RSLT:
RDF Stylesheet Language Transformations

Silvio Peroni and Fabio Vitali

Department of Computer Science and Engineering, University of Bologna (Italy)
silvio.peroni@unibo.it, fabio@cs.unibo.it

Abstract. In this paper we introduce RSLT, a simple transformation language for RDF data. RSLT organizes the rendering of RDF statements as transformation templates associated to properties or resource types and producing HTML. A prototype based on AngularJs is presented and we also discuss some implementation details and examples.

Keywords: RDF dataset visualization, RSLT, templates

1 Introduction

The visualization of RDF data is a hot topic for the Semantic Web community. Even if works have been presented in the past (e.g., [2, 3, 5–7]), and some visualisation interfaces (usually in tabular form) are supported by existing triplestores such as Virtuoso[1], visualisation tools for RDF triplestores in general are still far from the sophistication of reporting tools for traditional databases. In particular, most of such tools display triplestore content as tabular data where identificative, descriptive and secondary properties are shown in exactly the same way, where the order in which properties are displayed is totally arbitrary, where related entities are described by their plain IRI rather than textual descriptions, and where readability is completely absent.

This drove us to investigate the best tools to provide readable representations of RDF datasets for the general public, and to propose basic guidelines:

- the tool should be easy to integrate in a web-based application;
- it should be as easy to represent individual RDF statements as convoluted OWL entities;
- IRIs of entities should be rendered with readable and meaningful text or other representations – but they should still be available for special rendering needs (e.g., as destinations for links and HTML anchors);
- complex representations of main resources should be built by combining simpler representations of lesser resources and properties.

Reporting software tools have limitations, such as the complexity to integrate them in web-based architectures or the variety and complexity of approaches in generating and delivering the reports. Yet, we found XSLT [4] a fairly natural and sophisticated

[1] http://virtuoso.openlinksw.com/

approach to provide presentation support to XML documents, and sought to provide something similar for OWL and RDF data.

In this paper we introduce *RSLT* (*RDF Stylesheet Language Transformations,* pronounced *result*), a simple transformation language for RDF data. RSLT organizes rendering as transformation templates associated to resources, properties or resource types, produces HTML and can recursively call other templates. A browser-based prototype based on the *AngularJs* library has been created that allows client-based presentations of SPARQL constructs, and soon of Turtle datasets as well. The rest of the paper is organized as follows: in Section 2 we introduce the main constructs of RSLT. In Section 3 we introduce some implementation details of RSLT and few examples, and in Section 4 we conclude the paper by sketching out some future works.

2 RSLT

RDF Stylesheet Language Transformations (*RSLT*)[2], pronounced *result*, is a direct and trivial translation of (some parts of) the XSLT language into the RDF domain. Similarly to its noble ancestor, an RSLT document contains a number of templates that create fragments of output in some displayable format (e.g., HTML) when navigating through a graph of RDF statements. The fact that RDF graphs, differently than XML documents, lack a natural hierarchy in its internal organization. and that no obvious selector language for RDF exists[3] provide for some interesting complications of the original design, though.

Thus, while XSLT always starts transforming the root of an XML document, no such concept exists for RDF graphs, which therefore require a starting template such as the following one:

```
<template match='$start'>
  <div class="container">
    <applyTemplates select="??person foaf:familyName 'Horrocks'.">
    </applyTemplates>
  </div>
</template>
```

The above template will be fired at the beginning of the process and will create an HTML `div` element, and will first select all entities whose `foaf:familyNa me` is "Horrocks", and then look for a reasonable template for each of them.

The lack of a correspondence for XPath forced us to invent a new syntax for `<applyTemplates>` selectors, liberally derived from the SPARQL query syntax, that allow templates to distinguish statements from entities. Thus, the selector "`?person foaf:familyName 'Horrocks'.`" with a *single question mark* (*SQM*) in the variable selects all *RDF statements* whose predicate is `foaf:familyName` and whose object

[2] RSLT is available on its GitHub repository: `https://github.com/fvitali/rslt`. The tool as well as all the additional scripts are distributed under an ISC License, while the other related files such as HTML documents are distributed under a Creative Commons Attribution 4.0 International License.

[3] SPARQL cannot be considered as a pure selector language for RDF as XPath, used by XSLT, is. Rather, it is a full-featured query language, similar to XQuery for XML.

equals “Horrocks”. On the other hand, the selector “`??person foaf:familyName 'Horrocks'.`” with *double question marks* (*DQM*) selects a list of *RSLT entities*, where each RSLT entity is defined as the set of available statements that share the same subject. The DQM selector above can be converted as follows in SPARQL 1.1:

```
SELECT DISTINCT ?s ?p ?o WHERE {
  ?person foaf:familyName "Horrocks".
  { BIND(?person as ?s) ?s ?p ?o. }
}
```

RSLT templates can be either associated to RDF statements (by matching a triple with one or more unbound SQM variables), to a specific resource (by matching the IRI of the resource) or to resources of a particular type (by using a special syntax), as described in the following excerpts:

```
<!-- RSLT templates for RDF statements -->
<template match='?person foaf:familyName "Horrocks"'>...</template>
<template match='?person foaf:familyName ?string'>...</template>
<!-- RSLT templates for a particular resource -->
<template
  match='http://www.semanticlancet.eu/resource/person/ian-horrocks'>
  ...
</template>
<!-- RSLT templates for resources of a particular type -->
<template match="?person -> foaf:Person"> ... </template>
```

Within a template all unbound variables are bounded to the relevant RLST entities and all associated properties are available for presentation. Any markup or content can be placed inside templates, and RSLT constructs can be specified to refer to literals or other resources associated to the bounded entities. Values in literal statements can be rendered immediately through `<valueOf>` elements, while values of resource statements are additional RSLT entities, and therefore are rendered through additional templates. For instance, the following is a complete template of a class:

```
<template match="?person -> foaf:Person">
  <p>Found <valueOf select="count(?person pro:holdsRoleInTime ?r)">
    </valueOf> papers authored by
    <valueOf select="?person foaf:givenName ?g"></valueOf>
    <valueOf select="?person foaf:familyName ?f"></valueOf>:</p>
  <ul>
  <applyTemplates select='?person pro:holdsRoleInTime /
                          pro:relatesToDocument ??work'>
  </applyTemplates></ul>
</template>
```

Whenever the rendering engine comes across an entity of type `foaf:Person`, this template is triggered, creating an HTML fragment with a `<p>` element containing text and the values of data properties `foaf:givenName` and `foaf:family Name`, then an `<ul>` element containing the rendering recursively produced by selecting all entities

`??work` related to `?person` through the chain of properties `pro:holdsRoleInTime / pro:relatedToDocument`. This rendering is generated by looking for and executing the templates that match the `??work` entities just selected. In order to avoid circularity in the selection of templates, a simple approach is taken to consider the template as *spent* when applied in a recursion, so it cannot be chosen and applied again inside itself. Once all relevant templates have been spent, if a template for a specific entity is needed again, a default one is applied that does not recurse, thus halting any potential circularity.

Given the strong dependance of the current implementation of RSLT to the AngularJS framework, a shorter syntax exists that uses the classical double brackets of the framework. Additionally, since all entities are in fact Javascript objects and the colon ":" is a forbidden character in Javascript variables, Java-script's dot notation is used and the underscore is used instead:

```
<template match="?person -> foaf:Person">
  <p>Found {{ count(person.pro_holdsRoleInTime) }} papers authored
    by {{person.foaf_givenName}} {{person.foaf_familyName}}:</p>
  <ul>
    <applyTemplates select='?person pro_holdsRoleInTime /
                            pro_relatesToDocument ??work'>
    </applyTemplates>
  </ul>
</template>
```

In addition to `<template>`, `<applyTemplates>` and `<valueOf>`, at the moment the RSLT language inherits from XSLT also constructs such as `<callTempl ate>` and `<forEach>`, that have similar behaviour as their XSLT counterparts. Also, the template element inherits the `mode` and `priority` attributes, with similar functions to XSLT. Finally, the `<rslt>` element allows the specification of a triple store through the `triplestore` attribute, and element `<prefix>` allows the specification of prefixes for selectors and property names.

When executing `<applyTemplates>` and `<forEach>` instructions, RSLT verifies whether the requested entities are already locally available, and if not it proceeds to execute another SPARQL query to the triplestore. However, in order to avoid to run a large number of queries every time RSLT templates will match, RSLT implements a mechanism to preload (through the attribute `preload`) as many entities as needed, so that no further queries (or many less queries) need to be executed. In the following listing we show how this works, loading additional entities specified by the DQM variables in the `preload` attribute:

```
<template match='$start'>
  <applyTemplates select="??person foaf:familyName 'Horrocks'."
                  preload="?person pro:holdsRoleInTime ??role.
                           ?role pro:relatesToDocument ??work.">
  </applyTemplates>
</template>
```

In conclusion, RSLT is rather similar to its ancestor, XSLT, but for a few key differences, such as the specification of a different selector language, and the efficiency requirement that led us to include the *preload* attribute. The selector language is in itself just a subset of the SPARQL language, similarly to how the selector language XPath is but a subset of the query language XQuery, and for similar reasons. Would it be possible to just use an existing implementation of XSLT? Unfortunately, the requirements for non-circularity in template matching and the effort to adapt a new selector language to existing code probably makes this endeavour excessive and not worthy. We have rather reimplemented a few key commands of the language relying on a few peculiarities of AngularJs directives, as explained in the next section.

3 An implementation of RSLT

A working implementation of the RSLT engine has been developed to run as an AngularJS module. In its simplest incarnation, a simple specification of the libraries and one line of Javascript is enough to have a full working instance of RSLT in the browser, as shown in the following listing:

```
<html ng-app="simplestRSLT">
  <head>
    <script src="angular.js"> </script>
    <script src="rslt.js"> </script>
    <script>
      var app = angular.module('simplestRSLT', ['rslt']);
    </script>
  </head>
  <body>
    <rslt> ... </rslt>
  </body>
</html>
```

The `ng-app` attribute of element `<html>` is required by AngularJs to create an Angular application (named '`simplestRSLT`'), which corresponds to an empty AngularJs module that simply includes and uses the RSLT library. Anywhere in the body of the HTML document the `<rslt>` element creates the rendering of the entities selected in its `$start` template.

Each element of the RSLT language is defined as an AngularJs directive, which allows HTML to be extended with new markup and new element names. AngularJs directives create nested contexts where only selected objects are accessible. This is a very easy mechanism for managing context entities in templates: namely, each template can only access the entities that correspond to variables in the match attribute, thereby ensuring clean and controlled processing of entities and properties. Each variable name is in fact bound bidirectionally to a Javascript object that contains all triples of the corresponding entity, with the additional precaution of converting into underscores all colons separating the prefix from the actual name of the property. For instance, `foaf:familyName` of the DQM variable `??person` is always available as `person.foaf_familyName` within all directives/factories/filters in the current scope.

A little note on the RSLT syntax used in the current implementation: in AngularJs a normalization takes place for all directive names, where colons, dashes and underscores are removed and converted into camelCase. This means that `<apply-templates>`, `<apply:templates>` and `<apply_templates>` are equivalent to `<applyTemplates>`. The standard Angular recommendation is to "use name-with-dashes for attribute names and camelCase for the corresponding directive name", so in all our examples we are uing `<applyTemplates>` instead of `<apply-templates>` as XSLT introduced. The same goes for `<forEach>` and `<valueOf>` instead of `<for-each>` and `<value-of>`. They are all equivalent.

At the address `http://www.fabiovitali.it/rslt/` you can find four working examples of RSLT, all using as data source the Semantic Lancet Triplestore [1]. The first and the second examples are identical and show the simplest RSLT document accessing and rendering entities and properties across seven different classes – the only difference being in the rendering speed (which in example 1 is lacks any preload instruction in the queries).

The third example is a simple and straightforward reimplementation of the Lancet Data Browser (`http://www.semanticlancet.eu/browser`) already presented in [1], whose source code becomes incredibly simpler and more straightforward through the use of the RSLT library. Finally, example 4 (see Fig. 1) is a tool for the RSLT programmer that supports the interactive specifications of triplestores, start-with RSLT entities, pre-loaded RSLT entities, and templates (both in XML as described in this paper as well as in an experimental JSON format not yet documented) as well as checks the output of the RSLT transformation, the actual SPARQL queries sent to the triplestore and the list of entities that have been downloaded client-side by the full sequence of SPARQL queries generated through the execution of the RSLT templates.

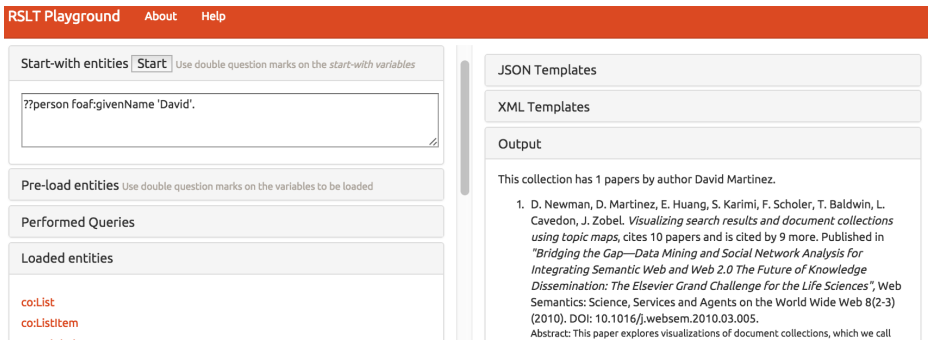

Fig. 1. The RSLT playground with a working presentation.

4 Conclusions

In this paper we have introduced *RSLT*, i.e., a simple transformation language for RDF data. RSLT organizes rendering of RDF statements in a triplestore through transformation templates recursively producing HTML. A prototype based on the *AngularJs* library has been presented and we have discussed implementation details and examples. In the future, there are ongoing plans to integrate support for the management of complete collections of RDF statements under the form of Turtle files, to extend the language to supporting more features of the XSLT language and of the query language. Finally, our plan is to study the requirements for providing support of the language on the server as well, by verifying how an architecture such as Node.js could be used to this end.

References

1. Bagnacani, A., Ciancarini, P., Di Iorio, A., Nuzzolese, A. G., Peroni, S., Vitali, F. (2015). The Semantic Lancet Project: a Linked Open Dataset for Scholarly Publishing. In Proc. of EKAW 2014 Satellite Events. DOI: 10.1007/978-3-319-17966-7_10
2. Bischof, S., Decker, S., Krennwallner, T., Lopes, N., Polleres, A. (2012). Mapping between RDF and XML with XSPARQL. Journal on Data Semantics, 1(3): 147–185.
3. Corby, O., Faron-Zucker, C., Gandon, F. (2014). SPARQL Template: A Transformation Language for RDF. `https://hal.inria.fr/hal-00969068v1`
4. Kay, M. (2007). XSL Transformations (XSLT) Version 2.0. W3C Recommendation, 23 January 2007. World Wide Web Consortium. `http://www.w3.org/TR/xslt20/`
5. Luggen, M., Gschwend, A., Bernhard, A., Cudré-Mauroux, P. (2015). Uduvudu: a Graph-Aware and Adaptive UI Engine for Linked Data. In Proc. of LDOW 2015.
6. Pietriga, E., Bizer, C., Karger, D., Lee, R. (2006). Fresnel: A Browser-Independent Presentation Vocabulary for RDF. In Proc. of ISWC 2006. DOI: 10.1007/11926078_12
7. Skjæveland, M. G. (2012). Sgvizler: A JavaScript Wrapper for Easy Visualization of SPARQL Result Sets. In Proc. of the Workshops and Demo tracks of ESWC 2012.