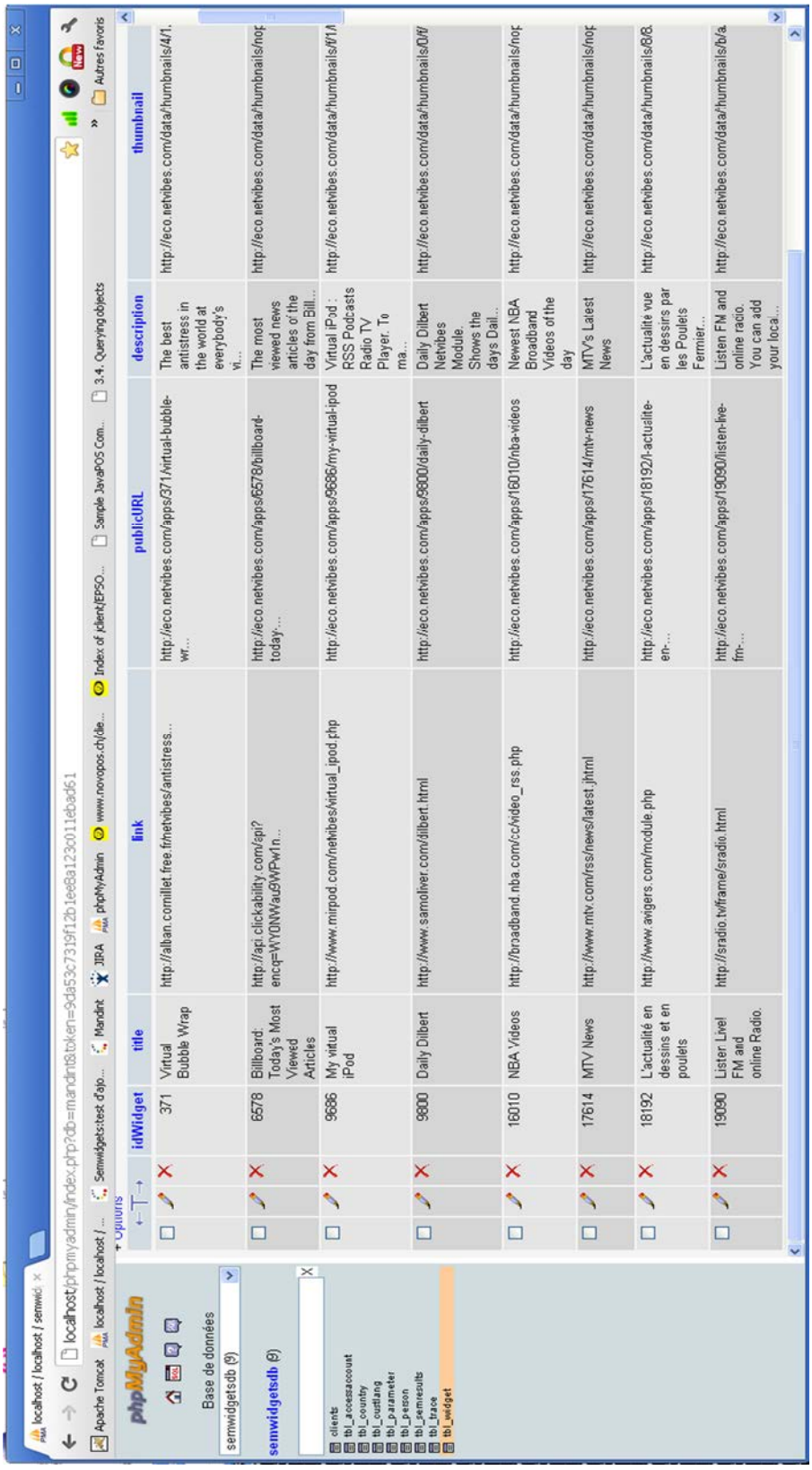


Figure 32 - Widgets enregistrés dans la base de données

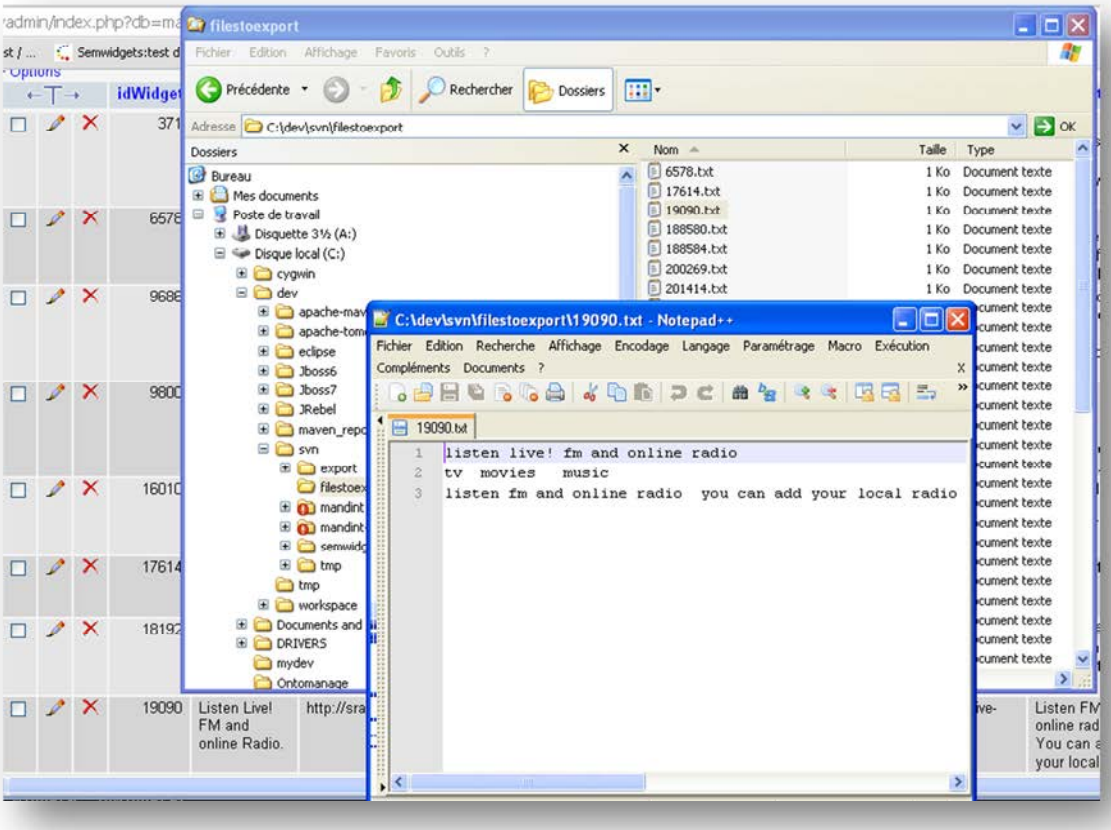


Figure 33 - Exemple des fichiers importés dans l'ontologie

6.4.3 Recherche

L'utilisateur peut effectuer une recherche à partir de la page "*Research*" ou en tapant directement les mots-clés dans la zone de recherche accessible à partir de toutes les pages web de l'application.

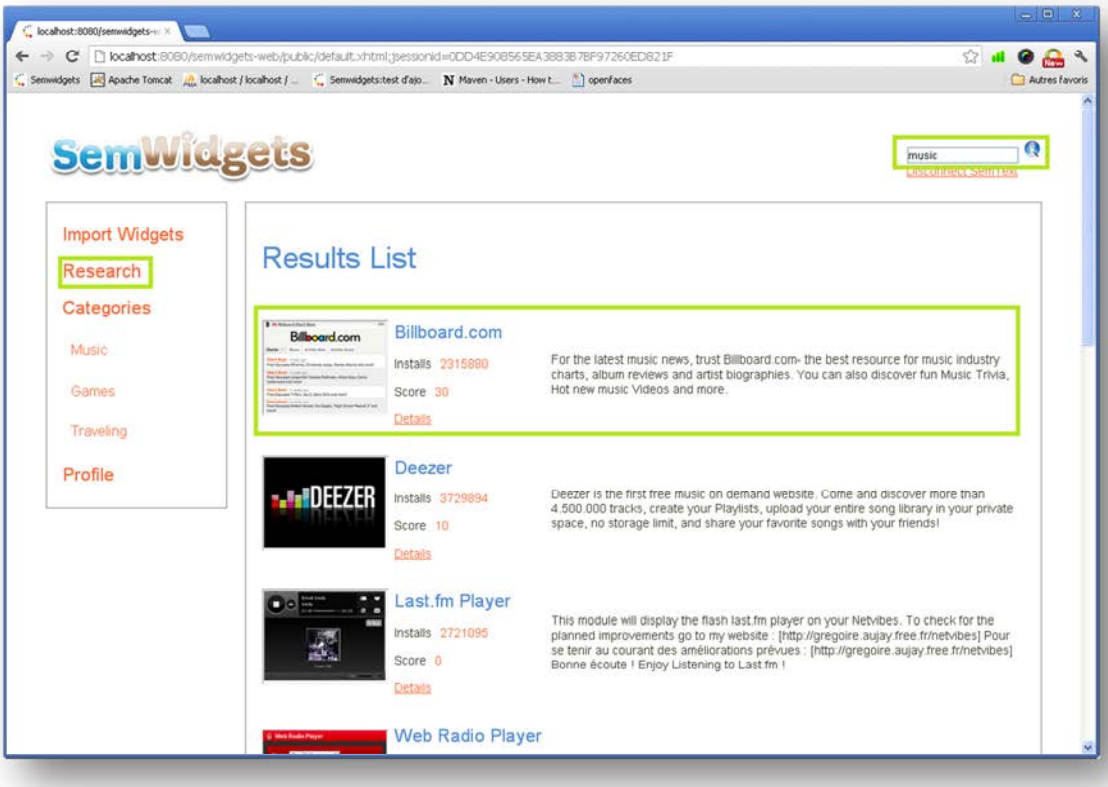


Figure 34 - Liste des résultats

Les Widgets correspondants à la recherche s'affichent dans une liste. Pour chaque Widget une description est affichée. Pour aiguiller l'utilisateur dans son choix des indices sont proposés :

- le nombre d'installations définissant la popularité du Widget ;
- le score calculé en fonction de la fréquence du ou des mots-clés dans le contenu des Widgets de la liste des résultats.

Pour avoir à la fois plus de détails et un aperçu du Widget, il suffit de cliquer sur "*Details*".

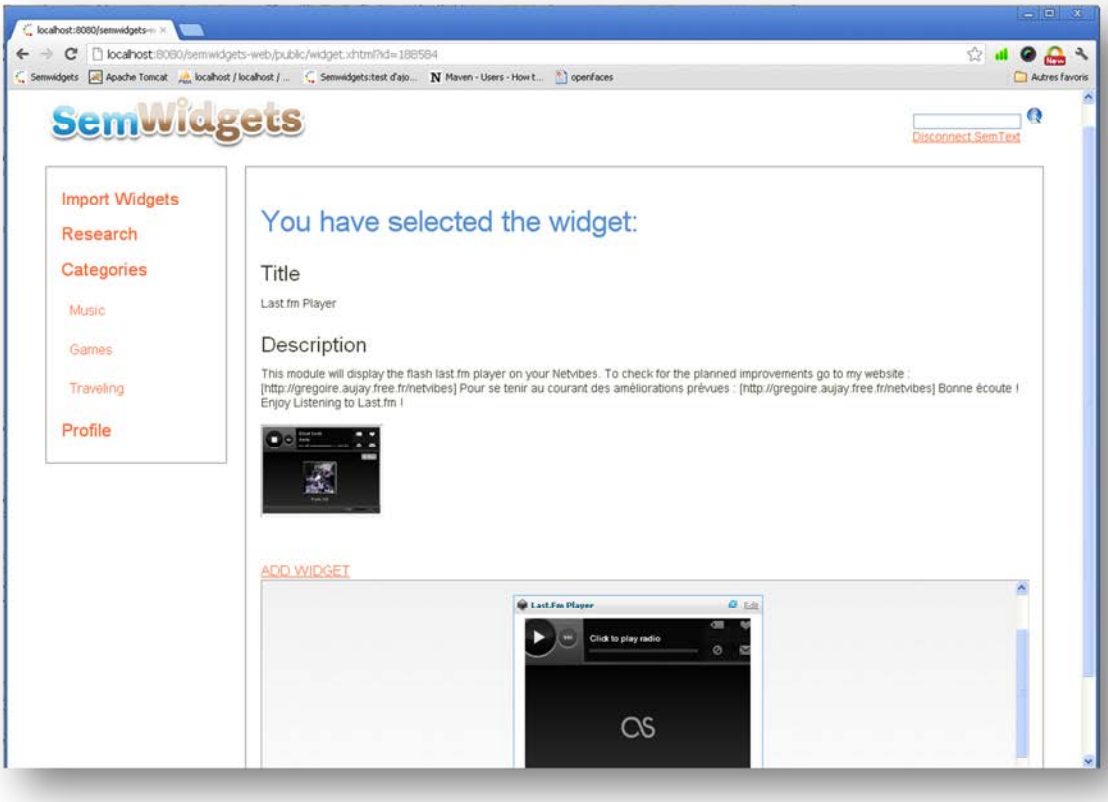


Figure 35 - Détails d'une Widget

Si le Widget plait à l'utilisateur il peut l'ajouter sur l'interface de son choix en cliquant sur "*Add Widget*".

Les interfaces supportées sont: Netvibes, iGoogle, Opera, Windows Desktop, Chome Extention, Blogger, MacOS Dashboard, Open Social.

Pour ajouter le Widget dans un site Web il faudra alors rajouter le script.

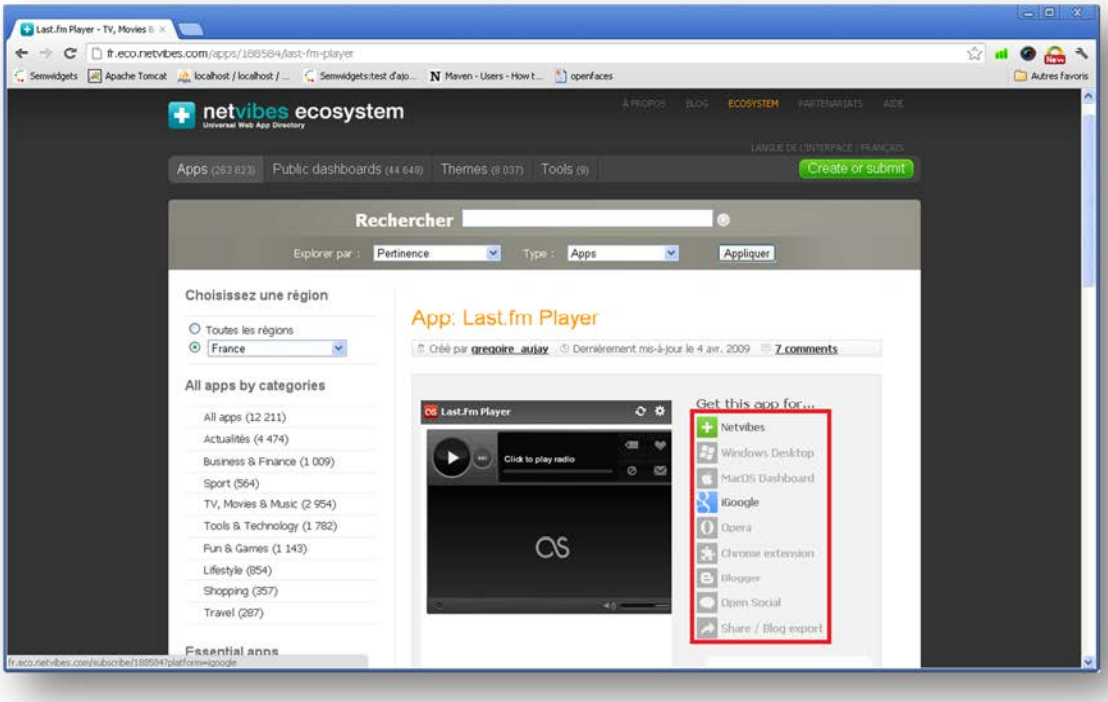


Figure 36 - Exporter le Widget sur une autre plateforme

7- Installation et configuration

Cette section décrit globalement les prérequis pour l'installation de l'application SemWidgets.

À titre indicatif, l'application SemWidgets et l'environnement de développement sont d'ores et déjà installés sur la machine virtuelle :

"**..\VM\semWidgets_JBoss\Windows_XP_Professional.vmx**" livrée avec ce travail.

7.1 Base de données

La base de données est une base MySQL. Le dump de la base est en annexe: **semwidgetsdb_dump.sql**.

7.2 JBoss

Le serveur JBoss est installé également sur la machine virtuelle. Si l'application SemWidgets doit être déployée sur un autre serveur, la version JBoss 6.0.1 Final doit y être installé et configuré.

Adresse téléchargement: http://www.jboss.org/jbossas/downloads/
Version: 6.1.0.Final

7.3 SemText

Ici sont énumérés les prérequis nécessaires au fonctionnement du module SemText embarqué dans l'application SemWidgets.

- Les fichiers d'installation de SemText sont sur le CD livré avec ce rapport, dans le répertoire :
  "**..\ Yacine Aghzaf Travail de Bachelor 2012\Code Source\Module SemText\Installation**"
- Les répertoires se trouvant dans :
  "**..\Yacine Aghzaf Travail de Bachelor 2012\Code Source\Module SemText\Installation\for the root folder of C drive**"
  doivent être copiés à la racine du lecteur C.

- Le Triplestore[47] avec lequel SemText interagit doit être installé sur un serveur 64bits ayant minimum 8Gb de RAM. La gestion des Ontologies étant gourmande en ressources, il convient de prévoir assez de mémoire pour faire fonctionner à la fois la machine virtuelle contenant l'application SemWidgets et le serveur Tomcat qui héberge Sesam OpenRDF.

IMPORTANT: le Triplestore n'est pas installé sur la machine virtuelle!

7.3.1 Gate

L'installation de Gate7[48] est nécessaire pour le fonctionnement de SemText. Ce dernier utilise des pipelines et des fonctionnalités depuis le dossier original d'installation de Gate.

Le lien avec Gate se fait grâce aux arguments de la machine virtuelle Java (JVM).

Les gazetteers [49] Wordnet se trouvent dans le répertoire:
"**..\semText\semTextFolder\gateApp\WN\Gazetteers**"

7.3.2 OWLIM

Développés par la société bulgare OntoText, OWLIM est un system de gestion de base de données relationnelles RDBMS (*Relational Database Management System*). C'est une implémentation java de la couche stockage et d'inférence qui améliore les capacités de Sesam en ajoutant le support du langage OWL RL[50]. Il est utilisé pour gérer les ontologies utilisées par l'application.

- OWLIM doit être installé (version Lite ou version SE avec licence).
- Un *repository* doit être créé pour SemText. (Le *repository* créé pour SemWidgets s'appelle "ontomanage")

[47] Le *Triplestore* est l'endroit où l'on stocke le triplet expliqué dans la section " RDF de l'introduction au Web sémantique"
[48] "Gate est un *framework* qui offre de multiple fonctionnalités pour traiter du texte et en extraire de l'information à travers son système d'annotation"
Source: http://dumas.ccsd.cnrs.fr/docs/00/56/88/04/PDF/MORARDO_Mikael_M2P.pdf
[49] "gazetteers : ce sont des listes de mots auxquels WordNet attachera une annotation particulière que nous aurons déclarée dans un fichier contenant des métadonnées sur un ou plusieurs gazetteers, voire une annotation qui se trouve au sein de l'un d'entre eux".
Source: http://dumas.ccsd.cnrs.fr/docs/00/56/88/04/PDF/MORARDO_Mikael_M2P.pdf
[50] Ontologie (voir section Ontologies /OWL du chapitre Introduction au web sémantique de ce document)

- L'URL de Sesam WorkBensh et le nom du *repository* doivent être passés en paramètres à l'instance SemText.

IMPORTANT: Le nom du *repository* et son adresse sur le serveur doivent être ajoutés comme paramètre dans la classe " SemTextServiceImpl.java"
Path:
" **/semwidgets-lib/src/main/java/ch/hevs/semwidgets/service/semtext/SemTextServiceImpl.java**"
Location (sur machine virtuelle):
"**C:\dev\svn\semwidgetsapp\semwidgets-lib\src\main\java\ch\hevs\semwidgets\service\semtext\SemTextServiceImpl.java** "

Exemple:

```
SemText   semText   =ch.hevs.semwidgets.semtext.SemText.SemTextSingletonInstance(
           "http://localhost:8080/openrdf-sesame","ontomanage");
```

Figure 37. Déclaration d'une instance SemText

7.3.3 Les fichiers de l'ontologie

Les ontologies employées par l'application doivent être chargées à partir des méthodes existantes dans la Classe SemText_Test, il faut " décommenter " également les lignes spécifiques à chaque opération qui ne doit, à priori, être lancée qu'une fois l'installation de Sesam Open RDF effectuée.

Les méthodes pour charger les fichiers sont citées ci-dessous:

- Pour Wordnet 3.0 l'emplacement est :"
  "**..\SemText\Installation\Ontology to be loaded in OWLIM\WordNet 3.0**\"

```
semText.loadRDFFilesToTripleStore(
"C://SemText//Installation//Ontology to be loaded in OWLIM//WordNet 3.0//",
"http://localhost:8080/openrdf-sesame/repositories/ontomanage",
 false, RDFFormat.TURTLE);
```

Figure 38. Charger les fichiers de l'ontologie Wornet3.0

- Pour charger les métadonnées des Widgets dans l'ontologie l'emplacement du fichier est : "**..\semTextFolder\ inText\ widgetsToImport** "
- Cet emplacement doit être mis dans la variable `pathInTexts`

Les fichiers TXT à placer dans ce répertoire sont compressés dans le fichier "**widgets_to_import.zip**"

```
String pathUserDescription = "C://dev//Workspace//TestSemText//semTextFolder//outTTL";
String pathInTexts=
"C://dev//Workspace//TestSemText//semTextFolder//inText//widgetsToImport" ;

semText.semGraph(pathInTexts, null,
                  pathUserDescription + "\\en\\", Lang.lngEn,
                  SemGraph.textURIPrefix_defaultValue,
                   SemGraph.toSingleFile_YES,
                   SemGraph.loadToStore_YES,
                   "file://userDescriptions_en",
                   SemGraph.oneContextPerFile_NO,
                  SemGraph.addDBKeyFromFileName_YES, "tt_userDesc");
```

Figure 39. Importer les métadonnées des Widgets dans l'ontologie

7.3.4 Arguments JVM

Des arguments de la JVM doivent être ajoutés:

-Dgate.home="**C:\Program Files (x86)\GATE_Developer_7.0" -Xmx1024M**

-Dshell.path="**C:\cygwin\bin\sh.exe**"

7.4 Déploiement de l'application SemWidgets

L'application est compressée dans un fichier EAR à déployer sur un serveur JBoss. Voici les étapes pour la génération et le déploiement de l'application sur la machine virtuelle contenant l'environnement développement.

Un raccourci pour chaque opération a été créé sur le bureau

Tableau 6 - Les raccourcis pour le lancement de l'application sur la machine virtuelle

Opération	Raccourci
1. Lancer WampServer (mySql) pour pouvoir accéder à la base de données	
2. La génération du fichier EAR se déclenche en cliquant sur le raccourci "*Build Semwidgets*" existant sur le bureau. l'opération prend une vingtaine de secondes.	
3. Il faut ensuite aller copier le fichier "semwidgets.ear" généré à l'emplacement "C:\dev\svn\semwidgetsapp\semwidgets\target", un raccourci sur le bureau ouvre ce répertoire.	
4. Coller le fichier dans le répertoire "*deploy*" de JBoss se trouvant à l'emplacement : "C:\dev\Jboss6\jboss-6.1.0.Final\server\default\deploy"	
5. Il ne reste plus qu'à lancer le serveur JBoss en cliquant sur le raccourci "JRbel" se trouvant sur le bureau.	
6. Lancer l'application URL: **http://localhost:8080/semwidgets-web/**	

8- Déroulement du projet

8.1 Planning et réalisation

La planification de ce projet a prévu quatre itérations de 2 à 4 semaines:

1ère Itération: Analyse Widgets

- Widgets: Fonctionnement, différents types, standards
- Framework, environnements
- Compréhension des différents concepts du Web sémantique
- Analyse et choix du processus d'extraction des métadonnées des Widgets vers du sémantique

2ème Itération: Choix et développement

- Développement de l'interface utilisateur
- Intégration des nouveaux Widgets
- Réflexion sur la fonctionnalité de recherche : résultats + recommandations

3ème Itération: Intégration

- Compréhension du logiciel OntoManag
- Extraction des métadonnées
- Interface entre l'application Web et le système OntoManag

4ème Itération: Finalisation

- Finalisation de l'interface
- Intégration de la fonctionnalité de recherche : résultats + recommandations
- Ajout de la fonctionnalité d'uploade des Widgets par les utilisateurs

Toutes les fonctionnalités obligatoires du cahier de charges ont été réalisées, excepté la mise en place d'un système de recommandation qui n'est pas encore aboutie.

Le planning prévu pour chaque itération a été respecté scrupuleusement à l'exception de la partie configuration de Maven et l'intégration SemText qui ont pris beaucoup plus de temps que prévu, et réduisant ainsi la durée planifié pour l'amélioration de l'interface graphique. L'accent a donc plutôt été mis sur la partie architecture et le côté fonctionnel de l'application pour livrer un produit exploitable et ergonomique.

8.2 Décompte des heures

Durant les différentes itérations de ce projet il y avait des tâches principales qui s'effectuaient en parallèle, il s'agit de:

- Recherche et lecture: documentation sur les concepts nouveaux, collecte des informations sur les technologies, recherche des solutions sur le web.
- Analyse comprenant également les tâches de configuration et d'installation:
  - analyse et choix des technologies à utiliser
  - Installation et configuration des outils (environnement développement, module SemText)
- Développement, test et débogage: construction de l'application, développement des différente fonctionnalités
- Rédaction du rapport
- Meeting/Chat/Email

Dans une optique de gestion de projet, un décompte des heures de chaque tâche a été maintenu à jour. Les détails de chaque semaine se trouvent en annexe à ce rapport (cf. YA _ décompte des heures TB 2012 sur le CD livré avec ce rapport).

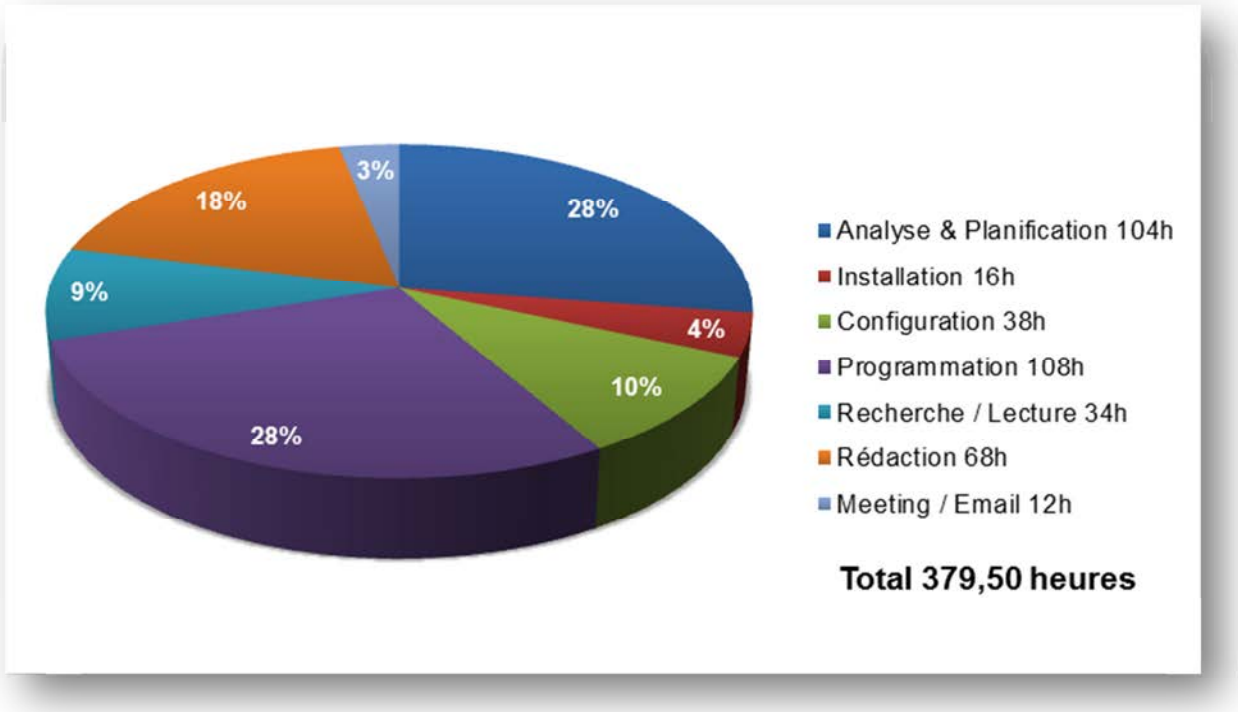


Figure 40 - Graphe du décompte des heures de la réalisation du projet

9- Conclusion

Cette section a pour but de dresser un bilan d'un point de vue technique et personnel portant sur une évaluation des résultats, les difficultés rencontrées suivi des possibles améliorations pouvant être apportées dans le futur.

9.1 Bilan technique

L'application SemWidgets a été développée dans le cadre du projet OntonManag (*Ontologies Management Tool*) son but premier est d'éprouver l'outil de classification d'informations textuelles dans des ontologies (SemText) et démontrer s'il est possible de l'intégrer dans un système fonctionnel.

Le nombre des Widgets sur la toile ne cesse d'augmenter, trouver une solution pour extraire leur métadonnées afin de les "sémantiser", les classer et de les regrouper par différents critères pour aider l'utilisateur de trouver celles qui lui convient est un défi relevé par le développement de l'application SemWidgets.

Pour démontrer si le projet a atteint ces objectifs, il est nécessaire d'évaluer l'application sur deux critères essentiels:

- Métadonnées: La qualité et la quantité de ces métadonnées extraites et exploitées par l'application.
- Recherche: Le temps de réponse ainsi que la pertinence des résultats des recherches effectuées en utilisant SemWidgets.

9.1.1 Métadonnées

Avec plus de 250000 Widgets de tout genre (*native, feed, video, flash*, etc.) et différentes catégories (*games, news, music, tv*, etc.) l'annuaire Netvibe Ecosystem constitue une source très importante de Widgets ayant des métadonnées bien structurées.

Le développement d'une solution pour donnant accès à la totalité de ces Widgets est l'un des points forts de l'application SemWidgets.

Pour les besoin du projet 200 Widgets de différentes catégories ont été extraites et référencées dans la base de données ainsi que dans l'ontologie.

9.1.2 Recherche

Le module SemText a été intégré dans l'application SemWidgets avec succès. Des services ont été créés pour permettre de lui passer les mots-clés et de manipuler les résultats en retour.

Dans le module SemText, la méthode standard générant les requêtes SPARQL ne permettait de faire qu'une recherche classique dite "exact match", et ne retourne que les Widgets qui contiennent le ou les mots-clés recherchés dans leurs métadonnées.

Pour exploiter un peu plus les possibilités offertes par le Web sémantique, une nouvelle méthode de génération de requêtes SPARQL a été développée. Elle permet de faire des recherches par synonymes. Le résultat de la recherche s'en retrouve enrichi.

Un système de *ranking* a été mis en place pour améliorer le classement des résultats ainsi les résultats les plus pertinents se retrouvent au début de la liste évitant à l'utilisateur de visionner la totalité de la liste pour trouver la Widget qui lui convient.

Ce système trie les résultats en se basant sur un nombre de points attribués selon le nombre d'occurrences des mots-clés dans les métadonnées de l'application et la position du mot dans la liste des mots-clés recherchés. Le nombre d'installations est le critère qui départage les résultats ayant le même nombre de points. Ainsi les résultats les plus pertinents et plus populaires se retrouvent au début de la liste.

Dans un souci d'évaluation de performance, le temps du traitement a été calculé grâce au *timestamp* logé pour chaque opération (cf. Annexe: Exemple log pour évaluation performance) :

- Génération de la requête
- Envoi à SemText
- Récupération et *processing* des résultats
- *Ranking*
- Affichage de la liste

Le temps calculé pour chaque requête est de moins d'une demi-seconde par requête (ce temps est calculé après la première instanciation de SemText, un singleton[51] a été mis en place pour utiliser qu'une seule instance SemText ainsi le temps d'instanciation est économisé quand l'utilisateur effectue des recherches.

9.2 Difficultés rencontrées

Comme dans tout projet visant à intégrer de nouveaux concepts, outils et technologies il est inévitable de se heurter à des obstacles qu'il faut surmonter. Chercher des solutions aux problèmes majeurs ou récurrents rencontrés le long de ce projet a constitué une grande partie de ce travail. Les principaux défis relevés sont:

- Nouvelles technologies, nouveaux concepts
- Création d'une application JEE from scratch
- Intégration SemText

9.2.1 Nouvelles technologies

Afin de répondre aux exigences de la qualité du développement de l'application, qui se doit d'être performante pour mettre en avant les possibilités offertes par le Web sémantique, les technologies et les outils choisis répondent aux standards les plus réputés en termes de qualité et de fiabilité. Cela n'exclue pas une certaine complexité dans l'intégration de ces outils, notamment ceux utilisés pour le développement des applications JEE. Ainsi, la mise en place de l'environnement de développement s'est avérée chronophage, vue la multitude de paramétrages devant être effectués avant de pouvoir commencer la construction de l'application.

9.2.2 Création d'une application JEE from scratch

SemWidgets est une application développée from scratch, incluant de nouveaux concepts à assimiler en un temps limité, ce qui a rendu difficile l'estimation du temps nécessaires pour chaque tâche et le respect des délais planifiés pour chaque étape.

[51] Un design pattern limitant l'instanciation d'une classe à un seul objet.

9.2.3 Intégration SemText

Les concepts du web sémantiques (RDF, Ontologies, etc.) m'étaient inconnus ce qui a constitué un challenge à relever en vue d'intégrer le module SemText. Une bonne partie du temps du projet a été consacré à la compréhension de ces concepts ainsi que le fonctionnement de SemText.

Comme c'est un module récemment développé, une documentation expliquant le fonctionnement du module SemText n'a pas encore été faite et il a fallu passer du temps pour en comprendre le mécanisme pour pouvoir l'intégrer dans le projet et surtout pouvoir le paramétrer et y apporter quelques changements notamment pour la génération des requêtes SPARQL.

Maven a été utilisé pour définir le squelette des différents projets Java faisant partie de l'application, les dépendances des libraires et l'ordre de compilation. Cet outil dispose de son propre référentiel de libraires ce qui a provoqué des conflits lors de l'intégration du module SemText qui utilise des versions antérieures de plusieurs des librairies existantes dans le référentiel Maven.

Le conflit des librairies a de loin été le problème le plus coriace auquel il a fallu remédier. Un problème bloquant qui a mis en péril la construction de l'application car le succès du projet reposait sur l'intégration du module SemText dans l'application.

Pour résoudre ce problème, une liste des librairies à importer et/ou à changer dans le référentiel Maven a été dressée. Ensuite, les librairies nécessaires ont été importées une à une et à chaque ajout, la compilation a été testée. Une tâche fastidieuse mais qui a fini par porter ses fruits, le module SemText a pu être intégré avec succès.

9.3 Les améliorations futures

Ce projet représente un essai d'intégration de quelques possibilités que propose le web sémantique et bien sûr il est loin d'exploiter tout le potentiel de ce dernier. Cette section expose quelques améliorations possibles à ce sujet ainsi que des extensions envisageables pour l'application SemWidgets.

9.3.1 Web sémantique

La base de données lexicale Wordnet3.0 a été utilisée pour la recherche par synonymes. Elle offre plus de types de relations pouvant être exploitées par des requêtes SPARQL, il s'agit de la métonymie, hyperonymie, l'homonymie et bien d'autre (cf. ces termes sont expliqués dans la rubrique WordNet 3.0 de ce document)

9.3.2 Ergonomie

La priorité portée sur l'aspect fonctionnel de l'application, ce qui a reporté au second plan le développement du design de l'interface graphique de l'application. L'utilisation d'APIs comme RichFaces ou PrimeFaces et JQuery permettrait d'améliorer l'aspect visuel de l'application et la rendre plus conviviale.

9.3.3 Application mobile

En optant pour une architecture basée sur des EJBs et respectant une séparation stricte des différentes couches de l'application, le développement d'une version mobile de l'application pourra se faire assez facilement car elle ne s'occupera que de la logique d'affichage puisque la couche métier est déjà prête côté serveur.

9.3.4 Widget

Un Widget permettant de profiter de la fonctionnalité de recherche SemWidgets pourra être développée. Le même argument que pour l'application mobile reste valable pour une autre interface visuelle comme un Widget.

9.4 Conclusion: avis personnel

Ce projet a été une expérience très enrichissante dans la mesure où il m'a permis de découvrir des concepts qui m'étaient jusqu'alors inconnus.

Construire une application JEE from scratch en utilisant des technologies récentes et souvent complexes a constitué un challenge pour le moins très intéressant à relever.

J’ai pu me tester durant la quête des solutions et me mettre à l’épreuve lorsque à chaque fois que j’étais confrontés à des points de blocage.

Je me suis enrichi concernant les diverses approches à adopter afin de résoudre l’ensemble des problèmes rencontrés et ainsi atteindre les objectifs prédéfinis.

L'intégration d'un module réalisé par un autre développeur a aiguisé mon sens de l'analyse et de la compréhension du code.

L'expérience de gestion d'un projet sur le long terme est également à compter parmi les points positifs de ce projet.

Les heures passées à chercher des solutions, à configurer des outils complexes comme Maven et à se documenter m'ont insufflé persévérance et patience.

Ma vision sur la réalisation de bout en bout d'un projet informatique a évolué.

Finalement ce projet est de loin le plus ambitieux et le plus stimulant qui m'est était donné de faire dans le cadre de ma formation à la HES-SO.

10- Bibliographie

(s.d.). Consulté le 04 12, 2012, sur http://www.w3.org/TR/widgets/

(s.d.). Consulté le 04 14, 2012, sur http://www.openajax.org/index.php

Alexander, K. (2007). *RDF For The Rest Of Us (RDF pour Nous Autres).* Digital Wab Magazine.

Anderruthy, J.-N. (2009). *du Web 2.0 au Web 3.0 Les nouveaux services internet.* st Herblain: Editions ENI.

Calvé, A. l. (2011). *11_05_23_RCSO 29450_OntoManag.*

Fellbaum, C. (1998). *An WordNet Electronic Lexical Database.* Massachusetts: MIT Press.

Goncalves, A. (2011). *Le cahier du programmeur Java EE 5.* Mayenne: editions eyrolles.

Lafosse, J. (2009). *Java EE Guide de développement d'applications web en Java.* st Herblain: Editions ENI.

10.1 Référence

[1] RDFa: Resource Description Framework – in – attributes [en ligne] – accès le 02 avril 2012

[2] Informations: http://schema.org/ [en ligne] – accès le 04 Juin 2012

[3] http://www.01net.com/editorial/566341/knowledge-graph-google-fait-un-pas-vers-le-web-semantique/ [en ligne] – accès le 02 août 2012

[4] http://wimmics.inria.fr/projects/dbpedia/doc/index.php/Accueil [en ligne] – accès le 02 août 2012

[5] (http://www.journaldunet.com/diaporama/0610-dicoweb2/1.shtml) [en ligne] – accès le 05 août 2012

[6] http://www.slideshare.net/srvwiz/explaining-the-semantic-web-1455176 [en ligne] – accès le 10 avril 2012

[7] http://www.yoyodesign.org/doc/digital-web/rdf-for-the-rest-of-us/ [en ligne] – accès le 30 avril 2012

[8] http://xmlfr.org/documentations/tutoriels/041015-0001 [en ligne] – accès le 15 juillet 2012

[9] http://www.yoyodesign.org/doc/w3c/rdf-sparql-query/ [en ligne] – accès le 20 avril 2012

[10] http://www.foaf-project.org/ [en ligne] – accès le 02 août 2012

[12]http://www.yoyodesign.org/doc/w3c/webont-req-20040210/index.htm [en ligne] – accès le l 05 août 2012

[13]http://www.w3.org/2004/OWL/ [en ligne] – accès le 08 août 2012

[14]http://fr.wikipedia.org/wiki/Th%C3%A9saurus [en ligne] – accès le 07 juillet 2012

[15] http://wordnet.princeton.edu/wordnet/ [en ligne] – accès le 04août 2012

[18] http://www.w3.org/TR/2008/WD-widgets-land-20080414/ [en ligne] – accès le 03 juillet 2012

[19] Microsoft : http://technet.microsoft.com/en-us/security/advisory/2719662 [en ligne] – accès le 29 juillet 2012
Yahoo : http://widgets.yahoo.net/blog/?p=26 [en ligne] – accès le 29 juillet 2012

[20] http://www.w3.org/TR/widgets-land/ [en ligne] – accès le 18 Juin 2012

[22] http://www.w3.org/Consortium/mission.html [en ligne] – accès le 04 avril 2012

[23] Index TIOBE: http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html [en ligne] – accès le 01 juillet 2012

[24]Documentation Oracle: http://docs.oracle.com/cd/E19798-01/821-1841/gipmb/index.html [en ligne] – accès le 01 juin 2012

[25] http://www.techno-science.net/?onglet=glossaire&definition=1471 [en ligne] – accès le 01 août 2012

[27] Source du logo: http://www.eclipse.org/ [en ligne] – accès le 28 juin 2012

[28] http://www.mysql.com/ [en ligne] – accès le 13 juillet 2012

[31] http://fr.wikipedia.org/wiki/Apache_Maven (visité le 28.05.2012) [en ligne] – accès le 13 juillet 2012

[33] Source Logo: maven.apache.org [en ligne] – accès le 15 juillet 2012

[36] Source logo: http://www.jboss.org/ [en ligne] – accès le 15 juillet 2012

[37] Source logo: http://tomcat.apache.org/ [en ligne] – accès le 14 juillet 2012

[39] Source logo: http://stax.codehaus.org/ [en ligne] – accès le 29 juillet 2012

[41] http://dev.netvibes.com/ [en ligne] – accès le 01 avril 2012

[45] http://dumas.ccsd.cnrs.fr/docs/00/56/88/04/PDF/MORARDO_Mikael_M2P.pdf [en ligne] – accès le 02 juillet 2012

11- Tableau du décompte des heures

Semaine	1	2	3	4	5	6	7	8	9	10
Recherche / Lecture	4,00	6,00	6,00	2,00	3,00	5,00	0,00	4,00	8,00	5,00
Analyse et planification	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	4,00	0,00
Configuration	0,00	0,00	0,00	0,00	0,00	0,00	0,00	2,00	4,00	4,00
Programmation	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
Installation	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	5,00
Rédaction	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
Meeting / Email	1,00	0,00	0,00	1,00	0,00	0,00	0,00	1,00	0,00	0,00
Total	**5,00**	**6,00**	**6,00**	**3,00**	**3,00**	**5,00**	0,00	**7,00**	**16,00**	**14,00**

11	12	13	14	15	16	17	18	19	20	21	22	**Total heures**
0,00	0,00	2,00	0,00	0,00	12,00	3,00	0,00	0,00	3,00	11,00	30,00	**104,00**
0,00	6,00	6,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	**16,00**
0,00	0,00	9,00	0,00	0,00	10,00	0,00	0,00	7,00	0,00	0,00	2,00	**38,00**
12,00	0,00	0,00	0,00	0,00	0,00	22,00	20,00	16,00	12,50	25,00	0,00	**107,50**
4,00	8,00	3,00	0,00	0,00	2,00	0,00	4,00	0,00	8,00	0,00	0,00	**34,00**
0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	1,00	4,00	27,00	36,00	**68,00**
2,00	2,50	0,00	0,00	0,00	0,00	0,00	2,00	0,00	1,50	1,00	0,00	**12,00**
18,00	**16,50**	**20,00**	0,00	0,00	**24,00**	**25,00**	**26,00**	**24,00**	**29,00**	**64,00**	**68,00**	**379,50**

12- Table des illustrations

12.1 Table des figures:

Figure 1- Mise en place du moteur de recherche..... VI

Figure 2 - Google Knowledge Graph 5

Figure 3 - Architecture du Web sémantique..... 7

Figure 4 - Evolution du web..... 8

Figure 5 - Exemple de graphe RDF..... 9

Figure 6 - Architecture Web sémantique Vs. Web classique 14

Figure 7 - Ensemble de Widgets pour Windows et Dashboard.....21

Figure 8 - Fonctionnement d'un Widget instancié 21

Figure 9 - Une pile typique de technologies et aspects standardisés.....24

Figure 10. Logo W3C 24

Figure 11. Logo Open Ajax Alliance 25

Figure 12 - Logo Eclipse.....30

Figure 13 - Logo MySQL 30

Figure 14 - Logo Hibernate.....31

Figure 15 - Logo Maven 31

Figure 16 - Logo JBoss 32

Figure 17 - Logo Tomcat 32

Figure 18 - Logo Codehaus.....32

Figure 19 - Architecture de l'application SemWidgets 35

Figure 20. Recherche sur Google Gadgets 41

Figure 21 - Google Gadgets - Code source des résultats de la recherche 41

Figure 22 - Schéma BPM: Extraction des Widgets à partir de Google Gadget 42

Figure 23. Code source d'une page Netvibes 43

Figure 24- Schéma BPM: Extraction des Widgets à partir de Netvibes 44

Figure 25 - Résultat de la requête Netvibes Ecosystem 46

Figure 26. Persistance des objets Widgets dans la base et création des fichiers textes 47

Figure 27. Importer les métadonnées des Widgets dans l'ontologie 48

Figure 28 – Exemple d'appel de la méthode SemQuery pour générer une requête SPARQL avec le mot "radio" 49

Figure 29 - Lancement de l'application à partir Google Chrome 51

Figure 30 - Page Web pour importer des Widgets 52

Figure 31 - Déroulement de l'import (sur console) 53

Figure 32 - Widgets enregistrés dans la base de données 54

Figure 33 - Exemple des fichiers importés dans l'ontologie 55

Figure 34 - Liste des résultats 56

Figure 35 - Détails d'une Widget 57

Figure 36 - Exporter le Widget sur une autre plateforme 58

Figure 37. Déclaration d'une instance SemText 61

Figure 38. Charger les fichiers de l'ontologie Wornet3.0 61

Figure 39. Importer les métadonnées des Widgets dans l'ontologie 62

Figure 40 - Graphe du décompte des heures de la réalisation du projet.....66

12.2 Table des tableaux

Tableau 1 - Modules OntoManag 2

Tableau 2 – Structure des fichiers dans l'environnement de développement Eclipse 37

Tableau 3 - Métadonnées et support de stockage.....40

Tableau 4 - Paramètre d'une requête Ecosystem Netvibes 45

Tableau 5 - Les résultats d'une requête SPARQL et leurs significations.....49

Tableau 6 - Les raccourcis pour le lancement de l'application sur la machine virtuelle 63

12.3 Liste des annexes

À consulter sur CD livré avec ce rapport (..\Classeur\Annexes)

- Annexe1 - 11_05_23_RCSO 29450_OntoManag.PDF
- Annexe2 - Cahier des charges.doc
- Annexe3 - YA _Compte rendu des heures TB2012.xlsx
- Annexe4 - Langage d'interrogation SPARQL pour RDF
- Annexe5 - LEA_Web3Widgets.pdf
- Annexe6 - OpenAjax Alliance.pdf
- Annexe7 - OWLIM on Sesame.pdf
- Annexe8 - Widget Packaging and XML Configuration.pdf
- Annexe9 - Semwidgetsdb.sql
- Annexe10 - Exemple Log pour évaluation performance.txt

Déclaration sur l'honneur

Je déclare, par ce document, que j'ai effectué le travail de bachelor ci-annexé seul, sans autre aide que celles dûment signalées dans les références, et que je n'ai utilisé que les sources expressément mentionnées. Je ne donnerai aucune copie de ce rapport à un tiers sans l'autorisation conjointe du RF et du professeur chargé du suivi du travail de bachelor, y compris au partenaire de recherche appliquée avec lequel j'ai collaboré, à l'exception des personnes qui m'ont fourni les principales informations nécessaires à la rédaction de ce travail.