

Brève introduction à RDF

Michel Gagnon

Table des matières

1	Introduction	2
2	Syntaxe de RDF	4
2.1	Syntaxe abstraite	4
2.2	Syntaxe N-Triples	6
2.3	Syntaxe Notation 3	7
2.4	Syntaxe RDF/XML	8
2.4.1	Représentation des noeuds vides	10
2.4.2	Littéraux	11
2.4.3	Identification de l'URI de base	11
2.5	Identification des types de ressources	12
2.6	Conteneurs	15
2.7	Collections	17
2.8	Réification	18
2.9	Exercices	20
3	RDF Schema	22
3.1	Classes	22
3.2	Propriétés	23
4	Union et fusion de graphes RDF	26
5	Sémantique de RDF	26
5.1	Interprétation simple	27
5.2	Interprétation simple et conséquence logique	28
5.3	Rdf-interprétation	29
5.4	Rdfs-interprétation	31
5.5	Exercices	33

6	Inférence	36
6.1	Inférence simple	36
6.2	Règles de généralisation de littéraux	38
6.3	Inférence RDF	39
6.4	Inférence RDFS	39
6.5	Exercices	41
7	Annexe A - Triplets axiomatiques de RDFS	42

1 Introduction

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 l'auteur de ce document, 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 Identifier), 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 1).

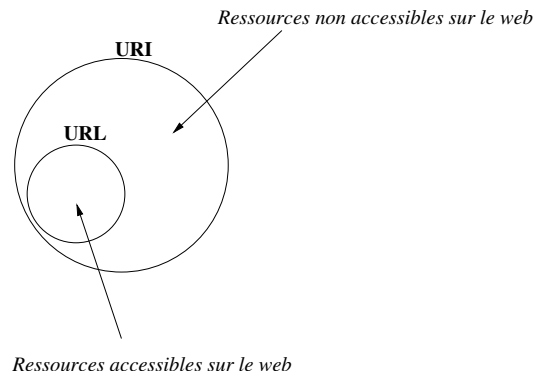


FIG. 1 – Ensembles des URI et des URL

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. Il existe aussi deux autres notations plus faciles à lire pour un humain : N-Triples et Notation 3.

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 les 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 pourrions 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	<code>http://www.polymtl.ca/Profs#MichelGagnon</code>
le nom de la personne	<code>"Michel Gagnon"</code>
le lieu de travail	<code>http://www.polymtl.ca/Vocabulary#dgi</code>
la page personnelle	<code>http://www.professeurs.polymtl.ca/michel.gagnon</code>

Veillez noter 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 noeuds. Certains noeuds, représentés par une ellipse, désignent une entité référée par une URI. D'autres noeuds, 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.

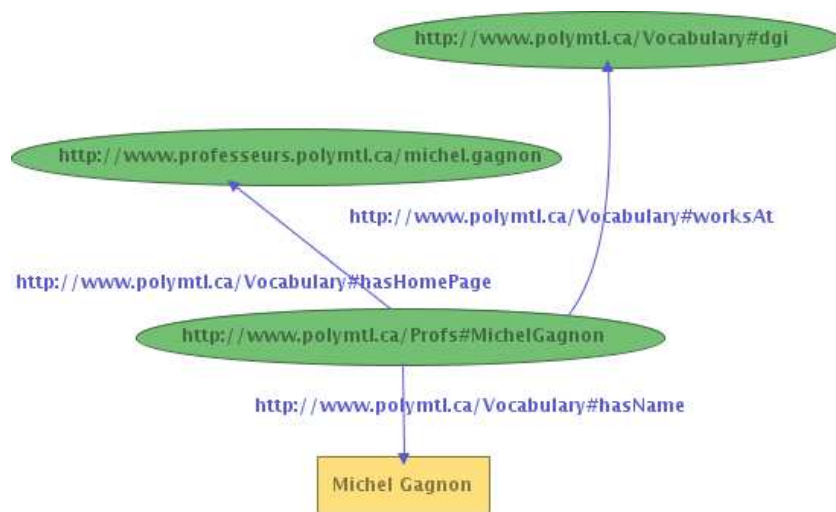


FIG. 2 – Un premier exemple

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 3.

2 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. Par la suite, nous verrons comment un graphe RDF peut être représenté dans les trois syntaxes suivantes : RDF/XML, N-Triples et Notation 3.

2.1 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 *noeud vide*, c’est-à-dire un noeud 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 noeud vide ou un littéral.

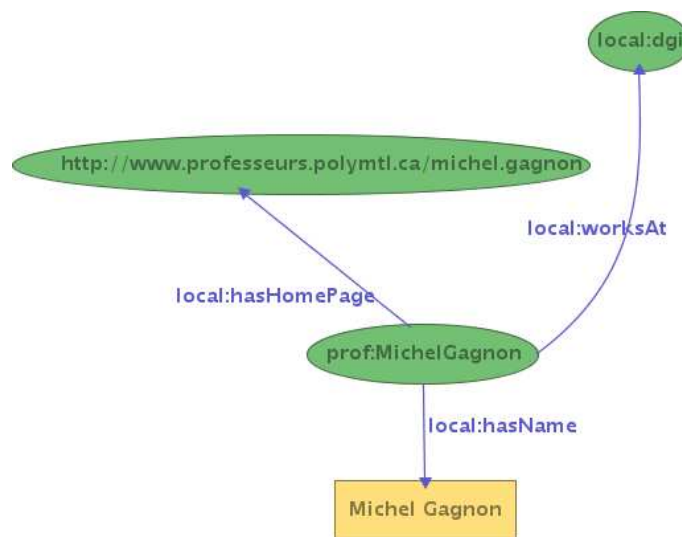


FIG. 3 – Exemple avec utilisation de préfixe

Notons qu'il y a plusieurs avantages à utiliser des URI pour désigner les ressources décrites par un 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. Nous verrons plusieurs situations de ce genre dans la suite de ce document.

Un noeud vide est tout noeud qui n'est ni une URI, ni un littéral. Il s'agit d'un noeud unique qui peut apparaître dans plusieurs triplets, et qui n'a pas de nom intrinsèque. En quelque sorte, un noeud 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.

Supposons par exemple, que nous voulions indiquer que la dernière date de mise à jour de la page personnelle de Michel Gagnon est le 20 mai 2005. On pourrait toujours le faire en ajoutant une propriété `local:derniereMaJ` et dont la valeur serait `2005-05-20`. Mais, comme il s'agit tout simplement d'une chaîne de caractère, il serait difficile d'effectuer une comparaison avec une autre date.

Il est donc préférable de fournir un type à ce littéral. La manière la plus simple de le faire serait d'utiliser le type `xml:date` prévu par la norme XML. En ajoutant ce type, on indique que le littéral ne doit pas être interprété comme tel, mais plutôt qu'on doit utiliser le type spécifié pour identifier la valeur correspondant à la chaîne fournie.

Plus formellement, un type de données en RDF est toujours identifié par une URI et suppose l'existence d'un espace lexical \mathcal{L} , d'un espace de valeurs \mathcal{V} et d'une fonction $m : \mathcal{L} \mapsto \mathcal{V}$. L'espace lexical \mathcal{L} est

l'ensemble de toutes les chaînes de caractères possibles pour ce type, alors que \mathcal{V} est l'ensemble de toutes les valeurs possibles. L'application m doit être définie telle que :

- chaque membre de \mathcal{L} est associé à exactement un membre de \mathcal{V} ;
- pour chaque valeur de \mathcal{V} , il peut y avoir zéro ou plusieurs membres de \mathcal{L} qui lui sont associés.

À part ces caractéristiques que tout littéral typé doit respecter en RDF, aucune autre contrainte n'est spécifiée et, surtout, aucune spécification sur la manière d'interpréter un littéral typé. Cette interprétation repose entièrement sur l'application qui manipule le document RDF.

2.2 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'identificateur 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'abréger 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 préfixe. Supposons maintenant que les préfixes `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`.

2.3 Syntaxe Notation 3

La syntaxe Notation 3 (abrégiée N3) permet de représenter un graphe RDF de manière flexible et lisible. À la base, elle utilise une notation similaire à la syntaxe N-Triples, sauf qu'elle permet de définir et d'utiliser des préfixes. Lorsque qu'une ressource est désignée par une URI dans une forme non abrégée, on la met entre crochets `< >`. Si on utilise un préfixe, on omet les crochets. Voici une manière de représenter l'exemple de la figure 2 avec la syntaxe N3 :

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

Comme il s'agit ici de trois descriptions d'une même ressource, il serait intéressant des les combiner en une seule description. Avec la notation N3, cela est possible. Il suffit de spécifier la ressource décrite, suivie de paires `<predicat> <objet>` séparées par des points-virgule. Notre exemple se retrouve alors avec la forme suivante :

```
@prefix local: <http://www.polymtl.ca/Vocabulary#> .  
@prefix prof: <http://www.polymtl.ca/Profs#> .
```

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

Un noeud vide est représenté par les crochets []. Ainsi, si dans notre exemple l'URI de la personne n'était pas connue, nous aurions la représentation suivante :

```
@prefix local: <http://www.polymtl.ca/Vocabulary#> .

[ ] local:hasHomePage <http://www.professeurs.polymtl.ca/michel.gagnon> ;
  local:hasName "Michel Gagnon" ;
  local:worksAt <http://www.polymtl.ca/Vocabulary#dgi> .
```

Remarquez qu'il est possible de mettre toutes les déclarations de propriété d'un noeud vide à l'intérieur des crochets :

```
@prefix local: <http://www.polymtl.ca/Vocabulary#> .

[ local:hasHomePage <http://www.professeurs.polymtl.ca/michel.gagnon> ;
  local:hasName "Michel Gagnon" ;
  local:worksAt <http://www.polymtl.ca/Vocabulary#dgi> ] .
```

L'avantage de cette dernière approche est qu'elle facilite l'écriture lorsque nous avons un noeud vide comme objet d'un triplet. Ainsi, si nous avons la relation `connait` et que nous voulions indiquer que le professeur Jean St-Jean connaît la personne décrite dans l'exemple précédent, nous aurions la représentation suivante :

```
@prefix local: <http://www.polymtl.ca/Vocabulary#> .
@prefix prof: <http://www.polymtl.ca/Profs#> .

prof:JeanStJean local:connait
  [ local:hasHomePage <http://www.professeurs.polymtl.ca/michel.gagnon> ;
    local:hasName "Michel Gagnon" ;
    local:worksAt <http://www.polymtl.ca/Vocabulary#dgi> ] .
```

2.4 Syntaxe RDF/XML

La meilleure manière de comprendre la syntaxe RDF/XML est de considérer un exemple. Voici donc une des représentations possibles de l'exemple de la figure 2 :

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

```

On remarque que tout le graphe RDF est décrit à l'intérieur des balises `rdf:RDF`. Plusieurs espaces de nommage y sont d'abord spécifiés, en particulier celui qui correspond à notre préfixe `local`. Viennent ensuite trois descriptions de la ressource `MichelGagnon` (pour simplifier l'écriture, j'utiliserai parfois un nom abrégé pour désigner une URI). Remarquons qu'une description d'une ressource est spécifiée par la balise `rdf:Description`, avec l'URI de la ressource indiquée par l'attribut `rdf:about`. On remarque aussi, pour les deux descriptions dont l'objet est une URI, que cette URI est elle aussi indiquée par un attribut, cette fois-ci dans une balise correspondant au prédicat. Dans le cas du littéral, la valeur est mise tout simplement entre les balises qui correspondent au prédicat. Finalement, notons que dans les attributs, les URI doivent être indiquées au long, tel que requis par la norme XML.

Normalement, lorsque plusieurs descriptions se réfèrent à une même ressource, on utilise une forme abrégée qui les réunit toutes en une seule description :

```

<?xml version="1.0"?>
<!DOCTYPE rdf:RDF [
  <!ENTITY local "http://www.polymtl.ca/Vocabulary#">
  <!ENTITY prof "http://www.polymtl.ca/Profs#">
]>
<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="&prof;MichelGagnon">
  <local:worksAt rdf:resource="&local;dgi"/>
  <local:hasName>Michel Gagnon</local:hasName>
  <local:hasHomePage
    rdf:resource="http://www.professeurs.polymtl.ca/michel.gagnon"/>
</rdf:Description>
</rdf:RDF>

```

Dans l'exemple précédent, nous remarquons que des entités `&prof;` et `&local;` ont été définies pour simplifier l'écriture des attributs.

2.4.1 Représentation des noeuds vides

Dans la syntaxe RDF/XML, un noeud vide est représenté tout simplement par une balise `rdf:Description` ne contenant aucun attribut `rdf:about`, ce qui revient à dire qu'il s'agit d'une description qui ne précise pas la ressource décrite. Tout comme dans la notation N-Triples, un noeud vide peut avoir un identificateur. Il suffit pour cela d'ajouter l'attribut `rdf:nodeID` dans la balise `Description` et d'y associer un identificateur unique. À tout endroit dans le même document où on retrouve dans une balise `rdf:Description` l'attribut `rdf:nodeID` avec le même identificateur, il s'agira toujours d'une référence au même noeud vide. À noter que tout comme dans la syntaxe N-Triples, l'identificateur d'un noeud vide ne sert qu'à désigner le même noeud vide dans une autre description qui se trouve dans le même document. Ainsi, notre exemple de la figure 2 pourrait être représenté de la manière suivante, si le noeud correspondant à la personne décrite était un noeud vide.

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

Supposons maintenant que la propriété *hasHomePage* est fournie dans une description séparée. Il faudrait alors utiliser un identificateur pour le noeud vide :

```
<?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:nodeID="p304">
    <local:worksAt rdf:resource="http://www.polymtl.ca/Vocabulary#dgi"/>
    <local:hasName>Michel Gagnon</local:hasName>
  </rdf:Description>
  <rdf:Description rdf:nodeID="p304">
    <local:hasHomePage
      rdf:resource="http://www.professeurs.polymtl.ca/michel.gagnon"/>
  </rdf:Description>
</rdf:RDF>
```

Un noeud vide ne peut être que le sujet ou l'objet d'un triplet. En aucun cas on ne pourra utiliser une propriété anonyme, ce qui signifie que le prédicat sera toujours désigné par une URI.

Un noeud vide n'est pas seulement utile pour décrire des ressources sans les nommer. Il permet aussi de contourner le fait qu'en RDF on ne peut exprimer que des relations binaires. Pour exprimer une relation R qui implique n arguments, où $n > 2$, il suffit de choisir un de ces arguments comme étant le sujet de la relation R (typiquement le premier argument) et de définir l'objet comme un noeud vide. Chaque argument qui reste est alors représenté comme étant l'objet d'une relation le liant à ce noeud vide.

2.4.2 Littéraux

Comme nous l'avons vu dans l'exemple précédent, qui contient le littéral "Michel Gagnon", un littéral simple est représenté en l'insérant tout simplement entre les balises qui désignent le prédicat ayant ce littéral comme valeur. S'il y a en plus la spécification de la langue utilisée, il suffit de mettre l'attribut `xml:lang` dans la balise du prédicat et de spécifier la langue en question.

En ce qui concerne les littéraux typés, ils sont représentés en RDF/XML par l'attribut `rdf:datatype`, dont la valeur indique le type utilisé pour interpréter le littéral. Si on ajoutait à notre exemple la date de dernière mise à jour de la page personnelle, on obtiendrait une représentation comme celle-ci :

```
<?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.polymtl.ca/Profs#MichelGagnon">
    <local:hasHomePage>
      <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>
    </local:hasHomePage>
    <local:worksAt rdf:resource="http://www.polymtl.ca/Vocabulary#dgi"/>
    <local:hasName>Michel Gagnon</local:hasName>
  </rdf:Description>
</rdf:RDF>
```

2.4.3 Identification de l'URI de base

Rappelons qu'une des caractéristiques importantes du web sémantique est la possibilité de décrire une même ressource dans des documents différents. Jusqu'à maintenant, nous avons toujours identifié une ressource en utilisant l'attribut `rdf:about`. Ainsi, dans notre exemple, le professeur qui est décrit est désigné par l'URI suivante :

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

N'importe quel document du web peut utiliser cette URI pour fournir des descriptions de cette ressource. Mais il n'y a pas à proprement parler d'endroit où cette URI est définie. RDF ne fournit pas de moyen de faire cela.

Par contre, on peut s'en rapprocher en utilisant un système similaire aux fragments de HTML. Au lieu de désigner une ressource en utilisant une URI complète associée à l'attribut `rdf:about`, on utilise plutôt l'attribut `rdf:ID` avec comme valeur un fragment. Ceci a pour effet de désigner implicitement la ressource par une URI qui résulte de la fusion de l'URI correspondant au document où se trouve la description avec le fragment en question. Dans notre exemple, il faudrait que la description se trouve dans un document dont l'URI (qui se trouve à être en fait son URL) est `http://www.polymtl.ca/Profs`, et que l'attribut `rdf:about` soit remplacé par `rdf:ID="MichelGagnon"`.

Un des intérêt d'utiliser ce mécanisme est que, tout comme en HTML, on a la contrainte que l'identificateur doit être unique dans le document. On s'assure donc ainsi que l'URI ne sera définie qu'à un seul endroit.

Cette solution, quoiqu'intéressante, crée un problème lorsqu'on veut définir une URI en utilisant la même base, mais dans des documents différents. Ce serait le cas par exemple si nous voulions que tous les professeurs aient une URI dont la base est `http://www.polymtl.ca/Profs`, mais que les descriptions soient dans des documents différents. Une solutions consisterait bien sûr à définir toutes les ressources dans un document ayant cette URI et de compléter les descriptions dans des documents séparés. Mais ceci n'est pas nécessairement la meilleure approche en terme de gestion.

Une autre approche, peut-être plus intéressante, est d'utiliser un mécanisme qui permet de spécifier explicitement la base qui sera utilisée pour former l'URI, en utilisant l'attribut `xml:base` dans la balise RDF du document. Chaque fois qu'on aura un attribut `rdf:ID` dans ce document, l'URI sera formée en prenant non pas l'URI du document, mais plutôt celle spécifiée par cet attribut. C'est ainsi qu'on pourra avoir la même base dans des documents différents.

2.5 Identification des types 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 définit la propriété suivante :

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

En utilisant l'alias `rdf:` pour désigner le préfixe `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
```

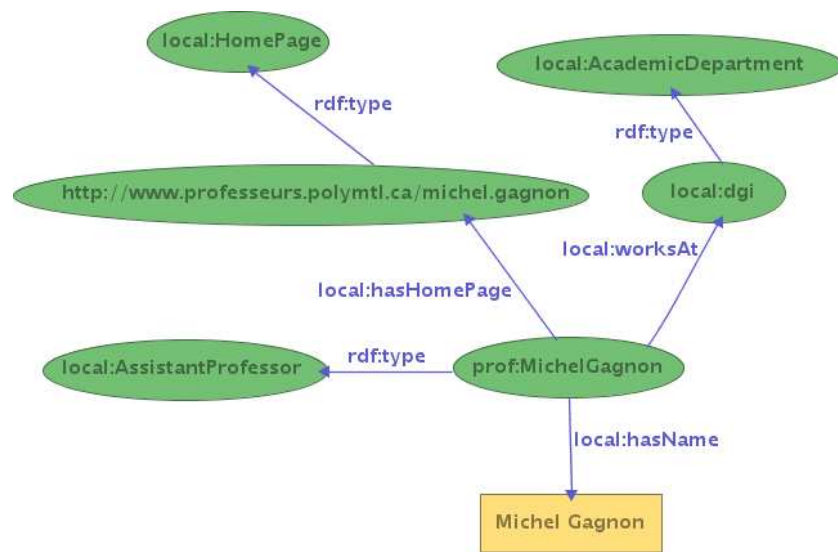


FIG. 4 – Exemple de graphe RDF avec types de ressources identifiés

```

    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>

```

Comme le type d'une ressource est une propriété importante et très utilisée, la syntaxe RDF/XML permet une abbréviation, qui consiste tout simplement à remplacer la balise `Description` par le type de la ressource et retirer la relation `rdf:type` associée à la ressource :

```

<?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#">
<local:AssistantProfessor

```

```

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

```

On peut aussi décrire séparément les ressources :

```

<?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#w">
<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">
  <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>
</local:AssistantProfessor>
</rdf:RDF>

```

Dans la notation N3, il existe aussi une abbréviation pour l'identification du type d'une ressource. Il s'agit tout simplement d'utiliser la propriété a :

```

@prefix local: <http://www.polymtl.ca/Vocabulary#> .
@prefix prof: <http://www.polymtl.ca/Profs#> .

local:dgi a local:AcademicDepartment .
<http://www.professeurs.polymtl.ca/michel.gagnon> a local:HomePage .
prof:MichelGagnon
  a local:AssistantProfessor ;
  local:hasHomePage <http://www.professeurs.polymtl.ca/michel.gagnon> ;
  local:hasName "Michel Gagnon" ;
  local:worksAt local:dgi .

```

À noter que RDF n'interdit pas qu'une ressource ait plus d'un type. Mais un seul de ses types pourra faire l'objet de l'abréviation de RDF/XML qui permet de remplacer la balise `rdf:Description` par le type. Tous les autres types devront être définis explicitement.

2.6 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. À noter qu'il n'y pas vraiment de contraintes sur la manière de décrire les conteneurs. Par exemple, RDF n'interdit pas d'avoir deux membres avec la même valeur n , même avec le conteneur `rdf:Seq`. Il n'interdit pas non plus qu'il y ait des sauts dans la numérotation. Par exemple, on peut très bien décrire un conteneur de trois membres avec les relations `rdf:_1`, `rdf:_2` et `rdf:_4`.

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 5.

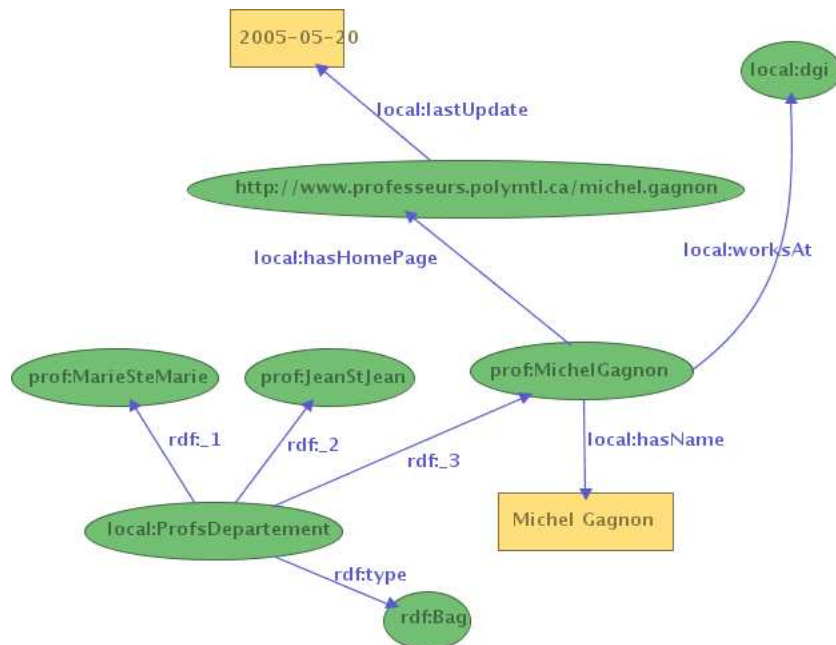


FIG. 5 – Exemple de description de conteneur

Pour simplifier l'écriture, RDF/XML fournit une abréviation, 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. Ainsi, les deux formes suivantes sont équivalentes :

```

<?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>

```

```

<?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:_1 rdf:resource="http://www.polymtl.ca/Profs#MarieSteMarie"/>
    <rdf:_2 rdf:resource="http://www.polymtl.ca/Profs#JeanStJean"/>
    <rdf:_3 rdf:resource="http://www.polymtl.ca/Profs#MichelGagnon"/>
  </rdf:Bag>
</rdf:RDF>

```

2.7 Collections

Un des désavantages avec les conteneurs est qu'il n'est pas possible de les considérer comme des ensembles fermés. Ainsi, si dans une description on spécifie l'existence de trois éléments dans un conteneur, on n'a jamais de garantie qu'il ne contient que ces trois éléments. Rien n'empêche de spécifier d'autres éléments de ce conteneur dans une autre description. Pour pallier cette lacune, RDF permet de définir des *collections*, qui sont en quelque sorte des listes. La liste vide est représentée par une ressource spéciale pré-définie 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 5 sont les seuls du département (un petit 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>
```

Comme cette syntaxe est très lourde, il existe donc une abbréviation, mais qui ne pourra être utilisée que si la liste est l'objet d'un triplet, ce qui n'est pas le cas dans notre exemple. Supposons donc qu'il existe une ressource `local:Departement` liée à notre liste par la relation `local:staff`. Nous pourrions

alors utiliser une l'attribut `rdf:parseType="Collection"` dans la balise qui représente la propriété, et fournir la liste des éléments. Ainsi, notre exemple aurait alors la forme suivante :

```
<?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>
  <local:Departement>
    <local:staff rdf:parseType="Collection">
      <rdf:Description rdf:about="http://www.polymtl.ca/Profs#MarieSteMarie"/>
      <rdf:Description rdf:about="http://www.polymtl.ca/Profs#JeanStJean"/>
      <rdf:Description rdf:about="http://www.polymtl.ca/Profs#MichelGagnon"/>
    </local:staff>
  </local:Departement>
</rdf:RDF>
```

Il est important de remarquer que si on utilise la forme non abrégée pour décrire une liste, il n'y a pas réellement de contraintes sur l'utilisation des prédicats `rdf:first` et `rdf:rest`. Rien n'empêche d'utiliser plus d'une occurrence de ces relations pour un même noeud. Par exemple, on pourrait les utiliser pour décrire un arbre au lieu d'une simple liste.

2.8 Réification

Supposons qu'un triplet est ajouté dans un modèle et que nous voulions spécifier l'auteur de ce triplet ou la date de cet ajout. Avec ce que nous connaissons jusqu'à maintenant, il ne serait pas possible de le faire, puisqu'on ne peut pas désigner un triplet au complet. Pour y arriver, il faut utiliser une technique appelée *réification*.

L'idée consiste à ajouter une ressource de type `rdf:Statement`. Une telle ressource représente un triplet. Pour spécifier les informations contenues dans ce triplet, on peut utiliser les propriétés `rdf:subject`, `rdf:predicate` et `rdf:objet`.

La figure 6 illustre un exemple de réification, où on précise que Michel Gagnon a lui-même fourni l'information sur son lieu de travail. Voici une représentation de ce modèle avec la syntaxe RDF/XML :

```
<?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#"
xmlns:dc="http://purl.org/dc/elements/1.1/">

<rdf:Description rdf:about="http://www.polymtl.ca/Profs#MichelGagnon">
  <local:worksAt rdf:resource="http://www.polymtl.ca/Vocabulary#dgi"/>
</rdf:Description>

<rdf:Statement>
  <rdf:predicate rdf:resource="http://www.polymtl.ca/Vocabulary#worksAt"/>
  <dc:creator rdf:resource="http://www.polymtl.ca/Profs#MichelGagnon"/>
  <rdf:object rdf:resource="http://www.polymtl.ca/Vocabulary#dgi"/>
  <rdf:subject rdf:resource="http://www.polymtl.ca/Profs#MichelGagnon"/>
</rdf:Statement>
</rdf:RDF>

```

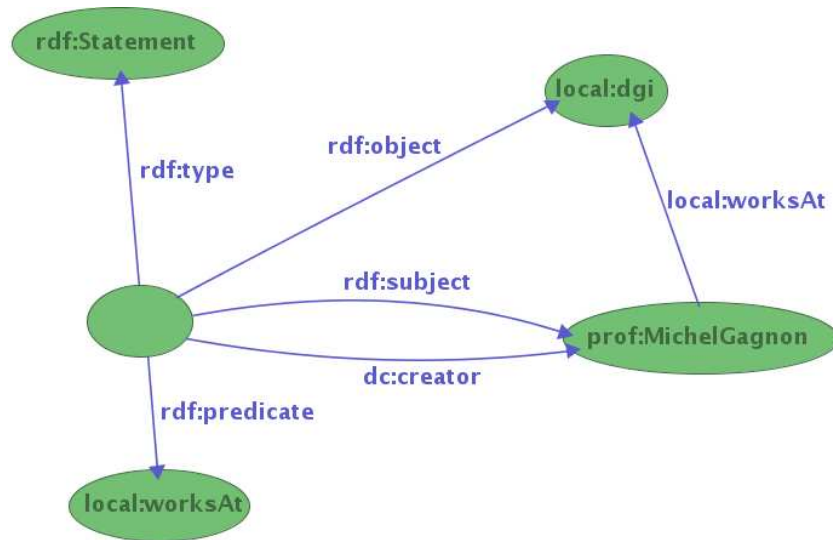


FIG. 6 – Exemple de réification

Un problème demeure, par contre. On peut identifier une ressource qui correspond à un triplet et indiquer quels sont le sujet, le prédicat et l'objet de ce triplet. Mais il n'est pas possible de dire que ce triplet correspond à un triplet spécifique dans un modèle RDF. Bien sûr, on peut rechercher un triplet qui possède le même sujet, le même prédicat et le même objet, mais ceci n'est pas une garantie qu'il s'agisse bien du même triplet. Après tout, rien n'empêche d'avoir deux triplets identiques dans un même modèle. Cela pourrait se produire par exemple, si deux personnes décrivent une même ressource dans deux documents différents. Il se pourra alors que certains triplets identiques se retrouvent dans les deux descriptions.

Dans la syntaxe RDF/XML, il existe un abbréviation qui permet de créer automatiquement la réification d'un triplet. Il s'agit tout simplement d'ajouter l'attribut `rdf:ID` à la propriété. Ceci aura pour effet de créer automatiquement une ressource de type `rdf:Statement` décrivant le triplet en question, et dont l'URI d'identification sera celle fournie à l'attribut `rdf:ID` de la propriété. Ceci permet donc de créer un lien indirect entre le triplet et la description réifiée de ce triplet. Mais il est important de noter que même si l'URI de la réification est la même que celle du prédicat du triplet, cela ne représente toujours pas une manière formelle d'indiquer l'identité entre ces deux entités. Pour y arriver, il faudrait vraiment que nous puissions identifier tous le triplet comme une ressource, ce qui, nous l'avons déjà vu, n'est pas possible en RDF. Voici donc comment notre exemple pourrait être représentée avec cet abbréviation :

```
<?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#"
  xmlns:dc="http://purl.org/dc/elements/1.1/">

  <rdf:Description rdf:about="http://www.polymtl.ca/Profs#MichelGagnon">
    <local:worksAt
      rdf:ID="enoncel" rdf:resource="http://www.polymtl.ca/Vocabulary#dgi"/>
  </rdf:Description>

  <rdf:Statement rdf:about="#enoncel">
    <dc:creator rdf:resource="http://www.polymtl.ca/Profs#MichelGagnon"/>
  </rdf:Statement>
</rdf:RDF>
```

2.9 Exercices

■ Exercice 2.1

Dessinez un graphe RDF représentant la situation suivante : une rencontre entre le président des États-Unis et le premier ministre du Canada, qui a eu lieu à Toronto, le 24 avril 2005.

■ Exercice 2.2

Soit le graphe RDF suivant représenté en utilisant le langage Notation 3 :

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix ex: <http://www.polymtl.ca#> .

ex:MichelGagnon
  rdf:type ex:Professeur, ex:Personne;
  ex:enseigne
    [rdf:first ex:inf6410 ;
     rdf:rest [rdf:first ex:inf1010 ;
               rdf:rest rdf:nil]] ;
```

```

ex:travaille ex:Poly .

[] ex:connait ex:MichelGagnon ;
  ex:travaille [rdf:type ex:Universite] .

```

- a) Dessinez ce graphe.
- b) Traduisez en représentation RDF/XML la plus abrégée possible.

■ Exercice 2.3

Soit la situation suivante :

Marie a eu deux enfants avec Robert : une fille, qui s'appelle Anne et un garçon, qui s'appelle André. Elle habite maintenant avec Luc, avec qui elle a eu une autre fille, qui s'appelle Mélanie. Robert habite à Montréal, avec André. Marie et Luc habitent à Québec avec Anne, Mélanie et Paul. Paul est le fils que Luc a eu avec Claudine dans son premier mariage.

- a) Dessinez un graphe RDF qui représente le mieux cette situation. Assurez-vous qu'un minimum de changements soit requis dans le graphe si un membre de cette famille décide de déménager, et si Marie déménage dans une autre ville avec tous ceux qui habitent avec elle.
- b) Représentez ce graphe en Notation 3.

■ Exercice 2.4

Dessinez le graphe RDF correspondant à la description suivante donnée en RDF/XML :

```

<?xml version="1.0"?>
<!DOCTYPE rdf:RDF [
  <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#" >
  <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#" >
  <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#" >
]>

<rdf:RDF
  xmlns="http://www.polymtl.ca#"
  xml:base="http://www.polymtl.ca#"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema#">
  <A rdf:ID="x1">
    <p rdf:parseType="Collection">
      <rdf:Description rdf:ID="x2"/>
      <rdf:Description>
        <q rdf:resource="x3"/>
      </rdf:Description>
    </p>
  </A>
</rdf:RDF>

```

3 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, abrégé 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 préfixe 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#
```

3.1 Classes

Le premier élément important de ce vocabulaire 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:HomePage` et `local:AssistantProfessor` ont toutes la valeur `rdfs:Class` pour la propriété `rdf:type`, comme illustré à la figure 7.

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 abréviation 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>
```

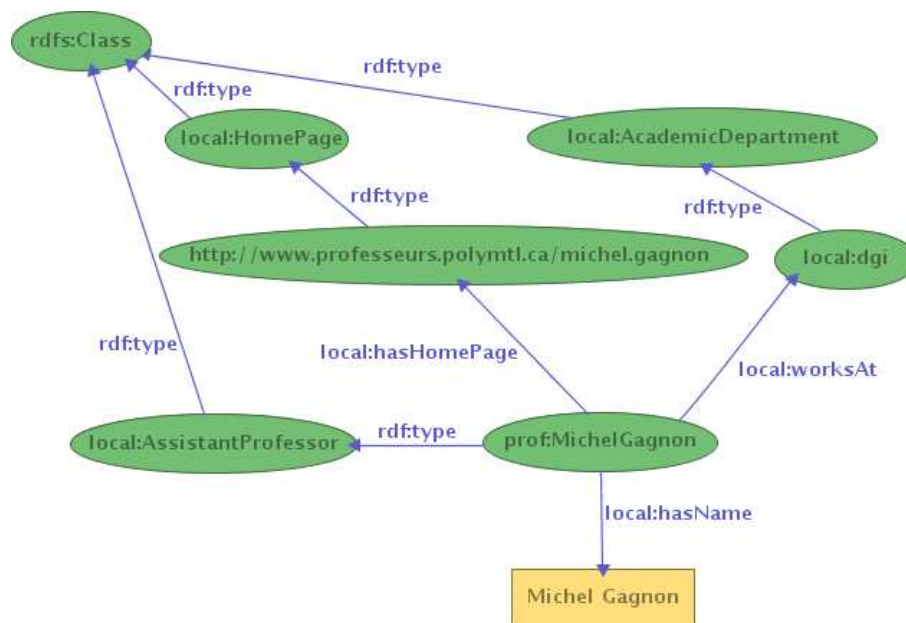


FIG. 7 – Classes

```

</local:AssistantProfessor>
</rdf:RDF>

```

RDFS permet de définir des hiérarchies de classes, en indiquant qu'une classe est sous-classe d'une autre classe, par le biais de la propriété `rdfs:subClassOf`. Si une classe C est sous-classe de C' , cela implique que si une ressource est décrite comme étant de type C , on peut déduire automatiquement qu'elle est aussi de type C' . À noter que la relation `rdfs:subClassOf` est transitive. La figure 8 illustre une extension de notre exemple comprenant une hiérarchie des classes de professeurs. On spécifie aussi qu'un département est un lieu de travail.

3.2 Propriétés

Il est naturel de supposer que toutes les ressources ont un type. Ceci vaut aussi pour les propriétés qui, rappelons-le, sont elles aussi considérées comme des ressources. En RDF, toutes les propriétés ont pour type la classe `rdf:Property`. Ceci signifie donc que notre exemple implique l'existence des triplets suivants :

```

local:hasHomePage rdf:type rdf:Property .
local:hasName rdf:type rdf:Property .
local:worksAt rdf:type rdf:Property .

```

Il va de soi que le triplet suivant est implicite pour tous les graphes RDF :

```

rdf:type rdf:type rdf:Property .

```

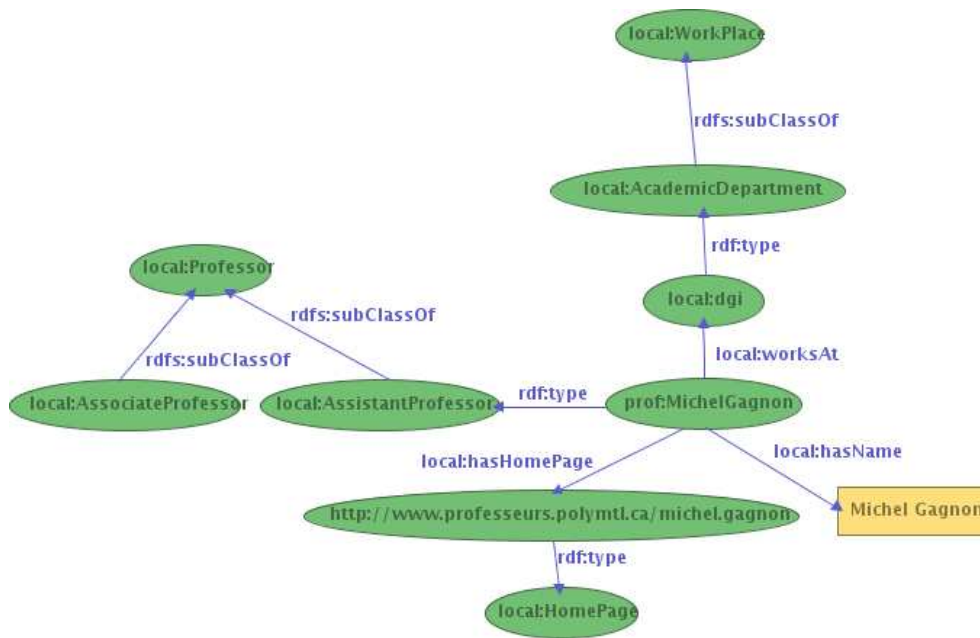


FIG. 8 – Hiérarchie de classes

Il est intéressant de noter que la classe `rdf:Property` ne fait pas partie du vocabulaire de RDFS, mais plutôt du vocabulaire de base de RDF. On comprend facilement ce choix en considérant le fait que la propriété `rdf:type` fait partie du vocabulaire de RDF et que toute ressource doit avoir un type. Il est alors naturel que la classe `rdf:Property` fasse partie du vocabulaire RDF.

De la même manière que l'on peut établir des hiérarchies de classes, on peut définir des hiérarchies de propriétés, en utilisant la relation `rdfs:subPropertyOf`. Mais le plus intéressant en RDFS est probablement la possibilité de spécifier le domaine ou l'image d'une propriété. On le fait en ajoutant un triplet qui indique un type de ressource qu'on peut retrouver comme sujet ou objet d'une propriété, en utilisant les relations `rdfs:domain` ou `rdfs:range`, respectivement. Ainsi, en revenant à notre exemple, on peut spécifier que c'est un membre de personnel (`StaffMember`) qui travaille (`worksAt`) à un lieu de travail (`WorkPlace`). On peut aussi spécifier que la propriété "travailler" est une sous-propriété de la propriété "avoir activité". Tout ceci est illustré à la figure 9.

Une conséquence du fait qu'une propriété P soit sous-propriété d'une propriété P' est que tout domaine (ou image) défini pour P' est automatiquement défini pour la propriété P .

Une propriété peut avoir plus d'un domaine, mais il faut bien comprendre ce que cela signifie : toute ressource à laquelle s'applique cette propriété appartient nécessairement à *toutes* les classes spécifiées comme faisant partie du domaine de la propriété. De même pour l'image. Supposons par exemple la propriété `local:hasMother` a pour domaine les classes `local:Woman` et `local:Person`. Supposons maintenant qu'on applique cette propriété à la ressource `prof:MarieSteMarie`. Automatiquement, on doit pouvoir déduire que cette personne est à la fois une personne et une femme.

Il est très important de comprendre que les spécifications de domaine ou d'image en RDF ne sont pas des contraintes. Quand on spécifie qu'une certaine classe est dans le domaine d'une propriété, on n'exige pas que toute ressource à laquelle la propriété est appliquée soit préalablement définie comme appartenant

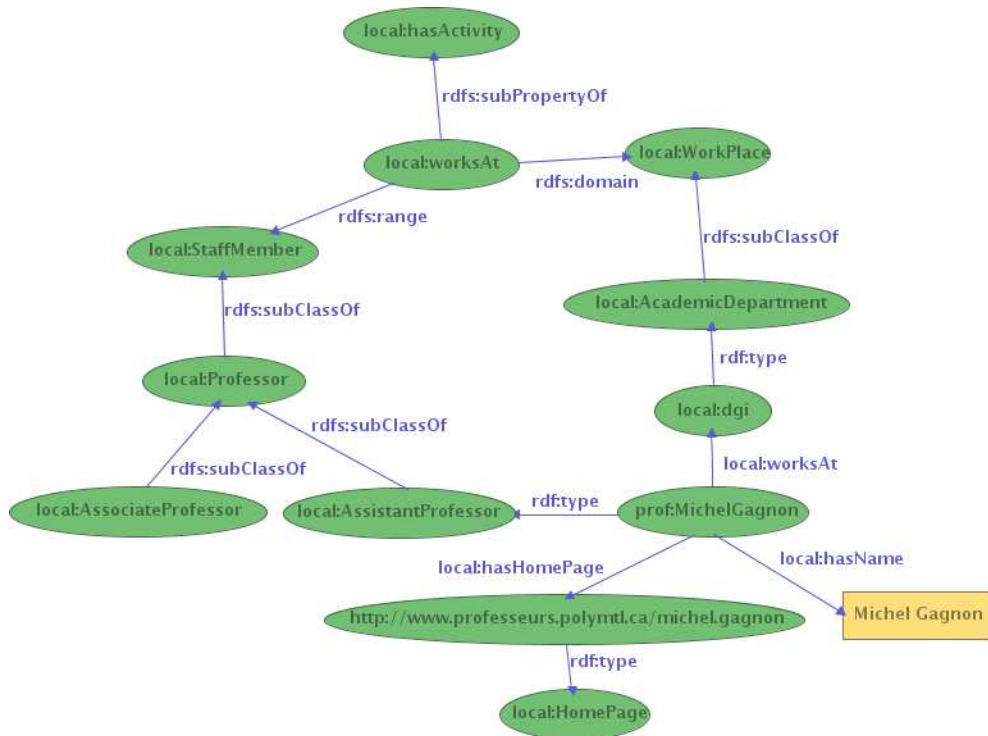


FIG. 9 – Hiérarchie de propriétés

à cette classe. On peut le voir comme une source d'information supplémentaire, c'est-à-dire que dès qu'on aura une ressource à laquelle la propriété est appliquée, on pourra déduire que cette ressource appartient à la classe en question, même si cela n'est spécifié nulle part de manière explicite.

On remarquera aussi que les propriétés en RDFS sont globales. On ne peut pas, par exemple, spécifier qu'une propriété est locale à une classe, comme on le ferait normalement dans un langage de programmation orienté objet. Ceci a pour effet qu'on pourra toujours spécifier d'autres domaines ou d'autres images d'une propriétés, à partir du moment où on connaît son URI. Cette caractéristique est très importante et démontre bien un biais dans le design de RDF, où on a voulu insister plus sur les relations entre les ressources que les ressources elles-mêmes.

4 Union et fusion de graphes RDF

Dans le web sémantique, il faut s'attendre à ce que les descriptions RDF se retrouvent dans des documents différents et éventuellement dispersées à travers le monde. Un agent intelligent dont les décisions dépendront de ces données pourra donc être appelé à regrouper des modèles RDF provenant de sources différentes. Ce qui nous oblige à nous interroger sur la manière de combiner ces données.

On ne pourra pas se contenter de faire une simple union des triplets contenus dans chaque source. Et cela à cause de la présence éventuelle de noeuds vides dans un graphe RDF. On peut penser qu'il suffit de considérer que deux noeuds vides situés dans des documents différents seront toujours distincts dans le modèle résultant. Mais cette solution ne fonctionnera pas si par coïncidence les deux graphes ont chacun un noeud vide avec un identificateur identique. Il faudra donc, avant de fusionner deux graphes RDF, renommer systématiquement les noeuds vides afin d'éviter les conflits d'identificateur. La fusion de deux graphes RDF est définie formellement de la manière suivante :

Soient G_1 et G_2 deux graphes RDF que l'on veut fusionner. Si les graphes n'ont aucun noeud vide avec le même identificateur, la fusion est simplement l'union des triplets des deux graphes. Sinon, il faut créer un graphe G'_2 égal à G_2 , à la différence près que tous les identificateurs des noeuds vides de G'_2 sont renommés de telle manière qu'aucun ne soit identique à un identificateur d'un noeud vide de G_1 . La fusion sera alors le résultat de l'union des triplets de G_1 et G'_2 .

5 Sémantique de RDF

Pour utiliser à bon escient une description RDF, il n'est pas suffisant de bien comprendre la syntaxe. Il faut aussi bien comprendre la sémantique de chaque construction utilisée, afin de s'assurer que ceux qui utilisent ces données aient bien la même interprétation de leur signification que ceux qui les ont créées.

Pour présenter la sémantique de RDF, nous procéderons en trois étapes. Dans un premier temps, nous verrons l'interprétation simple de RDF, c'est-à-dire une interprétation qui ne tient aucunement compte du vocabulaire utilisé. Ensuite, nous contraindrons plus la sémantique en expliquant comment l'interprétation prend en compte les éléments du vocabulaire RDF (c'est-à-dire les éléments `rdf:type`, `rdf:Property`, `rdf:List`, `rdf:Statement`, etc). Finalement, nous ajouterons l'interprétation du vocabulaire RDFS.

5.1 Interprétation simple

Pour interpréter un langage, il faut d'abord identifier les entités du monde auxquelles les constructions du langage se réfèrent. Dans le cas de RDF, cet univers comprend trois ensembles :

IR : Un ensemble non vide de ressources.

IP : Un ensemble de propriétés.

LV : Un sous-ensemble de IR qui représente les valeurs qui seront attribuées au littéraux simples.

La figure 10 illustre la relation entre ces trois ensembles. On remarque que puisque $LV \subset IR$, les valeurs des littéraux sont des entités qui existent, puisqu'elles sont des ressources. Mais à noter que ceci n'implique pas qu'un littéral puissent être identifié par une URI. En fait, RDF ne permet pas cela. Du moins, pas de manière directe.

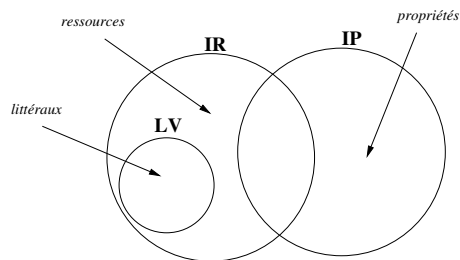


FIG. 10 – Ensembles de l'univers pour interprétation simple d'un vocabulaire RDF

Une fois l'univers défini, il faut définir une interprétation I , qui fait le lien entre chaque construction d'un graphe RDF et les entités comprises dans cet univers. Toute interprétation suppose d'abord un vocabulaire V qui constitue l'ensemble des noms que l'on peut retrouver dans un graphe RDF, c'est-à-dire les URI et les littéraux.

Toute interprétation suppose aussi l'existence des fonctions suivantes :

- $I_{EXT} : IP \mapsto IR \times IR$, qui associe à chaque propriété de IP l'ensemble des paires $\langle x, y \rangle$ telles que les ressources x et y appartiennent à la relation définie par la propriété en question.
- $I_S : URI \mapsto IR \cup IP$, qui associe chaque URI à une ressource ou une propriété.
- $I_L : \mathcal{L} \mapsto IR$ où \mathcal{L} est l'ensemble des littéraux typés. Cette fonction associe chaque littéral typé à sa valeur.

Contrairement à ce que l'on pourrait imaginer, l'image I_L est l'ensemble des ressources, plutôt que l'ensemble LV . La raison de ce choix est qu'il est possible qu'on retrouve dans un graphe RDF un littéral mal formé selon le type qui est spécifié, comme ce serait le cas par exemple avec l'expression "chien"^^xsd:int. Dans ce cas, la sémantique spécifie que la valeur associée soit une entité arbitraire qui n'est pas un littéral.

Avec ces définitions, nous pouvons maintenant fournir une interprétation d'un graphe RDF ne contenant aucun noeud vide. Soit V le vocabulaire utilisé dans le graphe RDF. Une interprétation retourne une valeur de vérité pour un graphe donné (la notation N-Triple est utilisée pour désigner les triplets du graphe) :

- Si E est un graphe ne contenant aucun noeud vide, $I(E) = faux$ si $I(E')$ est faux pour au moins un triplet E' contenu dans E . Sinon $I(E) = vrai$.
- Si E est un triplet $s \ p \ o$ ne contenant aucun noeud vide, alors $I(E) = vrai$ si s , p et o font partie de V , $I(p) \in IP$ et $\langle I(s), I(o) \rangle \in I_{EXT}(I(p))$. Sinon, $I(E) = faux$.
- Si E est un littéral simple "aaa" qui fait partie de V , alors $I(E) = aaa$.
- Si E est un littéral simple "aaa"@ll qui fait partie de V , où ll spécifie la langue, alors $I(E) = \langle aaa, ll \rangle$.
- Si E est un littéral typé et $E \in V$, alors $I(E) = I_L(E)$.
- Si E est une URI et $E \in V$, alors $I(E) = I_S(E)$.

Avec cette définition d'interprétation, la valeur obtenue est *faux* aussitôt qu'un graphe contient au moins un nom qui n'est pas dans le vocabulaire. Autrement dit, si un nom est utilisé dans graphe RDF, il faut absolument qu'il existe une entité correspondante dans l'univers. On remarquera aussi que la première condition implique qu'un graphe vide est trivialement vrai.

Pour terminer la présentation de l'interprétation simple, il ne nous reste plus qu'à voir comment on peut traiter les graphes qui contiennent au moins un noeud vide. L'idée consiste ici à traiter ces noeuds comme les variables existentielles en logique des prédicats. Ces noeuds vides doivent nécessairement désigner une entité de l'univers, sauf qu'on ne sait pas laquelle.

Pour interpréter un graphe contenant des noeuds vides, il suffit d'avoir une fonction A qui associe chaque noeud vide à une des ressources de l'ensemble IR . Ainsi, on peut définir une nouvelle interprétation $I + A$, qui est égale à l'interprétation I pour tout sauf les noeuds vides :

$$[I + A](E) = \begin{cases} A(E), & \text{si } E \text{ est un noeud vide,} \\ I(E), & \text{sinon.} \end{cases}$$

Pour généraliser notre définition d'interprétation à un graphe pouvant contenir des noeuds vides, il suffit d'ajouter le cas suivant :

- Si E est un graphe contenant des noeuds vides, alors $I(E) = vrai$ s'il existe une fonction A telle que $[I + A](E) = vrai$. Sinon, $E(I) = faux$.

Une des conséquences importantes de la manière dont est définie l'interprétation simple est qu'il est possible que deux propriétés aient la même extension, tout en étant distinctes. Supposons, par exemple, un univers comprenant les propriétés "posséder" et "acheter" et tel que tout objet possédé par quelqu'un a aussi été acheté par celui-ci. Les deux propriétés auront donc exactement les mêmes paires dans leur extension. Mais cela ne signifie aucunement que ces deux propriétés sont équivalentes.

Une autre conséquence est que rien n'empêche une propriété d'être elle-même incluse dans son extension. Comme nous le verrons plus loin, c'est le cas avec la propriété `rdf:type`, qui peut être appliquée à elle-même.

5.2 Interprétation simple et conséquence logique

On dit qu'une interprétation I satisfait E si $I(E) = vrai$, et qu'un ensemble S de graphes RDF implique (simplement) un graphe E si toute interprétation qui satisfait tous les membres de S satisfait aussi E . À partir de cette notion de conséquence logique, on peut prouver plusieurs lemmes, que l'on retrouve dans

le document officiel du W3C décrivant la sémantique de RDF. Ici, nous ne retiendrons que deux lemmes importants :

Lemme de la fusion : La fusion de tous les graphes d'un ensemble S de graphes est une conséquence logique de S .

Supposons maintenant que l'instance d'un graphe est définie comme étant le graphe obtenu en remplaçant un ou plusieurs noeuds vides de ce graphe par un autre noeud vide, une URI ou un littéral. On peut maintenant présenter le prochain lemme :

Lemme de l'interpolation : Un ensemble de graphes S implique logiquement un graphe E si et seulement si un sous-graphe de S est une instance de E .

5.3 Rdf-interprétation

Nous avons vu à la section précédente comment interpréter un graphe RDF sans tenir compte de la sémantique de son vocabulaire. Mais nous savons que RDF a déjà un vocabulaire pré-défini. Si dans un graphe RDF nous utilisons ce vocabulaire, il est important de respecter la sémantique associée à ce vocabulaire. Ceci aura pour effet de restreindre les interprétations possibles pour un tel graphe. Nous parlerons alors d'une *rdf-interprétation*, définie formellement de la manière suivante :

Soit $rdfV$ le vocabulaire RDF, constitué des items suivants :

```
rdf:type
rdf:Property
rdf:XMLLiteral
rdf:nil
rdf:List
rdf:Statement
rdf:subject
rdf:predicate
rdf:object
rdf:first
rdf:rest
rdf:Seq
rdf:Bag
rdf:Alt
rdf:_1
rdf:_2
...
rdf:value
```

Une *rdf-interprétation* d'un vocabulaire V est une interprétation simple de $V \cup rdfV$ qui satisfait les deux conditions concernant le terme `rdf:XMLLiteral` telles que décrites dans la spécification du W3C ainsi que la condition et les axiomes qui suivent :

Condition sémantique de RDF :

Soit une entité x de l'univers. $x \in IP$ si et seulement si $\langle x, I(\text{rdf:Property}) \rangle \in I_{EXT}(I(\text{rdf:type}))$.

Axiomes de RDF :

```
rdf:type rdf:type rdf:Property .
rdf:subject rdf:type rdf:Property .
rdf:predicate rdf:type rdf:Property .
rdf:object rdf:type rdf:Property .
rdf:first rdf:type rdf:Property .
rdf:rest rdf:type rdf:Property .
rdf:value rdf:type rdf:Property .
rdf:_1 rdf:type rdf:Property .
rdf:_2 rdf:type rdf:Property .
...
rdf:nil rdf:type rdf:List .
```

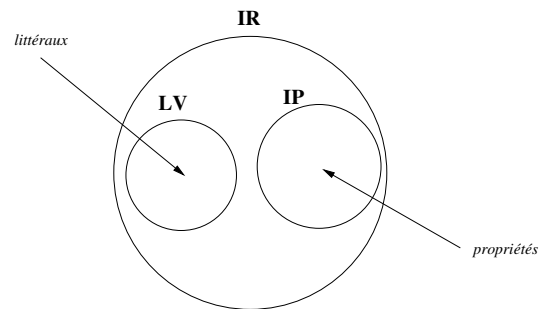


FIG. 11 – Ensembles de l'univers pour rdf-interprétation d'un vocabulaire RDF

Pour comprendre l'effet de l'ajout de cette condition, supposons l'existence d'un triplet de la forme suivante :

```
ex:u rdf:type rdf:Property .
```

L'entité qui correspond à cette $ex:u$ dans l'univers est une propriété. Supposons maintenant qu'un item $ex:p$ représente une propriété, ce qui sera nécessairement le cas si on le retrouve comme prédicat dans un triplet. La condition permet alors de déduire le triplet suivant :

```
ex:p rdf:type rdf:Property .
```

Une conséquence importante de cette condition est que toute propriété peut apparaître comme sujet d'un triplet. Or, par définition de l'interprétation simple, le sujet d'un triplet est toujours une ressource. Ce qui signifie qu'avec la rdf-interprétation, on a $IP \subseteq IR$. L'univers ressemblera plutôt à celui illustré à la figure 11.

Quant aux axiomes, ils établissent clairement quels éléments du vocabulaire RDF sont des propriétés, donc que l'on s'attend à retrouver dans les triplets à la position du prédicat. Le dernier axiome ne fait qu'établir l'existence de la liste vide, nécessaire pour la construction de toute liste.

Il est important de remarquer que la sémantique de RDF ne va pas au-delà ce qui est spécifié ici. Ce qui signifie qu'elle n'ajoute aucune contrainte sur les conteneurs, les collections et la réification.

5.4 Rdfs-interprétation

Nous avons vu à la section précédente que l'ajout apporté par une rdfs-interpretation, par rapport à une interprétation simple, est une spécification du concept de propriété. Une rdfs-interpretation ajoute à cela une spécification du concept de classe. Intuitivement, une classe est toute ressource qui peut se retrouver comme objet dans un triplet dont le prédicat est `rdf:type`. Nous définirons donc un sous-ensemble IC de l'ensemble des ressources pour représenter l'ensemble des classes de l'univers, tel qu'illustré à la figure 12.

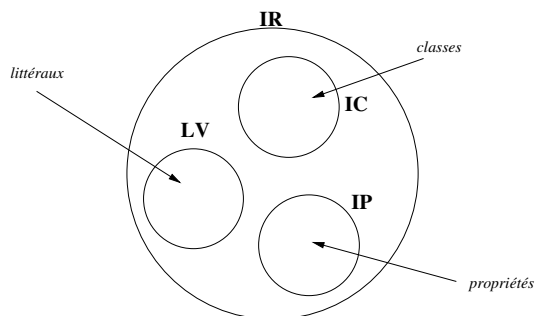


FIG. 12 – Ensembles de l'univers pour rdfs-interpretation d'un vocabulaire RDF

Il nous faudra alors une nouvelle fonction d'interprétation IC_{EXT} , qui associe des éléments du vocabulaire à un élément de l'ensemble IC . Une rdfs-interpretation devra respecter les conditions suivantes :

- $x \in IC_{EXT}(y)$ si et seulement si $\langle x, y \rangle \in I_{EXT}(I(\text{rdf:type}))$
- $IC = IC_{EXT}(I(\text{rdfs:Class}))$
- $IR = IC_{EXT}(I(\text{rdfs:Resource}))$
- $LV = IC_{EXT}(I(\text{rdfs:Literal}))$

La première condition spécifie que l'ensemble des classes dans l'univers correspond à toute ressource qui peut être l'objet de la propriété `rdf:type`. La deuxième condition peut être comprise de la manière suivante. Supposons l'existence d'un triplet de la forme suivante :

`ex:u rdf:type ex:c .`

Dans ce cas, `ex:c` appartient à l'ensemble des classes, par la première condition et, selon la deuxième condition, ce triplet implique logiquement le triplet suivant :

`ex:c rdf:type rdfs:Class .`

De même, par la définition de l'interprétation simple, $ex:u$ et $ex:c$ sont des ressources. Donc, par la troisième condition on peut déduire les triplets suivants :

```
ex:u rdf:type rdfs:Resource .
ex:c rdf:type rdfs:Resource .
```

La dernière condition indique que tout littéral appartient à la classe correspondant à `rdfs:Literal`. Mais étant donné qu'un littéral ne peut pas être le sujet d'un triplet, on ne peut pas spécifier directement sa classe, comme nous venons de le faire avec les classes et les ressources. Par contre, on pourrait dire que l'objet d'un triplet est un littéral, sans pouvoir spécifier ce littéral :

```
ex:u ex:prop _:x .
_:x rdf:type rdfs:Literal .
```

À noter qu'une classe peut avoir une extension vide. Aussi, une conséquence des deux premières conditions est que le triplet suivant est toujours valide :

```
rdfs:Class rdf:type rdfs:Class .
```

Voici maintenant d'autres conditions sur l'interprétation du vocabulaire de RDFS :

- Si $\langle x, y \rangle \in I_{EXT}(I(\text{rdfs:domain}))$ et $\langle u, v \rangle \in I_{EXT}(x)$, alors $u \in IC_{EXT}(y)$.
- Si $\langle x, y \rangle \in I_{EXT}(I(\text{rdfs:range}))$ et $\langle u, v \rangle \in I_{EXT}(x)$, alors $v \in IC_{EXT}(y)$.
- Si $\langle x, y \rangle \in I_{EXT}(I(\text{rdfs:subClassOf}))$, alors $x \in IC$, $y \in IC$ et $IC_{EXT}(x)$ est un sous-ensemble de $IC_{EXT}(y)$.
- $I_{EXT}(I(\text{rdfs:subClassOf}))$ est transitive et réflexive sur IC .
- Si $x \in IC$, alors $\langle x, I(\text{rdfs:Resource}) \rangle \in I_{EXT}(I(\text{rdfs:subClassOf}))$.
- $I_{EXT}(I(\text{rdfs:subPropertyOf}))$ est transitive et réflexive sur IP .
- Si $\langle x, y \rangle \in I_{EXT}(I(\text{rdfs:subPropertyOf}))$, alors $x \in IP$, $y \in IP$ et $I_{EXT}(x)$ est un sous-ensemble de $I_{EXT}(y)$.

La première de ces conditions spécifie que si on a un triplet de la forme suivante :

```
ex:p rdfs:domain ex:c .
```

alors $\langle ex:p \rangle$ est une propriété telle que pour tout triplet de la forme

```
ex:a ex:p ex:b
```

on a nécessairement que toute interprétation doit associer à $\langle ex:a \rangle$ un élément de l'univers qui appartient à la classe représentée par l'item $\langle ex:c \rangle$. En d'autres mots, cela signifie que nous pourrions déduire le triplet suivant :

ex:a rdf:type ex:c

La deuxième condition définit de manière similaire l'image d'une propriété. Selon la troisième condition, si une entité x est déclarée comme étant sous-classe d'une autre entité y , ces deux entités sont des classes et l'extension de x est un sous-ensemble de l'extension de y . Quant à la quatrième condition, elle spécifie que toute classe est sous-classe d'elle-même (réflexivité) et que si C est sous-classe de C' qui est elle-même sous-classe de C'' , alors C est sous-classe de C'' (transitivité). La condition suivante indique que toute classe est sous-classe de la classe des ressources. Finalement, les deux dernières conditions caractérisent les propriétés.

Voilà qui termine cette brève présentation de la sémantique de RDF. Pour être plus complet, il faudrait aussi présenter la sémantique formelle des types de donnée, qui a été exclue du présent document pour ne pas l'alourdir. On pourra se référer au document officiel du W3C pour plus de détails sur la sémantique des types de données. À noter aussi que la rdfs-interprétation contient un ensemble de triplets axiomatiques, dont la liste est fournie à l'annexe A.

5.5 Exercices

■ Exercice 5.1

Soit le graphe RDF illustré à la figure 13. Dites laquelle ou lesquelles des interprétations suivantes sont valides pour ce graphe :

Interprétation 1 :

$$\begin{aligned} IR &: \{p_1, p_2, r_1, r_2\} \\ IP &: \{p_1, p_2\} \\ I_{EXT} &: p_1 \rightarrow \{(p_1, r_1), (p_2, r_2)\} \\ & p_2 \rightarrow \{(r_1, r_2)\} \\ I_S &: \{(ex:A, p_1), (ex:B, r_1)\} \end{aligned}$$

Interprétation 2 :

$$\begin{aligned} IR &: \{p_1, r_1, r_2\} \\ IP &: \{p_1\} \\ I_{EXT} &: p_1 \rightarrow \{(p_1, p_1), (r_1, r_2)\} \\ I_S &: \{(ex:A, p_1), (ex:B, p_1)\} \end{aligned}$$



FIG. 13 – Graphe RDF

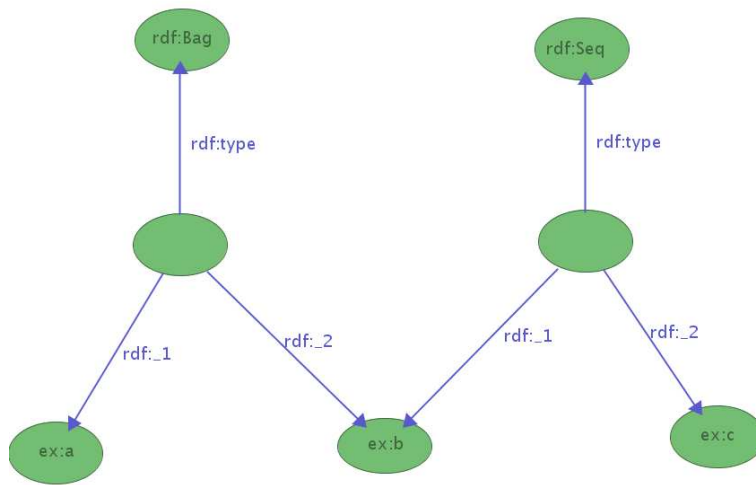


FIG. 14 – Graphe 1

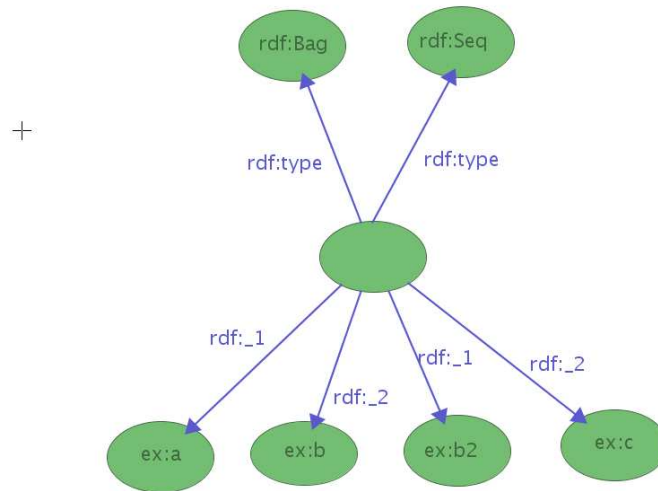


FIG. 15 – Graphe 2

■ Exercice 5.2

a) Considérez les deux graphes RDF illustrés aux figures 14 et 15. Est-il possible de fournir une interprétation qui soit la même pour les deux graphes ? (Justifiez votre réponse)

b) En vous servant des axiomes et des règles d'inférence pour RDF et RDFS, démontrez que le graphe RDF illustré à la figure 16 est une conséquence logique du graphe 1 (indiquez bien la règle utilisée à chaque étape de votre preuve) :

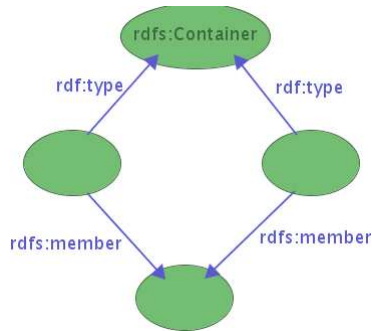


FIG. 16 – Graphe 3

■ Exercice 5.3

Soit la description suivante en RDF :

```
@prefix xmlns: <http://www.polymtl.ca/exemple#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix local: <http://www.polymtl.ca/voc#> .
```

```
local:a rdf:type local:b .
local:b rdf:type local:c .
```

Identifiez, parmi les triplets suivants, ceux qui sont conséquence logique de cette description :

- a) local:a rdf:type local:c .
- b) local:b rdf:type rdfs:Class .
- c) local:b rdfs:domain local:a .

Pour chacun de ces triplets, vous fournirez une preuve en utilisant les règles d'inférence, dans le cas où il est une conséquence logique. Dans le cas contraire, vous fournirez une interprétation du graphe original qui ne satisfait pas le triplet.

6 Inférence

La définition de la sémantique de RDF est utile pour déterminer l'expressivité de ce modèle de donnée. Elle permet d'établir clairement ce que nous voulons signifier lorsque nous construisons un graphe RDF. Nous allons maintenant aborder le problème de l'inférence. Plus précisément, nous voulons savoir si un graphe RDF est une conséquence logique d'un autre graphe RDF. De manière plus précise, supposons l'existence de deux graphes RDF G_1 et G_2 . Supposons maintenant que pour toute interprétation telle que G_1 est vrai, la même interprétation rend nécessairement vrai le graphe G_2 . On dira alors que G_2 est une conséquence logique de G_1 ou, de manière équivalente, que G_1 implique logiquement G_2 . En d'autres mots, cela signifie que chaque fois qu'on aura le graphe G_1 , on pourra déduire le graphe G_2 .

L'importance de l'inférence se révèle particulièrement lorsqu'on consulte un graphe RDF pour vérifier si un fait est vrai selon ce graphe. Il est possible que les faits ne soient pas tous exprimés explicitement par un graphe. Prenons par exemple le graphe suivant :

```
ex:a rdf:type ex:C1 .
ex:C1 rdfs:subClassOf ex:C2 .
```

Selon ce graphe, le fait `ex:a rdf:type ex:C2` est vrai, bien qu'il ne soit pas exprimé explicitement.

En général, pour appliquer une procédure d'inférence, on n'utilise pas la sémantique du langage. On utilise plutôt des règles de déduction qui respectent la sémantique du langage, c'est-à-dire des règles qui nous permettent de déduire de nouveaux faits qui sont nécessairement des conséquences logiques des faits originaux.

Nous allons maintenant présenter les règles de déduction de RDF. Nous utiliserons la même convention que celle adoptée dans le document du W3C : *aaa*, *bbb*, *cccc*, etc. représentent une URI, alors que *xxx*, *yyy*, etc. désignent une URI ou un noeud vide. Un terme *lll* représente un littéral quelconque, et *_ :nnn* un noeud vide, où *nnn* est son identificateur. Nous pouvons maintenant énumérer les règles d'inférence. Toutes les règles ont la forme **Si E contient le triplet T , alors ajouter le triplet T'** . À partir d'un graphe E , on obtient alors un graphe $E' = E \cup \{T'\}$ qui est conséquence logique de E .

6.1 Inférence simple

Voici les règles d'inférence simple :

Règle	Si E contient	Alors ajouter
se1	uuu aaa xxx .	uuu aaa _ :nnn . où _ :nnn désigne un noeud vide alloué à xxx par la règle se1 ou se2.
se2	uuu aaa xxx .	_ :nnn aaa xxx . où _ :nnn désigne un noeud vide alloué à uuu par la règle se1 ou se2.

Remarque importante : Dans ces règles et les suivantes, *allouer un noeud vide* signifie qu'on réutilise le même identificateur de noeud vide s'il a déjà été créé auparavant pour la même URI, ou un nouveau noeud vide si aucun n'a été créé auparavant pour cette URI.

Essentiellement, les règles d'inférence simple permettent de déterminer si un graphe est une instance d'un autre graphe. Par exemple, à partir du graphe suivant :

```
ex:a ex:p ex:b .
ex:c ex:q ex:a .
```

on peut déduire le graphe suivant, en appliquant les règles se1 et se2 :

```
ex:a ex:p ex:b .
ex:c ex:q ex:a .
_:id1 ex:p ex:b .
ex:c ex:q _:id1 .
```

Considérons maintenant le graphe suivant, qui est aussi conséquence logique du graphe original :

```
ex:a ex:p ex:b .
ex:c ex:q ex:a .
_:id1 ex:p ex:b .
ex:c ex:q _:id2 .
```

On ne peut pas obtenir ce graphe par une simple application de se2 suivie de se1. En effet, une fois que l'URI `ex:a` a été associée au noeud `_:id1`, on ne peut pas y associer un autre noeud vide. Pour y arriver, il faudra d'abord associer un noeud vide à `ex:a` dans les deux premiers triplets, puis associer un autre noeud vide à celui obtenu pour le deuxième triplet. Ce qui nous donne la dérivation suivante :

```
ex:a ex:p ex:b .
ex:c ex:q ex:a .
_:id1 ex:p ex:b .      (se2)
ex:c ex:q _:id1 .      (se1)
ex:c ex:q _:id2 .      (se1)
```

On ne peut pas déduire le graphe suivant, où l'on tente de réutiliser un même identificateur de noeud vide pour deux URI différentes :

```
ex:a ex:p ex:b .
ex:c ex:q ex:a .
_:id1 ex:p ex:b .
_:id1 ex:q ex:a .
```

Dans ce dernier cas, le graphe obtenu n'est pas une conséquence logique du graphe original. En effet, supposons que l'interprétation du graphe original est la suivante :

$$\begin{aligned}
 I_R &= \{1, 2, 3\} \\
 I_P &= \{r1, r2\} \\
 I_{EXT} &= r1 \rightarrow \{(1, 2)\} \\
 & \quad r2 \rightarrow \{(3, 1)\} \\
 I_S &= ex:a \rightarrow 1 \\
 & \quad ex:b \rightarrow 2 \\
 & \quad ex:c \rightarrow 3 \\
 & \quad ex:p \rightarrow r1 \\
 & \quad ex:q \rightarrow r2
 \end{aligned}$$

Il est clair que le graphe inféré est faux sous cette interprétation, puisqu'on ne trouve aucune entité x telle que $\langle x, 2 \rangle \in I_{EXT}(r1)$ et $\langle x, 1 \rangle \in I_{EXT}(r2)$.

6.2 Règles de généralisation de littéraux

Pour effectuer des inférences à partir de graphes contenant des littéraux, il nous faut deux règles supplémentaires :

Règle	Si E contient	Alors ajouter
lg	uuu aaa lll .	uuu aaa _ :nnn . où _ :nnn désigne un noeud vide alloué au littéral lll par cette règle
gl	uuu aaa _ :nnn . où _ :nnn désigne un noeud vide alloué au littéral lll par la règle lg	uuu aaa lll .

Ces règles ne prennent leur sens que lorsqu'on considère les règles d'inférence RDF et RDFS que nous verrons dans les sections qui suivent. La règle lg ne sert essentiellement qu'à ajouter une relation spécifiant la classe d'un littéral. Comme un littéral ne peut être le sujet d'un triplet, il faut passer par l'intermédiaire d'un noeud vide lui correspondant pour y arriver. Supposons par exemple le graphe suivant :

```
ex:a ex:p "Michel Gagnon" .
```

En utilisant la règle lg, on peut déduire le graphe suivant :

```
ex:a ex:p "Michel Gagnon" .  
ex:a ex:p _:lit1 .
```

Par le biais d'une règle que nous verrons plus loin, on peut étendre le graphe de la manière suivante, obtenant ainsi un graphe indiquant, de manière indirecte, le type du littéral :

```
ex:a ex:p "Michel Gagnon" .  
ex:a ex:p _:lit1 .  
_:lit1 rdf:type rdfs:Literal .
```

Pour comprendre la règle gl, il faut supposer que la règle lg a été appliquée pour transformer un littéral en un noeud vide. Supposons maintenant que l'on réussisse à déduire un nouveau triplet qui a pour objet ce noeud vide. Il nous faut cette règle pour spécifier que le nouveau triplet a en fait pour objet le littéral en question. Prenons par exemple le graphe suivant :

```
ex:a ex:p "Michel Gagnon" .  
ex:p rdfs:subPropertyOf ex:q .
```

Il n'y a pas, parmi les règles d'inférence que nous verrons plus loin, une règle permettant de déduire le triplet `ex:a ex:q "MichelGagnon"`, qui est une conséquence logique de ce graphe. Par contre, on peut y arriver grâce à la règle gl. Il faut d'abord allouer le noeud vide, en utilisant la règle lg :

```
ex:a ex:p "Michel Gagnon" .  
ex:p rdfs:subPropertyOf ex:q .  
ex:a ex:p _:lit1 .
```

Ensuite, il faut utiliser la règle `rdfs7` que nous verrons plus loin, obtenant ainsi le graphe suivant :

```
ex:a ex:p "Michel Gagnon" .
ex:p rdfs:subPropertyOf ex:q .
ex:a ex:p _:lt1 .
ex:a ex:q _:lt1 .
```

Finalement, par la règle `gl`, nous produisons le triplet désiré :

```
ex:a ex:p "Michel Gagnon" .
ex:p rdfs:subPropertyOf ex:q .
ex:a ex:p _:lt1 .
ex:a ex:q _:lt1 .
ex:a ex:q "Michel Gagnon" .
```

6.3 Inférence RDF

Les règles d'inférence RDF sont les suivantes :

Règle	Si E contient	Alors ajouter
<code>rdf1</code>	<code>uuu aaa yyy .</code>	<code>aaa rdf:type rdf:Property .</code>
<code>rdf2</code>	<code>uuu aaa lll .</code> où <code>lll</code> désigne un littéral XML bien typé	<code>_:nnn rdf:type rdf:XMLLiteral .</code> où <code>_:nnn</code> désigne un noeud vide alloué à <code>lll</code> par la règle <code>lg</code>

Ces règles n'ajoutent pas beaucoup de pouvoir au mécanisme d'inférence. La première règle spécifie que la classe d'un prédicat est nécessairement `rdf:Property`, alors que la seconde règle permet d'explicitement le type d'un littéral XML.

6.4 Inférence RDFS

Voici maintenant les règles d'inférence RDFS, qui complètent l'ensemble des règles requises pour notre mécanisme d'inférence :

Règle	Si E contient	Alors ajouter
rdfs1	uuu aaa lll . où lll désigne un littéral simple (avec ou sans étiquette de langue)	_ :nnn rdf:type rdfs:Literal . où _ :nnn désigne un noeud vide al- loué à lll par la règle lg
rdfs2	aaa rdfs:domain xxx . uuu aaa yyy	uuu rdf:type xxx .
rdfs3	aaa rdfs:range xxx . uuu aaa vvv	vvv rdf:type xxx .
rdfs4a	uuu aaa xxx .	uuu rdf:type rdfs:Resource ..
rdfs4b	uuu aaa vvv .	vvv rdf:type rdfs:Resource .
rdfs5	uuu rdfs:subPropertyOf vvv . vvv rdfs:subPropertyOf xxx .	uuu rdfs:subPropertyOf xxx .
rdfs6	uuu rdf:type rdf:Property .	uuu rdfs:subPropertyOf uuu .
rdfs7	aaa rdfs:subPropertyOf bbb . uuu aaa yyy .	uuu bbb yyy .
rdfs8	uuu rdf:type rdfs:Class .	uuu rdfs:subClassOf rdfs:Resource .
rdfs9	uuu rdfs:subClassOf xxx . vvv rdf:type uuu .	vvv rdf:type xxx .
rdfs10	uuu rdf:type rdfs:Class .	uuu rdfs:subClassOf uuu .
rdfs11	uuu rdfs:subClassOf vvv . vvv rdfs:subClassOf xxx .	uuu rdfs:subClassOf xxx .
rdfs12	uuu rdf:type rdfs:ContainerMembershipProperty .	uuu rdfs:subPropertyOf rdfs:member .
rdfs13	uuu rdf:type rdfs:Datatype .	uuu rdfs:subClassOf rdfs:Literal .

Avant de terminer cette discussion sur l'inférence en RDF et présenter la définition formelle de la conséquence logique en RDFS, il faut considérer une situation de graphe inconsistant. Il s'agit d'un cas où un littéral XML mal formé est déclaré, comme dans l'exemple suivant :

```
ex:a ex:p "<u> a <v> b </u>" ^^ rdf:XMLLiteral .
```

Dans un cas comme celui-ci, la sémantique de RDF impose que l'interprétation du graphe associe à cette chaîne une entité qui n'est pas un littéral. Or, nous pouvons obtenir une contradiction en déduisant de manière indirecte que la chaîne en question est un littéral. Considérons par exemple le graphe suivant :

```
ex:a rdfs:subClassOf rdfs:Literal .
ex:b rdfs:range ex:a
ex:c rdfs:subPropertyOf ex:b
ex:d ex:c "<u> a <v> b </u>" ^^ rdf:XMLLiteral .
```

À partir de ce graphe, on peut inférer le graphe suivant :

```
ex:a rdfs:subClassOf rdfs:Literal .
ex:b rdfs:range ex:a
ex:c rdfs:subPropertyOf ex:b
ex:d ex:c "<u> a <v> b </u>" ^^ rdf:XMLLiteral .
```

```

ex:d ex:c _:1 . (lg)
ex:d ex:b _:1 . (rdfs7)
_:1 rdf:type ex:a . (rdfs3)
_:1 rdf:type rdfs:Literal . (rdfs9)

```

On se retrouve donc avec une contradiction, c'est-à-dire qu'il n'y a aucune interprétation qui peut rendre un tel graphe vrai. Ceci a donc pour conséquence que ce graphe implique logiquement tout graphe RDF. Nous dirons que ce graphe contient un *conflit XML*.

Supposons maintenant qu'un graphe S ne contient pas de conflit XML. Soient A_{RDF} et A_{RDFS} les triplets axiomatiques de RDF et RDFS, respectivement. On dira que le graphe S implique logiquement un graphe E si et seulement si

il existe un graphe G' qu'on peut dériver de $S \cup A_{RDF} \cup A_{RDFS}$ par l'application des règles lg et gl, ainsi que les règles d'inférence RDF et RDFS, et qui implique logiquement E par les règles d'inférence simple.

6.5 Exercices

■ Exercice 6.1

Soit le graphe suivant :

```

ex:a ex:p ex:b .
ex:p rdfs:subPropertyOf ex:q .

```

Montrez que l'énoncé `ex:q rdf:type rdf:Property .` est une conséquence logique de ce graphe.

■ Exercice 6.2

Soit le graphe suivant :

```

ex:Paul ex:connait ex:Jean .
ex:Paul rdf:type ex:Personne .
ex:Jean rdf:type ex:Personne .
ex:Paul ex:nom "Gagnon" .
ex:Jean ex:nom "Gagnon" .

```

Montrez qu'à partir de ce graphe, on peut déduire que deux personnes ont le même nom, c'est-à-dire le graphe suivant :

```

[] rdf:type ex:Personne ;
   ex:nom _:l1 .
[] rdf:type ex:Personne ;
   ex:nom _:l1 .
_:l1 rdf:type rdfs:Literal .

```

■ Exercice 6.3

Soit le graphe suivant :

```
ex:detester rdfs:subPropertyOf ex:avoirSentimentEnvers .
ex:Chat rdfs:ClassOf ex:Animal .
ex:Tom rdf:type ex:Chat .
ex:Jerry rdf:type ex:Animal .
ex:Tom ex:detester ex:Jerry .
```

Montrez qu'à partir de ce graphe, on peut déduire qu'un animal a un sentiment envers un autre animal, c'est-à-dire le graphe suivant :

```
[ ] rdf:type ex:Animal ;
    ex:avoirSentimentEnvers
    [rdf:type ex:Animal] .
```

■ Exercice 6.4

Montrez que l'énoncé suivant est valide (c'est-à-dire vrai pour toutes les interprétations d'un graphe RDF) :

```
rdfs:Class rdf:type rdfs:Class .
```

7 Annexe A - Triplets axiomatiques de RDFS

```
rdf:type rdfs:domain rdfs:Resource .
rdfs:domain rdfs:domain rdf:Property .
rdfs:range rdfs:domain rdf:Property .
rdfs:subPropertyOf rdfs:domain rdf:Property .
rdfs:subClassOf rdfs:domain rdfs:Class .
rdf:subject rdfs:domain rdf:Statement .
rdf:predicate rdfs:domain rdf:Statement .
rdf:object rdfs:domain rdf:Statement .
rdfs:member rdfs:domain rdfs:Resource .
rdf:first rdfs:domain rdf:List .
rdf:rest rdfs:domain rdf:List .
rdfs:seeAlso rdfs:domain rdfs:Resource .
rdfs:isDefinedBy rdfs:domain rdfs:Resource .
rdfs:comment rdfs:domain rdfs:Resource .
rdfs:label rdfs:domain rdfs:Resource .
rdf:value rdfs:domain rdfs:Resource .
```

```
rdf:type rdfs:range rdfs:Class .
rdfs:domain rdfs:range rdfs:Class .
rdfs:range rdfs:range rdfs:Class .
rdfs:subPropertyOf rdfs:range rdf:Property .
rdfs:subClassOf rdfs:range rdfs:Class .
rdf:subject rdfs:range rdfs:Resource .
rdf:predicate rdfs:range rdfs:Resource .
rdf:object rdfs:range rdfs:Resource .
```

```
rdfs:member rdfs:range rdfs:Resource .
rdf:first rdfs:range rdfs:Resource .
rdf:rest rdfs:range rdf:List .
rdfs:seeAlso rdfs:range rdfs:Resource .
rdfs:isDefinedBy rdfs:range rdfs:Resource .
rdfs:comment rdfs:range rdfs:Literal .
rdfs:label rdfs:range rdfs:Literal .
rdf:value rdfs:range rdfs:Resource .

rdf:Alt rdfs:subClassOf rdfs:Container .
rdf:Bag rdfs:subClassOf rdfs:Container .
rdf:Seq rdfs:subClassOf rdfs:Container .
rdfs:ContainerMembershipProperty rdfs:subClassOf rdf:Property .

rdfs:isDefinedBy rdfs:subPropertyOf rdfs:seeAlso .

rdf:XMLLiteral rdf:type rdfs:Datatype .
rdf:XMLLiteral rdfs:subClassOf rdfs:Literal .
rdfs:Datatype rdfs:subClassOf rdfs:Class .

rdf:_1 rdf:type rdfs:ContainerMembershipProperty .
rdf:_1 rdfs:domain rdfs:Resource .
rdf:_1 rdfs:range rdfs:Resource .
rdf:_2 rdf:type rdfs:ContainerMembershipProperty .
rdf:_2 rdfs:domain rdfs:Resource .
rdf:_2 rdfs:range rdfs:Resource .
...
```