Mourad LOUKAM

Maître Assistant Chargé de cours
Département d'Informatique
Université Hassiba Ben Bouali Chlef

WEB SEMANTIQUE

Support de Cours

S O M M A I R E

CHAPITRE I : INTRODUCTION 4
1.1 Introduction : 4
1.2 Histoire du web sémantique : 4
1.3 Objectifs du web sémantique : 5
1.4 La pile du web sémantique : 5
1.5 La base du web sémantique : l'ontologie 6

CHAPITRE II : Le modèle RDF 8
2.1 Introduction : 8
2.2 Objectifs 8
2.3 Syntaxe de RDF 11
2.3.1 Syntaxe abstraite 11
2.3.2 Syntaxe N-Triples 11
2.4 Identification du type de ressources 13
2.5 Les conteneurs 14
2.6 Les collections 16
2.7 RDF Schema 16
2.7.1 Classes 17

CHAPITRE III : Les ontologies 19
3.1 Introduction : 19
3.2 Notions de base : 20
3.2.1 Le concept 20
1.4.2 Relations 21
Relation de subsomption 22
3.3 Classification d'ontologies : 23
3.4 Démarche pour la construction d'une ontologie : 23
3.5 Le langae owl (Web Ontology Language) : 25
3.5.1 Objectifs et motivation 25
3.5.2 Différentes déclinaisons de OWL 25
3.5.3 Structure d'une Ontologie OWL 26
3.5.4 Elément du langage OWL 27
3.6 Exemple d'ontologie : 30
3.7 Les éditeurs d'ontologies : 36
3.7.1 L'éditeur "Protégé" 36

CHAPITRE IV : Le langage de requêtes SPARQL 39
4.1 Introduction : 39
4.2 Requêtes de type SELECT 39
4.3 Requêtes de type CONSTRUCT 43
4.4 Contraintes par FILTER 45

CHAPITRE IV : Quelques moteurs de recheches sémantiques 47
4.1 Le moteur Corese : 47

CHAPITRE I : INTRODUCTION

1.1 INTRODUCTION :

Le Web sémantique (plus techniquement appelé « le Web de données ») permet aux machines de comprendre la sémantique, la signification de l'information sur le Web. Il étend le réseau des hyperliens entre des pages Web classiques par un réseau de lien entre données structurées permettant ainsi aux agents automatisés d'accéder plus intelligemment aux différentes sources de données contenues sur le Web et, de cette manière, d'effectuer des tâches (recherche, apprentissage, etc.) plus précises pour les utilisateurs. Le terme a été inventé par Tim Berners-Lee, co-inventeur du Web et directeur du W3C, qui supervise l'élaboration des propositions de standards du Web sémantique.

La plupart du temps, lorsque l'on prononce le terme de Web sémantique, on parle des différentes technologies qui se cachent derrière. Parmi les plus connues, on peut citer RDF (Ressource Description Framework) qui correspond à un modèle d'information, et les formats d'échanges de données en RDF pour communiquer entre différentes applications (RDF/XML, RDF/JSON, N3, Turtle, N-Triples et d'autres). Dans le domaine du Web sémantique, la sémantique des données est décrite par des ontologies - ce terme sera défini plus loin - avec des langages prévus pour fournir une description formelle de concepts, termes ou relations d'un domaine quelconque. Ces langages sont RDFS (Ressource Description Framework Schema) et OWL (Web Ontology Language). Il existe aussi des langages de description des données structurées dans du XHTML afin que des outils effectuent un traitement automatique de ces différentes données. Ces langages sont RDFa et Microformat et, nouvellement arrivé avec HTML 5, Microdata. Voici d'ailleurs un article qui vous introduit le langage RDFa et un autre sur les Microdata. Ensuite, pour finir avec la liste des technologies, il existe un langage de requête, au même titre que SQL pour les bases de données relationnelles, SPARQL, qui effectue des requêtes mais sur des triplets RDF.

1.2 HISTOIRE DU WEB SEMANTIQUE :

En 1994, lors de la première conférence WWW à Genève, plus précisément au CERN, a lieu l'annonce de la création du W3C. C'est d'ailleurs à cette période que Tim Berners-Lee dresse les objectifs du W3C et montre les besoins d'ajouter de la sémantique au Web futur. Il montre alors en quoi les liens hypertextes ou, plus précisément, la façon dont on met en relation les documents sur le Web est trop limitée pour permettre aux machines de relier automatiquement les données contenues sur le Web à la réalité. Compte tenu de l'ambition d'un tel projet, cette idée suscite quelques résistances et controverses qui sont classiquement rencontrées dès qu'on aborde des problématiques liées au domaine de l'intelligence artificielle.

Après cette conférence, mise à part la mise en place des recommandations nécessaires à la construction des documents, le W3C nouvellement créé entame les premières réflexions sur la mise en place du Web sémantique. Ces réflexions aboutissent à la publication d'un premier *draft* de

recommandations sur le Web sémantique en octobre 1997 et d'une seconde en avril 1998. Cette même année, Tim Berners-Lee publie un document sur les toutes premières moutures de ce qui sera plus tard appelé le Web sémantique. Ces moutures consistent à mettre en place les différentes technologies du Web sémantique. Dans ce document, il présente le Web sémantique comme une sorte d'extension du Web des documents, qui constitue une base de données à l'échelle mondiale, afin que toutes les machines puissent mieux lier les données du Web.

1.3 OBJECTIFS DU WEB SEMANTIQUE :

Un des principaux objectifs du Web sémantique est de permettre aux utilisateurs d'utiliser la totalité du potentiel du Web : ainsi, ils pourront trouver, partager et combiner des informations plus facilement. Aujourd'hui tout le monde est capable d'utiliser des forums, d'utiliser des réseaux sociaux, de chatter, de faire des recherches ou même d'acheter différents produits. Néanmoins, il serait mieux que la machine fasse tout ceci à la place de l'homme, car actuellement, les machines ont besoin de l'homme pour effectuer ces tâches. La raison principale est que les pages Web actuelles sont conçues pour être lisibles par des êtres humains et non par des machines. Le Web sémantique a donc comme principal objectif que ces mêmes machines puissent réaliser seules toutes les tâches fastidieuses comme la recherche ou l'association d'informations et d'agir sur le Web lui-même.

1.4 LA PILE DU WEB SEMANTIQUE :

Selon le W3c, le web sémantique répond à une architecture composée de la superposition et de la composition de plusieurs "briques" , comme le montre la figure suivante :

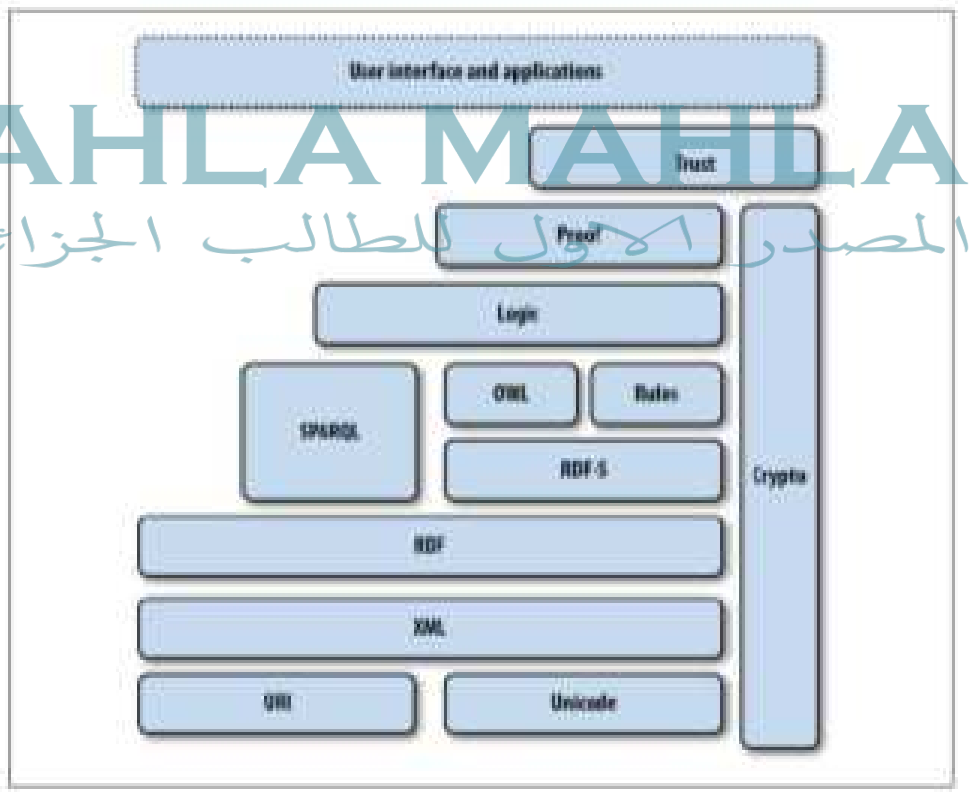


Figure . Pile du web sémantique , selon le W3C

- Le XML fournit une syntaxe élémentaire, pour la structure du contenu dans les documents, mais il ne décrit pas la sémantique du document. XML n'est pas à l'heure actuelle une composante nécessaire des technologies du Web sémantique dans la plupart des cas, comme syntaxes

alternatives il existe Turtle. Turtle est un standard de facto car moins verbeux que XML, mais n'a pas été choisi à travers un processus de normalisation formelle.

- Le XSD est un langage de description de format de document XML permettant de définir la structure et le type de contenu d'un document XML. Cette définition permet notamment de vérifier la validité de ce document.

- Le RDF est un langage simple pour exprimer des modèles de données sous forme d'objets (« ressources ») et de leurs relations. Un modèle basé sur RDF peut être représenté à travers plusieurs syntaxes d'échanges, par exemple, RDF/XML, N3, Turtle, et RDFa[25]. RDF est une norme fondamentale du Web sémantique[26,27,28].

- RDF Schema étend le RDF et son vocabulaire pour pouvoir structurer les propriétés et les classes au sein d'une ressource décrite en RDF.

- OWL ajoute plus de vocabulaire pour décrire les propriétés et les classes : comme avec les relations entre les classes, la cardinalité (par exemple « exactement un »), l'égalité, le typage des propriétés, les caractéristiques de propriétés (par exemple la symétrie), etc.

- SPARQL (prononcé *sparkle* ; en anglais : « étincelle »[29]) est un langage de requête et un protocole qui permettra de rechercher, d'ajouter, de modifier ou de supprimer des données RDF disponibles dans le Web à travers l'Internet.

1.5 LA BASE DU WEB SEMANTIQUE : L'ONTOLOGIE

L'ontologie est la base de ce que l'on appelle la représentation des connaissances. Ce domaine est né de la volonté des chercheurs de représenter diverses connaissances du monde actuel, de façon à ce qu'elles soient utilisables par des ordinateurs, pour qu'ils puissent effectuer des raisonnements sur ces connaissances. Ces connaissances sont exprimées sous forme de symboles auxquels on donne une « sémantique » (un sens).

Imaginons la problématique suivante : vous voulez interroger une base de données contenant diverses ressources (textes, images, vidéos...) et une requête (question ou mot(s) clé(s)), comment trouver les données se trouvant dans cette base qui correspondent à cette requête ?

Par exemple tapez dans votre moteur de recherche préféré les mots suivants : « ordinateur portable » puis « laptop ». Vous pouvez vous apercevoir que les résultats ne sont pas du tout les mêmes, alors que, vu que les deux mots représentent la même chose, on pourrait s'attendre à trouver les mêmes réponses.

De la même façon en lançant une recherche avec le mot clé "ibn badis" , le moteur de recherche affiche "pêle-mêle" des milliers de résultats hétérogènes.

Que se passe-t-il ? En fait, le moteur de recherche compare des mots sans prendre en compte leur sémantique (sens). Il exécute uniquement une recherche strictement syntaxique et donc sans réflexion car « ordinateur portable » et « laptop » représentent le même concept (la même chose), que nous appellerons maintenant des classes pour respecter la terminologie du Web sémantique. Plus précisément, on peut dire que la relation de spécialisation sur les classes n'est pas gérée. Par exemple, « notebook » est une spécialisation de la classe générale « laptop ». Ainsi, pour raisonner, il ne faut plus se baser sur les mots mais sur les classes. Mais que signifie raisonner ? Raisonner c'est utiliser sa raison pour démontrer quelque chose. C'est un terme très souvent employé en intelligence artificielle.

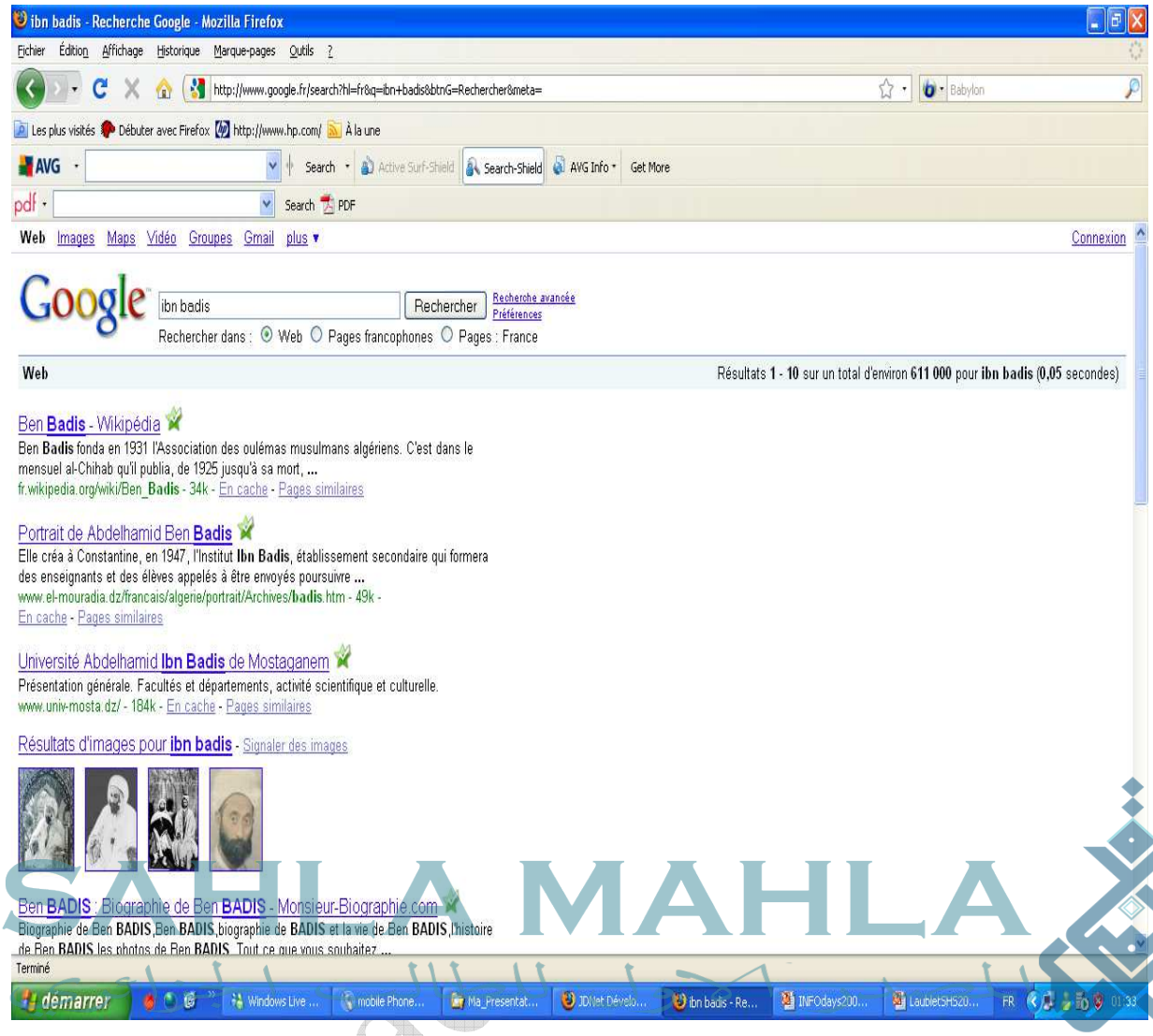


Fig 1 : Les résultats d'une recherche sur un moteur de recherche classique

CHAPITRE II : Le modèle RDF

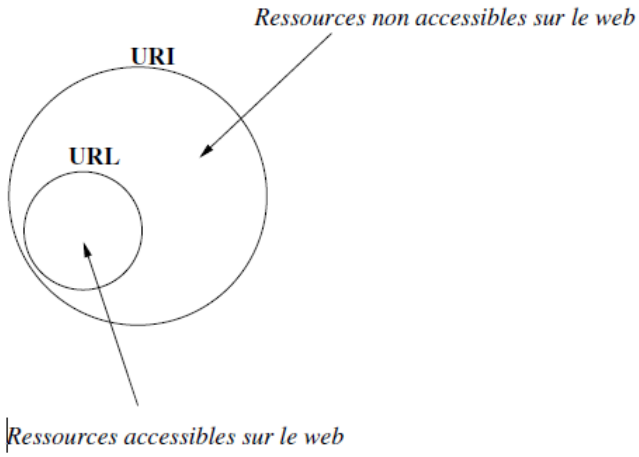
2.1 INTRODUCTION :

Si le Web a été conçu, au départ, pour permettre à des intelligences humaines de lire des documents – accessibles par des hyper-liens –, la démultiplication de l'information disponible nécessite aujourd'hui d'autres types d'accès, basés sur des automatismes gérés par des programmes informatiques. C'est dans ce cadre que l'on voit fleurir bon nombre de moteurs de recherche sur Internet et beaucoup de technologies comme celles des "*agents*".

RDF permet de rendre plus "intelligente" l'information nécessaire à ces moteurs de recherche et, plus généralement, nécessaire à tout outil informatique analysant de façon automatisée des pages Web. Pour ce faire, RDF propose d'associer à toute ressource du Web un ensemble de descriptifs qui caractérisent au mieux cette ressource : on parle alors de métadonnées (en anglais, *metadata*). Présupposant un monde ouvert ou tout le monde parle de tout, RDF permet d'agréger des informations issues de différents éditeurs, sur un même sujet.

2.2 OBJECTIFS

RDF (Resource Description Framework) n'est pas à proprement parler un langage. Il s'agit plutôt d'un modèle de données pour décrire des ressources sur le web. On entend par ressource toute entité que l'on veut décrire sur le web mais qui n'est pas nécessairement accessible sur le web. Par exemple, on pourrait fournir des informations sur un auteur, malgré que la personne décrite n'est pas accessible par le web. On trouve plutôt des ressources, comme sa page personnelle, ou une photo, qui peuvent être obtenues à partir de leur URL (Universal Resource Locator). Ces ressources sont reliées à cette personne, mais ne sont pas cette personne. Pour désigner cette personne, on utilisera une URI (Universal Resource Identifer), un nom unique qui ressemble syntaxiquement à une URL, mais à la différence près qu'il n'est pas nécessaire que celle-ci soit accessible sur le web. D'une certaine manière l'ensemble des URL est inclus dans l'ensemble des URI (voir figure).



La raison d'être de RDF est de permettre que les informations sur les ressources soient manipulées par des applications, plutôt que d'être simplement affichées aux utilisateurs du web. C'est entre autres pour cette raison qu'une syntaxe XML a été proposée pour véhiculer des informations modélisées en RDF.

Parmi les caractéristiques importantes de RDF, on retrouve sa flexibilité et son extensibilité. N'importe qui peut ajouter des informations sur une ressource, en autant que l'on connaisse son URI. Ainsi, deux documents distincts situés à des endroits différents peuvent fournir des descriptions d'une même ressource. Rien n'empêche une application d'extraire les descriptions fournies par ces documents et de le fusionner. Aussi, rien n'oblige qu'une ressource décrite existe réellement.
En fait, RDF impose peu de contraintes sur les descriptions possibles. Supposons par exemple que nous voulions décrire une personne qui s'appelle *Michel Gagnon*, qui travaille au département de génie informatique de l'École polytechnique de Montréal et dont la page personnelle se trouve à l'URL suivante :

```
http://www.professeurs.polymtl.ca/michel.gagnon.
```

Pour ce faire, il faut d'abord reconnaître qu'il y a quatre entités référées par la description : la personne en question, son nom, l'endroit où elle travaille et sa page personnelle. Pour chacune, sauf le nom, il faut donc une URI pour la représenter. Le nom est un cas spécial, puisqu'il s'agit en fait d'une chaîne de caractères. Nous pourrons dans ce cas utiliser la chaîne directement, sans passer par l'intermédiaire d'une URI pour la désigner.
Supposons donc que les quatre entités de notre description sont désignées de la manière suivante :

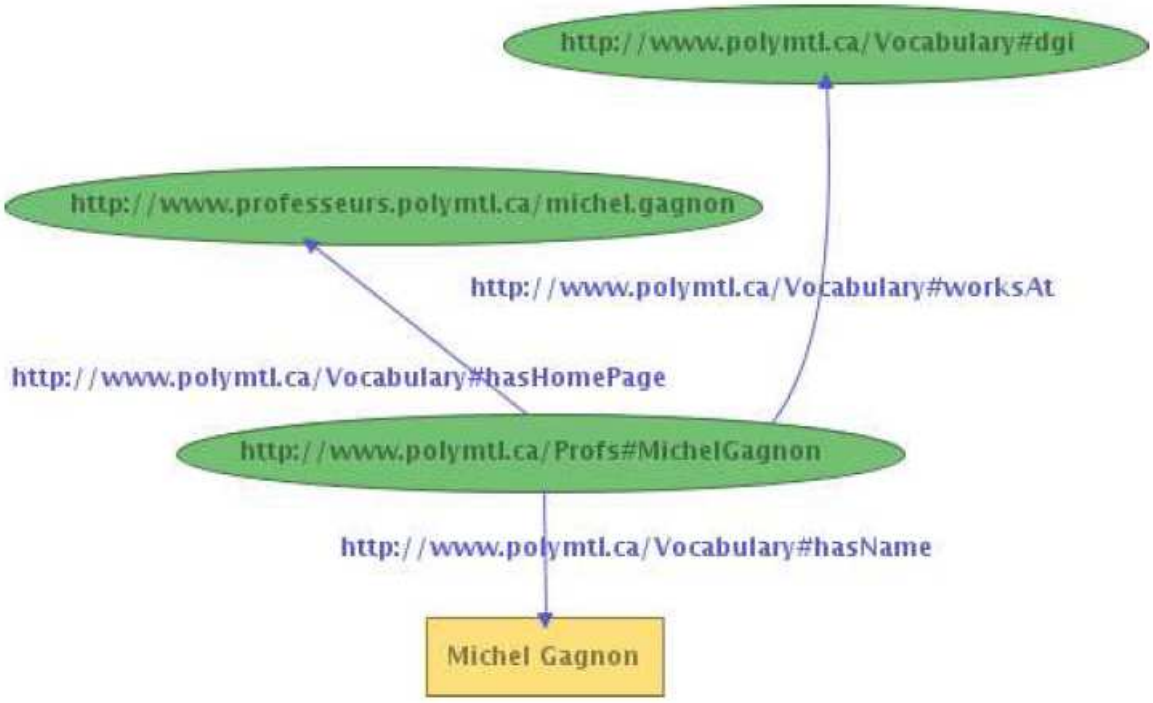
```
la personne décrite http://www.polymtl.ca/Profs#MichelGagnon
le nom de la personne "Michel Gagnon"
le lieu de travail http://www.polymtl.ca/Vocabulary#dgi
la page personnelle
http://www.professeurs.polymtl.ca/michel.gagnon
```


Veuillez notez que même si les URI désignant la personne et son lieu de travail ont la forme http://www..., cela ne signifie pas nécessairement qu'il s'agisse d'une URL correspondant à un document auquel on peut accéder sur le web. Il s'agit tout simplement d'une convention utilisée pour uniformiser les URI. L'intérêt de cette convention est qu'elle laisse la possibilité de traiter cette URI comme une URL et de mettre sur le web un document correspondant à cette URL. Un document, par exemple, qui expliquerait de manière informelle l'entité représentée par l'URI en question. Mais rappelons encore une fois qu'il n'est pas nécessaire qu'il y ait un document sur le web pour chaque URI.

Il nous faut maintenant représenter les relations entre ces entités. RDF prévoit l'existence de propriétés. Il s'agit d'entités dont le rôle est d'établir un lien entre deux autres entités. Dans notre exemple, il faudra utiliser trois propriétés pour relier la personne avec son nom, son lieu de travail, et sa page personnelle. Les propriétés sont elles aussi désignées par des URI. La figure 2 illustre notre description en RDF, en utilisant les propriétés suivantes, respectivement :

```
http://www.polymtl.ca/Vocabulary#hasName
http://www.polymtl.ca/Vocabulary#worksAt
http://www.polymtl.ca/Vocabulary#hasHomePage
```

On voit bien, avec cet exemple, qu'un modèle RDF est en fait un graphe, avec deux types de nœuds. Certains nœuds, représentés par une ellipse, désignent une entité référée par une URI. D'autres nœuds, représentés par un rectangle, représentent un littéral, c'est-à-dire une entité exprimée directement, une chaîne de caractères par exemple. Nous verrons plus loin qu'un littéral peut aussi être d'un autre type, comme un entier, une valeur booléenne, une date, etc.



Modèle RDF représenté par un graphe

On remarquera aussi, dans notre exemple, que plusieurs URI ont le même préfixe : http://www.polymtl.ca/Vocabulary#.
Il est possible d'alléger la représentation en utilisant un alias pour ce préfixe. Ainsi, si on utilise les alias local: et prof: pour les préfixes http://www.polymtl.ca/Vocabulary# et http://www.polymtl.ca/Profs#, respectivement, notre description sera telle qu'illustrée à la figure suivante :

Utilisation des alias dans un graphe

2.3 SYNTAXE DE RDF

RDF étant un modèle de données sous forme de graphe, il n'a pas à proprement parler de syntaxe. En fait, il y a plusieurs syntaxes possibles pour représenter une description RDF. La plus connue est la syntaxe RDF/XML qui, utilisant le méta-langage XML, permet de créer des descriptions facilement manipulables par la machine. Comme il faut bien pouvoir spécifier les contraintes de toute syntaxe utilisée pour représenter un graphe RDF, une syntaxe abstraite a donc été définie, que nous allons maintenant décrire brièvement. Un graphe RDF peut être représenté dans les trois syntaxes suivantes : RDF/XML, N-Triples et Notation 3.

2.3.1 SYNTAXE ABSTRAITE

La structure sous-jacente à toute expression RDF est une collection de triplets, chacun constitué d'un sujet, d'un prédicat et d'un objet. Un ensemble de tels triplets forme un graphe RDF. Le sujet d'un triplet peut être une URI ou un nœud vide, c'est-à-dire un nœud qui désigne une ressource sans la nommer. Le prédicat, qui représente une propriété, est toujours une URI. Finalement, l'objet, qui représente la valeur de la propriété lorsqu'appliquée au sujet, peut être une URI, un nœud vide ou un littéral.

Notons qu'il y a plusieurs avantages à utiliser des URI pour désigner les ressources décrites par une graphe RDF. Premièrement, elles permettent de désambiguïser les désignations utilisées et de permettre à plusieurs applications de partager le même vocabulaire tout en évitant les conflits de noms. De plus, le fait que les propriétés sont elles-mêmes désignées par des URI permet que celles-ci soient aussi utilisées comme des ressources. On peut donc ajouter des triplets RDF qui fournissent des informations sur ces propriétés.

Un nœud vide est tout nœud qui n'est ni une URI, ni un littéral. Il s'agit d'un nœud unique qui peut apparaître dans plusieurs triplets, et qui n'a pas de nom intrinsèque. En quelque sorte, un nœud vide représente une ressource anonyme.
Un graphe peut contenir deux types de littéral. Un littéral simple est constitué d'une chaîne de caractères, appelée forme lexicale, et facultativement d'un attribut indiquant la langue. Un littéral typé est formé d'une chaîne de caractères (qui constitue la forme lexicale) et d'une URI indiquant un type qui sera utilisé pour décoder cette chaîne.
Par exemple, la chaîne typée "10"^^xsd:integer indique que la chaîne "10" doit être interprétée comme un entier, selon le type xsd:integer défini dans la norme de XML Schema. À remarquer que RDF n'a pas de types pré-définis. Il faudra donc toujours utiliser une ressource externe à RDF pour interpréter un littéral typé. Rien n'empêche une application de définir ses propres types.

2.3.2 SYNTAXE N-TRIPLES

La notation N-Triples est la plus simple de toutes. Selon cette notation, un graphe RDF est représenté par une collection de triplets de la forme suivante (chaque ligne du document contient un seul triplet) :
Sujet Prédicat Objet .
Si le sujet, le prédicat ou l'objet est une URI, on le représente en mettant entre crochets <> la forme non abrégée de cet URI. Ainsi, on ne peut pas utiliser de préfixe dans la notation N-Triples.
Dans le cas où le sujet ou l'objet est un noeud vide, on utilise la forme :nom, où nom est tout simplement un identificateur unique pour ce noeud vide. La seule raison d'être de ce nom est de pouvoir référer au même noeud vide dans deux triplets différents lorsque le noeud vide est impliqué dans plus d'une relation.

Finalement, un littéral est représenté directement sans modification. En utilisant la notation N-Triples, nous obtenons la représentation suivante pour le graphe RDF de la figure 2 :

```
<http://www.polymtl.ca/Profs#MichelGagnon>
<http://www.polymtl.ca/Vocabulary#worksAt>
<http://www.polymtl.ca/Vocabulary#dgi> .
<http://www.polymtl.ca/Profs#MichelGagnon>
<http://www.polymtl.ca/Vocabulary#hasName> "Michel Gagnon"
.
<http://www.polymtl.ca/Profs#MichelGagnon>
<http://www.polymtl.ca/Vocabulary#hasHomePage>
<http://www.professeurs.polymtl.ca/michel.gagnon> .
```

Si le noeud dénoté par l'URI //www.polymtl.ca/Profs#MichelGagnon était un noeud vide, c'est-à-dire si l'on voulait désigner la personne en question sans la nommer, on aurait la représentation suivante (à noter que l'identi_cateur du noeud vide est totalement arbitraire) :

```
_:p234 <http://www.polymtl.ca/Vocabulary#worksAt>
<http://www.polymtl.ca/Vocabulary#dgi> .
_:p234 <http://www.polymtl.ca/Vocabulary#hasName> "Michel Gagnon" .
_:p234 <http://www.polymtl.ca/Vocabulary#hasHomePage>
<http://www.professeurs.polymtl.ca/michel.gagnon>
.
```

On voit bien que cette notation est fastidieuse, étant donné l'impossibilité d'abbréger les URI. Dans la suite de ce document, nous utiliserons une manière abrégée. Au lieu d'écrire l'URI au complet entre crochets, nous utiliserons la forme prefixe:URI. À noter qu'il s'agit d'un simple ajustement pour faciliter la lecture, que les triplets écrits de cette manière ne sont pas valides dans la notation N-triples. Ainsi, chaque fois qu'on verra la forme prefixe:URI dans un triplets, il faudra imaginer qu'elle ne fait que remplacer la forme correcte <URI>, où URI correspond à l'URI écrite au long sans pré_xe. Supposons maintenant que les pré_xes local: et prof: correspondent aux URI suivantes, respectivement :

```
http://www.polymtl.ca/Vocabulary#
http://www.polymtl.ca/Profs#
```

Notre exemple sera maintenant représenté de la manière suivante :

```
prof:MichelGagnon local:worksAt local :dgi .
prof:MichelGagnon local:hasName "Michel Gagnon" .
prof:MichelGagnon local:hasHomePage
<http://www.professeurs.polymtl.ca/michel.gagnon> .
```

Pour représenter un littéral simple, on l'écrit tout simplement en le mettant entre guillemets, comme dans l'exemple précédent, où "Michel Gagnon" est un littéral. Si le littéral a une étiquette de langue, on l'ajoute après le littéral, séparée par le symbole @. Ainsi le littéral "chien" en français serait représentée par l'expression suivante : "chien"@fr. Finalement, pour un littéral typé, on fait suivre le littéral par les symboles ^^ suivi du type. Nous avons déjà vu auparavant l'exemple du nombre 10, qui peut être représenté par le littéral "10"^^xsd:integer.

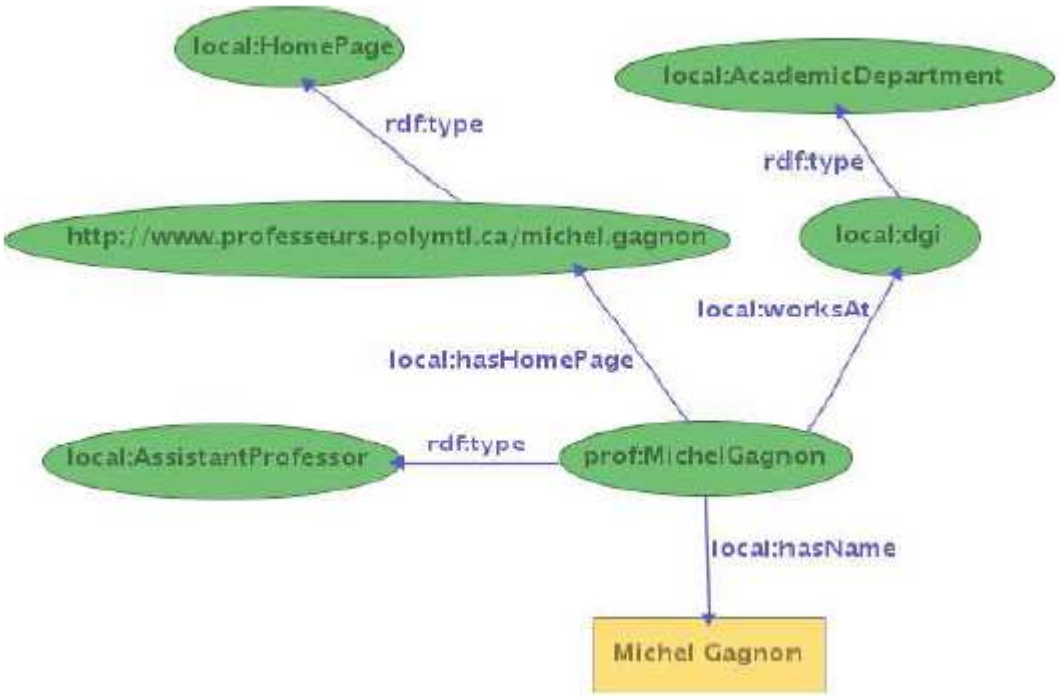
2.4 IDENTIFICATION DU TYPE DE RESSOURCES

Jusqu'à maintenant, nous nous sommes contentés de désigner des ressources et d'identifier des relations entre ces ressources. Mais il est important de noter que les ressources n'entrent pas toutes dans la même catégorie. Par exemple, un professeur, un département universitaire et une page personnelle sont des entités de types différents. Il serait donc intéressant de distinguer ces différents types. En RDF, il suffit tout simplement d'ajouter une propriété entre une ressource et son type. Pour ce faire la norme RDF dé_nit la propriété suivante :

```
http://www.w3.org/1999/02/22-rdf-syntax-ns#type
```

En utiliant l'alias rdf: pour désigner le pré_xe http://www.w3.org/1999/02/22-rdf-syntax-ns#, notre exemple peut être complété en indiquant les types des ressources, obtenant ainsi le graphe illustré à la figure 4. Dans la syntaxe RDF/XML le graphe sera représenté de la manière suivante :

```
<?xml version="1.0"?>
<rdf:RDF
xmlns:local="http://www.polymtl.ca/Vocabulary#"
xmlns:prof="http://www.polymtl.ca/Profs#"
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">
<rdf:Description rdf:about="http://www.polymtl.ca/Profs#MichelGagnon">
<rdf:type

rdf:resource="http://www.polymtl.ca/Vocabulary#AssistantProfessor"/>
<local:worksAt>
<rdf:Description rdf:about="http://www.polymtl.ca/Vocabulary#dgi">
<rdf:type
rdf:resource="http://www.polymtl.ca/Vocabulary#AcademicDepartment"/>
</rdf:Description>
</local:worksAt>
<local:hasName>Michel Gagnon</local:hasName>
<local:hasHomePage>
<rdf:Description
rdf:about="http://www.professeurs.polymtl.ca/michel.gagnon">
<rdf:type rdf:resource="http://www.polymtl.ca/Vocabulary#HomePage"/>
</rdf:Description>
</local:hasHomePage>
</rdf:Description>
</rdf:RDF>
```

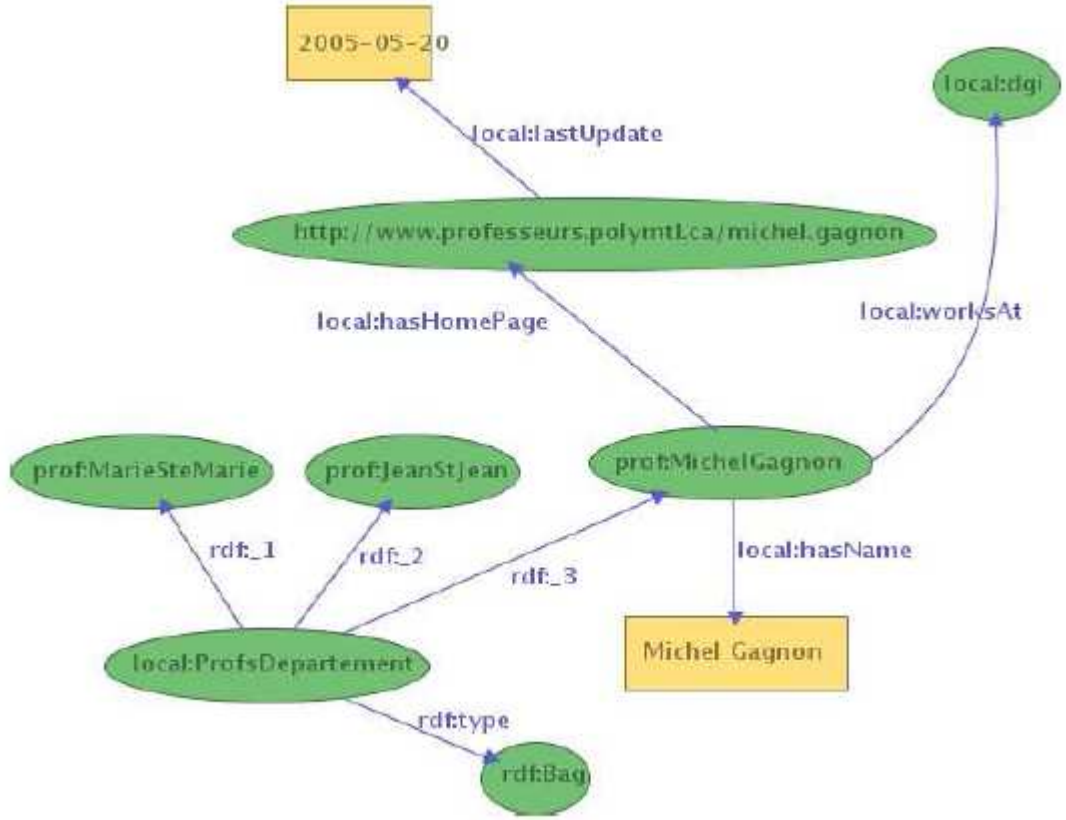


Exemple de graphe RDF avec type de ressource typé

2.5 LES CONTENEURS

Il n'est pas rare de vouloir représenter une ressource qui est en fait un conteneur, c'est-à-dire une ressource qui contient d'autres ressources. On n'a qu'à penser par exemple à un groupe de personnes. On aimerait donc pouvoir spécifier qu'une ressource est un conteneur et indiquer clairement les relations entre cette ressource et les entités qu'elle contient. RDF propose trois classes de conteneur : rdf:Bag, rdf:Seq et rdf:Alt. Le premier désigne un conteneur dont les membres n'ont aucun ordre entre eux, contrairement au second qui lui suppose l'existence d'un ordre. Le conteneur rdf:Alt désigne un conteneur présentant des alternatives parmi lesquelles on s'attend à ce qu'une seule soit sélectionnée. Le conteneur est relié à chacun des ses membres par une relation rdf:_n, où n est un entier. Il va de soi que dans le cas d'un conteneur de type rdf:Seq, on s'attend à ce que n représente l'ordre du membre en question.

Supposons maintenant que notre professeur soit membre d'un département qui contient trois professeurs.
La manière la plus naturelle de représenter cela est en utilisant un conteneur rdf:Bag, tel qu'illustré à la figure



exemple de conteneur

Pour simplifier l'écriture, RDF/XML fournit une abbréviation, en utilisant la relation rdf:li pour chaque membre, au lieu de la relation spécifique rdf:_n. Un telle notation suppose que les relations rdf:_1, rdf:_2, et ainsi de suite, sont générées automatiquement.

```
<?xml version="1.0"?>
<rdf:RDF
xmlns:prof="http://www.polymtl.ca/Profs#"
xmlns:local="http://www.polymtl.ca/Vocabulary#"
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">
<rdf:Description
rdf:about="http://www.professeurs.polymtl.ca/michel.gagnon">
<local:lastUpdate rdf:datatype="http://www.w3.org/2001/XMLSchema#date">
2005-05-20
</local:lastUpdate>
</rdf:Description>
<rdf:Description rdf:about="http://www.polymtl.ca/Profs#MichelGagnon">
<local:hasHomePage
rdf:resource="http://www.professeurs.polymtl.ca/michel.gagnon"/>
<local:worksAt rdf:resource="http://www.polymtl.ca/Vocabulary#dgi"/>
<local:hasName>Michel Gagnon</local:hasName>
</rdf:Description>
<rdf:Bag rdf:about="http://www.polymtl.ca/Vocabulary#ProfsDepartement">
<rdf:li rdf:resource="http://www.polymtl.ca/Profs#MarieSteMarie"/>
<rdf:li rdf:resource="http://www.polymtl.ca/Profs#JeanStJean"/>
```

```
<rdf:li rdf:resource="http://www.polymtl.ca/Profs#MichelGagnon"/>
</rdf:Bag>
</rdf:RDF>
```

2.6 LES COLLECTIONS

RDF permet de définir des collections. Les *collections*, qui sont en quelque sorte des listes. La liste vide est représentée par une ressource spéciale pré-dé_nie dont l'URI est rdf:nil. On construit une liste de manière récursive en utilisant le prédicat rdf:first pour indiquer le premier élément de la liste, et le prédicat rdf:rest pour indiquer le reste de la liste, qui est lui-même une liste.
Si, par exemple, on sait que les trois professeurs cités dans le modèle de la figure sont les seuls du département , on peut utiliser une liste au lieu d'un conteneur :

```
<?xml version="1.0"?>
<rdf:RDF
xmlns:prof="http://www.polymtl.ca/Profs#"
xmlns:local="http://www.polymtl.ca/Vocabulary#"
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">
<rdf:Description
rdf:about="http://www.professeurs.polymtl.ca/michel.gagnon">
<local:lastUpdate rdf:datatype="http://www.w3.org/2001/XMLSchema#date"
>2005-05-20</local:lastUpdate>
</rdf:Description>
<rdf:Description rdf:about="http://www.polymtl.ca/Profs#MichelGagnon">
<local:hasHomePage
rdf:resource="http://www.professeurs.polymtl.ca/michel.gagnon"/>
<local:worksAt rdf:resource="http://www.polymtl.ca/Vocabulary#dgi"/>
<local:hasName>Michel Gagnon</local:hasName>
</rdf:Description>
<rdf:List rdf:about="http://www.polymtl.ca/Vocabulary#ProfsDepartement">
<rdf:first rdf:resource="http://www.polymtl.ca/Profs#MarieSteMarie"/>
<rdf:rest>
<rdf:List>
<rdf:first rdf:resource="http://www.polymtl.ca/Profs#JeanStJean"/>
<rdf:rest>
<rdf:List>
<rdf:first rdf:resource="http://www.polymtl.ca/Profs#MichelGagnon"/>
<rdf:rest
rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#nil"/>
</rdf:List>
</rdf:rest>
</rdf:List>
</rdf:rest>
</rdf:List>
</rdf:RDF>
```

2.7 RDF SCHEMA

Nous avons vu, dans la section 2.5, que l'on peut utiliser la propriété rdf:type pour distinguer le type de chacune des ressources décrites. Mais que représente réellement la relation rdf:type? En d'autres mots, lorsqu'on applique la propriété rdf:type à une ressource, quel est le type de la ressource obtenue ?
Ou encore, dans la terminologie de RDF, quel est le type de l'objet dans un triplet dont le prédicat est rdf:type?

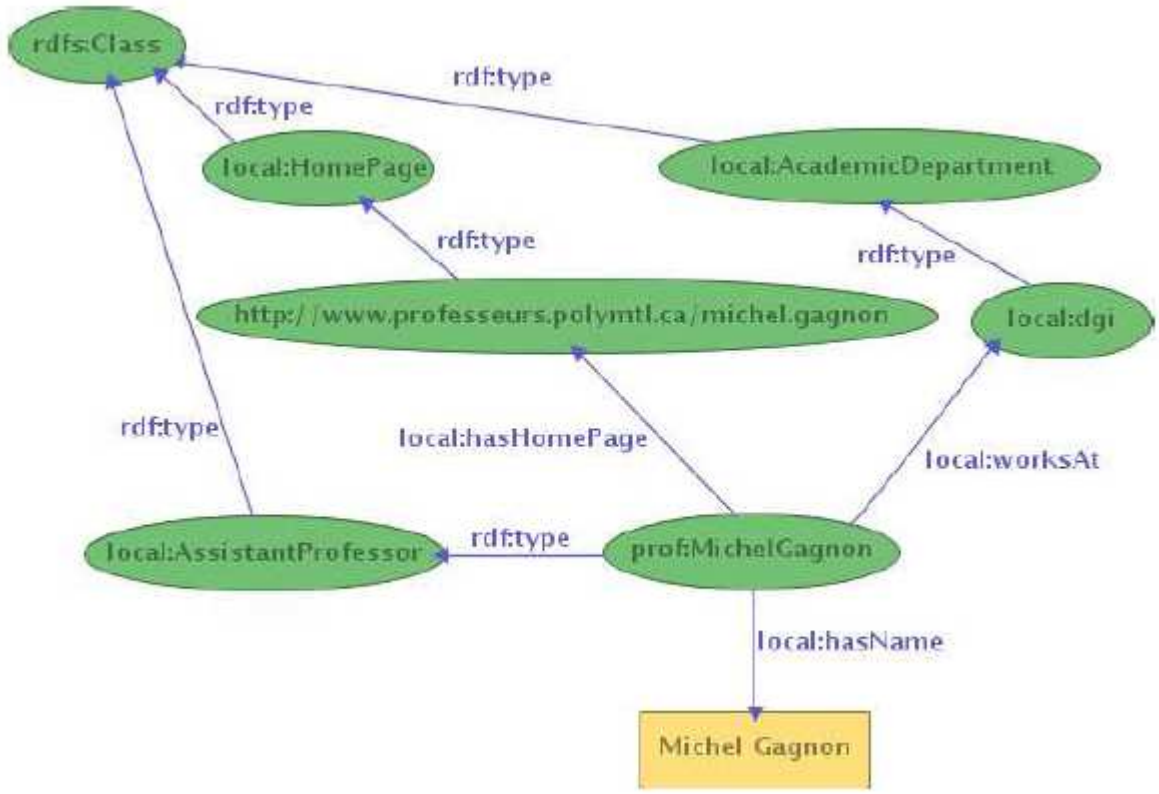
Il s'agit en fait d'un type de ressource spécial, puisqu'il ne s'agit pas d'une entité simple mais en fait d'une classe d'entités. Quand on dit qu'une ressource R et du type T, on se trouve à définir implicitement l'ensemble de toutes les ressources qui ont en commun d'avoir le type T. Pour exprimer ce genre de concepts et d'autres qui y sont reliés, un vocabulaire précis a été proposé. Pour des raisons historiques, ce vocabulaire a été dénommé RDF Schema, abbrégé RDFS. Comme cela porte à confusion avec XML Schema, qui n'a absolument rien à voir avec RDF Schema, on parle maintenant plutôt de RDF Vocabulary Language, même si le nom original est resté.
Le prefixe pour tous les éléments du vocabulaire RDFS est le suivant (dans la suite du texte, nous utiliserons l'alias rdfs: pour ce préfixe) :

```
http://www.w3.org/2000/01/rdf-schema#
```

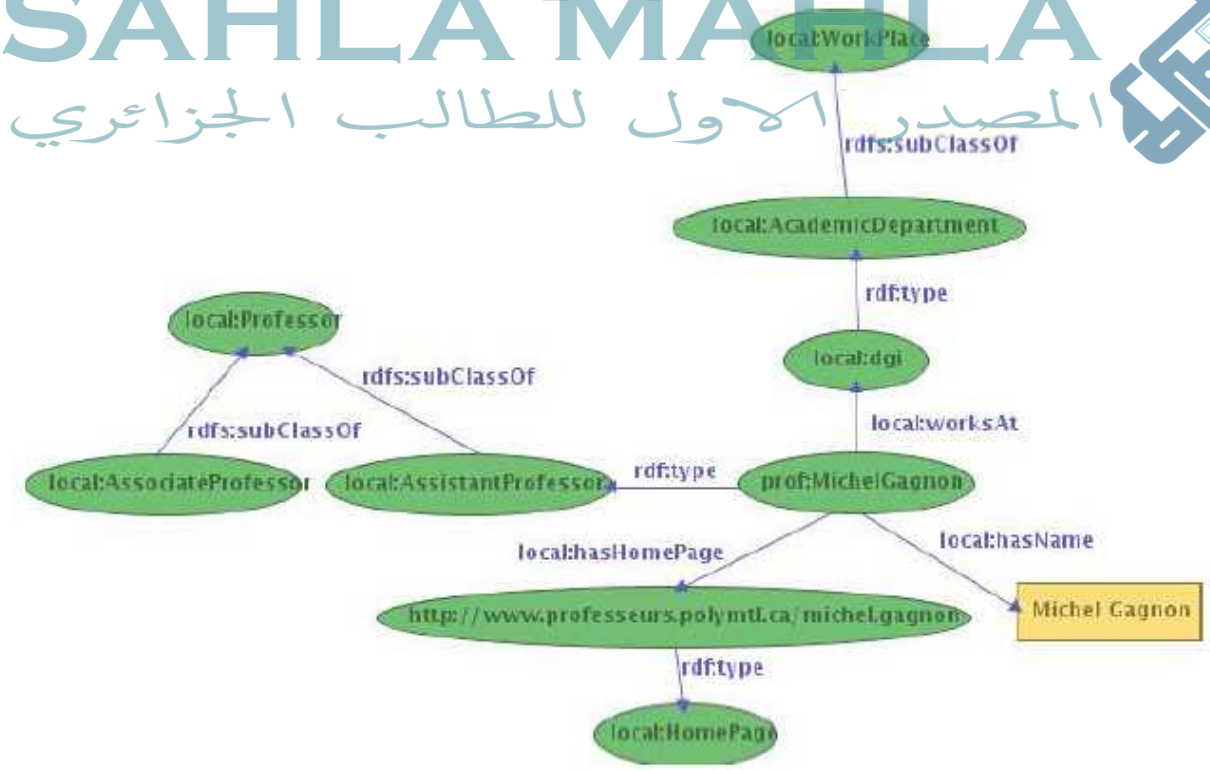
2.7.1 CLASSES

Le premier élément important de ce vocalulaire est rdfs:Class qui représente justement le type de la ressource obtenue comme valeur lorsque la propriété rdf:type est appliquée à une ressource. Ainsi, les ressources local:AcademicDepartment,local:HomePageet local:AssistantProfessor ont toutes la valeur rdfs:Class pour la propriété rdf:type, comme illustré à la figure .
Comme on l'a déjà vu plus tôt, une ressource peut appartenir à plus d'une classe. Par exemple, rien ne nous empêche d'ajouter un triplet indiquant que Michel Gagnon est aussi de type local:Humain (à noter que seul un des types peut être utilisé comme abdréviation à la place de la balise rdf:Description) :

```
<?xml version="1.0"?>
<rdf:RDF
xmlns:prof="http://www.polymtl.ca/Profs#"
xmlns:local="http://www.polymtl.ca/Vocabulary#"
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">
<rdfs:Class rdf:about="http://www.polymtl.ca/Vocabulary#HomePage"/>
<rdfs:Class rdf:about="http://www.polymtl.ca/Vocabulary#AssistantProfessor"/>
<rdfs:Class rdf:about="http://www.polymtl.ca/Vocabulary#AcademicDepartment"/>
<local:AcademicDepartment rdf:about="http://www.polymtl.ca/Vocabulary#dgi"/>
<local:HomePage rdf:about="http://www.professeurs.polymtl.ca/michel.gagnon"/>
<local:AssistantProfessor rdf:about="http://www.polymtl.ca/Profs#MichelGagnon">
<rdf:type rdf:resource="http://www.polymtl.ca/Vocabulary#Humain"/>
<local:hasHomePage
rdf:resource="http://www.professeurs.polymtl.ca/michel.gagnon"/>
<local:worksAt rdf:resource="http://www.polymtl.ca/Vocabulary#dgi"/>
<local:hasName>Michel Gagnon</local:hasName>
```



Les classes



Hiérarchies de classes

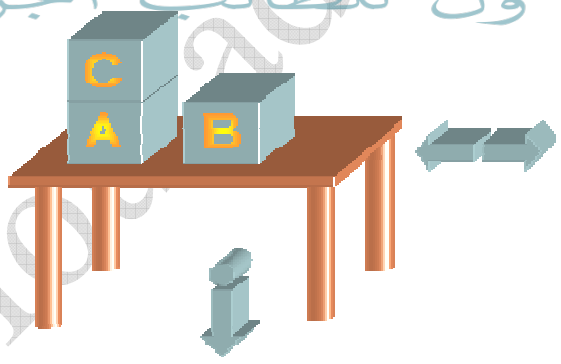
CHAPITRE III : Les ontologies

3.1 INTRODUCTION :

Le mot ontologie est construit à partir des racines grecques : *ontos* qui veut dire « ce qui existe », « l'existant », et *logos* pour « le discours », « l'étude ». En d'autres termes, Ontologie signifie l'étude de ce qui existe, la science de l'être.
Le terme ontologie recouvre deux usages dont le premier appartient à la philosophie classique et le second, plus récent, aux autres sciences cognitives.

L'objectif premier d'une ontologie est de modéliser un ensemble de connaissances dans un domaine donné, qui peut être réel ou imaginaire.
Les ontologies sont employées dans l'intelligence artificielle, le Web sémantique, le génie logiciel, l'informatique biomédicale et l'architecture de l'information comme une forme de représentation de la connaissance au sujet d'un monde ou d'une certaine partie de ce monde.

En informatique, une ontologie est une représentation de propriétés générales de ce qui existe dans un formalisme permettant un traitement automatique. C'est le résultat d'une formulation exhaustive et rigoureuse de la conceptualisation d'un domaine. Cette conceptualisation est souvent qualifiée de partielle car il est quasi-impossible de croire pouvoir capturer dans un formalisme toute la complexité d'un domaine. Notons aussi que le degré de formalisation d'une ontologie varie avec l'usage qui en est envisagé.




```
Cube (A)
Cube (B)
Cube (C)
Sur (A, Table)
Sur (C, A)
Sur (B, Table)
```

Cube (X) : L'entité X est parallélépipède à angles droits avec tous ses côtés de même longueur.
Table : Un objet global qui est un meuble composé d'une planche horizontale posée sur un ou plusieurs pieds.
Sur (Cube : X, Cube: Y / Table) : une relation énonçant qu'un cube X est sur un autre cube Y ou sur la table.

Figure Un exemple schématique d'ontologie : l'exemple des cubes.

Considérons l'exemple de la figure ci-dessus. On y voit une certaine « scène du monde ». La description de cette scène demande deux choses : (1) un vocabulaire non ambigu, aussi appelé

vocabulaire conceptuel ou ontologie ; (2) une énonciation des faits de la scène, reposant sur l'utilisation du vocabulaire de l'ontologie.

D'un point de vue pratique, une ontologie informatique permet, en particulier grâce aux travaux de l'intelligence artificielle sur les systèmes à base de connaissances et les moteurs d'inférence, d'implanter des mécanismes de raisonnement déductif, de classification automatique, de recherche d'information, …etc.

Plusieurs définitions existent des ontologies :

« *An ontology is an explicit specification of a conceptualization* ». (Gruber, 1993)
[Traduction: une ontologie est une spécification explicite d'une conceptualisation]. Il a introduit la notion de 'conceptualisation' qui réfère à un modèle abstrait d'un certain domaine du monde réel en identifiant les concepts pertinents décrivant ce domaine. Le terme 'explicite' signifie que les concepts utilisés ainsi que les contraintes sur leur emploi, sont réellement définis d'une manière claire et précise.

« *An ontology is a formal specification of a shared conceptualization* ».(Borst, 1997)
[Traduction: une ontologie est une spécification formelle d'une conceptualisation partagée]. Cette définition précise d'une part, le fait que l'ontologie doit être 'formelle', c'est à dire exprimée sous forme d'une logique pouvant être exploitable par une machine. D'autre part, elle doit être 'partagée' dans la mesure où elle doit capturer des connaissances partagées entre différents individus.

3.2 NOTIONS DE BASE :

Les ontologies rassemblent les connaissances propres à un domaine particulier. Ces connaissances sont formalisées en mettant en jeu les composants suivants: concepts ; relations ; axiomes et instances.

3.2.1 LE CONCEPT

un concept peut se définir comme une entité composée de trois éléments distincts :

- Le(s) terme(s) exprimant le concept en langue.
- La signification du concept, appelée également « notion » ou « intension » du concept.
- Le(s) objet(s) dénotés par le concept, appelé(s) également « réalisation » ou « extension » du concept.

Ces éléments sont habituellement présentés comme constituant les sommets d'un « triangle sémantique » (Ogden et Richards, 1923).

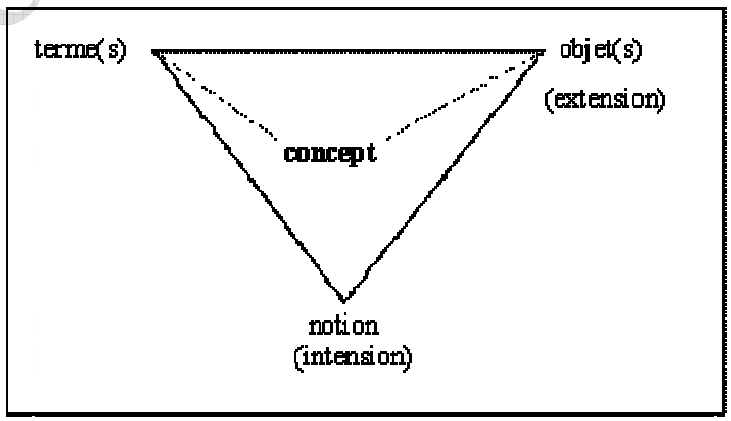


Fig – Le triangle sémantique

Prenons l'exemple du concept ÉTOILE :
- Son terme est le nom commun *étoile*.
- Sa notion *est d'être un point brillant dans le ciel, la nuit*.
- Son extension *est justement l'ensemble des points brillants* que l'on peut découvrir dans un ciel nocturne.

Le concept ÉTOILE a pour extension un ensemble d'objets. Nous le nommons pour cette raison concept générique.

A partir de la définition d'un concept, on peut dire sur la notion d'ontologie :

- Une ontologie est concernée en priorité par les *notions*, puisque l'objectif est de partager du sens. Il faut toutefois préserver un lien avec le(s) terme(s).
- Une ontologie concerne principalement *les notions* de concepts génériques qui ne dépendent pas de situations particulières du monde (l'équivalent d'une base de faits pour un SE) et sont donc plus susceptibles d'être partagées. On verra toutefois que les langages de représentation dédiés aux ontologies offrent la possibilité de représenter des concepts individuels.

1.4.2 RELATIONS

Pour présenter les relations entre concepts, examinons les définitions suivantes :

- Un « camion » est un véhicule automobile de grande taille transportant des marchandises.
- Un « cartable » est un sac d'écolier.
- Une « pneumonie » est une inflammation aiguë du poumon.
- Une « sole » est un poisson plat, ovale, qui vit couché sur les fonds sablonneux.

On peut observer dans la forme de ces définitions une régularité : le concept à définir (nous l'avons placé, plus exactement son terme (!), entre guillemets) est systématiquement défini par rapport à un autre concept (nous avons souligné son terme) en précisant une différence ; cette différence correspond à des propriétés vérifiées par les objets réalisant le nouveau concept (ex : CAMION) et que ne partagent pas les objets réalisant le concept de référence (ex : VÉHICULE AUTOMOBILE).
Ce mode de définition relève en fait d'une longue tradition, que l'on peut faire remonter à Porphyre (234-305, de notre ère). Pour classer l'ensemble des objets du monde en catégories abstraites, Porphyre avait adopté le mode de définition suivant : *nouveau genre = genre proche + différence*. L'arbre de Porphyre (cf. figure) peut être considéré comme un premier exemple d'ontologie.

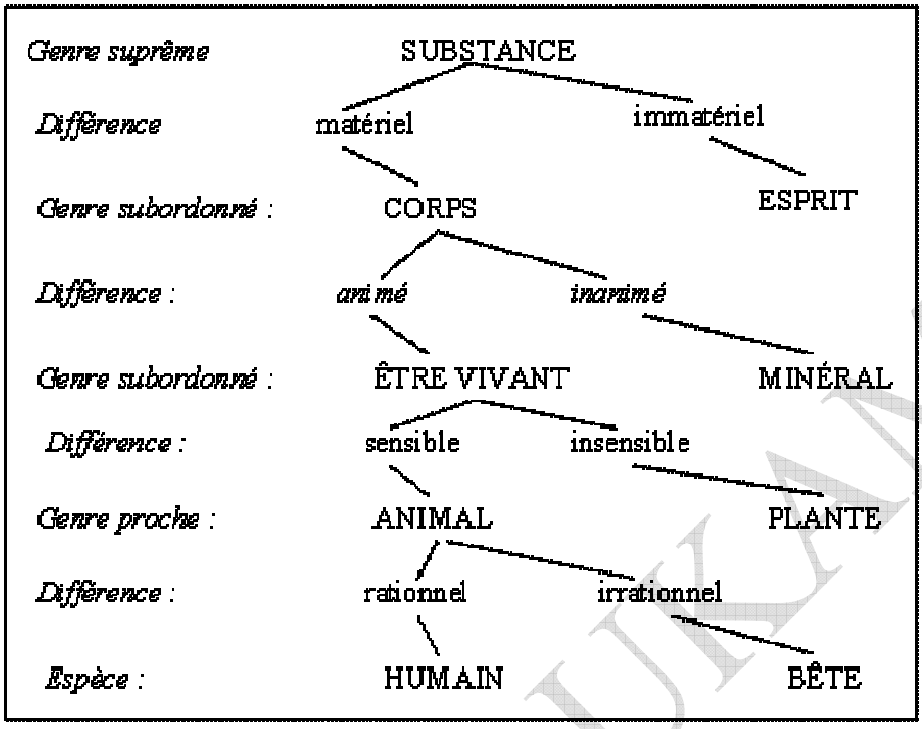


Fig – L'arbre de Porphyre

Définir un concept par rapport à un autre revient à introduire un ordre sur les concepts. Cette relation, exprimée en français par l'expression « *est un(e)* », est appelée relation de « *généralisation* », et son inverse relation de « *spécialisation* ». L'existence d'un tel lien entre concepts se justifie par l'existence d'une relation d'inclusion entre les extensions de ces concepts.

RELATION DE SUBSOMPTION

On dit qu'un concept CONCEPT1 **généralise** le concept CONCEPT2 (resp. le concept CONCEPT2 **spécialise** le concept CONCEPT1) si et seulement si l'extension du concept CONCEPT2 est incluse dans l'extension du concept CONCEPT1.

Dans l'arbre de Porphyre, le concept ÊTRE VIVANT généralise le concept ANIMAL, ce qui signifie que tout animal est un être vivant. Il faut noter qu'au fur et à mesure que l'on descend dans l'arbre, on ajoute des propriétés (les différences), ce qui a pour effet de réduire la taille des ensembles d'objets réalisant les concepts.

Le terme « subsomption » tend à remplacer le terme « généralisation » dans le domaine de l'ingénierie ontologique. On parlera ainsi de « liens de subsomption » et on dira, dans le cas de l'arbre de Porphyre, que le concept ÊTRE VIVANT « subsume » le concept ANIMAL, ou inversement, que le concept ANIMAL « est subsumé par » le concept ÊTRE VIVANT.

Dans les exemples que nous venons de voir, les notions sont exprimées au moyen de liens de généralisation/subsomption. Ces liens constituent le **principal mode de structuration d'une ontologie**. Pour autant, d'autres relations sont habituellement utilisées, comme le montrent les compléments de définition donnés ci-dessous (les termes exprimant les nouvelles relations sont soulignés) :

- Un « camion » est composé d'une cabine, d'essieux, *etc.*

- Un « cartable », généralement, est fabriqué en cuir.
- Une « pneumonie » est provoquée par une bactérie, appelée pneumocoque.
- Une « sole » possède une chair appréciée des gourmets.

3.3 CLASSIFICATION D'ONTOLOGIES :

Les ontologies peuvent être classifiées en fonction de plusieurs critères, à savoir :

- La richesse de la structure interne des ontologies ;
- L'objet de conceptualisation ;
- Le niveau de granularité ;
- Le niveau de formalisation de la représentation des connaissances.

Par exemple, selon le critère "objet de conceptualisation", les ontologies peuvent être :

- ontologies de représentation de connaissances ;
- ontologies générique / générale / commune ;
- ontologies de haut niveau / de niveau supérieur ;
- ontologies du domaine ;
- ontologies de tâche ;
- ontologies d'application.

3.4 DEMARCHE POUR LA CONSTRUCTION D'UNE ONTOLOGIE :

D'une manière générale, on peut dire que la démarche de construction d'une ontologie se décompose en plusieurs étapes :

1. Spécification
2. Acquisition de Connaissances
3. Formalisation
4. Évaluation
5. Documentation

La phase d'acquisition des connaissances est la plus longue et la plus fastidieuse.

1/Etape de **spécification** :

L'étape de spécification doit permettre de "cadrer" le domaine d'application de l'ontologie projeté en recensant toues les questions à prendre en charge lors de la conception. Les questions essentielles à poser sont :

- à quoi va servir l'ontologie?
    - Identifier le but de l'application
    - limiter le domaine
    - les objets
    - les usagers, les points de vues
    - les sources documentaires
- Trouver les questions auxquelles devra répondre l'ontologie.
- Trouver les scénarios d'utilisation des connaissances

2/ Etape de **l'acquisition des connaissances** :

C'est une étape où on doit faire des choix de conception et des choix techniques nécessaires pour la construction de l'ontologie. Les questions à trancher lors de cette étape sont :

- Réutiliser des ontologies existantes
    - top level ontology
    - Ontologie d'une partie du domaine ex. ontologie des unités
- Identifier les termes importants
    - normaliser le vocabulaire
- Identifier les concepts et les relations du domaine
    - Définition écrite en Langue Naturelle
    - Trouver les conditions minimales et suffisantes pour dire qu'un objet appartient à une classe donnée.

3/ Etape de **la formalisation :**

Il s'agit de formaliser la description de l'ontologie . Concrètement :

- Coder l'ontologie dans un langage formel en utilisant des outils: Protégé, Kaon, OntoEdit …
- Trouver les classes, les attributs, les types, les contraintes
- Peupler l'ontologie: instancier les classes

4/ Etape de **Validation** :

Il s'agit de vérifier la bonne construction de l'ontologie. Les principaux contrôles à faire :

- Valider la taxinomie :
    - Pas de cycle

- Toutes les instances d'une classe sont aussi les instances de la classe mère
- _ Hiérarchie homogène: pas de classe isolée.

– Tester l'application

5/ Etape de **documentation** :

Il s'agit de documenter le projet de l'ontologie, principalement il faut :

– Donner des explications

– Définition en langage naturel

– Lier les concepts aux sources dont sont issues les définitions

3.5 LE LANGAE OWL (WEB ONTOLOGY LANGUAGE) :

Le World Wide Web Consortium (W3C) a mis sur pieds, en Novembre 2001, le groupe de travail « WebOnt », chargé d'étudier la création d'un langage standard de manipulation d'ontologies web. Le premier Working Draft « OWL Web Ontology Language 1.0 Abstract Syntax » paraît en Juillet 2002 et, au final, OWL devient une Recommandation du W3C le 10 Février 2004.

3.5.1 OBJECTIFS ET MOTIVATION

OWL est, tout comme RDF, un langage XML profitant de l'universalité syntaxique de XML. Fondé sur la syntaxe de RDF/XML, OWL offre un moyen d'écrire des ontologies web. OWL se différencie du couple RDF/RDFS en ceci que, contrairement à RDF, il est justement un langage d'ontologies. SI RDF et RDFS apportent à l'utilisateur la capacité de décrire des classes (ie. avec des constructeurs) et des propriétés, OWL intègre, en plus, des outils de comparaison des propriétés et des classes : identité, équivalence, contraire, cardinalité, symétrie, transitivité, disjonction, etc.

Ainsi, OWL offre aux machines une plus grande capacité d'interprétation du contenu web que RDF et RDFS, grace à un vocabulaire plus large et à une vraie sémantique formelle.

3.5.2 DIFFERENTES DECLINAISONS DE OWL

Plus un outil est complet, plus il est, en général, complexe. C'est ce qu'a voulu éviter le groupe de travail WebOnt du W3C en dotant OWL de trois sous-langages offrant des capacités d'expression croissantes et, naturellement, destinés à des communautés différentes d'utilisateurs :

• OWL Lite est le sous langage de OWL le plus simple. Il est destiné aux utilisateurs qui ont besoin d'une hiérarchie de concepts simple.

• OWL DL est plus complexe que OWL Lite, permettant une expressivité bien plus importante. OWL DL est fondé sur la logique descriptive (d'où son nom, OWL Description Logics), un domaine de recherche étudiant la logique, et conférant donc à OWL DL son adaptation au raisonnement automatisé. Malgré sa complexité relative face à OWL Lite, OWL-DL garantit la complétude des raisonnements (toutes les inférences sont calculables) et leur décidabilité (leur calcul se fait en une durée finie).

• OWL Full est la version la plus complexe d'OWL, mais également celle qui permet le plus haut niveau d'expressivité. OWL Full est destiné aux situations où il est plus important d'avoir un haut niveau de capacité de description.

Il existe entre ces trois sous langage une dépendance de nature hiérarchique : toute ontologie OWL Lite valide est également une ontologie OWL DL valide, et toute ontologie OWL DL valide est également une ontologie OWL Full valide.

3.5.3 STRUCTURE D'UNE ONTOLOGIE OWL

Tout comme RDF dispose d'une structure logique de type RDF/XML est la sérialisation, les ontologies OWL se présentent sous forme de fichiers texte, de « documents OWL ». (omme suffixe de nom de fichier, les extensions « .rdf » ou « .owl »).

il est bien nécessaire de comprendre qu'OWL n'a pas été conçu dans un esprit fermé permettant de borner les frontières d'une ontologie. Au contraire, la conception d'OWL a pris en compte la nature distribuée du web sémantique et, de ce fait, a intégré la possibilité d'étendre des ontologies existantes, ou d'employer diverses ontologies existantes pour compléter la définition d'une nouvelle ontologie.

Espaces de nommage

Afin de pouvoir employer des termes dans une ontologie, il est nécessaire d'indiquer avec précision de quels vocabulaires ces termes proviennent. C'est la raison pour laquelle, comme tout autre document XML, une ontologie commence par une déclaration d'espace de nom (parfois appelée « de nommage ») contenue dans une balise rdf:RDF. Supposons que nous souhaitons écrire une ontologie sur une population de personnes ou, d'une manière plus générale, sur l'humanite. Voici la déclaration d'espace de nom qui pourrait être employée :

```
<rdf:RDF
xmlns = "http://domain.tld/path/humanite#"
xmlns:humanite= "http://domain.tld/path/humanite#"
xmlns:base = "http://domain.tld/path/humanite#"
xmlns:vivant = "http://otherdomain.tld/otherpath/vivant#"
xmlns:owl = "http://www.w3.org/2002/07/owl#"
xmlns:rdf = "http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:rdfs = "http://www.w3.org/2000/01/rdf-schema#"
xmlns:xsd = "http://www.w3.org/2001/XMLSchema#">
```

Les deux premières déclarations identifient l'espace de nommage propre à l'ontologie que nous sommes en train d'écrire. La première déclaration d'espace de nom indique à quelle ontologie se rapporter en cas d'utilisation de noms sans préfixe dans la suite de l'ontologie.

La troisième déclaration identifie l'URI de base de l'ontologie courante.

La quatrième déclaration signifie simplement que, au cours de la rédaction de l'ontologie humanité, on va employer des concepts développés dans une ontologie vivant, qui décrit ce qu'est un être vivant. Les quatre dernières déclaration introduisent le vocabulaire d'OWL et les objets définis dans l'espace de nommage de RDF, du schéma RDF et des types de données du Schéma XML. Afin de simplifier l'écriture des URI dans la déclaration d'espace de nom et, surtout, dans les valeurs des attributs de l'ontologie, il est conseillé de définir des abréviations au moyen d'entités de type de document :

```
<!DOCTYPE rdf:RDF [
<!ENTITY humanite "http://domain.tld/path/humanite#" >
<!ENTITY vivant "http://otherdomain.tld/otherpath/vivant#" >
]>
```

Les deux premières déclarations identifient l'espace de nommage propre à l'ontologie que nous sommes en train d'écrire. La première déclaration d'espace de nom indique à quelle ontologie se

rapporter en cas d'utilisation de noms sans préfixe dans la suite de l'ontologie. La troisième déclaration identifie l'URI de base de l'ontologie courante.

Ainsi, la déclaration d'espace de nom initiale devient :

```
<rdf:RDF
xmlns = "&humanite;"
xmlns:humanite= "&humanite;"
xmlns:base = "&humanite;"
xmlns:vivant = "&vivant;"
xmlns:owl = "http://www.w3.org/2002/07/owl#"
xmlns:rdf = "http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:rdfs = "http://www.w3.org/2000/01/rdf-schema#"
xmlns:xsd = "http://www.w3.org/2001/XMLSchema#">
```

Enfin , tout comme il existe une section d'en-tête <head>..</head> en haut de tout document XHTML bien formé, on peut écrire, à la suite de la déclaration d'espaces de nom, un en-tête décrivant le contenu de l'ontologie courante. C'est la balise owl:Ontology qui permet d'indiquer ces informations :

```
<owl:Ontology rdf:about="">
<rdfs:comment>Ontologie décrivant l'humanité</rdfs:comment>
<owl:imports
rdf:resource="http://otherdomain.tld/otherpath/vivant"/>
<rdfs:label>Ontologie sur l'humanité</rdfs:label>
```

3.5.4 ELEMENT DU LANGAGE OWL

Les classes

Une classe définit un groupe d'individus qui sont réunis parce qu'ils ont des caractéristiques similaires. L'ensemble des individus d'une classe est désigné par le terme « extension de classe », chacun de ces individus étant alors une « instance » de la classe. Les trois versions d'OWL comportent les mêmes mécanismes de classe, à ceci près que OWL FULL est la seule version à permettre qu'une classe soit l'instance d'une autre classe (d'une métaclasse). A l'inverse, OWL Lite et OWL DL n'autorisent pas qu'une instance de classe soit elle-même une classe.

La déclaration d'une classe se fait de différentes manières , dont la création par le mot clé "Class" . Exemple : Une classe « humain » se déclare de la manière suivante :

```
<owl:Class rdf:ID="Humain" />
```

L'héritage

Il existe dans toute ontologie OWL une superclasse, nommée Thing, dont toutes les autres classes sont des sous-classes. Ceci nous amène directement au concept d'héritage, disponible à l'aide de la propriété subClassOf :

```
<owl:Class rdf:ID="Humain">
<rdfs:subClassOf rdf:resource="&etre;#EtreVivant" />
</owl:Class>
<owl:Class rdf:ID="Homme">
<rdfs:subClassOf rdf:resource="#Humain" />
</owl:Class>
<owl:Class rdf:ID="Femme">
<rdfs:subClassOf rdf:resource="#Humain" />
```

```
</owl:Class>
```

Enfin, il existe également une classe nommée noThing, qui est sous-classe de toutes les classes OWL. Cette classe ne peut avoir aucune instance.

Les instances de classe

Définition d'un individu

La définition d'un individu consiste à énoncer un « fait », encore appelé « axiome d'individu ».

On peut distinguer deux types de faits :

1/ les faits concernant **l'appartenance à une classe**

La plupart des faits concerne généralement la déclaration de l'appartenance à une classe d'in individu et les valeurs de propriété de cet individu. Un fait s'exprime de la manière suivante :

```
<Humain rdf:ID="Pierre">
        <aPourPere rdf:resource="#Jacques" />
        <aPourFrere rdf:resource="#Paul" />
</Humain>
```

Le fait écrit dans cet exemple exprime l'existence d'un Humain nommé « Pierre » dont le père s'appelle « Jacques », et qu'il a un frère nommé « Paul ».

On peut également instancier un individu anonyme en omettant son identifiant :

```
<Humain>
        <aPourPere rdf:resource="#Jacques" />
        <aPourFrere rdf:resource="#Paul" />
</Humain>
```

Ce fait décrit, dans ce cas, l'existence d'un Humain dont le père se nomme « Jacques » et qui a un frère nommé « Paul ».

2/ les faits concernant **l'identité des individus**

Une difficulté qui peut éventuellement apparaître dans le nommage des individus concerne la non-unicité éventuelle des noms attribués aux individus. Par exemple, un même individu pourrait être désigné de plusieurs façons différentes.

C'est la raison pour laquelle OWL propose un mécanisme permettant de lever cette ambiguité, à l'aide des propriétés owl:sameAs, owl:diffrentFrom et owl:allDifferent. L'exemple suivant permet de déclarer que les noms « Louis_XIV » et « Le_Roi_Soleil » désignent la même personne :

```
<rdf:Description rdf:about="#Louis_XIV">
        <owl:sameAs rdf:resource="#Le_Roi_Soleil" />
</rdf:Description>
```

Une fois que l'on sait écrire une classe en OWL, la création d'une ontologie se fait par l'écriture d'instances de ces objets, et la description des relations qui lient ces instances.

Les propriétés

Maintenant que l'on sait écrire des classes OWL, il ne manque plus que la capacité à exprimer des faits au sujet de ces classes et de leurs instances. C'est ce que permettent de faire les propriétés OWL. OWL fait la distinction entre deux types de propriétés :

• les propriétés d'objet permettent de relier des instances à d'autres instances

• les propriétés de type de donnée permettent de relier des individus à des valeurs de données.

Une propriété d'objet est une instance de la classe owl:ObjectProperty, une propriété de type de donnée étant une instance de la classe owl:DatatypeProperty. Ces deux classes sont elles-même sous-classes de la classe RDF rdf:Property.

La définition des caractéristiques d'une propriété se fait à l'aide d'un axiome de propriété qui, dans sa forme minimale, ne fait qu'affirmer l'existence de la propriété :

<owl:ObjectProperty rdf:ID="aPourParent" />

Cependant, il est possible de définir beaucoup d'autres caractéristiques d'une propriété dans un axiome de propriété.

Définition d'une propriété

Afin de spécifier une propriété, il existe différentes manières de restreindre la relation qu'elle symbolise. Par exemple, si on considère que l'existence d'une propriété pour un individu donné de l'ontologie constitue une fonction faisant correspondre à cet individu un autre individu ou une valeur de donnée, alors on peut préciser le domaine et l'image de la propriété. Une propriété peut également être définie comme la spécialisation d'une autre propriété.

```
<owl:ObjectProperty rdf:ID="habite">
        <rdfs:domain rdf:resource="#Humain" />
        <rdfs:range rdf:resource="#Pays" />
</owl:ObjectProperty>
```

Dans l'exemple ci-dessus, on apprend que la propriété habite a pour domaine la classe Humain et pour image la classe Pays : elle relie des instances de la classe Humain à des instances de la classe Pays. Dans le cas d'une propriété de type de donnée, l'image de la propriété peut être un type de donnée, comme défini dans le Schéma XML. Par exemple, on peut définir la propriété de type de données anneeDeNaissance :

```
<owl:Class rdf:ID="dateDeNaissance" />
        <owl:DatatypeProperty rdf:ID="anneeDeNaissance">
        <rdfs:domain rdf:resource="#dateDeNaissance" />
        <rdfs:range rdf:resource="&xsd;positiveInteger"/>
</owl:DatatypeProperty>
```

Dans ce cas, anneDeNaissance fait correspondre aux instances de la classe de dateDeNaissance des entiers positifs.

On peut également employer un mécanisme de hiérarchie entre les propriétés, exactement comme il existe un mécanisme d'héritage sur les classes :

```
<owl:Class rdf:ID="Humain" />
<owl:ObjectProperty rdf:ID="estDeLaFamilleDe">
<rdfs:domain rdf:resource="#Humain" />
<rdfs:range rdf:resource="#Humain" />
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="aPourFrere">
```

```
<rdfs:subPropertyOf rdf:resource="#estDeLaFamilleDe" />
<rdfs:range rdf:resource="#Humain" />
...
</owl:ObjectProperty>
</owl:Class>
```

La propriété aPourFrere est une sous-propriété de estDeLaFamilleDe, ce qui signifie que toute entité ayant une propriété aPourFrere d'une certaine valeur a aussi une propriété estDeLaFamilleDe de même valeur.

Caractéristiques des propriétés

En plus de ce mécanisme d'héritage et de restriction du domaine et de l'image d'une propriété, il existe divers moyens d'attacher des caractéristiques aux propriétés, ce qui permet d'affiner grandement la qualité des raisonnements liés à cette propriété.

Parmi les caractéristiques de propriétés principales, on trouve la transitivité, la symétrie, la fonctionnalité, l'inverse, etc. L'ajout d'une caractéristique à une propriété de l'ontologie se fait par l'emploi de la balise OWL type :

Exemple de propriété non symétrique :

```
<owl:ObjectProperty rdf:ID="aPourPere">
<rdfs:domain rdf:resource="#Humain" />
<rdfs:range rdf:resource="#Humain" />
</owl:ObjectProperty>
```

Exemple de propriété non symétrique :

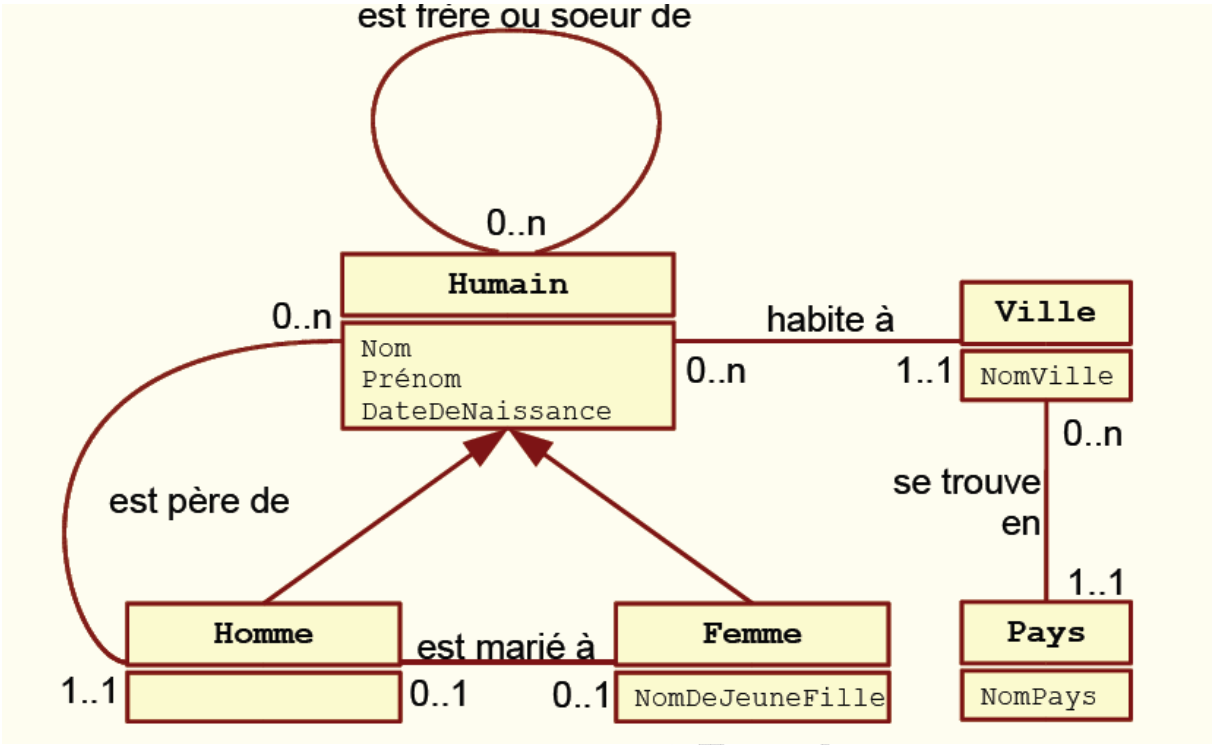
```
<owl:ObjectProperty rdf:ID="aPourFrere">
<rdf:type rdf:resource="&owl;SymmetricProperty" />
<rdfs:domain rdf:resource="#Humain" />
<rdfs:range rdf:resource="#Humain" />
</owl:ObjectProperty>

<Humain rdf:ID="Pierre">
<aPourFrere rdf:resource="#Paul" />
<aPourPere rdf:resource="#Jacques" />
</Humain>
```

L'Humain Pierre a pour frère Paul, de même que (symétrie) l'Humain Paul a pour frère Pierre. Par contre, si Pierre a pour père Jacques, l'inverse n'est pas vrai (aPourPere n'est pas symétrique).

3.6 EXEMPLE D'ONTOLOGIE :

L'ontologie OWL présentée dans ce paragraphe décrit un groupe de personnes et leurs relations de parenté, ainsi que leur lieu d'habitation.

Le diagramme de classes UML de la figure suivante indique les classes qui composent la population dont on souhaite écrire une ontologie OWL :



La population que l'on souhaite décrire est composée d'humains, divisés en deux sous-classes Homme et Femme, et habitant dans une certaine ville. Un humain peut avoir un lien de fraternité avec un autre humain. Un homme peut être père d'un autre humain, et un homme et une femme peuvent être mariés.

Le schéma suivant indique plus précisément les individus qui existent dans l'ontologie que nous allons écrire :

Le monde que l'on va décrire comporte trois instances de la classe « Homme » (en vert), une instance de la classe « Femme » (en rouge), ainsi que deux instances de « Ville » (en bleu) et de « Pays » (en mauve).

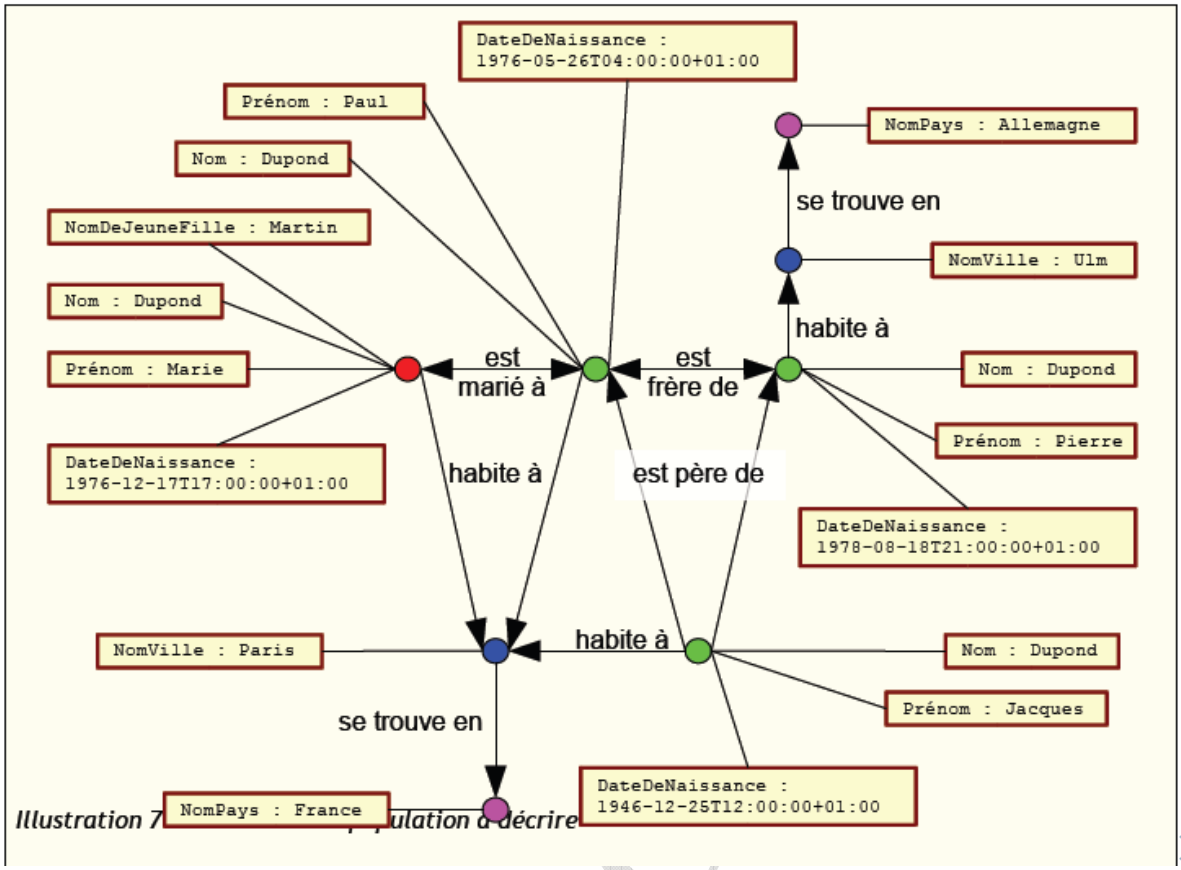


Illustration 7 [illegible] ulation à décrire

Ecriture des classes

La première étape de l'écriture de l'ontologie OWL représentant cette population consiste à écrire les classes du monde. Il est possible de définir avec plus ou moins de précision la nature des éléments qui composent le monde que l'on décrit, en créant plus ou moins de classes. Par exemple, il serait possible d'écrire une classe « date » dont seraient instances toutes les dates de l'ontologie. Cela pourrait avoir un interêt dans le cadre d'une population plus vaste, pour rechercher, par exemple, les personnes nées un même jour.

```
<!-- Défintion des classes -->
<owl:Class rdf:ID="Humain" />
<owl:Class rdf:ID="Homme">
<rdfs:subClassOf rdf:resource="#Humain" />
</owl:Class>
<owl:Class rdf:ID="Femme">
<rdfs:subClassOf rdf:resource="#Humain" />
</owl:Class>
<owl:Class rdf:ID="Ville" />
<owl:Class rdf:ID="Pays" />
```

Cette première définition des classes de l'ontologie est satisfaisante, mais elle est néanmoins perfectible. En effet, on peut ajouter des contraintes, notamment sur les classes Humain et Ville :

• un humain a toujours un père.

• Une ville se trouve forcément dans un pays.

D'autres contraintes sont imaginables, notamment les contraintes d'unicité des noms, dates de naissance, etc. Ces contraintes sont consultables dans le code complet de l'ontologie OWL, consultable en Annexe de ce document.

Voici la nouvelle écriture, plus détaillée, des deux classes « Humain » et « Ville » :

```
<owl:Class rdf:ID="Humain">
        <rdfs:subClassOf>
                <owl:Restriction>
                        <owl:onProperty rdf:resource="#aPourPere" />
                        <owl:cardinality
                        rdf:datatype="&xsd;nonNegativeInteger">1</owl:cardinality>
                </owl:Restriction>
        </rdfs:subClassOf>
</owl:Class>

<owl:Class rdf:ID="Ville">
        <rdfs:subClassOf>
                <owl:Restriction>
                        <owl:onProperty rdf:resource="#seTrouveEn" />
                        <owl:cardinality
                        rdf:datatype="&xsd;nonNegativeInteger">1</owl:cardinality>
                </owl:Restriction>
        </rdfs:subClassOf>
</owl:Class>
```

Ecriture des propriétés

L'écriture des propriétés est l'étape qui va permettre de détailler la population que l'on veut décrire. Ecrivons tout d'abord les propriétés d'objet : habiteA, aPourPere, aUnLienDeFraternite, estMarieA et seTrouveEn :


```
<!-- Propriétés d'objet -->
        <owl:ObjectProperty rdf:ID="habiteA">
        <rdfs:domain rdf:resource="#Humain" />
        <rdfs:range rdf:resource="#Ville" />
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="aPourPere">
        <rdfs:domain rdf:resource="#Humain" />
        <rdfs:range rdf:resource="#Homme" />
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="aUnLienDeFraternite">
        <rdf:type rdf:resource="&owl;SymmetricProperty" />
        <rdfs:domain rdf:resource="#Humain" />
        <rdfs:range rdf:resource="#Humain" />
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="estMarieA">
        <rdf:type rdf:resource="&owl;SymmetricProperty" />
        <rdfs:domain rdf:resource="#Humain" />
        <rdfs:range rdf:resource="#Humain" />
```

```
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="seTrouveEn">
        <rdfs:domain rdf:resource="#Ville" />
        <rdfs:range rdf:resource="#Pays" />

</owl:ObjectProperty>
```

Les propriétés aUnLienDeFraternite et estMarieA sont définies comme des propriétés symétriques : si une telle relation lie un individu A à un individu B, alors la même relation lie également l'individu B à l'individu A.

Passons maintenant aux propriétés de type de donnée. Ce sont ces propriétés qui, concrètement, permettent d'affecter des propriétés quantifiées aux instances des classes (par exemple, un nom, un prénom, une date de naissance, etc.) :

```
<!-- Propriétés de type de donnée -->
<owl:DatatypeProperty rdf:ID="nom">
        <rdfs:domain rdf:resource="#Humain" />
        <rdfs:range rdf:resource="&xsd;string" />
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="prenom">
        <rdfs:domain rdf:resource="#Humain" />
        <rdfs:range rdf:resource="&xsd;string" />
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="nomDeJeuneFille">
        <rdfs:domain rdf:resource="#Femme" />
        <rdfs:range rdf:resource="&xsd;string" />
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="dateDeNaissance">
        <rdfs:domain rdf:resource="#Humain" />
        <rdfs:range rdf:resource="&xsd;date" />

</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:ID="nomVille">
        <rdfs:domain rdf:resource="#Ville" />
        <rdfs:range rdf:resource="&xsd;string" />
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="nomPays">
        <rdfs:domain rdf:resource="#Pays" />
        <rdfs:range rdf:resource="&xsd;string" />

</owl:DatatypeProperty>
```


Assertion de faits caractéristisant la population

La dernière étape concerne l'assertion des faits caractérisant la population, présentés sur la figure « Individus de la population à décrire ». Il s'agit donc non seulement de l'instanciation des individus de la population, mais également de leur description par l'énonciation de leurs propriétés :

```
<!-- Humains -->
        <Homme rdf:ID="unknown">
        <aPourPere rdf:resource="#unknown" />
</Homme>
<Homme rdf:ID="Pierre">
        <nom>Dupond</nom>
        <prenom>Pierre</prenom>
        <dateDeNaissance>1978-08-18</dateDeNaissance>
        <aUnLienDeFraternite rdf:resource="#Paul" />
        <aPourPere rdf:resource="#Jacques" />
        <habiteA rdf:resource="#Ulm" />
</Homme>
<Homme rdf:ID="Paul">
        <nom>Dupond</nom>
        <prenom>Paul</prenom>
        <dateDeNaissance>1976-05-26</dateDeNaissance>
        <estMarieA rdf:resource="#Marie" />
        <aPourPere rdf:resource="#Jacques" />
        <habiteA rdf:resource="#Paris" />
</Homme>
<Homme rdf:ID="Jacques">
        <nom>Dupond</nom>
        <prenom>Jacques</prenom>
        <dateDeNaissance>1946-12-25</dateDeNaissance>
        <aPourPere rdf:resource="#unknown" />
        <habiteA rdf:resource="#Paris" />
</Homme>
<Femme rdf:ID="Marie">
        <nom>Dupond</nom>
        <prenom>Marie</prenom>
        <dateDeNaissance>1976-12-17</dateDeNaissance>
        <nomDeJeuneFille>Martin</nomDeJeuneFille>
        <aPourPere rdf:resource="#unknown" />
        <habiteA rdf:resource="#Paris" />
</Femme>

<!-- Pays -->
<Pays rdf:ID="Allemagne">
        <nomPays>Allemagne</nomPays>
        </Pays>
        <Pays rdf:ID="France">
        <nomPays>France</nomPays>
</Pays>

<!-- Villes -->
<Ville rdf:ID="Paris">
        <nomVille>Paris</nomVille>
        <seTrouveEn rdf:resource="#France" />
</Ville>

<Ville rdf:ID="Ulm">
        <nomVille>Ulm</nomVille>
        <seTrouveEn rdf:resource="#Allemagne" />
```

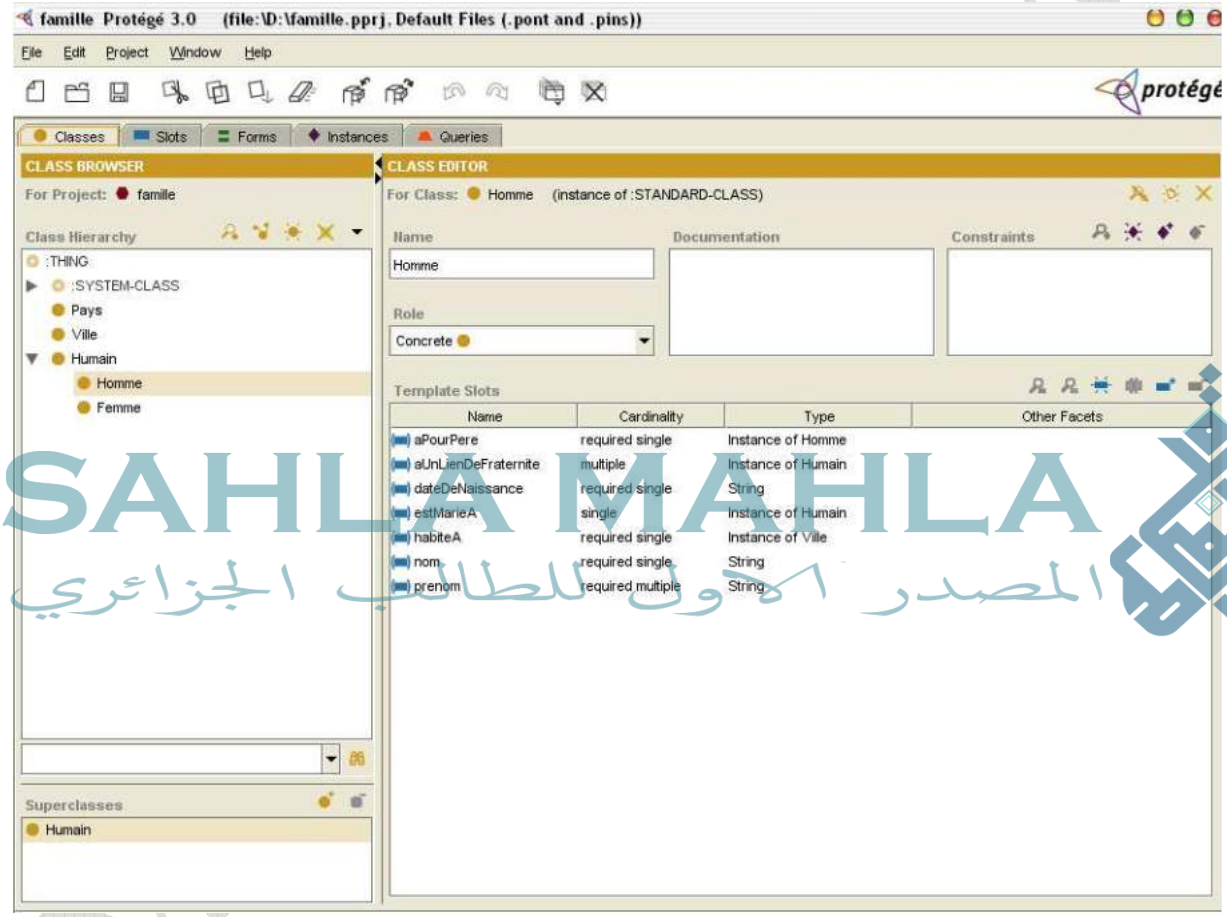
</Ville>

3.7 LES EDITEURS D'ONTOLOGIES :

3.7.1 L'EDITEUR "PROTEGE"

Protégé est un éditeur d'ontologies distribué en open source par l'université en informatique médicale de Stanford.

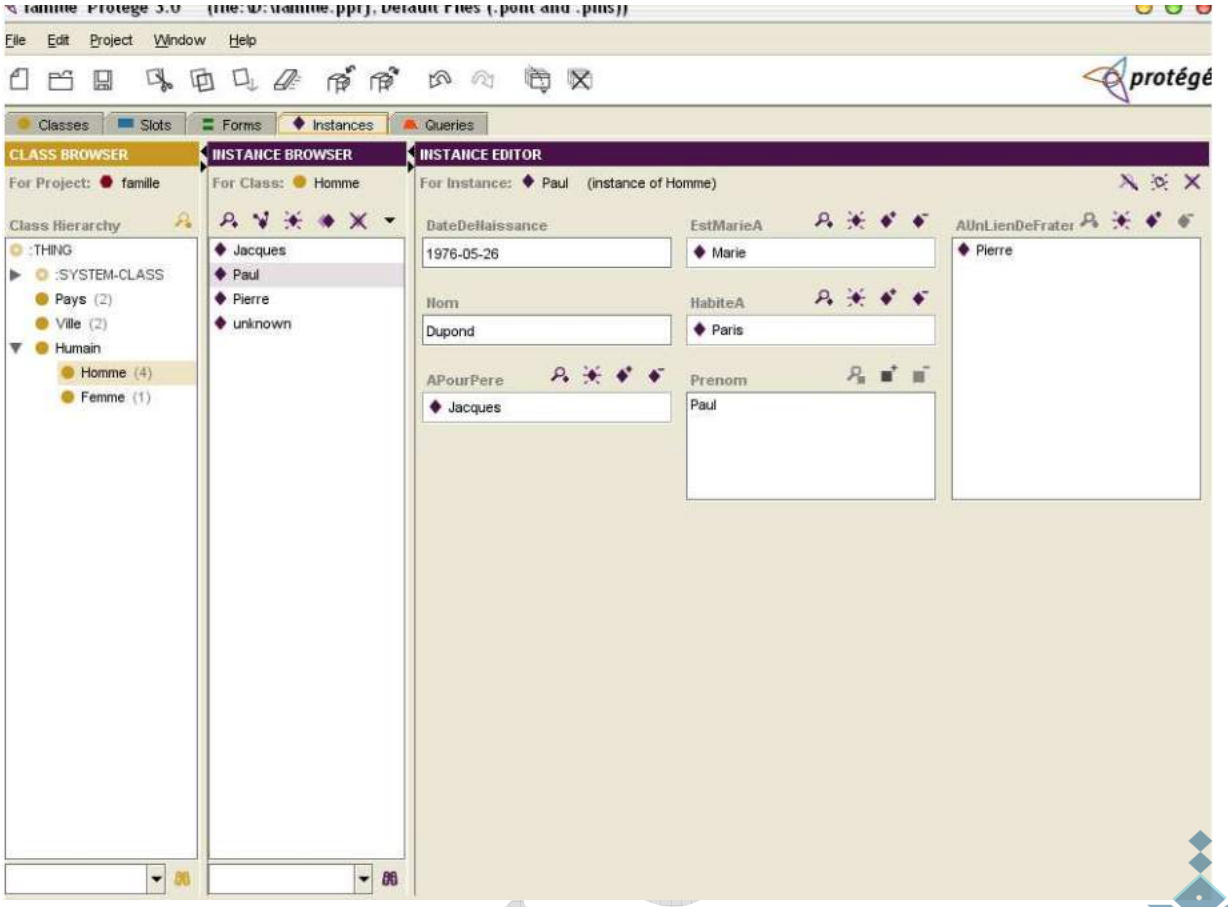
L'interface de Protégé est assez simple, l'ensemble des fonctionnalités de l'éditeur étant regroupé en cinq onglets. Le premier onglet présente les classes de l'ontologie. Il est possible de créer, modifier, supprimer une classe, et de lui attacher des propriétés. Ces propriétés peuvent elles-même être caractérisées.



Désignation des classes

Gestion des instances de classe et de leurs propriétés

L'onglet « Instances » permet de créer des instances et de leur affecter des propriétés, conformément à la définition des classes et des propriétés effectuée dans l'onglet « Classes » :



Création d'instance

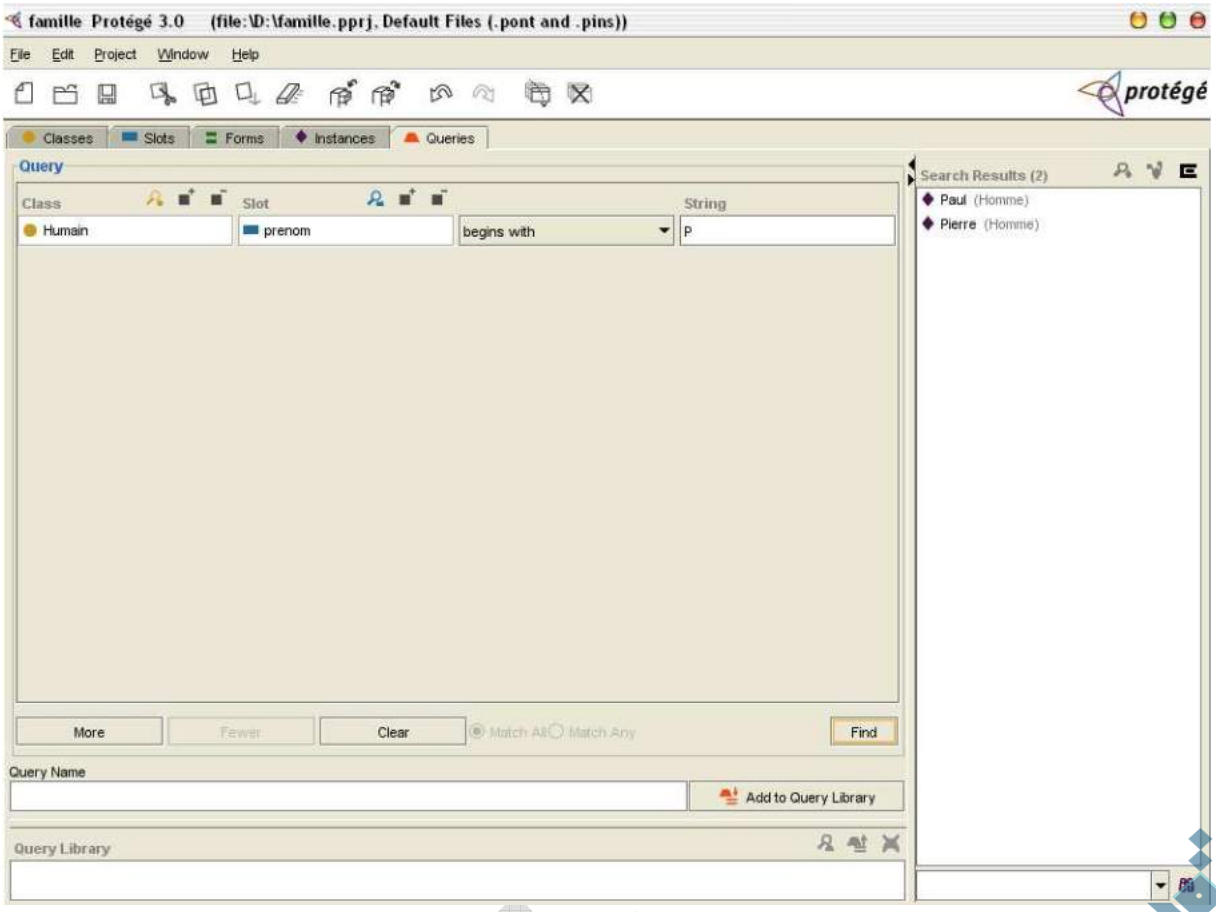
Sur l'écran présenté, il est par exemple possible d'éditer les informations concernant l'individu « Paul ». Il est important de comprendre que, dans l'Instance Browser, un individu est désigné par son identifiant (son « rdf:ID »), et non par son prénom.

Possibilité d'effectuer des requêtes

Enfin, une fonctionnalité intéressante de Protégé concerne la possibilité d'effectuer des requêtes sur l'ontologie en cours d'édition. La capture d'écran de la figure suivante :

Gestionnaire de requêtes de Protégé » présente une requête. En l'occurence, il s'agit de chercher toutes les instances de la classe « Humain » dont le « prenom » commence par « P ».

Le résultat de cette requête est évidemment l'ensemble {Pierre, Paul}. Le gestionnaire de requêtes de Protégé, bien que pratique, reste cependant limité en fonctionnalités.



Requêtes

CHAPITRE IV : Le langage de requêtes SPARQL

4.1 INTRODUCTION :

SPARQL (acronyme obscur dérivé de SQL) est un langage de requêtes destiné à interroger les bases de données (fichiers) RDF, standardisé par le W3C et implémenté en divers langages, notamment en java avec le framework Jena et Twinkle.

4.2 REQUETES DE TYPE SELECT

Comme son nom l'indique, le langage de requêtes SPARQL suit son devancier SQL.

Exemple simple

```
PREFIX vcard: <http://www.w3.org/2001/vcard-rdf/3.0#>
SELECT ?prenom
FROM <http://pagesperso-systeme.lip6.fr/Jean-
Francois.Perrot/inalco/XML/RDF/Exemples/vcardCat.rdf>
WHERE {
    ?x vcard:N ?c .
    ?c vcard:Given ?prenom .
}
```


ce qui se lit :
chercher dans le fichier "vcardCat.rdf" (ci-dessus) toutes les chaînes (ici désignées par "?prenom")
telles qu'il existe dans le graphe correspondant deux triplets de la forme "?x vcard:N ?c" et "?c vcard:Given ?prenom",
avec la même valeur pour "?c".

Exécution :

```
%/$JENAROOT/bin/sparql --query vcard1.rq
-------------
| prenom    |
=============
| "Rebecca" |
| "Matthew" |
| "Sarah"   |
| "John"    |
-------------
```

Les valeurs recherchables ne se limitent pas aux chaînes terminales, mais on obtiendra toutjours des chaînes représentatives :

Exemple : fichier `vcard3.rq`

```
PREFIX vcard: <http://www.w3.org/2001/vcard-rdf/3.0#>
SELECT ?prenom ?x ?c
FROM <vcardCat.rdf>
WHERE {
    ?x vcard:N ?c .
    ?c vcard:Given ?prenom .
}
```

```
%/$JENAROOT/bin/sparql --query vcard3.rq
-------------------------------------------------------------
| prenom    | x                                   | c     |
=============================================================
| "Rebecca" | <http://somewhere/RebeccaSmith/>    | _:b0  |
| "Matthew" | <http://somewhere/MattJones/>       | _:b1  |
| "Sarah"   | <http://somewhere/SarahJones/>      | _:b2  |
| "John"    | <http://somewhere/JohnSmith/>       | _:b3  |
-------------------------------------------------------------
```

Et le résultat obtenu peut aussi avoir une forme propre à une exploitation ultérieure, par exemple en XML :

```
%/$JENAROOT/bin/sparql --query vcard1.rq --results XML
<?xml version="1.0"?>
<sparql xmlns="http://www.w3.org/2005/sparql-results#">
  <head>
    <variable name="prenom"/>
  </head>
  <results>
    <result>
      <binding name="prenom">
        <literal>Rebecca</literal>
      </binding>
    </result>
    <result>
      <binding name="prenom">
        <literal>Matthew</literal>
      </binding>
    </result>
    <result>
      <binding name="prenom">
        <literal>Sarah</literal>
      </binding>
    </result>
    <result>
      <binding name="prenom">
        <literal>John</literal>
      </binding>
    </result>
  </results>
</sparql>
```


Un exemple un peu plus élaboré :

[Exemple de requête SPARQL retournant une liste de photos avec le nom de la personne qu'elles représentent et une description, à partir d'une base de données de type RDF utilisant

l'ontologie (vocabulaire) FOAF (une des plus connues et utilisée pour décrire les personnes et les liens entre elles).]

fichier ExWiki.rdf (base-source)

```
<?xml version="1.0" ?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-
ns#"
  xmlns:foaf="http://xmlns.com/foaf/0.1/"
  xmlns:dc="http://purl.org/dc/elements/1.1/">
    <foaf:Person rdf:about="http://example.net/Paul_Dupont">
        <foaf:name>Paul Dupont</foaf:name>
        <foaf:img
rdf:resource="http://example.net/Paul_Dupont.jpg"/>
        <foaf:knows
rdf:resource="http://example.net/Pierre_Dumoulin"/>
    </foaf:Person>
    <foaf:Person
rdf:about="http://example.net/Pierre_Dumoulin">
        <foaf:name>Pierre Dumoulin</foaf:name>
        <foaf:img
rdf:resource="http://example.net/Pierre_Dumoulin.jpg"/>
    </foaf:Person>
    <foaf:Image rdf:about="http://example.net/Paul_Dupont.jpg">
        <dc:description>Photo d'identité de Paul
Dupont</dc:description>
    </foaf:Image>
    <foaf:Image
rdf:about="http://example.net/Pierre_Dumoulin.jpg">
        <dc:description>Photo d'identité de Pierre
Dumoulin</dc:description>
    </foaf:Image>
</rdf:RDF>
```

ou si vous préférez le lire en format N3 :

```
@prefix dc:    <http://purl.org/dc/elements/1.1/> .
@prefix rdf:   <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
.
@prefix foaf:    <http://xmlns.com/foaf/0.1/> .

<http://example.net/Pierre_Dumoulin>
      a       foaf:Person ;
      foaf:img <http://example.net/Pierre_Dumoulin.jpg> ;
      foaf:name "Pierre Dumoulin" .

<http://example.net/Paul_Dupont>
      a       foaf:Person ;
      foaf:img <http://example.net/Paul_Dupont.jpg> ;
      foaf:knows <http://example.net/Pierre_Dumoulin> ;
      foaf:name "Paul Dupont" .

<http://example.net/Paul_Dupont.jpg>
      a       foaf:Image ;
      dc:description "Photo d'identité de Paul Dupont" .

<http://example.net/Pierre_Dumoulin.jpg>
      a       foaf:Image ;
      dc:description "Photo d'identité de Pierre Dumoulin" .
```

fichier ExWiki.rq (requête SPARQL)

```
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
FROM <http://pagesperso-systeme.lip6.fr/Jean-
Francois.Perrot/inalco/XML/RDF/ExWiki.rdf>
WHERE {
    ?personne rdf:type foaf:Person .
    ?personne foaf:name ?nom .
    ?image rdf:type foaf:Image .
    ?personne foaf:img ?image .
    ?image dc:description ?description
}
```

le résultat en mode texte :

```
-------------------------------------------------------------------
-----------------------------------------
| nom                | image
| description                                                      |
===================================================================
=========================================
| "Pierre Dumoulin" | <http://example.net/Pierre_Dumoulin.jpg>
| "Photo d'identité de Pierre Dumoulin" |
| "Paul Dupont"      | <http://example.net/Paul_Dupont.jpg>
| "Photo d'identité de Paul Dupont"     |
-------------------------------------------------------------------
-----------------------------------------
```

et en syntaxe XML (fichier ExWiki.sparql)

```
<?xml version="1.0" ?>
<sparql xmlns="http://www.w3.org/2005/sparql-results#">
    <head>
        <variable name="nom"/>
        <variable name="image"/>
        <variable name="description"/>
    </head>
    <results ordered="false" distinct="true">
        <result>
            <binding name="nom">
                <literal>Pierre Dumoulin</literal>
            </binding>
            <binding name="image">

<uri>http://example.net/Pierre_Dumoulin.jpg</uri>
            </binding>
            <binding name="description">
                <literal>Photo d'identité de Pierre
Dumoulin</literal>
            </binding>
        </result>
        <result>
            <binding name="nom">
                <literal>Paul Dupont</literal>
            </binding>
            <binding name="image">
                <uri>http://example.net/Paul_Dupont.jpg</uri>
```

```
            </binding>
            <binding name="description">
                <literal>Photo d'identité de Paul
Dupont</literal>
            </binding>
        </result>
    </results>
</sparql>
```

4.3 REQUETES DE TYPE CONSTRUCT

- Une requête `SELECT` renvoie la liste plate des systèmes de liaisons variable - valeur.
  La suite du traitement est ouverte... Très souvent elle commencera par construire un nouveau graphe RDF à partir de ces liaisons.

- L'idée de `CONSTRUCT` est de produire directement ce graphe RDF contenant les valeurs des variables.
  Pour cela on donnera un **gabarit** (*template*), à savoir un graphe dans lequel on va en insérer les valeurs en question.

Exemple très simple

Le graphe-gabarit se réduit à un triplet dont seule la relation est spécifiée (`vcard:FN`), le sujet et l'objet étant des variables.

```
PREFIX foaf:    <http://xmlns.com/foaf/0.1/>
PREFIX vcard:   <http://www.w3.org/2001/vcard-rdf/3.0#>

CONSTRUCT { ?x  vcard:FN ?y }
FROM <http://pagesperso-systeme.lip6.fr/Jean-
Francois.Perrot/inalco/XML/RDF/ExWiki.rdf>
WHERE  { ?x foaf:name ?y }
```

résultat en N3

```
@prefix vcard:   <http://www.w3.org/2001/vcard-rdf/3.0#> .
@prefix foaf:    <http://xmlns.com/foaf/0.1/> .

<http://example.net/Pierre_Dumoulin>
      vcard:FN      "Pierre Dumoulin" .

<http://example.net/Paul_Dupont>
      vcard:FN      "Paul Dupont" .
```

qui se traduit en XML comme

```
<?xml version="1.0" ?>
<rdf:RDF
    xmlns:foaf="http://xmlns.com/foaf/0.1/"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:vcard="http://www.w3.org/2001/vcard-rdf/3.0#" >
  <rdf:Description
rdf:about="http://example.net/Pierre_Dumoulin">
    <vcard:FN>Pierre Dumoulin</vcard:FN>
  </rdf:Description>
```

```
  <rdf:Description rdf:about="http://example.net/Paul_Dupont">
    <vcard:FN>Paul Dupont</vcard:FN>
  </rdf:Description>
</rdf:RDF>
```

Exemple simple

avec un graphe-gabarit à un seul sujet, deux triplets.

```
PREFIX foaf:    <http://xmlns.com/foaf/0.1/>
PREFIX vcard:   <http://www.w3.org/2001/vcard-rdf/3.0#>

CONSTRUCT { ?x foaf:firstName ?y .
            ?x foaf:lastName ?z .
          }

FROM <http://pagesperso-systeme.lip6.fr/Jean-
Francois.Perrot/inalco/XML/RDF/Exemples/vcardCat.rdf>
WHERE
 {        ?x  vcard:N ?u .
          ?u vcard:Given ?y .
          ?u vcard:Family ?z.

 }
```

résultat en N3

```
@prefix vcard:   <http://www.w3.org/2001/vcard-rdf/3.0#> .
@prefix foaf:    <http://xmlns.com/foaf/0.1/> .

<http://somewhere/RebeccaSmith/>
      foaf:firstName  "Rebecca" ;
      foaf:lastName  "Smith" .

<http://somewhere/JohnSmith/>
      foaf:firstName  "John" ;
      foaf:lastName  "Smith" .

<http://somewhere/SarahJones/>
      foaf:firstName  "Sarah" ;
      foaf:lastName  "Jones" .

<http://somewhere/MattJones/>
      foaf:firstName  "Matthew" ;
      foaf:lastName  "Jones" .
```

Exemple un peu moins simple

avec une ressource anonyme dans le graphe-gabarit

```
PREFIX foaf:    <http://xmlns.com/foaf/0.1/>
PREFIX dc:      <http://purl.org/dc/elements/1.1/>
PREFIX vcard:   <http://www.w3.org/2001/vcard-rdf/3.0#>

CONSTRUCT { ?auteur foaf:pastProject _:oeuvre .
            _:oeuvre dc:title ?nom .
            _:oeuvre dc:publisher ?editeur .
            ?auteur foaf:firstName ?p .
            ?auteur foaf:lastName ?n .
```

```
          }

FROM <http://pagesperso-systeme.lip6.fr/Jean-
Francois.Perrot/inalco/XML/RDF/Frantext/RRb.rdf>
WHERE
 {          ?oe  dc:creator ?auteur .
            ?oe  dc:title ?nom .
            ?oe  dc:publisher ?editeur .
            ?auteur  vcard:N ?vc .
            ?vc vcard:Family ?n .
            ?vb vcard:Given ?p .
 }
```

résultat en N3

```
@prefix vcard:   <http://www.w3.org/2001/vcard-rdf/3.0#> .
@prefix dc:      <http://purl.org/dc/elements/1.1/> .
@prefix foaf:    <http://xmlns.com/foaf/0.1/> .

<http://inalco/M2-Trad/poetes#Théodore_de_BANVILLE>
      foaf:firstName  "Théodore" ;
      foaf:lastName  "de BANVILLE" ;
      foaf:pastProject
                  [ dc:publisher  "Paris : M. Levy, 1859." ;
                    dc:title      "Odes funambulesques"
                  ] .
```

4.4 CONTRAINTES PAR FILTER

La clause `FILTER` introduit une expression booléenne que les solutions à la requête doivent satisfaire. Le langage dans lequel s'écrit cette expression est très riche. Il contient des opérateurs de comparaison et des fonctions d'accès.

Comparaison : exiger une différence

Nous faisons l'hypothèse que deux personnes qui ont le même nom de famille se connaissent (`foaf:knows`).
Mais nous voulons éviter les résultats banals "`X foaf:knows X`".
Pour cela, nous pouvons demander que les deux ressources anonymes désignant les "`vcard:N`" soient différentes.

```
PREFIX foaf:    <http://xmlns.com/foaf/0.1/>
PREFIX vcard:   <http://www.w3.org/2001/vcard-rdf/3.0#>

CONSTRUCT { ?x foaf:knows ?y }

FROM <http://pagesperso-systeme.lip6.fr/Jean-
Francois.Perrot/inalco/XML/RDF/Exemples/vcardCat.rdf>
WHERE
 {          ?x  vcard:N ?va .
            ?va vcard:Family ?f .
            ?y  vcard:N ?vb .
            ?vb vcard:Family ?f .
      FILTER (?va != ?vb )
 }
```

résultat en N3

```
@prefix vcard:   <http://www.w3.org/2001/vcard-rdf/3.0#> .
@prefix foaf:    <http://xmlns.com/foaf/0.1/> .

<http://somewhere/RebeccaSmith/>
      foaf:knows    <http://somewhere/JohnSmith/> .

<http://somewhere/JohnSmith/>
      foaf:knows    <http://somewhere/RebeccaSmith/> .

<http://somewhere/SarahJones/>
      foaf:knows    <http://somewhere/MattJones/> .

<http://somewhere/MattJones/>
      foaf:knows    <http://somewhere/SarahJones/> .
```

Bon, mais cette réponse est encore trop verbeuse, car nous savons bien que la relation est symétrique !
Si on nous dit "`X foaf:knows Y`", inutile d'ajouter "`Y foaf:knows X`" !

Conversion des URI et comparaison de chaînes

La fonction `STR` va convertir une URI en chaîne, à laquelle on va pouvoir appliquer la comparaison selon l'ordre lexicographique.

même requête que la précédente, avec

```
   FILTER (?va != ?vb && STR(?x) < STR(?y) )
```

résultat en N3

```
@prefix vcard:   <http://www.w3.org/2001/vcard-rdf/3.0#> .
@prefix foaf:    <http://xmlns.com/foaf/0.1/> .

<http://somewhere/JohnSmith/>
      foaf:knows    <http://somewhere/RebeccaSmith/> .

<http://somewhere/MattJones/>
      foaf:knows    <http://somewhere/SarahJones/> .
```

CHAPITRE IV : Quelques moteurs de recheches sémantiques

4.1 LE MOTEUR CORESE :

Corese est un moteur de recherche sémantique pour le langage RDF (Resource Description Framework), modèle standard du web sémantique proposé par le W3C. Ce moteur est dédié à des applications de web sémantique communautaire ou d'entreprise.