Validating Ontologies Against OWL 2 Profiles with the SPARQL Template Transformation Language

Olivier Corby(✉), Catherine Faron-Zucker(✉), and Raphaël Gazzotti

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

Abstract. In this paper we address the general research question of *How can we express constraints on RDF data and how can we check that an RDF graph satisfies some given constraints?* and we focus on expressing constraints defining OWL 2 profiles and checking these constraints for OWL validation. We propose an approach based on the SPARQL Template Transformation language (STTL). An STTL template is a transformation rule that applies to a given RDF graph and the recursive call of a set of STTL templates on an RDF graph outputs some textual data resulting from the transformation of this graph. We show that STTL can be used as a constraint language for RDF and we use it to implement the semantics of OWL 2 profiles: each profile is represented by a set of STTL templates that a valid ontology must satisfy.

1 Introduction

OWL 2 profiles [6] can be seen as restrictions of OWL 2 statements and the validation of ontologies against OWL 2 profiles as the checking of syntactic constraints on OWL 2 axiom declarations. In this paper we address the general research question of *How can we express constraints on RDF data and how can we check that an RDF graph satisfies some given constraints?* and we focus on expressing constraints defining OWL 2 profiles and checking these constraints for OWL validation.

We propose an approach based on the SPARQL Template Transformation language (STTL), which we originally designed in order to enable the transformation of RDF data into any data format. An STTL template can be viewed as a transformation rule that applies to a given RDF graph just like an XSL template applies to an XML tree, and the recursive call of a set of STTL templates on a whole RDF graph outputs some textual data resulting from the transformation of this graph.

We show that STTL can be used as a constraint language for RDF: each STTL template is viewed as representing a constraint and an RDF graph is checked against a set of constraints by applying the set of STTL templates representing these constraints on the RDF graph. The output of the application of a set of STTL templates can be a simple boolean value or a convenient textual

M. Ortiz and S. Schlobach (Eds.): RR 2016, LNCS 9898, pp. 39–45, 2016.
DOI: 10.1007/978-3-319-45276-0_4

view of the data, where for instance, the subgraphs violating the constraints are highlighted. This is done by defining a "Visitor" design patten associated to the set of STTL templates in order to collect illegal RDF sub-graphs, and a generic design pattern to display the result to the user.

As a result, we apply our approach to implement the semantics of OWL 2 profiles, each viewed as a set of constraints to be validated: we defined an STTL transformation to represent each of the three OWL 2 profiles (OWL RL, OWL QL and OWL EL). The application of one of these STTL transformations to an ontology (expressed in RDF) enables to validate it against the OWL 2 profile this transformation represents.

The paper is organized as follows. Section 2 provides an overview of the STTL language. Section 3 presents the STTL transformation implementing the semantics of the OWL 2 profiles. Section 4 shows how an additional STTL transformation enables to provide the user with a visual presentation of the results of the OWL validation results. Section 5 describes our experiments conducted on several OWL ontologies of the Linked Data. Section 6 concludes.

2 SPARQL Template Transformation Language (STTL)

STTL is a generic transformation rule language for RDF which relies on two extensions of SPARQL: an additional TEMPLATE query form to express transformation rules and extension functions to recursively call the processing of a template from another one. A TEMPLATE query is made of a standard WHERE clause and a TEMPLATE clause. The WHERE clause is the condition part of a rule, specifying the nodes in the RDF graph to be selected for the transformation. The TEMPLATE clause is the presentation part of the rule, specifying the output of the transformation performed on the solution sequence of the WHERE part. For instance, let us consider the OWL axiom stating that the class of parents is equivalent to the class of individuals having a person as child. Here are its expressions in Functional syntax and in Turtle:

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

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

The template below enables to transform the above `equivalentClass` statement from RDF into Functional syntax:

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

The value matching variable `?in` is `a:Parent` which is expected in the transformation output (the Functional syntax of the OWL 2 statement), while the value

matching variable `?c` is a blank node whose property values are used to build the expected output. This is defined in another template to be applied on this focus node. The `st:apply-templates` extension function enables this recursive call of templates, where `st` is the prefix of STTL namespace[1].

More generally, function `st:apply-templates` can be used in the TEMPLATE clause of any template t_1 to execute another template t_2 that can itself execute a template t_3, etc. Hence, templates call themselves one another, in a series of calls, enabling a hierarchical processing of templates and a recursive traversing of the target RDF graph. The STTL interpreter keeps track of templates and focus nodes in order to prevent loops as RDF graphs may have cycles. Similarly, function `st:call-template` can be used to recursively call named templates.

STTL is compiled into standard SPARQL. The compilation keeps the WHERE clause, the solution modifiers and the VALUES clause of the template unchanged and the TEMPLATE clause is compiled into a SELECT clause.

A complete description of STTL language is provided in [1]. We implemented the STTL language and transformer engine within the Corese Semantic Web Factory [3] which now comprises an `STTL` RESTful Web service to process STTL transformations and output the result of transforming an RDF dataset. This implementation is described in [2].

3 Validating OWL 2 Profiles with STTL Transformations

OWL 2 profiles are logic fragments, or sublanguages, trading expressive representation power for efficient reasoning capabilities. There are three profiles predefined in the recommendation: EL, QL and RL. As stated in the W3C recommendation, each OWL 2 profile is defined as a set of restrictions on the structure of OWL 2 statements, i.e. syntactic constraints on OWL 2 axioms definitions[2]. Each profile is defined as (1) a set of restrictions on the type of class expressions that can be used in axioms and on the place in which they can be used, (2) the set of OWL axioms supported when restricted to the allowed set of class expressions, (3) the set of OWL constructs which are not supported. For example, in OWL 2 RL, the constructs in the subclass and superclass expressions in `SubClassOf` axioms must follow some usage patterns and OWL 2 RL axioms are undirectly constrained by these restrictions.

We defined an STTL transformation to represent each of the three OWL 2 profiles defined in the W3C recommendation. Each STTL template participating to these transformations enables to check a specific OWL 2 model constraint and returns a boolean, the value of which depends on whether the constraint is verified or not. When traversing the RDF graph representing the ontology to be validated against a given OWL 2 profile, the boolean results of the templates applied to the graph nodes are aggregated by using a conjunction instead of a concatenation, so that the final result is a boolean value indicating whether type checking succeeds or fails.

[1] http://ns.inria.fr/sparql-template/.
[2] https://www.w3.org/TR/owl2-profiles/.

Considering that each OWL 2 profile is defined by a set of constraints for the declaration of axioms (some axioms are not supported, some are supported with restrictions) and a set of constraints on class expressions, we defined *modular* STTL transformations to represent OWL 2 profiles. Basically, each one consists in a single template calling a transformation gathering templates representing constraints on axioms and these transformations call several other transformations gathering templates representing constraints on class expressions.

Let us focus on the `st:owlrl`[3] transformation which comprises 36 STTL templates representing the constraints defining the OWL 2 RL profile. It consists of a start template calling the `st:axiom` transformation whose templates themselves call the `st:subexp`, `st:superexp`, and `st:equivexp` transformations. Transformation `st:axiom` comprises 10 templates representing restrictions on class axioms to use the appropriate form of class expressions, restrictions on property domain and range axioms to only use class expressions of type `superClassExpression`, restriction on positive assertions to only use class expressions of type `superClassExpression` and restrictions on keys to only use `subClassExpression`.

The result of each template is a boolean value that represents the conformance of the axiom arguments. For instance, the following template represents the restriction on `subClassOf` axioms to use a class expression of type `superClassExpression` (respectively `subClassExpression`) for the superclass (respectively the subclass). These two types of class expressions are each defined by another STTL transformation which is recursively called in the WHERE clause of the template. More precisely, a `subClassOf` axiom is represented by an RDF triple whose property is `rdfs:subClassOf`, whose subject `?in` is passed as argument to transformation `st:subClassExpression` and whose object `?y` is passed as argument to transformation `st:superClassExpression`. Both transformations return a boolean whose value corresponds to the conformance of the class expressions. The template returns the conjunction of these two booleans. In addition, a "Visitor" design pattern is used to report axioms which are not conform to the profile.

```
TEMPLATE { ?suc }
WHERE {
  ?in 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) }
```

In addition, `st:axiom` comprises one template representing the disallowance of the `DisjointUnion` axiom and of reflexive properties. This template returns `false` if such an axiom or property occurs in the ontology.

[3] http://ns.inria.fr/sparql-template/owlrl/owlrl.

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

We defined an STTL transformation for each of the three types of class expressions in OWL 2 RL: `subClassExpression`, `superClassExpression` and `equivClassExpression`. For instance, let us consider the `st:subexp` transformation representing the `subClassExpression` type of class expressions, that can occur as subclass expressions in `SubClassOf` axioms. In this transformation, the following named template `st:subClassExpression` calls for all the other templates in the transformation. It enables to checks whether the argument is a URI, in which case it must not be `owl:Thing`; otherwise it checks whether all the templates matching the argument return true. In addition, a "Visitor" design pattern is used to report expressions that do not conform.

```
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) }
```

4 Validation Result Presentation

In order to provide the user with a visualization of the result of the validation, we wrote an STTL transformation to present in a HTML document the RDF graph (in the Turtle syntax) representing the ontology to be validated, where non valid triples are highlighted. For instance, Fig. 1 shows the visualization of an ontology represented in Turtle and tested against the OWL 2 RL profile with `owl:complementOf` in red since OWL 2 RL does not allow this within a class intersection inside a class equivalence.

During the traversal of the RDF graph representing the tested ontology, a visitor records the subjects of RDF triples corresponding to failing statements. After type check resumes, the visitor is given to an STTL transformation RDF2Turtle which enables to pretty-print RDF graphs in Turtle. The template below is the key of the STTL transformation. It uses the `st:visited(?in)` extension function which returns true if the node has been visited (and hence represents a failing statement). When processing a node of the RDF graph representing the vocabulary to be validated, in case this node represents a failing OWL statement, the STTL template generates a `<span class='fail'> ... </span>` HTML element to embed the transformation of the node, i.e. its pretty-print in Turtle embeded in HTML. A CSS stylesheet associates a specific presentation format to the `fail` class, e.g. a red font color.

```
<http://example.com/owl/families/ChildlessPerson>
  a owl:Class ;
  rdfs:subClassOf [a owl:Class ;
    owl:intersectionOf (<http://example.com/owl/families/Person>
    [a owl:Class ;
      owl:complementOf [a owl:Restriction ;
        owl:onProperty [a owl:ObjectProperty ;
          owl:inverseOf <http://example.com/owl/families/hasParent>] ;
        owl:someValuesFrom owl:Thing]])] ;
  owl:equivalentClass [a owl:Class ;
    owl:intersectionOf ([a owl:Class ;
      owl:complementOf <http://example.com/owl/families/Parent>]
    "OWL RL: Statement not supported in an Equivalent Class Expression."

    <http://example.com/owl/families/Person>)]
    "OWL RL: Class Expression not supported with owl:equivalentClass or owl:intersectionOf."
    .
```

Fig. 1. Visualizing the validation result of an ontology against OWL 2 RL

```
TEMPLATE { FORMAT {
  if (st:visited(?in),"[<span class='fail'>%s</span>].","[%s].")
  ibox { st:call-template(st:type, ?in)
         st:call-template(st:value, ?in) } }}
WHERE { ?in ?p ?y  FILTER isBlank(?in) } LIMIT 1
```

5 Implementation and Experiments

We have written an STTL transformation for the three OWL profiles defined in the W3C recommendation: OWL RL (36 templates), OWL QL (24 templates) and OWL EL (20 templates)[4]. These transformations, like any other STTL transformations, can be applied to an OWL ontology to be validated by using the Corese Semantic Web Factory which comprises an STTL engine. This is an open-source development that can be freely downloaded[5]. We also wrote and deployed a dedicated Web service that can validate an OWL ontology against OWL 2 profiles given the URL of the ontology as an argument in the HTTP request (in RDF)[6]. We also have tested the STTL transformations on a proprietary ontology in the e-Education domain, owned by the Educlever company. It comprises 57,174 triples and its validation took 0.5 s on a laptop (HP EliteBook 840 G2, 2.6 GHz, 16 GB RAM). Finally, we have tested the STTL constraint checking transformations on the open source Foundational Model of Anatomy (FMA) ontology[7]. It comprises 1,743,162 triples and its validation against OWL RL takes 3.3 s, against OWL QL 4.8 s, and against OWL EL 4.6 s.

6 Conclusion

We have shown how to answer the problem of OWL 2 RL Profile conformance checking by using the STTL language. We have designed an STTL

[4] http://ns.inria.fr/sparql-template/.
[5] http://wimmics.inria.fr/corese.
[6] http://corese.inria.fr/.
[7] http://sig.biostr.washington.edu/projects/fma/release/index.html.

transformation for each of the OWL 2 profiles in the W3C recommendation. The STTL engine as well as the STTL transformations are freely available and open-source and a Web service enables to test our validators with any ontology (in RDF). We have created a design pattern that enables transformations to perform type checking by returning boolean values and pretty-print the result of the validation. As future work, we will provide a comparison of our OWL 2 validator to the validator developed by the University of Manchester[8] which relies on the OWL API [5].

Our approach to represent OWL 2 profiles by STTL transformations is not specific to the problem of OWL validation and STTL can be used to represent other kinds of constraints on RDF data. Therefore, as future work, we will compare our approach to related works on RDF constraint checking, among which [4]. Relatedly, the W3C hosts a RDF Data Shapes[9] working group for describing structural constraints and validate RDF data against those and we are currently designing an STTL transformation implementing the current version of W3C RDF Data Shapes.

References

1. Corby, O., Faron-Zucker, C.: STTL: a SPARQL-based transformation language for RDF. In: 11th International Conference on Web Information Systems and Technologies, WEBIST 2015, Lisbon, Portugal, May 2015
2. Corby, O., Faron-Zucker, C., Gandon, F.: A generic RDF transformation software and its application to an online translation service for common languages of linked data. In: Arenas, M., et al. (eds.) ISWC 2015. LNCS, vol. 9367, pp. 150–165. Springer, Heidelberg (2015)
3. Corby, O., Gaignard, A., Faron-Zucker, C., Montagnat, J.: KGRAM versatile data graphs querying and inference engine. In: IEEE/WIC/ACM International Conference on Web Intelligence, Macau, China (2012)
4. Fischer, P.M., Lausen, G., Schätzle, A., Schmidt, M.: RDF constraint checking. In: Proceedings of the Workshops of the EDBT/ICDT 2015 Joint Conference. CEUR Workshop Proceedings, Brussels, Belgium, vol. 1330, pp. 205–212 (2015)
5. Horridge, M., Bechhofer, S.: The OWL API: a Java API for OWL ontologies. Semantic Web J. **2**, 11–21 (2011)
6. Motik, B., Grau, B.C., Horrocks, I., Zhe, W., Fokoue, A., Lutz, C.: OWL 2 Web ontology language profiles. Recommendation, W3C (2012). http://www.w3.org/TR/owl2-profiles/

[8] http://mowl-power.cs.man.ac.uk:8080/validator/.
[9] http://www.w3.org/2014/data-shapes/wiki/Main_Page.