Un validateur d'ontologies par rapport à des profils OWL implémenté dans le langage STTL

Raphaël Gazzotti, Olivier Corby, Catherine Faron-Zucker

Université Côte d'Azur, Inria, CNRS, I3S, France
olivier.corby@inria.fr, faron@unice.fr, gazzotti@i3s.unice.fr

Résumé : Dans cet article, nous abordons la question de recherche générale *Comment exprimer des contraintes sur des données RDF et comment vérifier qu'un graphe RDF satisfasse un certain nombre de contraintes ?* Nous nous concentrons sur le cas particulier de l'expression des contraintes telles que définies par les profils de OWL 2 et nous vérifions ces contraintes pour déterminer la conformité d'une ontologie OWL et mettre en évidence la présence éventuelle d'énoncés sources de non conformité. Nous proposons une approche basée sur le langage SPARQL Template Transformation Language (STTL). Un template STTL est une règle de transformation qui s'applique sur un graphe RDF donné et par le biais d'appels récursifs de templates STTL sur un graphe RDF nous obtenons une sortie textuelle, résultante de la transformation de ce même graphe. Nous montrons que STTL peut être utilisé comme un langage de contraintes sur RDF et nous l'utilisons afin d'implémenter la sémantique propre à chaque profil de OWL 2, chacun pouvant être interprété comme un ensemble de contraintes à respecter sur les définitions de classes et de propriétés. Chaque profil de OWL 2 est ainsi représenté par un ensemble de templates STTL qu'une ontologie valide se doit de satisfaire.

Mots-clés : Ontologies, Langage de contrainte, Validation, RDF, OWL 2, STTL

1 Introduction

Les profils de OWL 2 (Horridge & Bechhofer, 2011) peuvent être interprétés comme des restrictions sur les déclarations de OWL 2 et la validation d'une ontologie par rapport à des profils de OWL 2 comme la vérification des contraintes syntaxiques des axiomes de OWL 2. Dans cet article, nous abordons la question de recherche générale *Comment exprimer des contraintes sur des données RDF et comment vérifier qu'un graphe RDF satisfait un certain nombre de contraintes ?* Nous nous concentrons sur la question particulière de l'expression des contraintes telles que définies par les profils de OWL 2 et nous vérifions ces mêmes contraintes pour déterminer la conformité d'une ontologie OWL et mettre en évidence la présence éventuelle d'énoncés sources de non conformité.

Nous proposons une approche basée sur le langage SPARQL Template Transformation Language (STTL) que nous avons initialement conçu afin de transformer des données RDF dans n'importe quel autre format de données. Un template STTL peut être vu comme une règle de transformation qui s'applique à un graphe RDF donné, à l'instar de XSL qui s'applique à un arbre XML, et les appels récursifs d'un ensemble de templates STTL sur un graphe RDF engendrent une sortie textuelle résultante de la transformation de ce graphe.

Nous montrons que STTL peut être utilisé comme un langage de contrainte pour RDF : chaque template STTL peut être vu comme représentant une contrainte et la conformité d'un graphe RDF par rapport à un ensemble de contraintes est vérifié en appliquant sur ce graphe l'ensemble des templates STTL qui représentent les contraintes à vérifier. Le résultat de cette application peut être une simple valeur booléenne ou un texte adapté à la visualisation de ces données, où les sous-graphes violant certaines contraintes peuvent, par exemple, se voir surlignés. Ceci est rendu possible par la définition d'un patron de conception “Visiteur" associée à un ensemble de

templates STTL ayant pour objectif de réunir les sous-graphes RDF incorrects, et un patron de conception générique pour présenter le résultat à l'utilisateur.

En conséquence de quoi, notre approche implique d'implémenter la sémantique des profils de OWL 2, chaque profil étant considéré comme un ensemble de contraintes devant être vérifiées : nous avons ainsi défini une transformation STTL afin de représenter chacun des profils de OWL 2 (OWL RL, OWL QL et OWL EL). L'application de l'une de ces trois transformations STTL sur une ontologie (exprimée en RDF) permet de la valider par rapport à l'un ou l'autre des profils de OWL 2 que ces transformations représentent.

Cet article reprend et étend une première présentation courte de notre approche dans Corby *et al.* (2016). Il est organisé de la manière suivante. La section 2 propose un aperçu du langage STTL. La section 3 présente les transformations STTL qui implémentent la sémantique des profils de OWL. La section 4 montre comment une transformation STTL supplémentaire permet de fournir aux utilisateurs une représentation visuelle des résultats de la validation d'un document OWL. La section 5 décrit les expérimentations qui ont été conduites sur de nombreuses ontologies OWL issues du Web de données. La section 6 présente une conclusion.

2 SPARQL Template Transformation Language (STTL)

STTL est un langage de règles de transformations générique pour RDF qui dépend de deux extensions de SPARQL : une requête `TEMPLATE` est faite d'une clause standard `WHERE` et d'une clause `TEMPLATE`. La clause `WHERE` n'est autre que la partie pour la condition d'une règle qui va spécifier les nœuds à sélectionner au sein du graphe RDF pour la transformation. La clause `TEMPLATE` est la partie servant à présenter la règle, elle va spécifier la sortie de la transformation effectuée par la solution donnée par la clause `WHERE`. À titre d'exemple, considérons l'axiome exprimant que la classe `a:Parent` est équivalente à celle de la classe des individus ayant une personne (`a:Person`) comme enfant (relation `a:hasChild`). Cet axiome s'exprime ainsi en syntaxe fonctionnelle :

```
EquivalentClasses(a:Parent
  ObjectSomeValuesFrom(a:hasChild a:Person))
```

de la sorte en Turtle :

```
a:Parent a owl:Class ; owl:equivalentClass
    [ a owl:Restriction ; owl:onProperty a:hasChild ;
      owl:someValuesFrom a:Person ]
```

Le template suivant permet de transformer de manière directe la déclaration `equivalentClass` en RDF vers la syntaxe fonctionnelle :

```
TEMPLATE { FORMAT {"EquivalentClasses(%s %s)"
    st:apply-templates(?in) st:apply-templates(?c) }}
WHERE { ?in owl:equivalentClass ?c }
```

La ressource qui s'apparie avec la variable `?in` est `a:Parent`, il s'agit là de la valeur attendue pour le résultat de la transformation (la syntaxe fonctionnelle de la déclaration OWL 2). La ressource qui s'apparie avec la variable `?c` est une ressource anonyme dont les valeurs des propriétés servent à construire la sortie attendue. Celle-ci est définie dans un autre template afin de se concentrer sur le nœud d'intérêt. La fonction `st:apply-templates` permet l'appel récursif de templates, où `st` est le préfixe désignant l'espace de nommage de STTL [1].

De manière générale, la fonction `st:apply-templates` peut être utilisée dans n'importe quelle clause `TEMPLATE` de n'importe quel template t_1 afin d'exécuter un autre template t_2 qui peut lui-même exécuter un template t_3, et ainsi de suite. En conséquence de quoi, les templates s'appellent les uns les autres, permettant une hiérarchisation dans le traitement des templates ainsi qu'un parcours récursif du graphe RDF visé. De façon similaire, la fonction `st:call-template` peut être utilisée afin d'appeler récursivement des templates.

STTL est compilé en SPARQL standard. La compilation conserve la clause `WHERE`, les modificateurs de séquence de solutions et la clause `VALUES` du template demeurent inchangés et la clause `TEMPLATE` est compilée en une clause `SELECT`. À titre d'exemple, la clause `TEMPLATE` du template STTL suivant :

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

est compilée en la clause standard `SELECT` suivante de SPARQL :

```
SELECT
(CONTACT("ObjectSomeValuesFrom(",
    st:process(?p), " ",
    st:process(!,c), ")") AS ?out)
```

la clause `WHERE` reste inchangée.

Une description complète du langage STTL est présentée dans (Corby & Faron-Zucker, 2014) et (Corby & Faron-Zucker, 2015).

Nous avons implémenté le langage STTL dans un moteur de transformation qui fait partie de Corese Semantic Web Factory [2] (Corby *et al.*, 2012; Corby & Faron-Zucker, 2010) qui comprend un service Web RESTful pour traiter les transformations STTL et produire les résultats de la transformation de données RDF. L'implémentation de STTL dans ce moteur de transformation est décrite dans (Corby *et al.*, 2015) ; son utilisation pour développer un navigateur pour le Web de données est présentée dans (Corby & Faron-Zucker, 2015) ; une présentation globale du langage STTL et du serveur de transformation de graphes pour le Web de données est proposée dans (Corby & Faron-Zucker, 2016).

1. http ://ns.inria.fr/sparql-template/
2. http ://wimmics.inria.fr/corese

3 Validation d'ontologies auprès des profils de OWL 2 avec des transformations STTL

Dans cette section nous présentons comment se servir du langage STTL afin de représenter les profils de OWL 2 et le moteur de STTL pour vérifier le vocabulaire auprès des profils de OWL 2. De façon globale, chaque template de la transformation STTL représente un profil capable de vérifier les spécificités des contraintes d'un modèle de OWL 2 et renvoyer un booléen, dont la valeur dépendra de la validation ou non des contraintes. Lors de l'exploration du graphe RDF représentant l'ontologie OWL à vérifier, les booléens résultants de l'application des templates sur les nœuds du graphe sont agrégés en se servant d'une conjonction au lieu d'une concaténation, ce qui pour effet que le résultat final est un booléen qui indique si la validation du typage réussie ou échoue.

3.1 OWL 2 profiles

Les profils de OWL 2 sont des fragments logiques, ou des sous-langages, compensant le pouvoir d'expressivité par des capacités de raisonnement. Trois profils sont prédéfinis dans la recommandation : El, QL et RL. OWL EL a été défini pour des applications exploitant de très larges ontologies : il permet de se servir d'algorithmes en temps polynomial pour des tâches standards de raisonnement. OWL 2 QL et OWL 2 RL ont été définis pour des applications se servant d'un très grand nombre d'individus dans des ontologies relativement légères. OWL 2 QL sied à des applications demandant l'accès à des données par des requêtes relationnelles : il permet de répondre à des requêtes conjonctives de façon logarithmique en se servant de bases de données relationnelles. OWL 2 RL est adapté pour les applications se servant directement des données RDF : il permet de raisonner en temps polynomial grâce à un système de règles. Comme indiqué par la recommandation du W3C, chaque profil de OWL 2 possède un ensemble de restrictions sur les déclarations de OWL 2, c.-à-d. les contraintes syntaxiques sur la définition des axiomes de OWL 2 [3]. Chaque profil est défini comme (1) un ensemble de restrictions sur le type des expressions de classes qui peuvent être appliquées sur les axiomes et en lieu où ces expressions sont utilisées, (2) l'ensemble des axiomes de OWL supporté lorsque celui-ci est contraint aux expressions de classes autorisées, (3) l'ensemble des constructions de OWL non supportées. Par exemple, avec OWL RL, la construction de l'axiome `SubClassOf` composée d'une sous-classe et d'une superclasse suit des contraintes d'usages et les axiomes de OWL 2 RL suivent donc indirectement ces restrictions. De façon plus précise, il existe trois types d'expressions de classe en OWL RL : les expressions de classes présentent en tant qu'expression de sous-classe dans un axiome SubClassOf (subClassExpression), les expressions de classe présentes en tant qu'expression de super classe dans un axiome SubClassOf (SuperClassExpression) et les expressions de classe présentent dans les axiomes EquivalentClasses (equivClassExpression). Les axiomes de OWL 2 RL sont contraints d'utiliser l'expression de classe appropriée (subClassExpression, superClassExpression ou equivClassExpression). Tous les axiomes de OWL 2 sont supportés à l'exception de l'union disjointe des classes (DisjointUnion).

3. https ://www.w3.org/TF/owl2-profiles/

3.2 Transformations STTL représentant les profils de OWL 2

Nous avons défini une transformation STTL afin de représenter chacun des trois profils de OWL 2 définis dans la recommandation du W3C. Chaque transformation englobe les contraintes définies par les profils par un ensemble de templates STTL. Comme décrit auparavant, chaque profil est défini selon un ensemble de contraintes lors de la déclaration des axiomes (certains ne sont tout simplement pas supportés, d'autres le sont avec des restrictions) et également, un ensemble de contraintes sur les expressions de classe. Suivant cela, nous avons défini les transformations et les avons organisées de façon `modulaire` afin de représenter les profils : chacun est composé d'un template appelant une transformation qui va centraliser les templates représentant les contraintes sur les axiomes et ces transformations appellent de nombreuses autres transformations centralisant les templates représentant les contraintes sur les expressions de classe.

À titre d'exemple, la transformation `st:owlrl` est composée d'un ensemble de 36 templates représentant les contraintes telles que définies dans le profil de OWL 2 RL. Pour résumer, cela consiste en un seul template appelant la transformation `st:axiom` composée de templates qui à leurs tours vont appeler les transformations `st:subexp`, `st:superexp`, et `st:equivexp`. `st:owlrl` est visible en ligne [4].

3.2.1 Représenter les contraintes des axiomes de OWL 2 RL

La transformation `st:axiom` comprend 10 templates représentant les restrictions sur les axiomes de classe afin d'utiliser les expressions de classe correctes, les contraintes sur les propriétés des axiomes domain et range pour ne se servir que d'expressions de classe de type `superClassExpression`, restreindre les assertions pour n'utiliser que les expressions de classe de type `superClassExpression` et les keys pour n'utiliser que `subClassExpression`. Le résultat des templates est une valeur booléenne représentant la conformité des arguments de l'axiome. À titre d'exemple, le template suivant représente la contrainte sur l'axiome `subClassOf` pour utiliser une expression de classe du type `superClassExpression` pour la super classe et sous-classe pour l'expression de type `subClassExpression`. Ces deux types d'expressions sont définis par une autre transformation STTL qui est appelée récursivement par la clause `WHERE` du template. Plus précisément, un axiome `subClassOf` est représenté par un triplet RDF avec la propriété `rdfs:subClassOf`, la transformation `st:subClassExpression` est appelée par le sujet du triplet `?in` et la transformation `st:superClassExpression` est appelée par l'objet `?y`. Ces deux transformations renvoient un booléen dont la valeur sera dépendante de la validité des expressions de classe. Le template renvoie la conjonction de ces deux booléens. En addition à cela, le patron de conception du "Visiteur" est utilisé afin d'indiquer les axiomes ne se conformant pas aux contraintes exprimées par les profils.

```
TEMPLATE {
?suc
}
WHERE {
```

4. http ://ns.inria.fr/sparql-template/owlrl/owlrl

```
?i rdfs:subClassOf ?y
    BIND (
     st:call-template-with(st:subexp, st:subClassExpression, ?in) &&
     st:call-template-with(st:superexp, st:superClassExpression, ?y)
    AS ?suc)
    FILTER (st:alreadyVisited(?in, "subClass", ?suc))
}
```

De plus, `st:axiom` comprend un template représentant l'interdiction d'utiliser l'axiome DisjointUnion et les propriétés réflexives. Ce template renvoie la valeur `false` si l'un de ces axiomes ou propriétés est présent dans l'ontologie à valider :

```
TEMPLATE {
false
}
WHERE {
{ ?in owl:disjointUnionOf ?y }
    UNION
    { ?in a owl:ReflexiveProperty }
    FILTER (st:alreadyVisited(?in, "fail", false))
}
LIMIT 1
```

3.2.2 Représenter les contraintes sur les expressions de classe de OWL 2 RL

Comme détaillé ci-dessus, il existe trois type d'expressions de classe en OWL 2 RL : `subClassExpression`, `superClassExpression` et `equivClassExpression`. Nous avons défini pour chacun d'entre eux une transformation STTL. À titre d'exemple, prenons la transformation `st:subexp` qui représente le type d'expression de classe `subClassExpression` qui peut être utilisé dans le cadre d'une sous-classe expression avec l'axiome `SubClassOf`. Au sein de cette transformation, le template nommé `st:subClassExpression` appelle les autres templates pour cette transformation. Il permet de vérifier si l'argument est une URI, auquel cas cette dernière ne doit pas être de type `owl:Thing`, par ailleurs il vérifie que tous les templates impactés renvoient bien la valeur `true`. De plus, le patron de conception du "Visiteur" est utilisé afin de signaler les expressions non conformes.

```
TEMPLATE st:subClassExpression(?x) {
?suc
}
WHERE {
BIND (
     IF (isURI(?x), ?x != owl:Thing,
         st:apply-templates-all(?x))
    AS ?suc)
    BIND (st:visit(st:sub, ?x ?suc) as ?b)
}
```

Parmi tous les templates qui peuvent être appelés lors de la transformation, le template suivant représente les restrictions sur les expressions de sous-classes. Si l'une des propriétés d'énumération est rencontrée, le template renvoie la valeur `false`.

```
TEMPLATE {
false
}
WHERE {
?in a owl:Restriction ;
     owl:onProperty ?p

BIND (
     EXISTS {
         { ?in owl:hasValue ?v }
            UNION
            { ?in owl:someValuesFrom ?e
              FILTER (?e = owl:Thing ||
              st:call-template(st:subClassExpression, ?e)) }
        }
    AS ?suc)
}
```

Le template STTL suivant vérifie qu'une expression de classe présente dans une classe intersection ou union est de type `subClassExpression`. La clause `WHERE` sélectionne en premier lieu les déclarations `owl:intersectionOf` et `owl:unionOf`. Par la suite, cette dernière vérifie que tous les éléments `?e` dans la liste d'objets `?z` correspondent au type `subClassExpression` ou non en appelant le template `st:subClassExpression` sur tous ces éléments.

```
TEMPLATE {
?suc
}
WHERE {
?in owl:intersectionOf|owl:unionOf ?z
    ?z rdf:rest*/rdf:first ?e

    BIND (
     st:call-template(st:subClassExpression, ?e)
        AS ?suc
    )
}
```

De façon similaire à `subClassExpression`, nous avons représenté les contraintes propres aux expressions `superClassExpression` et `equivClassExpression` dans OWL 2 RL avec les templates `st:superClassExpression` et `st:equivClassExpression`. La définition de ces templates STTL suit de manière assez directe la recommandation du profil

OWL 2 RL. De façon similaire à la transformation STTL `st:owlrl` représentant le profil de OWL 2 RL, nous avons défini les transformations `st:owlql` et `st:owlel` représentant les profils OWL 2 QL et OWL 2 EL. Tous deux sont également visibles en ligne [5]. `st:owlql` comprend 24 templates et `st:owlel` 20 templates.

4 Affichage des résultats de la validation

Dans le but de fournir à l'utilisateur les résultats de la validation d'une ontologie au regard des profils de OWL 2, nous avons écrit une transformation supplémentaire afin de présenter dans un document HTML le graphe RDF (dans une syntaxe Turtle) représentant l'ontologie à valider, où les triplets incorrects se voient surlignés en rouge.

Le code permettant de présenter les déclarations sur lesquelles la validation échoue est à la fois simple et puissant. Durant le parcours du graphe RDF représentant l'ontologie à valider, un visiteur enregistre les sujets des triplets RDF correspondants aux déclarations incorrectes. Après validation, le visiteur est utilisé par une transformation STTL RDF2Turtle qui permet de représenter un graphe RDF en Turtle. Le template ci-dessous est la pierre angulaire de la transformation STTL. Il se sert de la fonction d'extension `st:visited(?in)` qui renvoie `true` lorsqu'un nœud a été visité (ce qui signifie que la déclaration en question est incorrecte). Lorsqu'un nœud du graphe RDF du vocabulaire à vérifié est traité, dans le cas où ce nœud échoue lors de la validation d'une déclaration OWL, le template STTL génère l'élément HTML `<span class='fail'> ... </span>` afin de prendre en compte la transformation du nœud en question, c.-à-d. qu'elle permet de présenter du Turtle en HTML. Une feuille de style CSS permet d'appliquer une présentation spécifique sur la classe `fail`, p. ex. une fonte de couleur rouge.

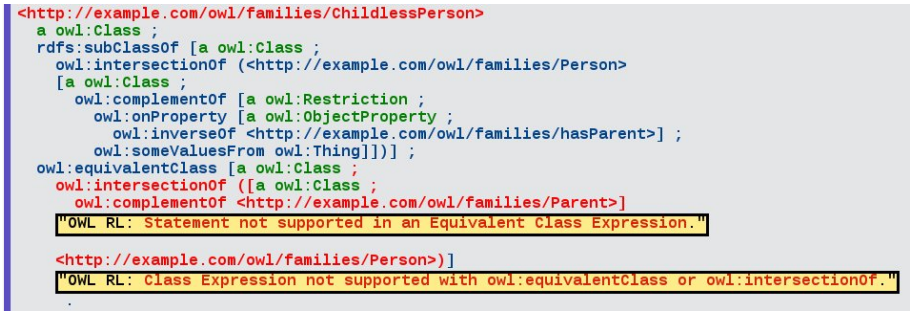

FIGURE 1 – Présentation des résultats de la validation d'une ontologie sur le profil OWL 2 RL

```
TEMPLATE {
format {
     if (st:visited(?in), "[<span class='fail'>%s</span>].", "[%].")
     ibox {
          st:call-template(st:type, ?in)
          st:call-template(st:value, ?in)
```

5. http ://ns.inria.fr/sparql-template/owlql/owlqltc et http ://ns.inria.fr/sparql-template/owlel/owleltc

```
        }
    }
}
WHERE {
?in ?p ?y
    FILTER isBlank(?in)
}
LIMIT 1
```

Ce patron de conception est générique et peut être réutilisé pour n'importe quel couple de transformation STTL permettant de valider et de visualiser les résultats. À titre d'exemple, un autre cas d'utilisation qui peut être traité avec cette même approche est le surlignement de triplets d'un graphe RDF en fonction d'une propriété donnée, p. ex. la provenance de triplets RDF : leurs couleurs peuvent permettre de distinguer ceux originellement présents au sein d'un graphe RDF et ceux venant de contraintes sur OWL.

5 Implémentation et expérimentation

Nous avons écrit une transformation STTL pour chacun des trois profils définis dans la recommandation du W3C : OWL RL (36 templates), OWL QL (24 templates) et OWL El (20 templates) [6]. Ces transformations, comme n'importe quelle autre STTL transformation, peuvent être appliquées afin de valider une ontologie OWL en se servant de Corese Semantic Web Factory qui comprend un moteur STTL. Il s'agit d'un projet open source libre en téléchargement [7]. Nous avons également écrit et déployé un service Web consacré à la validation d'ontologies OWL auprès des profils de OWL 2 avec avec comme argument une URL de l'ontologie en requête HTTP (en RDF) [8].

Nous avons en premier lieu testé nos transformations STTL sur l'échantillon d'ontologie de OWL du OWL 2 Web Ontology Language Primer [9]. Elle contient un ensemble de déclarations OWL 2. Elle décrit les concepts propres à une famille ainsi que ses relations et des individus. Elle contient 32 classes OWL, 18 relations et 7 individus. Par défaut le Web service de STTL vérifie cet échantillon d'ontologie, donc la visualisation des résultats de la validation sur cette ontologie est disponible en ligne [10]

Nous avons également testé les transformations STTL sur une ontologie propriétaire dans le domaine de l'e-Éducation, détenue par l'entreprise Educlever. Elle comprend 57 174 triplets et sa validation prend 0,5 seconde sur un ordinateur portable (HP EliteBook 840 G2, 2,6 GHz, 16GB RAM). Nous avons aussi testé les transformations afin de vérifier les contraintes des profils de OWL 2 sur l'ontologie open source Foundational Model of Anatomy (FMA) [11]. Celle-ci

6. http ://wimmics.inria.fr/corese
7. http ://wimmics.inria.fr/corese
8. consultez l'onglet OWL en en-tête de la page : htt//corese.inria.fr/
9. http ://corese.inria.fr/data/primer.owl
10. N'importe quelle ontologie peut être validée avec le Web service de Corese en changeant l'URI du service (la requête HTTP)
11. http ://sig.biostr.washington.edu/projects/fma/release/index.html

comprend 1 743 162 triplets et sa validation auprès du profil OWL 2 RL prend 3,3 secondes, auprès de OWL 2 QL 4,8 secondes et auprès de OWL 2 EL 4,6 secondes.

6 Conclusion

Nous avons montré comment répondre au problème de la validation d'ontologies par rapport à des profils de OWL 2 RL en se servant du langage STTL. Nous avons élaboré des transformations STTL pour chacun des profils de OWL 2 de la recommandation du W3C. Le moteur STTL tout comme les transformations STTL sont en libre accès et open source. Un Web service permet aussi de tester nos validateurs sur n'importe quelle ontologie (en RDF). Nous avons élaboré un patron de conception qui permet aux transformations de procéder à de la validation en renvoyant une valeur booléenne et en affichant le résultat de la validation.

L'approche que l'on utilise afin de représenter les profils de OWL 2 par le biais de transformations STTL n'est pas un problème spécifique à la validation de documents OWL, STTL peut-être utilisé afin de représenter d'autres types de contraintes sur des données RDF. Nous travaillons actuellement sur l'élaboration de transformations STTL afin d'implémenter le langage SHACL [12], soumis au W3C au stade de *candidate recommendation*, qui propose un standard pour exprimer des contraintes sur des données RDF.

Références

CORBY O. & FARON-ZUCKER C. (2010). The kgram abstract machine for knowledge graph querying. In *IEEE/WIC/ACM International Conference on Web Intelligence, WI 2010*, volume 1, p. 338–341 : IEEE.

CORBY O. & FARON-ZUCKER C. (2014). SPARQL template : un langage de pretty printing pour RDF. In *25es Journées francophones d'Ingénierie des Connaissances, IC 2014, Clermont Ferrand, France*, p. 213–224.

CORBY O. & FARON-ZUCKER C. (2015). STTL : A SPARQL-based Transformation Language for RDF. In *11th International Conference on Web Information Systems and Technologies, WEBIST 2015*, Lisbon, Portugal.

CORBY O. & FARON-ZUCKER C. (2015). Un navigateur pour les données liées du web. In *26es Journées francophones d'Ingénierie des Connaissances, IC 2015, Rennes, France*.

CORBY O. & FARON-ZUCKER C. (2016). Un langage et un serveur de transformation de graphes pour le web de données. *Revue d'Intelligence Artificielle*, **30**(5), 607–627.

CORBY O., FARON-ZUCKER C. & GANDON F. (2015). A Generic RDF Transformation Software and its Application to an Online Translation Service for Common Languages of Linked Data. In *14th International Semantic Web Conference, ISWC 2015*, Bethlehem, Pennsylvania, USA.

CORBY O., FARON-ZUCKER C. & GAZZOTTI R. (2016). Validating ontologies against OWL 2 profiles with the SPARQL template transformation language. In *10th International Conference on Web Reasoning and Rule Systems, RR 2016, Aberdeen, UK*, volume 9898 of *Lecture Notes in Computer Science*, p. 39–45 : Springer.

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

12. https ://www.w3.org/TR/shacl/

HORRIDGE M. & BECHHOFER S. (2011). The OWL API : A Java API for OWL ontologies. *Semantic Web Journal*, p. 11–21.