SPARQL Template : un langage de Pretty Printing pour RDF

Olivier Corby, Catherine Faron Zucker

SPARQL Template : un langage de Pretty Printing pour RDF

Olivier Corby[1] & Catherine Faron-Zucker[2]

[1] INRIA Sophia-Antipolis Méditerranée, 06900 Sophia Antipolis, France
olivier.corby@inria.fr

[2] Univ. Nice Sophia Antipolis, CNRS, I3S, UMR 7271, 06900 Sophia Antipolis, France
faron@i3s.unice.fr

Abstract : RDF est un langage de représentation de connaissances basé sur des graphes étiquetés, conçu par le W3C pour le Web sémantique et le Web des données. En tant que langage d'échange pivot, il peut être utilisé pour représenter des arbres de syntaxe abstraite (AST) de langages. Par exemple le langage OWL a plusieurs syntaxes dont une syntaxe fonctionnelle et une syntaxe RDF, de même que le langage RIF (Rule Interchange Format) ; SPIN est une notation qui permet de représenter des requêtes SPARQL en RDF.

Cet article traite du problème de la transformation d'un arbre abstrait RDF d'un langage dans sa syntaxe concrète (appelée *pretty print*). Nous proposons une approche générique pour écrire des pretty printers basés sur SPARQL pour des AST RDF. Nous définissons un pretty printer comme un ensemble de règles de transformation traitées par un moteur de pretty print. Nous proposons une extension syntaxique de SPARQL, appelée SPARQL Template, pour faciliter l'écriture des règles de transformation et l'implémentation du moteur de transformation. Nous montrons la faisabilité de notre approche en présentant deux exemples de pretty printers opérationnels pour les langages OWL et SPIN.

Mots-clés : RDF Pretty Printing, RDF AST, SPARQL Template

1 Introduction

RDF est un langage de représentation de connaissances basé sur des graphes étiquetés, conçu pour le Web sémantique et le Web des données. En tant que langage d'échange pivot, il peut être utilisé pour représenter des arbres de syntaxe abstraite (AST) de langages.

1.1 AST RDF

Un AST est le résultat de l'analyse syntaxique (parsing) d'un texte ou d'un programme dans un langage donné, depuis sa syntaxe concrète vers un arbre de syntaxe abstraite.

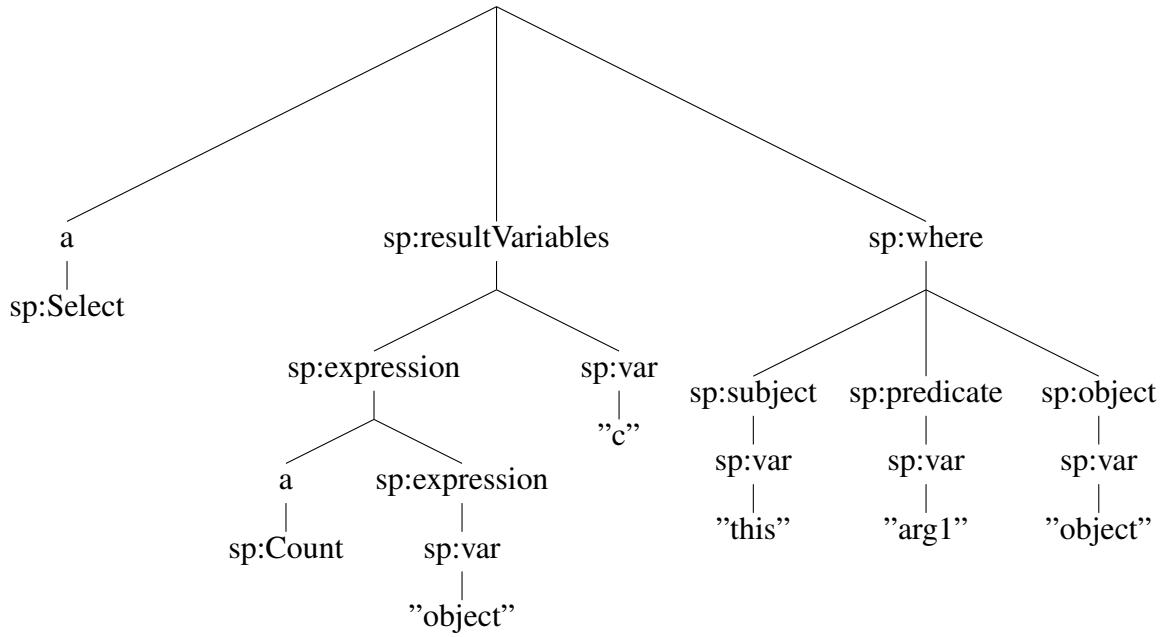
Les standards OWL et RIF sont des langages bien connus du web sémantique, munis d'une syntaxe RDF : *OWL 2 Mapping to RDF*, Patel-Schneider & Motik (2012) et *RIF in RDF*, Hawke & Polleres (2012). Un autre exemple d'AST en RDF est le format SPIN, Knublauch (2011), qui permet de représenter des requêtes SPARQL en RDF. Voici un exemple issu du document "W3C Member submission SPIN" :

```
SELECT (COUNT(?object) as ?c)
WHERE {
  ?this ?arg1 ?object
}
```

La requête SPARQL précédente est représentée par le graphe SPIN RDF suivant, où les lignes (03-09) représentent la clause SELECT et les lignes (10-14) la clause WHERE :

```
(01) @prefix sp: <http://spinrdf.org/sp#> .
(02) [] a  sp:Select ;
(03)       sp:resultVariables ([
(04)         sp:expression
(05)           [ a sp:Count ;
(06)             sp:expression [ sp:varName "object" ]
(07)           ] ;
(08)         sp:varName "c"
(09)       ]) ;
(10)      sp:where ([
(11)         sp:subject   [ sp:varName "this" ] ;
(12)         sp:predicate [ sp:varName "arg1" ] ;
(13)         sp:object    [ sp:varName "object" ]
(14)       ]) .
```

La requête SPIN précédente représente l'AST suivant :



Un AST en RDF est constitué de sommets et d'arcs où les sommets sont des ressources (URI), des blank nodes (ressources anonymes représentant des variables existentielles) ou des littéraux (des constantes). Les arcs sont des triplets RDF étiquetés par des noms de propriétés. En SPIN, les sommets intermédiaires sont des blank nodes et les feuilles sont des URI et des littéraux.
Remarquons que dans un graphe RDF les arcs d'un sommet ne sont pas ordonnés alors que dans un AST ils peuvent l'être. Quand un ordre est nécessaire, on peut utiliser une liste RDF ou

bien le Container RDF rdf:Seq. SPIN utilise une convention de nommage ad hoc : `sp:arg1`, `sp:arg2`, etc. pour ordonner les arguments.

Nous constatons donc qu'il existe un certain nombre de standards ayant une représentation sous forme d'AST RDF tels que OWL, SPIN ou RIF. Etant donné le caractère standard de RDF pour l'échange d'information, nous pouvons imaginer qu'il existe (et qu'il existera) d'autres langages ayant un AST en RDF. Ce principe de représentation d'AST RDF peut être généralisé à n'importe quel langage. Par exemple, Follenfant *et al.* (2012) proposent une syntaxe RDF pour des requêtes SQL. Egalement, nous avons prototypé un AST RDF pour des expressions mathématiques (en notation préfixe) dont la représentation pourrait ainsi être embarquée dans l'annotation RDF d'un article scientifique par exemple.

Si l'on considère le fait que RDF peut servir à décrire des énoncés de langages sous forme d'AST, on constate immédiatement que l'on peut interroger ces énoncés à l'aide de requêtes SPARQL. Par exemple : trouver les requêtes SPIN opérant sur la propriété `foaf:name`, rechercher les classes OWL sous-classes de `foaf:Human`, etc. Se pose alors le problème de la présentation des résultats des requêtes SPARQL. En effet, une requête SPIN est représentée sous forme d'un blank node RDF, de même pour les classes OWL définies et le résultat d'une requête SPARQL recherchant une requête SPIN ou une classe OWL est donc le blank node les représentant. Pour améliorer l'interaction avec l'utilisateur il est souhaitable de produire plutôt les requêtes SPIN ou les classes OWL résultats dans la syntaxe concrète de ces langages, c'est à dire dans la syntaxe concrète de SPARQL ou la syntaxe fonctionnelle de OWL.

Ainsi la possibilité de représenter des énoncés de langage sous forme d'AST RDF mène naturellement à la question de la présentation de tels énoncés dans leur syntaxe concrète, et donc de la possibilité de pouvoir réaliser une opération appelée *pretty printing* en théorie de langages. En généralisant cette question, nous posons le problème de la présentation de graphe RDF dans un format compréhensible par un utilisateur. Un autre scénario relatif à la présentation de résultats de requêtes SPARQL est par exemple la production de pages HTML issues de descriptions RDF, par exemple pour réaliser une navigation hypertexte dans une base RDF, telle que DBpedia.

1.2 Langages de Pretty Printing

Comme nous l'avons dit précédemment, un AST est le résultat d'une analyse syntaxique (parsing). Le problème inverse du *parsing*, appelé *pretty printing*, consiste à engendrer la syntaxe concrète d'un énoncé d'un langage à partir de son AST. Des langages spécifiques ont été conçus pour résoudre ce problème comme PPML, Théry (2003), pour les langages de programmation et XSLT, Kay (2007), pour XML. Ces langages reposent sur des règles de transformation déclaratives qui s'appliquent aux différents énoncés du langage cible.

PPML a été conçu dans le cadre du générateur d'environnement de programmation *Centaur*, Borras *et al.* (1988). Une règle a une partie gauche qui est une description du sous-arbre à pretty printer, et une partie droite qui est la spécification de la présentation à engendrer. L'exemple suivant montre une règle pour un énoncé addition :

```
plus(*x, *y) -> [<h> *x "+" *y];
```

XSLT a été conçu pour transformer et pretty printer des arbres XML. La même règle pour l'addition est montrée ci-dessous :

```
<xsl:template match='plus'>
  <xsl:apply-templates select='x' />
  <xsl:text> + </xsl:text>
  <xsl:apply-templates select='y' />
</xsl:template>
```

Plus particulièrement, nous avons identifié des langages de pretty print pour le Web sémantique. Le plus connu est Fresnel, Bizer *et al.* (2005), conçu pour engendrer des formats de présentation pour RDF. Il est muni d'un vocabulaire RDF permettant de définir des formats de présentation pour des types de ressources RDF et de spécifier quelles propriétés doivent être affichées et comment. Voici un exemple d'énoncé Fresnel qui indique les propriétés à afficher pour une ressource de type `foaf:Person` :

```
PersonLens a fresnel:Lens ;
  fresnel:classLensDomain foaf:Person ;
  fresnel:showProperties (
  foaf:name
  foaf:mbox
  foaf:depiction
) .
```

Voici un format de présentation pour la propriété `foaf:name` :

```
:nameFormat a fresnel:Format ;
  fresnel:label "Name" ;
  fresnel:propertyFormatDomain foaf:name .
```

Xenon RDF Stylesheet, Quan (2005), est un langage inspiré de XSLT, conçu pour appliquer des lentilles (*lenses*) sur des données RDF. La syntaxe de Xenon est RDF/Turtle utilisé comme un AST. Un énoncé de base du langage est ainsi un triplet RDF avec la propriété `xe:applyTemplates`. Le langage SPARQL est utilisé pour sélectionner les templates et les ressources dans le graphe à afficher. Le format du résultat est XHTML.

OWL-PL, Brophy & Heflin (2009), est une adaptation et une extension de XSLT pour transformer RDF/OWL en XHTML. Il vise à adapter le traitement des arbres XML aux graphes RDF. En particulier il permet d'apparier des propriétés de ressources au lieu de nœuds XML.

Alkhateeb & Laborie (2008) proposent une extension du langage SPARQL pour engendrer un document XML en réponse à une requête. Une requête SPARQL est complétée avec un template XML qui référence des variables SPARQL. Ces variables sont liées à des valeurs solutions par l'appariement d'une clause WHERE standard et le template est complété avec les valeurs de ces variables. C'est une requête de type CONSTRUCT où le template est en XML au lieu d'être en RDF.

1.3 Problèmes de recherche posés

Dans cet article, nous répondons aux problèmes de recherche suivants :

1. Comment engendrer la syntaxe concrète d'un langage depuis un AST en RDF, par exemple pour engendrer des énoncés OWL en syntaxe fonctionnelle à partir de la représentation en RDF d'une ontologie OWL.

2. Comment proposer une solution *générique* au pretty-print d'AST RDF, indépendante du langage traité et donc de la syntaxe concrète à engendrer.

3. Comment utiliser SPARQL pour écrire des règles de pretty printing.

2 SPARQL comme langage de Pretty Printing

De manière générale, une règle de pretty print comporte une condition qui s'apparie à un énoncé et une présentation qui définit le résultat du pretty print de l'énoncé filtré par la condition. Le langage SPARQL est un bon candidat pour un langage de pretty-print : dans une requête de la forme SELECT, la clause WHERE joue le rôle de la condition qui sélectionne un sommet de l'AST et la clause SELECT retourne un résultat. Considérons par exemple l'énoncé OWL suivant représentant une *restriction* de propriété dans la syntaxe RDF :

```
[
 a owl:Restriction ;
   owl:onProperty ex:hasFather ;
   owl:allValuesFrom ex:Man
]
```

La requête ci-dessous sélectionne de telles *restrictions* OWL :

```
(01) SELECT ?p ?c
(02) WHERE {
(03)  ?in a owl:Restriction ;
(04)    owl:onProperty ?p ;
(05)    owl:allValuesFrom ?c .
(06) }
```

La clause SELECT retourne les valeurs associées aux variables, mais pas (encore) le pretty print de la *restriction* en syntaxe fonctionnelle.

Supposons maintenant que nous disposons d'un ensemble de règles de pretty print, telles que celle ci-dessus, pour tous les énoncés du langage considéré, ici OWL. Supposons également que nous disposons d'une fonction `st:apply-templates` qui, appelée sur un sommet, retourne le résultat de son pretty print par les règles de présentation appropriées. Nous pouvons alors compléter la règle comme suit :

```
(01) SELECT {
(02)  (st:apply-templates(?p) as ?pp)
(03)  (st:apply-templates(?c) as ?pc)
(04) }
(05) WHERE {
(06)  ?in a owl:Restriction ;
(07)    owl:onProperty ?p ;
(08)    owl:allValuesFrom ?c .
(09) }
```

Nous avons alors (quasiment) résolu le problème. `?pp` est le résultat du pretty-print de `onProperty` et `?pc` de `allValuesFrom`. Il reste simplement à compléter par le pretty-print de la restriction. Par convention le résultat final est lié à la variable `?out`, le sommet courant étant `?in`. Ainsi, la requête finale permettant le pretty-print de la restriction est la suivante :

```
(01) SELECT {
(02)  (st:apply-templates(?p) as ?pp)
(03)  (st:apply-templates(?c) as ?pc)
(04)  (concat("ObjectAllValuesFrom(",?pp," ",?pc,")") as ?out)
(05) }
(06) WHERE {
(07)  ?in a owl:Restriction ;
(08)    owl:onProperty ?p ;
(09)    owl:allValuesFrom ?c .
(10) }
```

SPARQL permet donc d'écrire des règles de pretty-print, modulo l'écriture de la fonction d'extension `st:apply-templates` que nous allons détailler dans la suite de l'article.

3 Extension de SPARQL pour représenter des templates de transformation

Pour simplifier l'écriture des règles de pretty print, nous proposons une extension de la syntaxe du langage SPARQL, appelée SPARQL Template. La partie WHERE est standard et la partie TEMPLATE permet d'écrire directement le format de présentation à engendrer. Le template ci-dessous correspond à la requête ci-dessus.

```
(01) TEMPLATE {
(02)   "ObjectAllValuesFrom(" ?p " " ?c ")"
(03) }
(04) WHERE {
(05)  ?in a owl:Restriction ;
(06)    owl:onProperty ?p ;
(07)    owl:allValuesFrom ?c .
(08) }
```

Les règles de pretty print se focalisent sur un sommet de l'arbre pour lequel il s'agit de retourner un format de présentation. On appelle *focus node* le sommet courant qui fait l'objet d'un calcul de présentation à un instant donné du parcours de l'arbre. Le focus node est matérialisé dans une règle de pretty print par la variable `?in`, c'est une convention syntaxique car la notion de focus node n'existe pas en SPARQL. A partir d'un focus node `?in` dans la partie WHERE, le pretty printer parcourt l'arbre dans son voisinage à la recherche de ses voisins d_i. Toute mention d'un voisin d_i dans la partie TEMPLATE fait de celui-ci un nouveau focus node du moteur de pretty print, et cela récursivement jusqu'aux feuilles de l'arbre. Une occurrence d'un d_i dans la partie TEMPLATE est interprétée comme : *insérer ici le résultat du pretty print de* d_i.

Le résultat retourné par un template est par convention la valeur liée à la variable `?out` ; c'est le résultat retourné par la fonction `st:apply-templates` appliquée à cette variable. Etant donnée une liste de templates, la fonction `st:apply-templates` évalue successivement les templates jusqu'à ce qu'un template réussisse sur le focus node, c'est-à-dire retourne un résultat. Si aucun template ne réussit, le sommet lui-même est retourné. Dans le cas où la partie WHERE retourne plusieurs solutions, la partie TEMPLATE est appliquée autant de fois qu'il y a de solutions et le résultat final est la concaténation des résultats élémentaires.

L'évaluateur SPARQL est appelé avec une liaison dynamique de la variable IN (`?in`) qui est liée au focus node. Les appels récursifs à la fonction `st:apply-templates` réalisent le parcours récursif de l'AST sur les focus nodes. Le pseudo-code ci-dessous donne un aperçu du principe de la fonction `st:apply-templates`.

```
(01) Node st:apply-templates(Node node){
(02)   for (Query q : getTemplates()){
(03)     Mappings map = eval(q, IN, node);
(04)     Node res = map.getResult(OUT);
(05)     if (res != null) return res;
(06)   }
(07)   return node;
(08) }
```

Outre ce code, le pretty printer teste et gère les éventuels cycles dans le cas où le graphe RDF a des cycles (cas général). Une pile garde la trace des templates appliqués et des focus nodes de telle sorte que l'on n'applique pas deux fois le même template sur le même sommet.

3.1 Syntaxe

Nous présentons ici la syntaxe de l'extension TEMPLATE dans la syntaxe de SPARQL 1.1 Query Language, Harris & Seaborne (2013).
`Prologue` définit les préfixes, `PrimaryExpression` est une constante, une variable, un appel de fonction ou une expression parenthésée.

```
Template ::= Prologue
             'template' (iri) ? '{'
                  ( PrimaryExpression | Group ) *
                  ( Separator ) ? '}'
             WhereClause
             SolutionModifier
             ValuesClause

Group ::= 'group' ( 'distinct' ) ? '{'
             PrimaryExpression *
             ( Separator ) ? '}'

Separator ::= ';' 'separator' '=' String
```

Le namespace du langage est le suivant :

```
prefix st: <http://ns.inria.fr/sparql-template/>
```

3.2 Compilation

Nous avons développé un compilateur qui traduit les templates en requêtes SPARQL standard SELECT-WHERE. Nous présentons ici le schéma de compilation d'un template sous forme d'une requête SELECT-WHERE, au moyen d'une fonction de traduction `tr`. Cette fonction remplace les variables `V` par `coalesce(st:apply-templates(V), "")` et concatène les résultats avec `concat()`.

```
(01) tr(template(List l)) -> select (concat(tr(l)) as ?out)
(02) tr(List l) -> List(tr(ei)) pour ei élément de l
(03) tr(Literal d) -> d
(04) tr(Variable v) -> coalesce(st:apply-templates(v), "")
(05) tr(group(List l)) -> group_concat(tr(l))
(06) tr(Exp f) -> f
```

La clause `WHERE` et tout ce qui la suit sont laissés tels quels.

Soit Q la requête SPARQL SELECT-WHERE résultat de la compilation d'un template T. Soit M le multiset de solutions résultat de l'évaluation de Q. Le résultat de l'évaluation de T est calculé en appliquant un aggrégat `group_concat` sur la variable `?out` sur le multiset M :

```
Aggregation((?out), group_concat, separator, M)
```

3.3 Fonctions de pretty print

Les appels de fonction sont autorisés dans la clause TEMPLATE, comme dans une requête SPARQL standard, comme par exemple `xsd:string(?x)`. Nous avons défini un ensemble de fonctions de "pretty print" :

- `st:turtle(term)` renvoie un terme RDF au format Turtle.
- `st:uri(term)` renvoie `st:turtle(term)` si l'argument est un URI ; sinon renvoie `st:apply-templates(term)`.
- `st:call-template(name, term)` exécute un template nommé sur un focus node. Elle s'apparente à la fonction XSL `xsl:call-template`.
- `st:call-templates-with(pp, name, term)` similaire à la fonction précédente en précisant un pretty printer.
- `st:apply-templates(term)` appelle le pretty printer sur un focus node. Elle s'apparente à la fonction XSL `xsl:apply-templates`.
- `st:apply-templates-with(pp, term)` similaire à la fonction précédente en précisant un pretty printer.

- `st:apply-all-templates(term)` appelle le pretty printer sur un focus node et exécute tous les templates qui s'appliquent ; renvoie la concaténation des résultats des différents templates. Il est possible de spécifier un séparateur :
  `st:apply-all-templates(?x ; separator = ", ").`

Nous avons étendu l'interprète SPARQL de telle manière qu'il puisse évaluer *directement* une requête SPARQL de la forme TEMPLATE (en plus des cinq formes SELECT, CONSTRUCT, DESCRIBE, ASK, UPDATE).

3.4 Patrons de conception

Si la base de templates considérée contient un template nommé `st:start`, il est utilisé comme template de départ par la fonction `st:apply-templates-with`. Dans le cas contraire, le premier template qui réussit est le point de départ.

Les templates nommés sont appelés explicitement avec la fonction `st:call-template`. Il est possible d'exécuter tous les templates s'appliquant sur un focus node avec la fonction `st:apply-all-templates()` :

```
(01) TEMPLATE {
(02)    st:apply-all-templates(?x ; separator = "\n")
(03) }
(04) WHERE {
(05)   ...
(06) }
```

Il est possible d'effectuer des calculs en utilisant des appels récursifs sur des templates nommés. Il est par exemple possible d'engendrer le développement de la fonction factorielle avec le template suivant :

```
(01 TEMPLATE st:rec {
(02)  if (?i = 1, 1,
(03)    concat(?i, " * ", st:call-template(st:rec, ?i - 1)))
(04) }
(05) WHERE {
(06)    bind(?in as ?i)
(07) }
```

Le résultat de l'exécution de `st:call-template(st:rec, 3)` est : `3 * 2 * 1`

4 Applications et validation

SPARQL Template est implémenté dans la plate-forme Web sémantique Corese[1], Corby *et al.* (2012); Corby & Faron-Zucker (2010). Cinq pretty printers sont actuellement définis, pour les

[1] `http://wimmics.inria.fr/corese`

langages OWL, SPIN, SQL, Turtle et HTML, ainsi qu'un pretty printer d'expressions mathématiques en RDF vers Latex. Ces pretty printers sont accessibles en ligne[2].

Le pretty printer SPIN a été testé avec succès sur la base de tests du W3C pour SPARQL 1.1 Query & Update[3] (444 requêtes testées). Le pretty printer OWL a été testé avec succès sur l'ontology du W3C OWL 2 Primer[4], avec un chargement de la syntaxe fonctionnelle engendrée dans Protégé pour validation.

5 Discussions

5.1 Liaison dynamique de variables

Pour exécuter des templates, l'interprète SPARQL doit être capable de traiter les liaisons dynamiques de variables pour le focus node. En principe, cela pourrait être fait en SPARQL 1.1 en utilisant la clause `VALUES` qui est faite pour cela. Mais cette clause ne permet pas de passer des blank nodes en argument. Or les AST reposent en grande partie sur les blank nodes. Il est possible de réaliser le passage dynamique d'argument en utilisant une fonction d'extension dans un énoncé `bind` dans la partie WHERE comme le montre l'exemple suivant :

```
TEMPLATE { ... }
WHERE {
   bind(st:getFocusNode() as ?in)
   ...
}
```

5.2 Ordre de priorité sur les templates

Il peut être nécessaire de spécifier une priorité sur les templates pour pouvoir les considérer dans un certain ordre. Pour cela, nous utilisons une clause PRAGMA non standard avec laquelle nous avons étendu le langage SPARQL, dans laquelle nous exprimons une priorité avec une propriété `st:priority`:

```
TEMPLATE { }
WHERE { }
PRAGMA { st:template st:priority 1 }
```

5.3 Templates et régimes d'inférence

Notre interprète SPARQL implémente pour les requêtes de la forme TEMPLATE les mêmes régimes d'inférence que pour les formes standards de requêtes SPARQL (SPARQL 1.1 Entailment Regimes, Glimm & Ogbuji (2013)). Un AST peut donc être typé par une ontologie du langage cible (e.g. des classes d'énoncés) et les templates peuvent exploiter les inférences issues de cette ontologie.

[2] `ftp://ftp-sop.inria.fr/wimmics/soft/pprint`

[3] `http://www.w3.org/2009/sparql/docs/tests/data-sparql11/`

[4] `http://www.w3.org/TR/owl2-primer/`

5.4 Usages particuliers de pretty printers

Etant donné un langage muni d'une syntaxe RDF, SPARQL peut être utilisé pour rechercher des énoncés dans un tel langage, comme par exemple interroger une ontologie OWL pour rechercher les classes OWL reliées à la classe `foaf:Person`. Notre pretty printer permet alors de retourner à l'utilisateur les résultats dans la syntaxe concrète du langage cible, dans le cas de OWL, dans sa syntaxe fonctionnelle. Ceci ouvre des perspectives en termes d'interactions homme-machine, comme le montre l'exemple ci-dessous à la ligne (02) :

```
(01) SELECT ?x
(02)   (st:apply-templates-with(st:owl, ?x) as ?t)
(03) WHERE {
(04)   ?x a owl:Class ;
(05)      rdfs:subClassOf* foaf:Person
(06) }
```

Le pretty printer peut également être utilisé dans un filtre pour faire une recherche plein texte sur des énoncés dans la syntaxe concrète de leur langage.

```
(01) SELECT * WHERE {
(02)   ?x a owl:Class
(03)   FILTER(contains(
(04)      st:apply-templates-with(st:owl, ?x),
(05)      "intersectionOf"))
(06) }
```

6 Conclusion

Nous avons présenté une méthode générale pour écrire des pretty printers pour des arbres de syntaxe abstraite écrits en RDF. La méthode s'applique également à des graphes RDF quelconques, c'est-à-dire qu'il est possible d'écrire un pretty printer dédié à un domaine de connaissance quelconque modélisé en RDF. Un pretty printer est un ensemble de règles de transformation écrites dans une extension de SPARQL appelée SPARQL Template.

Nous avons validé cette méthode en réalisant des pretty printers opérationnels pour SPIN, OWL, SQL Turtle. La méthode est implémentée dans la plate-forme Web sémantique Corese. Un pretty printer peut être écrit pour engendrer la syntaxe concrète d'un langage quelconque (pourvu que ce langage soit muni d'une syntaxe RDF), mais aussi des énoncés en langue naturelle à partir de données RDF, du code HTML, etc. Nous travaillons actuellement au développement d'un pretty-printer permettant d'engendrer une représentation en Latex d'expressions mathématiques contenues dans des données RDF.

Remerciements

Nous remercions Abdoul Macina et Corentin Follenfant pour leur participation à l'écriture du pretty printer pour SQL.

References

ALKHATEEB F. & LABORIE S. (2008). Towards Extending and Using SPARQL for Modular Document Generation. In *Proc. of the 8th ACM Symposium on Document Engineering*, p. 164–172, Sao Paulo, Brésil: ACM Press.

BIZER C., LEE R. & PIETRIGA E. (2005). Fresnel - A Browser-Independent Presentation Vocabulary for RDF. In *Second International Workshop on Interaction Design and the Semantic Web @ ISWC'05*, Galway, Ireland.

BORRAS P., CLÉMENT D., DESPEYROUX T., INCERPI J., KAHN G., LANG B. & PASCUAL V. (1988). Centaur: the system. In *Proc. SIGSOFT, 3rd Annual Symposium on Software Development Environments*, Boston, USA.

BROPHY M. & HEFLIN J. (2009). *OWL-PL: A Presentation Language for Displaying Semantic Data on the Web*. Technical report, Department of Computer Science and Engineering, Lehigh University.

CORBY O. & FARON-ZUCKER C. (2010). The KGRAM Abstract Machine for Knowledge Graph Querying. In *IEEE/WIC/ACM International Conference*, Toronto, Canada.

CORBY O., GAIGNARD A., FARON-ZUCKER C. & MONTAGNAT J. (2012). KGRAM Versatile Data Graphs Querying and Inference Engine. In *Proc. IEEE/WIC/ACM International Conference on Web Intelligence*, Macau.

FOLLENFANT C., CORBY O., GANDON F. & TRASTOUR D. (2012). RDF Modelling and SPARQL Processing of SQL Abstract Syntax Trees. In *Programming the Semantic Web, ISWC Workshop*, Boston, USA.

GLIMM B. & OGBUJI C. (2013). *SPARQL 1.1 Entailment Regimes*. W3C Recommendation, W3C. http://www.w3.org/TR/sparql11-entailment/.

HARRIS S. & SEABORNE A. (2013). *SPARQL 1.1 Query Language*. Recommendation, W3C. http://www.w3.org/TR/sparql11-query/.

HAWKE S. & POLLERES A. (2012). *RIF In RDF*. Working Group Note, W3C. http://www.w3.org/TR/rif-in-rdf/.

KAY M. (2007). *XSL Transformations (XSLT) Version 2.0*. Recommendation, W3C. http://www.w3.org/TR/xslt20/.

KNUBLAUCH H. (2011). *SPIN - SPARQL Syntax*. Member Submission, W3C. http://www.w3.org/Submission/2011/SUBM-spin-sparql-20110222/.

PATEL-SCHNEIDER P. & MOTIK B. (2012). *OWL 2 Web Ontology Language Mapping to RDF Graphs (Second Edition)*. Recommendation, W3C. http://www.w3.org/TR/owl-mapping-to-rdf/.

QUAN D. (2005). Xenon: An RDF Stylesheet Ontology. In *Proc. WWW*.

THÉRY L. (2003). *A Table-Driven Compiler for Pretty Printing Specifications*. Technical Report RT 0288, Inria. http://hal.inria.fr/docs/00/06/98/91/PDF/RT-0288.pdf.