A Generic RDF Transformation Software and its Application to an Online Translation Service for Common Languages of Linked Data

Olivier Corby[1], Catherine Faron-Zucker[2], and Fabien Gandon[1]

[1] Inria, France
[2] Univ. Nice Sophia Antipolis, CNRS, I3S, France

Abstract. In this article we present a generic template and software solution for developers to support the many cases where we need to transform RDF. It relies on the SPARQL Template Transformation Language (STTL) which enables Semantic Web developers to write specific yet compact RDF transformers toward other languages and formats. We first briefly recall the STTL principles and software features. We then demonstrate the support it provides to programmers by presenting a selection of STTL-based RDF transformers for common languages. The software is available online as a Web service and all the RDF transformers presented in this paper can be tested online.

Keywords: RDF, SPARQL, STTL, Transformation, Software, Tool

1 Introduction

The RDF standard [8] provides us with a general purpose graph-oriented data model recommended by the W3C to represent and interchange data on the Web. While the potential of a world-wide Semantic Web of linked data and linked data schemas is now widely recognized, the transformation and presentation of RDF data is still an open issue. Among the initiatives to answer this question there are extensive works for providing RDF with several varied syntaxes (XML, N-Triples, Turtle, RDFa, TriG, N-Quads, JSON-LD) and for linking it to other data sources (R2RML, CSV-LD, etc.). With the multiplication of data sources and data formats, developers of the Web of data now spend a lot of time and energy to build transformations to present RDF data to users and transform data from one source to another. Moreover, a special case of RDF data holds a very special potential as RDF is more and more used as a syntax to represent other languages. For instance in the domain of the Semantic Web alone, this is the case of three W3C standards: OWL 2 [13] is provided with several syntaxes, among which the Functional syntax, the Manchester syntax used in several ontology editors and RDF/XML and RDF/Turtle; the Rule Interchange Format (RIF) [10] is provided with several syntaxes among which an XML syntax, two compact syntaxes for RIF-BLD and RIF-PRD and an RDF syntax; SPARQL Inference Notation (SPIN) is a W3C member submission [12] to represent SPARQL rules

in RDF, to facilitate storage and maintenance. Many other languages can (and will) be "serialized" into RDF. For instance [9] is an attempt to represent SQL expressions in RDF, and we consider that RDF can then be viewed as a pivot language to represent the abstract syntax trees of expressions of other languages.

For this reason, we present a software solution for developers to support the many cases where we need to transform RDF. We rely on the SPARQL Template Transformation Language (STTL) which enables Semantic Web developers to write specific yet compact RDF transformers toward other languages and formats. As an example of application of STTL we can mention the implementation of a SPARQL tutorial that was successfully used in a Semantic Web MOOC with over 4000 students[3].

Since the applications of STTL are varied we categorized them into five main families: the case of transformation between RDF syntaxes (e.g., RDF/XML to Turtle); the generation of presentation formats (e.g., HTML pages); the exports towards other data formats (e.g., CSV); the pretty-printing of statements of a given language represented in RDF syntax (e.g., SPIN to SPARQL); and the use of RDF as a pivot model between two languages. In addition to the above cited transformations which we already have developed, we are receiving expressions of interest for new transformations including the translation to graph file formats like DOT[4], to Topic Maps[5], the generation of PROV-O and Ratio4TA annotations, the anonymization of data, etc. and several of these transformation are currently being developed. An alternative or complementary categorization is the distinction between domain independent transformations (e.g., transformations between RDF syntaxes, generic presentations of RDF triples into HTML tables, OWL/RDF to OWL/FS transformation) and domain or application dependent transformations (e.g., a customized export to CSV or HTML rendering).

In this article we briefly recall STTL principles. Then we present the software features and we demonstrate the support this software provides to programmers by presenting a selection of STTL-based RDF transformers for common languages. The software is available online as a Web service and all the RDF transformers presented in this paper can be tested online[6]. The paper is organized as follows: Section 2.1 presents the STTL language. Section 3 presents the generic transformation rule engine we developed to implement STTL and the Web service encapsulating it. The following sections present several specific RDF transformers written in STTL, illustrating the different families of transformations we identified. Section 7 concludes.

[3] https://www.france-universite-numerique-mooc.fr/courses/inria/41002/Trimestre_1_2015/about
[4] http://www.graphviz.org/Documentation.php
[5] http://www.topicmaps.org/
[6] http://corese.inria.fr

2 Transforming RDF

2.1 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 condition. 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 following template enables to transform the above `equivalentClass` statement from RDF into Functional syntax:

```
TEMPLATE {
 "EquivalentClasses("
 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[7] 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[8].

More generally, `st:apply-templates` function can be used in any template t_1 to execute another template t_2 that can itself execute a template t_3, etc. Hence,

[7] Let us note that blank nodes are handled like any other node (URIs and literals). If needed, function `isBlank()` enables to detect them to apply specifically written templates on them.

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

templates call themselves one another, in a series of call, enabling a hierarchical processing of templates and a recursive traversing of the target RDF graph. Similarly, `st:call-template` function can be used to call named templates. Table 1 summerizes the list of the Core STTL functions.

Name	Description
st:apply-templates	Apply current transformation on focus node
st:apply-templates-with	Apply given transformation on focus node
st:apply-templates-all	Apply all templates on focus node
st:call-template	Apply named template on focus node
st:call-template-with	Apply named template of given transformation on focus node
st:define	Define a template function (e.g. st:process)
st:process	Define the processing of template variables
st:get	Get a property value from Context
st:set	Set a property value into Context
st:turtle	Display Turtle syntax of focus node
st:nl	Insert newline

Table 1. STTL Core Functions

Following the layer-cake standardization of the semantic Web, STTL is compiled into standard SPARQL. This allows the approach to be usable with different implementations of the standard, to benefit from its expressiveness, from the native extension mechanisms and also from the optimizations of the implementations. 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. This also allows STTL to benefit from all SPARQL features for instance, when needed, the DISTINCT solution modifier can be used in a nested subquery to avoid duplicates. For instance, the TEMPLATE clause of the following STTL template:

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

is compiled into the following standard SPARQL SELECT clause:

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

The WHERE clause is unchanged.

2.2 Work related to STTL

A complete description of STTL language is provided in [6] together with an extended presentation of the state-of-the-art approaches addressing the problem of RDF transformation. We briefly summarize here this state-of-the-art of languages addressing the problem of RDF transformation stressing that STTL is independent of the syntax of its RDF input and addresses the general problem of the *transformation* of RDF data into any output format.

OWL-PL [4] is an extension of XSLT for transforming RDF/OWL into XHTML; it is both tied to its RDF/XML input format and its output format. Fresnel [3] is an RDF vocabulary for specifying in RDF which data contained in an RDF graph should be displayed and how. Again, it is tied to a specific display paradigm and an XHTML-like output format.

SPARQL is provided with a CONSTRUCT query form which enables to extract and *transform* RDF data into RDF according to any other schema. [1] addresses the problem of generating XML from RDF data with an extended SPARQL query. Here again, the solution is specific to one output format. XSPARQL [2] is a combination of SPARQL and XQuery [15] enabling to query both XML and RDF data and to transform data from one format into the other. [16] proposes an XML-based transformation language, inspired by XSLT, that mainly matches types of RDF resources. [14] proposes an XML-based stylesheet language also inspired by XLST where templates match triple patterns and generate HTML.

Finally, there is quite a wide range of ad hoc RDF parsers and validators[9], some of which enable to transform RDF data from one syntax into another. Among them, let us cite RDF Distiller[10] and RDF Translator[11]. A review of these RDF-to-RDF converters can be found in [17]. Another famous example of specific-purpose RDF transformer is the RDF/XML parser in OWL API[12] [11] which enable to transform OWL 2 statements in RDF/XML into the Functional syntax of the language.

3 STTL Engine

We implemented a STTL engine within the Corese Semantic Web Factory[13] [7,5]. It comprises an `STTL` RESTful Web service to process STTL transformations. In this section, we first describe our implementation of STTL then the Web service which encapsulate the STTL engine.

3.1 Implementation of STTL

Algorithm. Basically, the STTL engine is called by `st:apply-templates` or other alike extension functions. Given an RDF graph with a focus node to be

[9] http://www.w3.org/2001/sw/wiki/Category:Tool
[10] http://rdf.greggkellogg.net/distiller
[11] http://rdf-translator.appspot.com/
[12] http://owlapi.sourceforge.net/
[13] http://wimmics.inria.fr/corese

transformed and a list of templates, it successively tries to apply them to the focus node until one of them succeeds. A template succeeds if the matching of the WHERE clause succeeds, i.e., returns a result. If no template succeeds, the `st:default` named template (if any) is applied to the focus node. Recursive calls to `st:apply-templates` within templates implements the graph recursive traversal with successive focus nodes. The engine keeps track of the templates applied to nodes in order to avoid cycles, i.e. to avoid to apply the same template on the same node twice in case the RDF graph is cyclic. If there is no template to be applied on a focus node that has not previously been applied on it, the transformer calls the `st:default` named template if any, otherwise the Turtle format of the focus node is returned.

Template Selection. By default, the STTL engine considers templates in order: given a focus node, in response to a call to the `st:apply-templates` function, it considers the first template that matches this node. Alternatively, the processing of a named template is commanded by a call to the `st:call-template` function. In both cases, the result of the transformation of the focus node is the result of the template.

In some other cases, it is worth writing *several* templates for a type of node, in particular when the node holds different graph patterns that should be transformed according to different presentation rules. Executing several templates on the focus node is done by calling the `st:apply-templates-all` function. The result of the transformation is the concatenation of the results of the successful templates.

A transformer can be used to transform a whole RDF graph — without any distinguished root node in the graph. For this purpose, the `st:apply-templates-with` function can be called without focus node and the transformer must then determine it. By default, the first template that succeeds is the starting point of the transformer; or a `st:start` named template can be defined to be executed first.

Transformation Settings. The `st:start` named template, if any, is selected at the beginning of the transformation process when no focus node is available. In that case, it is the first template executed by the template engine. The `st:default` template, if any, is executed when all templates fail to match the focus node.

The processing of a variable in the TEMPLATE clause by default consists in outputting its value in the Turtle format. The `st:profile` template can be used to overload this default transformation behaviour. For example, the following definition of `st:profile` specifies that processing variables, denoted by `st:process(?x)`, consists in the application of the `st:apply-templates` function to it.

```
TEMPLATE st:profile {
  st:define( st:process(?x) = st:apply-templates(?x) )
WHERE { }
```

3.2 STTL-based RDF Transformers

In our approach of RDF transformation based on STTL, the STTL engine, i.e. the template processor, is generic: it applies to any RDF data with any set of STTL templates. What is specific to each transformation is the set of STTL templates defining it. In other words, each RDF transformer specific to an output format is defined by a specific set of STTL templates processed by the generic template processor implementing STTL. In Sections 4 to 6, we present specific STTL-based RDF transformers. Each RDF transformer may be accessed as a Web service. We present in Section 3.4 our implementation of a STTL RESTfull Web service in the Corese Semantic Web Factory.

3.3 STTL Development Environment

The Corese Semantic Web Factory provides a standalone environment with a GUI, enabling the user to load RDF data and ontologies, to load or write SPARQL queries and STTL templates as well and easily test them against the loaded RDF data. This tool enables developers an easy handling, with a fast learning curve, of Semantic Web technologies in general, and of STTL in particular.

3.4 STTL service

The Corese Semantic Web Factory provides a SPARQL endpoint by means of a RESTfull Web service which implements SPARQL 1.1 Protocol[14]. In addition to this standard implementation, Corese proposes an STTL RESTful Web service to process STTL transformations on local or distant RDF dataset. Figure 1 presents the general architecture of the server.

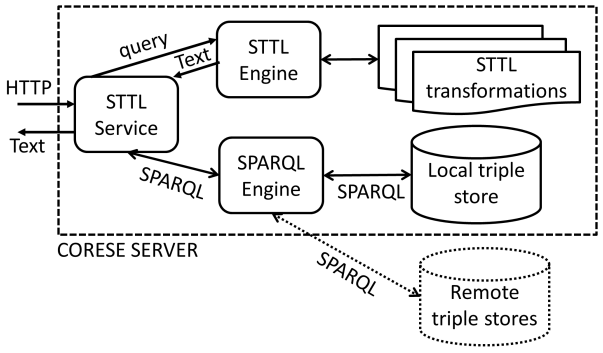

Fig. 1. Architecture of a Corese STTL-based RDF Transformation Server

[14] http://www.w3.org/TR/sparql11-protocol/

A request for an STTL transformation of RDF data is conveyed to the transformation service in a URL whose hierarchical part ends with `/template`[15] and whose query part comprises key-value pairs specifying the request to the service. The `query` key is reused from SPARQL 1.1 Protocol to indicate a SPARQL query to be performed by the server on its RDF dataset. The URL of the STTL transformation to be applied to the result of the SPARQL query (or to the RDF dataset) is specified as the value of a `transform` key. For instance, the following URL asks for the transformation of all the triples of the RDF dataset with a STTL transformation specified at `st:sparql`.

```
http://localhost:8080/template?
  query=SELECT * WHERE { ?x ?p ?y }&transform=st:sparql
```

In order to simplify the interaction with a `STTL` service, we define the notion of *profile* of a transformation, assembling an optional SPARQL query and a STTL transformation into a simple workflow. A profile is described in RDF as follows: a `st:query` property and a `st:transform` property associate a SPARQL query and a STTL transformation to a profile. Here is an example of a profile description:

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

with `q1.rq` containing for instance the following SPARQL query:

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

In the URL conveying the request for a STTL transformation, the URI of a profile is indicated to the STTL service as the value of a `profile` key. Here is an example of such a URL:

```
http://localhost:8080/template?profile=st:dbpedia
```

We defined the notion of *context* of a transformation that enables the STTL service to send parameters to the transformer. This context is set up by the service and passed to the transformer. The transformer can access the context with the `st:get` extension function which we have defined to return the value of a context parameter. Context parameters are the name of the service, the service profile, the transformation, the query, and the focus URI. For instance here is the context of an RDF-to-HTML transformation described in section 5.

```
st:get(st:service)   = /template
st:get(st:profile)   = st:dbpedia
st:get(st:transform) = st:navlab
st:get(st:query) =
  CONSTRUCT { ?x ?p ?y }
  WHERE { SERVICE <http://fr.dbpedia.org/sparql> {?x ?p ?y} }
st:get(st:uri) = http://fr.dbpedia.org/resource/Antibes
```

[15] e.g., http://corese.inria.fr/template

4 RDF-to-RDF Transformers

The first family of RDF transformations we identified in the introduction of this paper comprises RDF-to-RDF transformations, i.e., the transformation of RDF data from any RDF syntax into any other RDF syntax, e.g., RDF/XML-to-Turtle. In this family the transformations are completely domain-independent. There exist several online transformation services such as RDF Translator for instance but we show here how to implement your own transformation in a declarative way with STTL. As an exercice in style, we wrote RDF-to-Turtle and RDF-to-RDF/XML transformations. The main template of the RDF-to-Turtle transformation is shown below, it is available online[16]:

```
TEMPLATE st:start {
  st:apply-templates(?x) }
WHERE {
  SELECT DISTINCT ?x WHERE {
    ?x ?p ?y
    FILTER (
      isURI(?x) ||
      NOT EXISTS { ?z ?q ?x } ||
      ( isBlank(?x) &&
        ( EXISTS { ?a ?q ?x . ?b ?r ?x . FILTER(?a != ?b) } ||
          EXISTS { ?x ?q ?x } ) ) )
  } ORDER BY ?x}
```

A demo RDF/XML-to-Turtle transformation service is available online[17].

We wrote an RDF-to-RDF/XML transformation as a set of 21 STTL templates available online[18]. Here is a key template of it to express the description of a resource as the subject of some RDF triples:

```
TEMPLATE st:resource(?in) {
  "<rdf:Description " ?att "='" str(?in) "'>"
    box { st:call-template(st:property, ?in) }
  "</rdf:Description>"
}
WHERE { BIND ( if (isBlank(?in), rdf:nodeID, rdf:about) as ?att) }
```

Based on these STTL transformations, the Corese Semantic Web Factory enables to deploy RDF/XML-to-Turtle, RDFa-to-Turtle, Turtle-to-RDF/XML and RDFa-to-RDF/XML transformation services since the distribution is provided with a Turtle parser, an RDF/XML parser, and an RDFa parser.

5 RDF-to-HTML Transformers

The second family of RDF transformations identified in the introduction of this paper gathers the transformations of RDF data for presentation purposes, in

[16] http://ns.inria.fr/sparql-template/turtle
[17] http://corese.inria.fr
[18] http://ns.inria.fr/sparql-template/rdfxml

any given presentation format. In particular, it comprises RDF-to-HTML transformations. In this section, we present three examples of RDF-to-HTML transformations which enables to design Linked Data navigators. A demo server is accessible online[19]. Its source code is freely available within the Corese Semantic Web Factory[20].

5.1 Principles of Linked Data Navigation: Dynamic HyperLinks to the STTL Service

The keys to build a Linked Data navigator is (1) to generate HTML pages for RDF resources and (2) to generate hyperlinks in the HTML code output. This is achieved with *href* attributes having as value a URL conveying a request for STTL transformation to the transformation service. Here is an example of a named STTL template to construct a hyperlink to a focus URI `?x`:

```
TEMPLATE st:link(?x) {
 "<a href='/template?profile=st:dbpedia&uri=" str(?x) "'>" str(?x) "</a>"}
WHERE { }
```

In order to avoid hardcoding the service and the profile names, these can be extracted from the context of the transformation. Here is an example of such a generic template:

```
TEMPLATE st:link(?x) {
   "<a href='" st:get(st:service) "?profile=" st:get(st:profile)
   "&uri=" str(?x) "'>" str(?x) "</a>" }
WHERE { }
```

When applied on a given URI the two above templates would produce for instance the following output code:

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

5.2 Three Examples of Linked Data Navigators

Basic Navigator. The simplest navigator enables the user to send a SPARQL query to the server and get back the results presented in an HTML table in a completely domain-independent way. The results of SELECT queries are translated in RDF by using the Vocabulary for recording query result set published by the W3C RDF Data Access Working Group[21]. The `st:sparql` transformation[22] is then applied on this RDF result graph. The output HTML code contains

[19] http://corese.inria.fr
[20] http://wimmics.inria.fr/corese
[21] http://www.w3.org/2001/sw/DataAccess/tests/result-set.n3
[22] http://ns.inria.fr/sparql-template/sparql

hyperlinks to URLs, conveying further requests to the STTL server to apply a transformation to the resources involved in the description produced.

Here is an example of query solution in RDF:

```
prefix rs:<http://www.w3.org/2001/sw/DataAccess/tests/result-set#>
[] rs:resultVariable "x", "n" ;
   rs:solution [
     rs:binding [
       [rs:variable "x" ; rs:value ex:Auguste],
       [rs:variable "n" ; rs:value "Auguste"] ] ] .
```

Here is the main template of `st:sparql` that processes such an RDF graph solution of a SELECT query processed by the server.

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

The WHERE clause focus is a solution `?in` which is a set of variable bindings. The OPTIONAL clause enumerates these bindings. This enumeration is optional because some variables (`?var`) may be unbound. For each binding, the TEMPLATE clause generates a cell, in an HTML table, with the value of the variable (`?val`). For unbound variables, a space character is generated in the cell.

DBpedia Navigator. Another kind of navigators are domain-specific Linked Data Navigators. We developed such a navigator — a server with its STTL service and the `st:navlab` RDF-to-HTML transformation — to browse the DBpedia dataset, specifically on persons and places. Figure 2 is the screenshot of an HTML page produced by this navigator. We wrote the `st:navlab` transformation as a set of 24 STTL templates which are available online[23]. Here is a template in `st:navlab`, to construct the table of resource descriptions; it recursively calls the `st:title` named template to output the title in HTML and the `st:descresource` to build the description of each resource selected in DBpedia.

```
TEMPLATE {
  st:call-template(st:title, ?in, ?label, (coalesce(?ic, "")))
  "<table>" st:call-template(st:descresource, ?in) "</table>" }
WHERE {
  ?in a <http://dbpedia.org/ontology/Resource> .
  ?in rdfs:label ?label FILTER(lang(?label) = 'fr')
  OPTIONAL { ?in <http://dbpedia.org/ontology/thumbnail>  ?ic } }
```

[23] http://ns.inria.fr/sparql-template/navlab

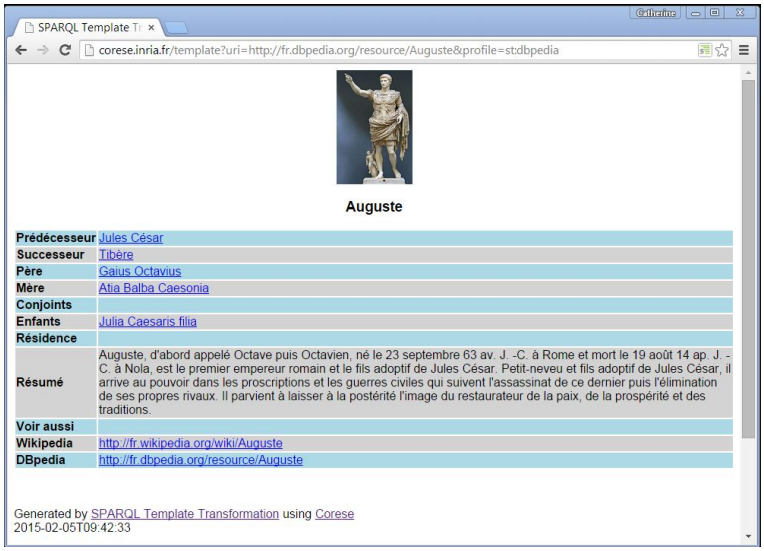

Fig. 2. DBpedia Navigator

The DBpedia SPARQL endpoint is accessed through a SERVICE clause in a predefined CONSTRUCT query to retrieve relevant data according to the types of resources that the application is interested in: our navigator focuses on historical people and places. Then the `st:navlab` transformation is applied to the resulting RDF graph. It generates a presentation in HTML format of the retrieved data, adapted to the type of targeted resources — people and places. In particular, the transformation localizes places on a map.

As it can be viewed in Figure 2, when following the hyperlink generated by the DBpedia navigator, a request is sent to the STTL server to produce an HTML presentation of the DBpedia resource on Augustus, according to the `st:dbpedia` profile (embeddding the `st:navlab` transformation). The interest of this STTL-based approach of DBpedia-to-HTML transformation is that it is declarative and can therefore easily be extended to handle the presentation of other types or resources by adding new dedicated templates.

History Timeline Navigator. We developed a third demonstrator of our service to browse an RDF graph combining a local RDF dataset about history linked with DBpedia. The local dataset contains a set of historical events and personalities with dates. The URIs of the resources are those of DBpedia (fr.dbpedia.org) in order to link data. Resource descriptions are stored in named graphs which are tagged with topics such as “France” or “Empire”. The `st:cdn` transformation generates an HTML page for each century, where resources are displayed in columns according to the topic of their named graph and in ascending order of dates. Hyperlinks to DBpedia resources are generated which are processed with

the former `st:navlab` transformation. Figure 3 is a screenshot of an HTML page generated by the server.

Fig. 3. History Navigator

6 RDF-Syntax to Another-Syntax Transformers

The fourth family of RDF transformations identified in the introduction of this paper gathers the transformations of statements in languages with an RDF syntax into another syntax of the language. In that sense the transformations of this family are completely domain-independent. Here we give two examples of such transformations: the transformation of OWL statements from RDF syntax into OWL Functional syntax and the transformation of SPARQL statements from SPIN/RDF syntax into SPARQL concrete syntax.

6.1 OWL/RDF to OWL/Functional Syntax Transformer

In this section we present the `st:owl` transformation of OWL 2 statements from the OWL/RDF syntax into the OWL 2 Functional Syntax which belongs to this family. The transformation follows the W3C Recommentation *OWL 2 Web Ontology Language Mapping to RDF Graphs*[24]. We wrote it as a set of 73 STTL templates, structured into 5 subsets, available online[25].

[24] http://www.w3.org/TR/owl2-mapping-to-rdf
[25] http://ns.inria.fr/sparql-template/owl

Among them, here is one template enabling to transform OWL/RDF statements presented in section 2.1 in OWL 2 Functional syntax. The template handles the transformation of equivalent classes (and datatype definitions) axiom, possibly involving an intersection or union class expression. For this case, the STTL engine is recursively called on variables `?in` and `?y`.

```
TEMPLATE {
  IF (bound(?t), "DatatypeDefinition", "EquivalentClasses")"("
  st:call-template(st:annotate, ?in, owl:equivalentClass, ?y)
  ?in ?y ")" }
WHERE {
  ?in owl:equivalentClass  ?y
  OPTIONAL { ?y a ?t FILTER(?t = rdfs:Datatype) }}
```

A demo OWL transformation service is available online[26].

We validated the `st:owl` transformation on the OWL 2 Primer ontology[27] containing 350 RDF triples: we first transformed this set of triples into OWL Functional Syntax with the `st:owl` transformation, then we loaded the output into Protégé and saved it in RDF/XML, then we transformed it again with `st:owl` and we checked that the two transformation outputs were equivalent. Let us note that the results are equivalent but not identical because some statements are not printed in the same order, due to the fact that Protégé does not save RDF/XML statements exactly in the same order and hence blank nodes are not allocated in the same order.

We tested this OWL/RDF transformer on several real world ontologies, among which a subset of the *Galen* ontology. The RDF graph representing it contains 33,080 triples, the size of the result is 0.58 MB and the (average) transformation time is 1.75 seconds. We also have tested our pretty-printer on the *HAO* ontology. The RDF graph representing it contains 38,842 triples, the size of the result is 1.63 MB, the (average) transformation time is 3.1 seconds.

In addition to the transformation of an RDF dataset representing OWL statements, the `st:owl` transformation can also be used when querying an OWL ontology stored in its RDF syntax, to present the results to the user in OWL 2 Functional syntax. This is done by calling in the SELECT clause of the query the STTL extension functions launching the transformer. As an example, the following query retrieves specific classes and displays them in Functional syntax:

```
SELECT (st:apply-templates-with(st:owl, ?x) as ?t)
WHERE { ?x a owl:Class ; rdfs:subClassOf* f:Human }
```

6.2 SPIN/RDF to SPARQL Concrete Syntax Transformer

In this section we present the `st:spin` transformation of SPARQL queries in SPIN RDF syntax[12], into SPARQL concrete syntax. This transformation of SPARQL statements, belongs to the same family as the transformation of OWL

[26] http://corese.inria.fr
[27] http://www.w3.org/TR/owl2-primer

statements: the fourth one identified in the introduction of this paper gathering the transformations of statements in languages with an RDF syntax into another syntax. We wrote the `st:spin` transformation as a set of 64 STTL templates which are available online[28]. Among them, the following template translates a union of alternative graph patterns in SPIN into SPARQL concrete syntax:

```
prefix sp: <http://spinrdf.org/sp#> .
TEMPLATE { ?e1 st:nl() "union" st:nl() ?e2 }
WHERE { ?in a sp:Union ; sp:elements (?e1 ?e2)}
```

We validated this transformation on SPARQL 1.1 W3C test suite: we translated each SPARQL query into SPIN by using the Corese library, then we translated it back into SPARQL using the `st:spin` transformation, and finally we executed it with the Corese SPARQL engine.

7 Conclusion and Future Work

In this paper we recalled the STTL principles, we presented the STTL engine and Web service and we demonstrated the support this software provides to programmers by presenting a selection of STTL-based RDF transformers for common languages: RDT-to-Turtle, RDF-to-HTML, OWL/RDF-to-OWL/FunctionalSyntax and SPIN-to-SPARQL/ConcreteSyntax transformations. The source code of this software is freely available within the Corese Semantic Web Factory; it is also available online as a Web service, and all the RDF transformations presented in this paper can be tested online[29].

As future work, regarding the performance of our generic transformation rule engine, we intend to improve it by implementing heuristics to optimize the selection of templates. We will also compare in the short term the performance of our generic transformation rule engine with that of existing tools for specific RDF transformations. For instance, we may compare the performance of our engine with that of the parser of the well known OWL API[30] for transforming large OWL 2 ontologies from RDF/XML syntax into Functional syntax.

Regarding the exploitation of our generic transformation rule engine to implement RDF transformers into specific languages, we intend to augment the number of STTL transformations available by writing STTL template sets for other formats and domains. In particular, we intend to define an RDF-to-CSV transformation and an RDF-to-JSON transformation. Finally, we will consider a sixth family of RDF transformations gathering RDF-to-RDF transformations for special purposes, e.g., to anonymize RDF datasets.

Acknowledgements We thank Fuqi Song (Inria), Alban Gaignard (CNRS) and Eric Toguem (U. of Yaoundé, Cameroun) for the setup of the HTTP server.

[28] http://ns.inria.fr/sparql-template/spin
[29] http://corese.inria.fr
[30] http://owlapi.sourceforge.net/

References

1. Faisal Alkhateeb and Sébastien Laborie. Towards Extending and Using SPARQL for Modular Document Generation. In *Proc. of the 8th ACM Symposium on Document Engineering*. ACM Press, 2008.
2. Stefan Bischof, Stefan Decker, Thomas Krennwallner, Nuno Lopes, and Axel Polleres. Mapping between RDF and XML with XSPARQL. *J. Data Semantics*, 1(3), 2012.
3. Christian Bizer, Ryan Lee, and Emmanuel Pietriga. Fresnel - A Browser-Independent Presentation Vocabulary for RDF. In *Second International Workshop on Interaction Design and the Semantic Web*, Galway, Ireland, 2005.
4. Matt Brophy and Jeff Heflin. OWL-PL: A Presentation Language for Displaying Semantic Data on the Web. Technical report, Lehigh University, 2009.
5. Olivier Corby and Catherine Faron-Zucker. The KGRAM Abstract Machine for Knowledge Graph Querying. In *Proc. of IEEE/WIC/ACM International Conference on Web Intelligence*, Toronto, Canada, 2010.
6. Olivier Corby and Catherine Faron-Zucker. STTL: A SPARQL-based Transformation Language for RDF. In *Proc. of the 11th International Conference on Web Information Systems and Technologies (WEBIST)*, Lisbon, Portugal, May 2015.
7. Olivier Corby, Alban Gaignard, Catherine Faron-Zucker, and Johan Montagnat. KGRAM Versatile Data Graphs Querying and Inference Engine. In *Proc. IEEE/WIC/ACM International Conference on Web Intelligence*, Macau, 2012.
8. Richard Cyganiak, David Wood, and Markus Lanthaler. RDF 1.1 Concepts and Abstract Syntax. Recommendation, W3C, 2014.
9. Corentin Follenfant, Olivier Corby, Fabien Gandon, and David Trastour. RDF Modelling and SPARQL Processing of SQL Abstract Syntax Trees. In *Programming the Semantic Web, ISWC Workshop*, Boston, USA, 2012.
10. Sandro Hawke and Axel Polleres. RIF In RDF. Working Group Note, W3C, 2012.
11. Matthew Horridge and Sean Bechhofer. The OWL API: A java API for OWL ontologies. *Semantic Web*, 2(1), 2011.
12. Holger Knublauch. SPIN - SPARQL Syntax. Member Submission, W3C, 2011.
13. Peter F. Patel-Schneider and Boris Motik. OWL 2 Web Ontology Language Mapping to RDF Graphs (Second Edition). Recommendation, W3C, 2012.
14. Silvio Peroni and Fabio Vitali. RSLT: RDF Stylesheet Language Transformations. In *Proc. of 12th ESWC Developers Workshop*, Portoroz, Slovenia, June 2015.
15. Jonathan Robie, Don Chamberlin, Michael Dyck, and John Snelson. XQuery 3.0: An XML Query Language. Recommendation, W3C, 2014.
16. Pavel Shapkin and Leonid Shumsky. A Language for Transforming the RDF Data on the Basis of Ontologies. In *Proc. of the 11th International Conference on Web Information Systems and Technologies (WEBIST)*, Lisbon, Portugal, May 2015.
17. Alex Stolz, Bene Rodriguez-Castro, and Martin Hepp. RDF Translator: A RESTful Multi-Format Data Converter for the Semantic Web. Technical report, U. der Bundeswehr, Munich, 2013.