Un langage et un serveur de transformation de graphes pour le Web de données

Olivier Corby, Catherine Faron Zucker

Université Côte d'Azur, Inria, CNRS, I3S, France
olivier.corby@inria.fr,faron@unice.fr

RÉSUMÉ. *Dans cet article nous commençons par présenter le langage STTL (SPARQL Template Transformation Language) pour la transformation de graphes RDF, conçu comme une extension du langage SPARQL. Nous présentons ensuite une plate-forme qui implémente ce langage pour permettre la conception de navigateurs Web offrant une navigation hypertextuelle en HTML dans des graphes RDF sur le Web de données. La plateforme se présente sous la forme d'un serveur HTTP embarquant, outre un service SPARQL, un moteur de transformation STTL et des services Web permettant d'exécuter des transformations. Nous montrons les capacités du système en présentant quatre navigateurs : un premier pour exécuter des requêtes SPARQL sur un graphe RDF local ou sur le Web et pour présenter les résultats en HTML ; un second pour naviguer sur le graphe RDF de DBpedia avec des formats de présentation dédiés à certains types de ressources ; un troisième pour présenter une vue unifiée d'un graphe local lié au graphe de DBpedia ; et un dernier pour gérer et exploiter une base de requêtes SPARQL prédéfinies.*

ABSTRACT. *In this paper we first present the SPARQL Template Transformation Language (STTL) for the transformation of RDF graphs, designed as an extension of SPARQL. Then we present a STTL-based platform for the design of Web browsers for the Linked Data. This platform is a SPARQL endpoint augmented with an STTL engine and services to run STTL transformations. Finally, we show the capabilities of the platform by presenting several Web browsers: a first one to execute SPARQL queries and present their results in HTML; a second one to navigate on the DBpedia RDF graph with HTML presentation formats adapted to certain types of data; a third one that generates a mashup of a local RDF graph with the DBpedia RDF graph and provides a unified view; and a last one to manage and exploit a base of predefined SPARQL queries.*

MOTS-CLÉS : *Transformation RDF, Web de données, Web sémantique.*

KEYWORDS: *RDF transformation, Web of Data, Semantic Web.*

DOI:10.3166/RIA.30.607-627

1. Introduction

Le Web de documents structurés qui repose sur le standard XML a été muni du standard XSLT pour engendrer des formats de présentation tels que HTML ou bien pour écrire des transformations de données XML d'un schéma XML vers un autre. De la même manière, le Web de données qui repose sur le standard RDF a maintenant besoin d'un langage de transformation pour présenter les données RDF dans des formats lisibles tels que HTML et pour écrire des transformations de RDF vers RDF.

La transformation et la présentation de données RDF est encore un problème ouvert. Nous avons proposé dans (Corby, Faron-Zucker, 2014) et (Corby, Faron-Zucker, 2015b) un langage de règles de transformation pour RDF, *SPARQL Template Transformation Language* (STTL), dont le format de sortie est textuel, sans restriction a priori sur le format de texte. L'état de l'art sur les autres solutions existantes pour transformer des données RDF présenté dans les articles ci-dessus mentionnés montre qu'elles sont toutes liées à la syntaxe XML ou à un format spécifique de sortie, ou aux deux, sauf Fresnel (Bizer *et al.*, 2005). Mais Fresnel s'intéresse à la présentation des données RDF et ne traite pas du problème plus général de leur transformation. STTL permet une approche générique pour écrire des transformations RDF vers différents formats de sortie.

Nous nous intéressons dans cet article plus particulièrement à la transformation de données RDF en HTML et nous présentons une plate-forme basée sur le langage STTL, nommée ALIGATOR (acronyme de *A LInked data naviGATOR*), et permettant de concevoir des navigateurs pour le Web des données liées et de présenter ces données en HTML. ALIGATOR est constitué d'un serveur HTTP, un service REST, un moteur de transformations STTL et une bibliothèque de transformations STTL. Le code HTML engendré par les transformations contient des liens hypertextes vers le serveur ce qui permet d'offrir une navigation hypertextuelle sur des graphes RDF.

Cet article est organisé comme suit : la section 2 présente brièvement le langage STTL, la section 3 présente le service REST appelant le moteur de transformations STTL. La section 4 présente différentes applications Web permettant de présenter et naviguer sur des données RDF.

2. Le langage de transformation STTL

Le langage STTL que nous avons conçu est une extension du langage *SPARQL 1.1 Query* avec une clause TEMPLATE permettant d'engendrer un résultat sous forme de texte à partir des solutions d'une clause WHERE. La clause TEMPLATE peut contenir du texte, des variables et des expressions. Les variables sont remplacées par les valeurs trouvées dans les solutions, les expressions sont évaluées et le tout est concaténé sous forme d'une chaîne de caractères qui est le résultat retourné par la clause TEMPLATE. L'exemple suivant montre la traduction d'un énoncé OWL exprimé dans la syntaxe RDF de OWL — une restriction de type `allValuesFrom` — dans la syntaxe fonctionnelle de OWL.

```
TEMPLATE {
  "allValuesFrom(" ?p " " ?c ")"
}
WHERE {
  ?in a owl:Restriction ;
    owl:onProperty ?p ;
    owl:allValuesFrom ?c
}
```

Une transformation STTL est un ensemble de tels patrons de transformation dédiés à la transformation d'énoncés RDF dans un certain modèle (e.g. les données DBpedia sur les rois de France) ou pour un certain langage (e.g. OWL/RDF) dans un format textuel (e.g. une présentation des rois de France en HTML ou une représentation d'énoncés OWL dans la syntaxe fonctionnelle du langage). L'exécution d'un patron peut récursivement déclencher l'exécution d'autres patrons de transformation ; l'appel à un patron se fait au moyen d'une fonction d'extension `st:apply-templates` [1] appelée dans la clause TEMPLATE. Une variante de l'exemple précédent est le patron suivant :

```
TEMPLATE {
   "allValuesFrom("
     st:apply-templates(?p) " "
     st:apply-templates(?c) ")"
}
WHERE {
  ?in a owl:Restriction ;
    owl:onProperty ?p ;
    owl:allValuesFrom ?c
}
```

Dans ce patron de transformation, l'appel de la fonction `st:apply-templates` avec la variable `?p` en paramètre déclenchera la sélection d'un autre patron de transformation qui sera appliqué sur le résultat (*binding*) associé à la variable `?p` lors de l'appariement du graphe requête de la clause WHERE avec le graphe RDF à transformer. La même chose se produira pour la variable `?c`. Des patrons nommés sont également prédéfinis (et d'autres peuvent être définis et nommés par le programmeur STTL) qui peuvent être exécutés par un appel à la fonction d'extension `st:call-template`. Voici un patron de transformation équivalent au précédent où les patrons appelés dans la clause TEMPLATE sont indiqués explicitement : le patron `st:property` dédié à la présentation d'une propriété dans la syntaxe fonctionnelle de OWL est appelé avec la variable `?p` en paramètre et le patron `st:value` dédié à la présentation des valeurs de propriété est appelé avec la variable `?c` en paramètre.

1. Le préfixe `st:` correspond à l'espace de nommage (*namespace*) `http://ns.inria.fr/sparql-template/`

```
TEMPLATE {
   "allValuesFrom("
     st:call-template(st:property, ?p) " "
     st:call-template(st:value, ?c) ")"
}
WHERE {
  ?in a owl:Restriction ;
    owl:onProperty ?p ;
    owl:allValuesFrom ?c
}
```

2.1. Grammaire de STTL

Voici la grammaire de STTL exprimée selon celle de SPARQL 1.1 (Harris, Seaborne, 2013). STTL est ainsi vu comme une extension de SPARQL avec la clause `Template`. Dans cette grammaire `Prologue` définit les préfixes, `PrimaryExpression` est une constante, une variable, un appel de fonction ou une expression parenthésée.

```
Template ::= Prologue TemplateClause
   DatasetClause* WhereClause
   SolutionModifier ValuesClause

TemplateClause ::=
  'TEMPLATE' (iri VarList ?) ?
  '{' Term* Separator? '}'

VarList ::= '(' Var+ ')'

Term ::= PrimaryExpression | Group | Format

Group ::= 'GROUP' 'DISTINCT'? '{'
   PrimaryExpression* Separator? '}'

Format ::= 'FORMAT' '{'PrimaryExpression PrimaryExpression+'}'

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

La table ci-dessous présente les fonctions principales de STTL.

Nom	**Description**
`apply-templates`	Applique la transformation courante sur le *focus*
`apply-templates-with`	Applique une transformation sur le *focus*
`apply-templates-all`	Applique tous les templates sur le *focus*
`call-template`	Applique un template sur le *focus*
`call-template-with`	Applique un template d'une transformation sur le *focus*
`process`	Définit le traitement des variables
`get`	Trouve une propriété du contexte
`set`	Assigne une propriété au contexte
`turtle`	Retourne un terme en syntaxe Turtle

Le schéma de traduction suivant décrit comment les expressions du langages sont traduites en SPARQL. La règle (1) traduit une clause TEMPLATE en une clause SELECT. La règle (2) traduit un groupe de termes en une aggrégation `group_concat`. La règle (3) traduit un format. La règle (4) traduit une variable en appel à la fonction `st:process`. La règle (5) traduit une expression conditionnelle. La règle (6) laisse telle quelle toute autre expression.

```
(1) cp(TemplateClause(Term(t1), ... Term(tn), sep)) =
    SELECT (concat(cp(t1), ... cp(tn)) AS ?out)

(2) cp(Group(Term(t1), ... Term(tn), sep)) =
    group_concat(concat(cp(t1), ... cp(tn)), sep)

(3) cp(Format(Term(t1), ... Term(tn)) =
    st:format(t1, cp(t2), ... cp(tn))

(4) cp(Var(v)) = st:process(v)

(5) cp(PrimaryExpression(if(e1, e2, e3))) =
    if(e1, cp(e2), cp(e3))

(6) cp(PrimaryExpression(e)) = e
```

Par exemple, le patron STTL suivant :

```
TEMPLATE {
  "ObjectAllValuesFrom(" ?p " " ?c ")"
}
WHERE {
  ?in a owl:Restriction ;
    owl:onProperty ?p ;
    owl:allValuesFrom ?c
}
```

est compilé dans la requête SPARQL suivante :

```
SELECT
  (concat("ObjectAllValuesFrom(",
   st:process(?p), " ",
   st:process(?c), ")") AS ?out)
WHERE {
  ?in a owl:Restriction ;
    owl:onProperty ?p ;
    owl:allValuesFrom ?c
}
```

STTL étant ainsi compilé en SPARQL 1.1, la sémantique de l'évaluation d'un patron STTL est celle de l'évaluation d'une requête SPARQL [2]. Soit Ω la séquence de solutions résultant de l'évaluation de la requête SPARQL issue de la compilation d'un patron STTL. Le patron échoue si la séquence est vide. Si elle n'est pas vide, le résultat du patron est le résultat de l'opération d'aggrégation `group_concat` de l'algèbre de SPARQL [3] suivant :

```
Aggregation((?out), group_concat, scalarvals, {1 -> Ω})
```

où `scalarvals` correspond à l'argument `sep` de la clause TEMPLATE.

2.2. Traitement par défaut

Par défaut, la fonction `st:process` retourne le terme RDF en argument dans la syntaxe Turtle. Il est possible de redéfinir le comportement de cette fonction pour, par exemple, exécuter la fonction `st:apply-templates`. Pour cela, on dispose du langage FunSPARQL de définition de fonctions directement en SPARQL (Corby, Faron-Zucker, 2015a). L'exemple ci-dessous montre une redéfinition de la fonction `st:process` qui appelle le moteur de transformation quand l'argument est une ressource anonyme (*blank node*) et qui retourne le format Turtle sinon.

```
function st:process(?t){
   if (isBlank(?t)){
     st:apply-templates(?t)
   }
   else { st:turtle(?t) }
}
```

2.3. Comparaison de STTL et XSLT

STTL et XSLT sont relativement similaires dans leurs fonctionnalités et leur expressivité. Toutefois, XSLT traite des arbres XML où les arcs sont ordonnés alors que STTL traite des graphes RDF dans lesquels les arcs ne sont pas ordonnés. Ainsi, une requête XSLT/XPath qui recherche le troisième fils d'un sommet n'a pas d'équivalent direct avec STTL mais cela peut être simulé avec des listes ou des conteneurs.

Les deux langages sont basés sur des règles de transformation déclaratives et possèdent les opérations `apply-templates` et `call-template` permet l'appel récursif du moteur de transformation. Ils offrent tous deux une structure conditionnelle, le groupement et le tri des solutions. XSLT offre une structure répétitive explicite `xsl:for-each` tandis que STTL permet d'itérer une clause TEMPLATE sur toutes les solutions d'une clause WHERE.

---

2. http://www.w3.org/TR/sparql11-query/#sparqlAlgebraEval
3. http://www.w3.org/TR/sparql11-query/#aggregateAlgebra

Un *template* XSLT peut traiter plusieurs patrons successifs tandis que STTL ne possède qu'une seule clause WHERE. STTL hérite de tous les énoncés SPARQL 1.1, y compris les chemins de propriétés (*property paths*) et les clauses services qui permettent d'interroger des graphes distants selon les principes du *Linked Data*.

3. Le serveur de transformation

Nous présentons dans cette partie la plate-forme ALIGATOR qui repose sur le langage STTL et permet de réaliser des navigateurs hypertextes pour le Web de données. Plus précisément, le système repose sur un serveur HTTP Jetty avec des services Web REST qui implémentent le protocole SPARQL 1.1 [4]. Un SPARQL endpoint exécute des requêtes SPARQL convoyées par HTTP et retourne les résultats également par HTTP. Ces résultats sont exprimés dans les formats standards SPARQL Query Results XML Format, Turtle, RDF/XML ou JSON-LD.

Outre cette implémentation standard, ALIGATOR fournit un service Web qui permet d'exécuter des transformations STTL pour engendrer du code HTML à partir de données RDF. Nous appelons *RDF2HTML* ces transformations particulières. Ce service permet de réaliser des navigateurs hypertextes sur un graphe RDF local ou distant.

3.1. Le service STTL

Le service `STTL` répond à deux scénarios décrits sur la figure 1. Dans le Scénario 1, une transformation RDF2HTML est exécutée sur un graphe RDF. Etant donné un URI, la transformation engendre une page HTML décrivant la ressource RDF correspondante. Le code HTML contient des liens hypertextes, vers le service STTL, pour décrire les ressources liées à cet URI. Cliquer sur ces liens hypertextes déclenche un appel au service STTL qui applique la transformation STTL et engendre en retour une nouvelle page HTML. On implémente ainsi une navigation hypertexte sur un graphe RDF. La transformation peut, par exemple, engendrer une table HTML avec les triplets dont l'URI est le sujet ou l'objet, comme le fait DBpedia [5].

Dans le scénario 2, le service STTL exécute une requête SPARQL sur le graphe puis applique une transformation RDF2HTML sur le résultat de la requête. Pour les requêtes de la forme CONSTRUCT ou DESCRIBE, la transformation est exécutée directement sur le graphe RDF résultat de la requête. Pour les requêtes de la forme SELECT ou ASK, le résultat de la requête est traduit en graphe RDF en utilisant le vocabulaire publié par le W3C, RDF Data Access Working Group [6]. La transformation STTL est alors appliquée sur le graphe RDF ainsi produit.

4. http://www.w3.org/TR/sparql11-protocol/
5. e.g. http://dbpedia.org/resource/Berlin
6. http://www.w3.org/2001/sw/DataAccess/tests/result-set.n3

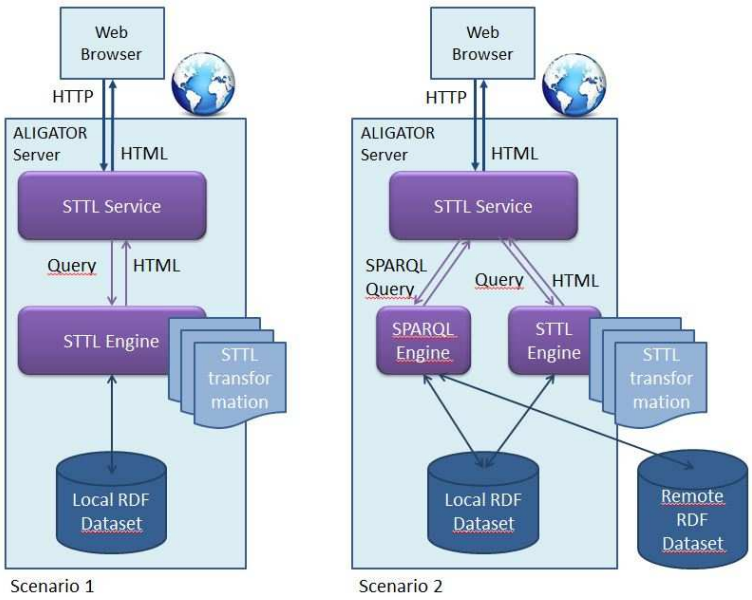

Figure 1. Architecture

Une requête pour une transformation STTL est envoyée à un service ALIGATOR dans un URL dont la partie hiérarchique se termine par `/template` et dont la partie requête contient des paires clé-valeur spécifiant la requête au serveur. La clé `query`, reprise du protocole SPARQL, permet d'indiquer une requête SPARQL à exécuter. La clé `transform` permet de spécifier l'URL de la transformation à appliquer sur le résultat de la requête ou sur le graphe du serveur. Par exemple, l'URL suivant demande l'exécution d'une transformation STTL spécifiée par `st:sparql` sur le résultat d'une requête SPARQL qui retourne tous les triplets d'un graphe RDF donné.

```
http://corese.inria.fr/template?
  query=SELECT * WHERE { ?x ?p ?y }&
  transform=st:sparql
```

3.2. Profil de transformation

Un URL peut donc comporter une requête, une transformation ou les deux. Pour simplifier l'interaction avec le service STTL, nous introduisons la notion de *profil* de transformation RDF2HTML. Un *profil* permet de définir une chaîne de traitements simple, composée d'une requête SPARQL optionnelle et d'une transformation STTL, et de nommer ce traitement avec un URI. Un profil est décrit sous forme d'énoncés RDF : la classe `st:Profile` caractérise un profil, la propriété `st:query` définit un

chemin (un URL) vers une requête SPARQL et la propriété `st:transform` spécifie l'URL d'une transformation STTL. Voici un exemple de profil :

```
st:dbpedia a st:Profile ;
  st:query <q1.rq> ;
  st:transform st:navlab .
```

Dans cette description, `q1.rq` contient par exemple la requête SPARQL suivante qui interroge le serveur DBpedia :

```
CONSTRUCT { ?x ?p ?y }
WHERE { service <http://fr.dbpedia.org/sparql> { ?x ?p ?y } }
```

Dans l'URL transportant une requête pour une transformation STTL, l'URI d'un profil est indiqué par un argument `profile` dont la valeur est l'URI du profil :

```
http://corese.inria.fr/template?profile=st:dbpedia
```

Une requête peut préciser l'URI d'une ressource sur lequel focaliser l'exécution du service :

```
http://corese.inria.fr/template?profile=st:dbpedia
&uri=http://fr.dbpedia.org/resource/Antibes
```

La requête SPARQL exécutée par le serveur peut être complétée par une clause de la forme `st:get(st:uri)` qui permet de récupérer dans le contexte d'exécution l'URI donné comme valeur de l'argument `uri`.

```
CONSTRUCT { ?x ?p ?y }
WHERE {
  bind (st:get(st:uri) as ?x)
  service <http://fr.dbpedia.org/sparql> { ?x ?p ?y }}
```

La définition d'un profil peut contenir une requête interrogeant un serveur distant, e.g. DBpedia, sur une ressource particulière, e.g. la ville d'Antibes, et peut appliquer au graphe résultat une transformation RDF2HTML qui engendre une page HTML décrivant la ressource. Le code HTML engendré peut contenir des liens hypertextes sur les autres ressources reliées à la ressource courante. Nous obtenons ainsi un navigateur hypertexte sur un graphe RDF distant (celui de DBpedia). Cette technologie permet donc d'engendrer un navigateur hypertexte sur un SPARQL endpoint. On peut aussi coupler l'interrogation d'un graphe local avec un SPARQL endpoint distant et réaliser ainsi un *mashup* de données liées.

3.3. Contexte de transformation

Nous définissons la notion de contexte de transformation qui permet au serveur de transmettre au moteur de transformation STTL des informations relatives au contexte d'exécution de la transformation. Par exemple, pour engendrer des liens hypertextes, le moteur de transformation peut avoir besoin du nom du service. La spécification d'un contexte de transformation évite de "coder en dur" de telles informations dans la transformation, de manière à la rendre le plus générique possible.

Le contexte peut être consulté par le moteur de transformation au moyen d'une fonction d'extension SPARQL `st:get`. Les paramètres possibles du contexte sont les suivants : le nom du service, le nom du profil, le nom de la transformation, l'URI de la ressource courante. Voici un exemple de contexte :

```
st:get(st:service)   = template
st:get(st:profile)   = st:dbpedia
st:get(st:transform) = st:navlab
st:get(st:uri) = http://fr.dbpedia.org/resource/Antibes
```

Le contexte peut être utilisé par la transformation pour stocker des informations au moyen de la fonction `st:set` :

```
st:set(st:mode, "debug")
```

Nous généralisons le contexte en permettant aux utilisateurs de spécifier des paramètres supplémentaires dans un profil, paramètres qui sont transmis au contexte de la transformation. Pour cela nous introduisons une propriété `st:param` dans le profil. La valeur de cette propriété est un nœud vide qui possède un ensemble de propriétés qui représentent les paramètres supplémentaires à transmettre au contexte. L'exemple suivant définit une propriété `st:lang` qui permet de spécifier la langue de la transformation. La transformation peut alors consulter la valeur de ce paramètre au moyen de l'appel de fonction `st:get(st:lang)`.

```
st:dbpedia a st:Profile ;
  st:transform st:navlab ;
  st:param [
    st:lang "fr"
  ]
```

Le contexte peut ainsi être utilisé pour paramétrer des informations relatives au format de présentation, par exemple HTML et CSS. Il peut être utilisé pour spécifier le comportement initial d'objets graphiques tels que des cartes : zoom, taille de la carte, icône de marqueur sur la carte, etc.

Des propriétés relatives au *Linked Open Data* peuvent être définies. Nous avons par exemple introduit une propriété `st:lodprofile` qui permet de spécifier le

profil à utiliser pour un espace de nommage particulier. Par exemple, une ressource appartenant à l'espace de nommage de DBpedia "http://fr.dbpedia.fr/resource" peut être traitée par le profil spécifique du navigateur DBpedia :

```
st:sparql a st:Profile ;
  st:transform st:sparql ;
  st:param [
    st:lodprofile
      ((<http://fr.dbpedia.fr/resource/> st:dbpedia)
       ( "*" st:sparql ))
  ] .
```

Différents types de ressources peuvent ainsi être traités avec des profils appropriés.

3.4. Liens hypertextes dynamiques

Une des clés de notre approche et du système développé réside en la capacité à engendrer dynamiquement des liens hypertextes dans le code HTML produit. Lorsque l'un de ces liens est suivi, un nouvel appel au serveur est produit, pour engendrer de nouvelles pages HTML relatives à une nouvelle ressource (avec de nouveaux liens hypertextes), grâce à une nouvelle transformation RDF2HTML.

Voici un exemple d'un tel lien hypertexte ; l'attribut `href` de l'élément `a` a pour valeur un URL qui contient une requête au service STTL du serveur :

```
<a href='/template?profile=st:dbpedia
&uri=http://fr.dbpedia.org/resource/Antibes'>Antibes</a>
```

Voici un exemple de patron qui engendre un lien hypertexte vers une ressource `?x` avec un titre `?t` :

```
TEMPLATE st:link(?x, ?t) {
  format {
    "<a href='/template?profile=st:dbpedia&uri=%s'>%s</a>"
    encode_for_uri(?x)
    str(?t)
  }
}
WHERE { }
```

Les informations relatives au serveur peuvent être représentées dans le contexte :

```
TEMPLATE st:link(?x, ?t) {
  format {
    "<a href='/%s?profile=%s&uri=%s'>%s</a>"
```

```
    st:get(st:service)
    st:get(st:profile)
    encode_for_uri(?x)
    str(?t)
  }
}
WHERE { }
```

4. Quatre navigateurs ALIGATOR

Plusieurs exemples de navigateurs construits avec ALIGATOR en réponse à différents besoins sont disponibles en ligne sur un serveur de démonstration [7]. Nous présentons ici quatre d'entre eux. Le code source est disponible dans la distribution de Corese-KGRAM [8] (Corby, Faron-Zucker, 2010) (Corby *et al*., 2012).

4.1. Navigateur pour un service SPARQL

Nous avons conçu un navigateur ALIGATOR permettant d'exécuter une requête SPARQL et d'en présenter le résultat en HTML. Le résultat d'une requête de la forme SELECT ou ASK est traduit en RDF en utilisant le vocabulaire W3C RDF Data Access Working Group [9]. Une transformation RDF2HTML est ensuite appliquée sur ce graphe. Nous avons défini le profil `st:sparql` pour identifier cette transformation. Pour les requêtes de forme CONSTRUCT ou DESCRIBE, la transformation `st:sparql` est directement appliquée sur le graphe RDF résultat. Les règles de cette transformation sont disponibles en ligne [10].

La figure 2 est une capture d'écran de la transformation du profil `st:sparql` appliquée au résultat d'une requête SPARQL (cela est visible dans l'URI entrée dans le navigateur Chrome utilisé). Les liens hypertextes visibles sur la page HTML générée sont des liens vers le serveur ALIGATOR comme expliqué précédemment.

Considérons par exemple la requête SPARQL suivante :

```
SELECT ?x ?n WHERE  ?x rdfs:label ?n
```

et l'exemple suivant de solution à cette requête, exprimée en RDF.

7. http://corese.inria.fr
8. http://wimmics.inria.fr/corese
9. http://www.w3.org/2001/sw/DataAccess/tests/result-set.n3
10. http://ns.inria.fr/sparql-template/

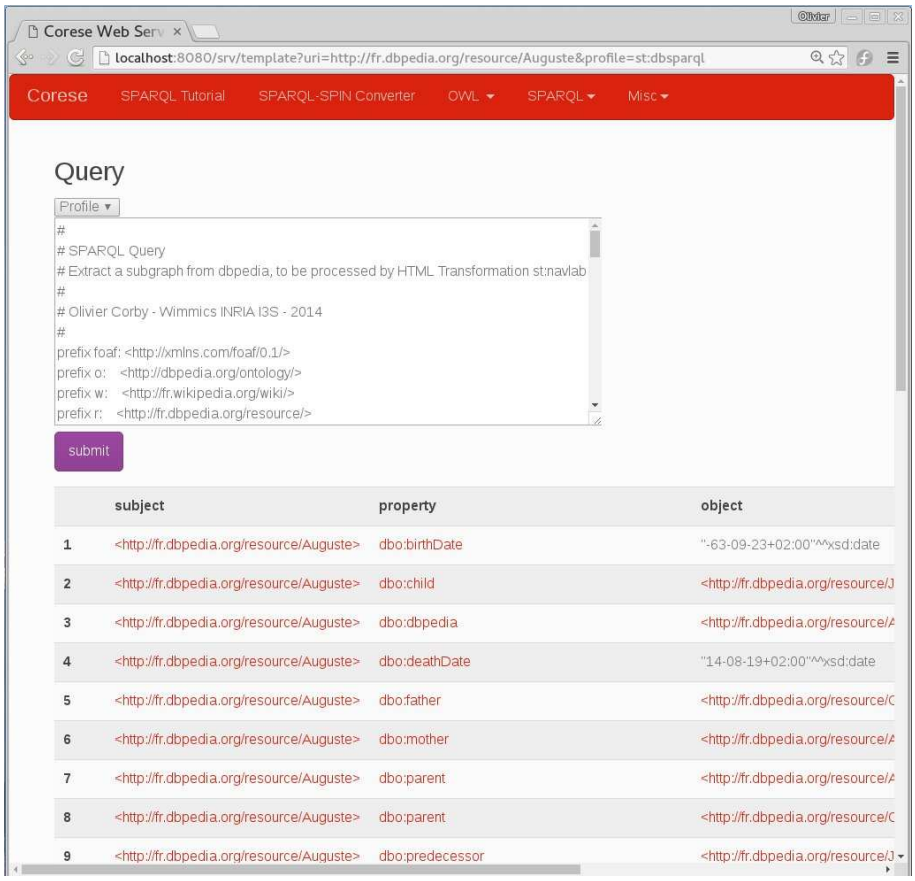

Figure 2. Navigation dans les résultats d'une requête SPARQL

```
@prefix rs:
  <http://www.w3.org/2001/sw/DataAccess/tests/result-set#>
@prefix ex: <http://fr.dbpedia.org/resource/>

[] rs:resultVariable "x", "n" ;
   rs:solution
     [ rs:binding [ rs:variable "x" ; rs:value ex:Auguste ],
                  [ rs:variable "n" ; rs:value "Auguste"  ] ],
     [ rs:binding [ rs:variable "x" ; rs:value ex:Tibère  ],
                  [ rs:variable "n" ; rs:value "Tibère"   ] ].
```

Voici le patron principal de la transformation `st:sparql` qui traite les résultats des requêtes de la forme SELECT :

```
prefix rs:
  <http://www.w3.org/2001/sw/DataAccess/tests/result-set#>
TEMPLATE {
  "<td>"
  coalesce(st:call-template(st:display, ?val), " ")
  "</td>" ; separator = " " }
WHERE {
  ?x rs:solution ?in
  ?x rs:resultVariable ?var
  OPTIONAL {?in rs:binding [rs:variable ?var ; rs:value ?val]}}
ORDER BY ?var
```

La clause WHERE de ce patron se focalise sur une solution `?in` qui est un ensemble de liaisons de variables. La clause OPTIONAL énumère ces liaisons de variables ; cette énumération est dans un sous-graphe optionnel car il se peut que certaines variables (`?val`) n'aient pas de valeur. La clause TEMPLATE engendre une cellule de table HTML pour chaque variable (`?val`) avec le résultat de la présentation de la valeur `st:call-template(st:display, ?val)`, ou bien un espace s'il n'y a pas de valeur disponible.

4.2. Navigateur DBpedia

Avec la même technologie, il est également possible de concevoir des navigateurs dédiés à des domaines ou des applications spécifiques. Nous avons ainsi développé un navigateur hypertexte dédié à certaines ressources de DBpedia : les personnes et les lieux. Il repose sur un serveur ALIGATOR offrant un service de transformation STTL avec une nouvelle transformation dédiée, dans le profil `st:navlab`. Son principe de fonctionnement est le suivant. Une requête SPARQL de la forme CONSTRUCT interroge le graphe distant DBpedia, avec une clause SERVICE, sur une personne ou un lieu et retourne un graphe RDF résultat. La transformation `st:navlab` est ensuite appliquée sur ce graphe résultat. Elle engendre une page HTML dédiée à la présentation des ressources retournées par la requête SPARQL. Les lieux sont géolocalisés sur une carte interactive. La figure 3 est une capture d'écran d'une page HTML générée par le navigateur ALIGATOR pour DBpedia. L'URI entrée dans le navigateur Chrome est un exemple de requête HTTP envoyée au serveur ALIGATOR. Celui-ci produit en réponse une page HTML engendrée dynamiquement avec le profil `st:dbpedia` qui utilise la transformation `st:navlab`.

4.3. Navigateur historique

Nous avons développé un troisième type de navigateur qui permet de présenter les données issues d'un graphe local lié à un graphe distant, selon le principe du Web de données liées. La source distante est encore une fois DBpedia ; le graphe RDF local contient un ensemble d'événements et de personnages historiques dont les URI sont

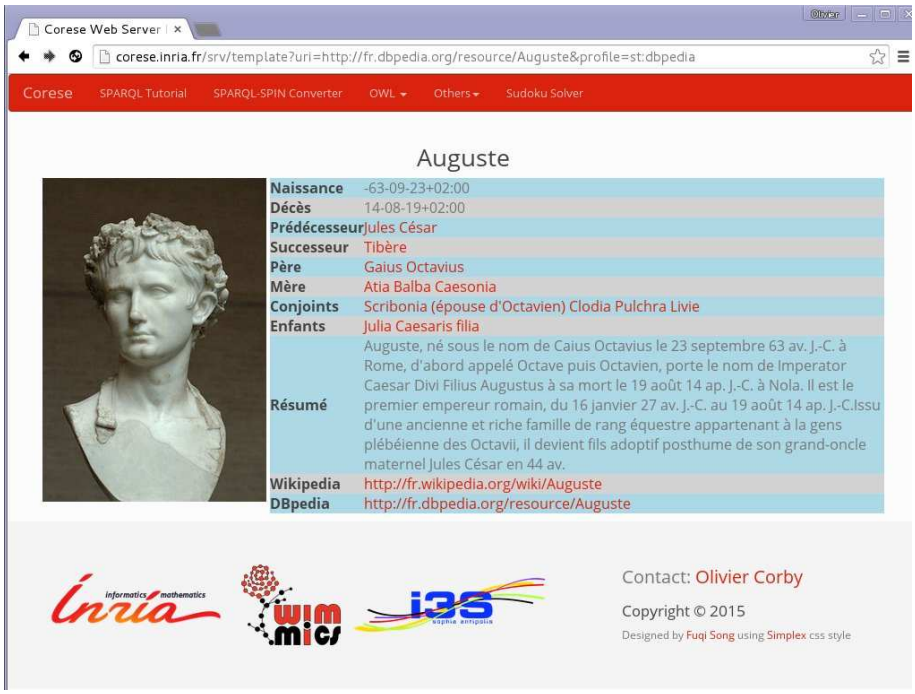

Figure 3. Navigateur DBpedia

ceux de DBpedia [11]. Les énoncés RDF locaux sont stockés dans des graphes nommés annotés avec des thèmes tels que la France, l'Empire, etc.

La transformation RDF2HTML que nous avons développée engendre une page HTML par siècle. Dans chaque siècle, les ressources sont classées par ordre chronologique et rangées dans les colonnes d'une table en fonction du thème de leur graphe nommé. Par exemple, une colonne "France" pour les descriptions dans le graphe annoté par le thème "France". Des liens hypertextes vers les ressources correspondantes de DBpedia sont engendrés selon le même principe décrit dans la section précédente (avec la transformation `st:navlab`). La figure 4 est une copie d'écran d'une page HTML engendrée par ce navigateur historique.

Le patron suivant joue un rôle essentiel dans la transformation `st:navlab`. Sa clause WHERE retourne les dates (`?d`) comprises dans un intervalle (par exemple un siècle), triées par ordre chronologique. Sa clause TEMPLATE permet d'engendrer une ligne de table HTML pour chaque date et une cellule de table pour chaque thème dans laquelle sont affichées les éventuelles ressources correspondant à la date et au thème.

11. `http://fr.dbpedia.org/resource/`

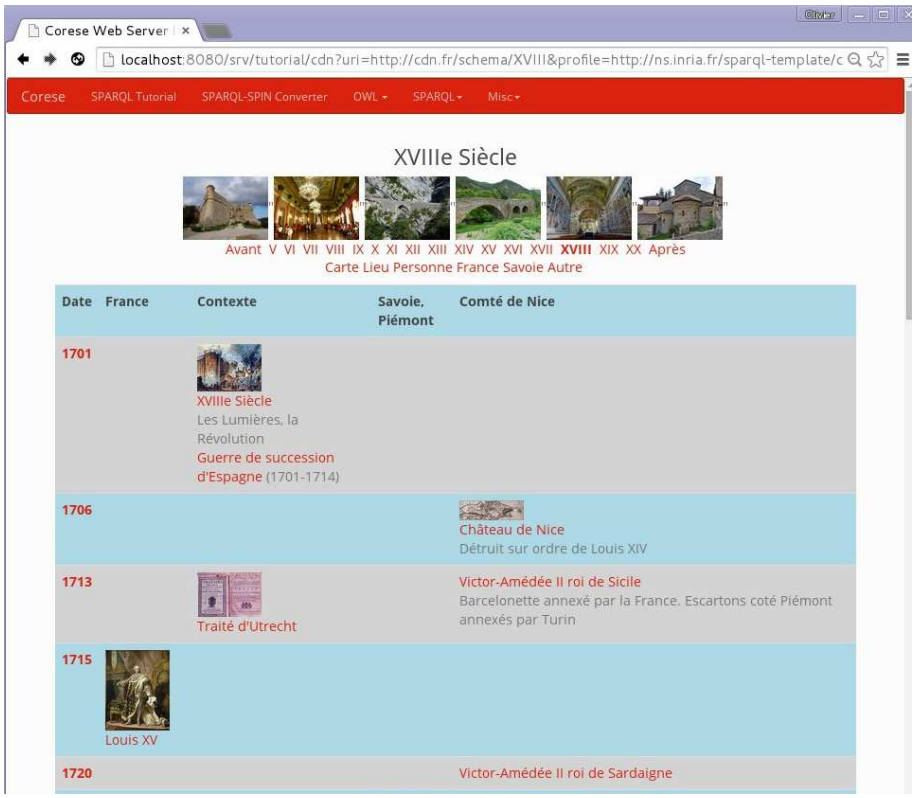

Figure 4. Navigateur historique

```
TEMPLATE  cn:table(?min, ?max) {
FORMAT {
  """<tr><th class='date'>%s</th>
  <td>%s</td><td>%s</td><td>%s</td><td>%s</td></tr>\n"""
  st:call-template(cn:wikidate, ?d)
  st:call-template(cn:date, ?d, cn:fr)
  st:call-template(cn:date, ?d, cn:context)
  st:call-template(cn:date, ?d, cn:mds)
  st:call-template(cn:date, ?d, cn:cdn)
}}
WHERE {
  { SELECT DISTINCT ?d  WHERE { ?uri cn:date ?d }}
  FILTER(?min <= ?d && ?d <= ?max)
}
ORDER BY asc(?d)
```

4.4. Navigateur sur une base de requêtes SPARQL

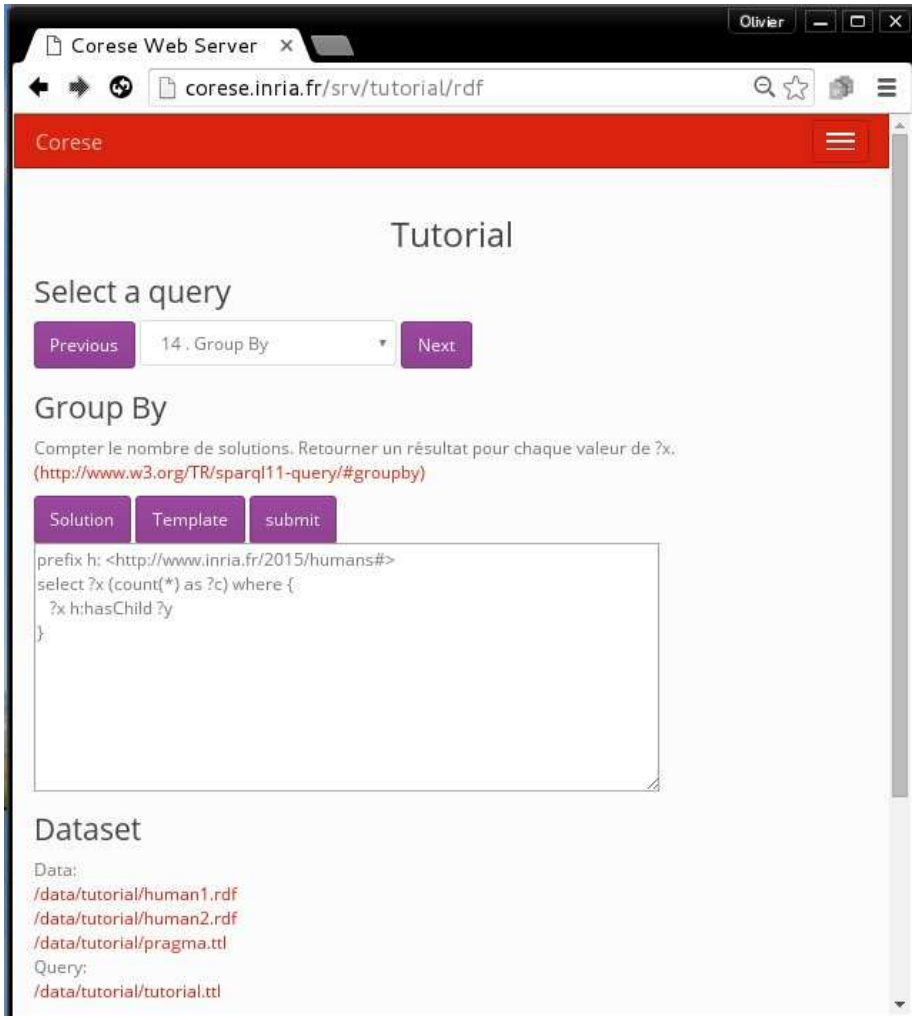

Figure 5. Tutoriel SPARQL

Nous avons écrit une transformation STTL `st:web` permettant de gérer une base de requêtes SPARQL prédéfinies. Celles-ci sont stockées dans un fichier RDF et sont gérées sous forme d'un graphe RDF qui peut ainsi être l'objet d'une transformation. Le patron suivant est central dans la transformation. Il récupère les requêtes prédéfinies et engendre un sélecteur HTML avec les noms des requêtes (valeurs des propriétés `st:name`).

```
TEMPLATE st:list(?t) {
  FORMAT {
    "<option value='%s' %s>%s . %s</option>"
    ?q
    if (?q = ?t, "selected='true'", "")
    str(?i)
    str(?title)
  }
}
WHERE {
  BIND (st:get(st:context) as ?g)
  GRAPH ?g {
    VALUES ?qt {st:Query sw:Query}
    [] a ?qt ;
      st:index ?i ;
      st:name ?q ;
      rdfs:label ?title
  }
}
ORDER BY ?i
```

Nous avons développé deux applications à partir de cette transformation générique. La première application est un tutoriel sur le langage SPARQL avec 56 requêtes prédéfinies [12] et des liens hypertextes vers la recommandation W3C. La transformation permet d'afficher à l'utilisateur une liste de requêtes SPARQL prédéfinies. L'utilisateur peut en sélectionner une qui s'affiche alors dans un champ d'édition dédié ; il peut éventuellement en modifier le texte avant de la soumettre au serveur SPARQL. La requête est alors exécutée sur le graphe RDF de l'application. Le résultat de la requête retourné par le serveur est présenté dans un format HTML calculé par une deuxième transformation STTL. Ce format contient des liens hypertextes et permet ainsi de naviguer dans le graphe RDF de l'application. Le résultat de la requête est affiché sous le texte de la requête soumise; il est ainsi possible d'analyser le résultat au vu de la requête qui est elle-même toujours éditable. Cela permet à l'utilisateur de progresser dans son apprentissage du langage SPARQL en modifiant pas à pas sa requête et en constatant l'effet de ses modifications sur les résultats retournés par le serveur SPARQL. La figure 5 montre une copie d'écran du tutoriel. Cette application est utilisée dans le MOOC Inria-uTOP "Web sémantique et web de données" [13]. Le tutoriel propose des exercices à l'aide d'un énoncé, d'un patron de requête SPARQL à compléter et la requête solution. Un lien hypertexte permet d'accéder à la solution de l'exercice, un autre lien permet de revenir au patron de requête à compléter et un troisième lien dynamique permet de soumettre la requête.

12. http://corese.inria.fr/data/tutorial/workflow.ttl
13. http://corese.inria.fr/srv/tutorial/rdf

La seconde application de la transformation STTL permettant de gérer des requêtes SPARQL est un navigateur Linked Data pour visualiser des données RDF extraites des textes du rapport d'activité de l'Inria. Ce navigateur a été développé dans le cadre d'un projet Inria interne. La transformation permet de gérer un ensemble de requêtes prédéfinies sur les équipes de recherche, les thématiques de recherche et les coopérations nationales ou internationales. Les requêtes sont résolues sur les données extraites du rapport d'activité, des bases internes de description des équipes et de DBpedia par exemple pour savoir que Manchester est en Angleterre. Les ressources géographiques issues de DBpedia sont géolocalisées sur la base de leur URI [14]. La figure 6 montre une copie d'écran de l'application.

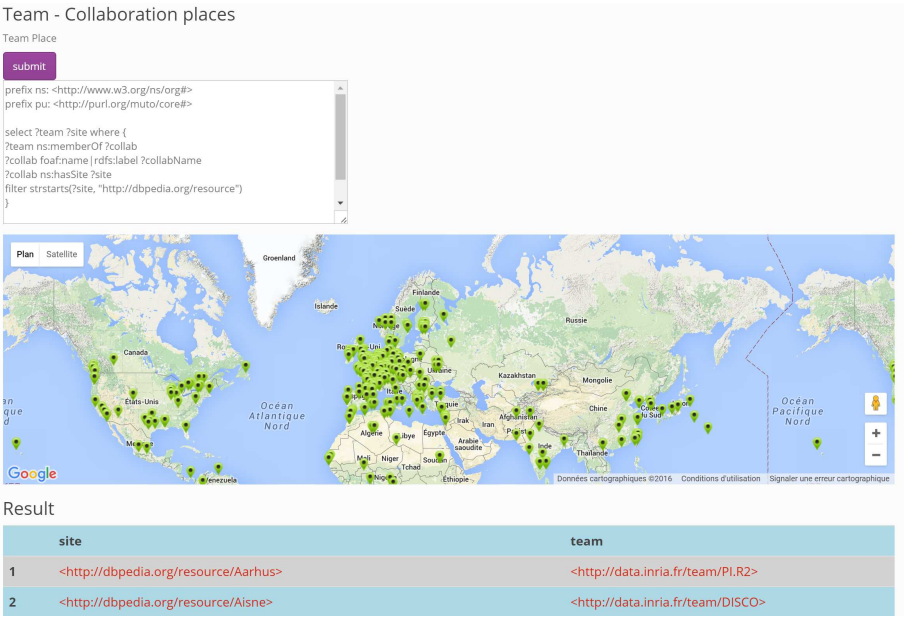

Figure 6. Navigateur sur le rapport d'activité de l'Inria

14. http://fr.dbpedia.org/resource

5. Conclusion

Nous avons présenté la plate-forme ALIGATOR permettant de concevoir des navigateurs pour les données liées du Web sémantique. Elle repose sur le langage STTL qui est une extension de SPARQL permettant d'écrire des transformations déclaratives de RDF vers des formats textuels, en particulier ici vers HTML. Le moteur de transformation STTL et le serveur STTL qui constituent le navigateur sont disponibles (en open source) dans la plate-forme Corese. Plusieurs applications de cette technologie sont disponibles en ligne sur un serveur de démonstration à l'adresse http://corese .inria.fr.

Un avantage de cette approche, d'un point de vue technique, est qu'elle ne nécessite pas d'apprendre un nouveau *framework* Web. Il suffit de connaître SPARQL et HTML (et éventuellement JavaScript). STTL permet ainsi de passer "directement" de RDF à HTML avec SPARQL, sans autre langage de programmation.

En termes d'ingénierie des connaissances, les avantages de cette approche sont multiples. Tout d'abord, le fait de reposer sur le langage SPARQL est un atout du point de vue de la réutilisation possible de connaissances expertes : avec une nouvelle forme de requête dont la clause WHERE est commune aux autres formes de requêtes SPARQL, STTL permet d'envisager de réutiliser des patrons de toutes formes de requêtes lorsque ceux-ci ont été capitalisés. Egalement, comme pour l'écriture de requêtes SPARQL en général, les patrons de conception mis en œuvre dans la construction des bases RDF sur lesquelles opèrera une transformation STTL peuvent être réutilisés pour écrire les patrons qui composent celle-ci. Enfin, la déclarativité du langage STTL permet de capturer dans les clauses TEMPLATE les connaissances expertes nécessaires pour opérer des transformations sur des données RDF. Les transformations STTL peuvent être vues comme des connaissances de *présentation* capitalisables, partageables, réutilisables dans des scénarios ou selon des points de vue sur les données similaires.

Dans la continuité de ces conclusions, nous envisageons d'une part d'exploiter les fonctions de HTML 5 couplées avec JavaScript ainsi que d'engendrer des vues graphiques avec des librairie dédiées (e.g. 3D.js). Nous projetons d'autre part d'explorer plus avant la capitalisation de patrons de requêtes SPARQL et patrons de présentation HTML et le couplage de ces deux types de patrons.

Remerciements

Nous remercions Eric Toguem (U. de Yaoundé, Cameroun) et Alban Gaignard (CNRS) pour la première version du serveur HTTP qui embarque Corese ainsi que Fuqi Song pour le déploiement du serveur de démonstration `corese.inria.fr`.

Bibliographie

Bizer C., Lee R., Pietriga E. (2005, November). Fresnel - A Browser-Independent Presentation Vocabulary for RDF. In *Second International Workshop on Interaction Design and the Semantic Web @ ISWC'05.* Galway, Ireland. http://hal.inria.fr/inria-00001057

Corby O., Faron-Zucker C. (2010, September). The KGRAM Abstract Machine for Knowledge Graph Querying. In *Ieee/wic/acm international conference on web intelligence.* Toronto, Canada.

Corby O., Faron-Zucker C. (2014, May). SPARQL Template : un langage de Pretty-Printing pour RDF. In *Proc. 25e Journées francophones d'Ingénierie des Connaissances.* Clermont-Ferrand.

Corby O., Faron-Zucker C. (2015a, December). *FunSPARQL: Extension of SPARQL with a Functional Language*. Research Report nº RR-8814. INRIA. (http://hal.inria.fr/hal-01236947)

Corby O., Faron-Zucker C. (2015b, May). STTL: A SPARQL-based Transformation Language for RDF. In *Proc. 11th international conference on web information systems and technologies, webist 2015.* Lisbon, Portugal.

Corby O., Gaignard A., Faron-Zucker C., Montagnat J. (2012, December). KGRAM Versatile Data Graphs Querying and Inference Engine. In *Proc. ieee/wic/acm international conference on web intelligence.* Macau, China.

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