Tal4Rdf: lightweight presentation for the Semantic Web

Pierre-Antoine Champin

LIRIS, Université de Lyon, CNRS, UMR5205,
Université Claude Bernard Lyon 1, F-69622, Villeurbanne, France
pchampin@liris.cnrs.fr

Abstract. As RDF data becomes increasingly available on the Web, there is a need to render this data in different formats, aimed at end-users or applications. We propose Tal4Rdf, a template based language, implemented as an open-source project and an online demo. Tal4Rdf uses intuitive path-based expressions for querying RDF data, and allows to easily generate any XML or textual output format. We believe it has the potential to become a "scripting language for presentation".

1 Introduction

More and more RDF data has become available in the recent years, thanks to different efforts to export linked data from legacy databases [1] or existing content [2, 3], tag published content with machine-readable metadata [4, 5], or ease the collaborative and flexible authoring of RDF data [6, 7]. Although many of these efforts focus on content primarily intended for human consumption, we argue that this does not reduces the need for flexible and versatile presentation tools for RDF data. Indeed, the main point of all those efforts is to make the available information *machine-processable*, and the results of the involved processes also has to be presented to an end-used. Adaptation or aggregation of are immediate scenarios where RDF data, even when extracted from presentation-ready documents, needs to be re-presented in a different way to the user. Another scenario is the formatting to other machine-processable formats, like application-specific XML or JSON. Indeed, adding RDF support in existing applications is not always feasible (closed proprietary applications) or practical (lightweight embedded applications)[1].

In this paper, we present Tal4Rdf (T4R), a lightweight template language for rendering RDF data in any XML or textual format, for which an open-source implementation and an interactive demo are available at `http://champin.net/t4r/`. It is based on TAL (Template Attribute Language), an existing template language that we have already reused successfully in the Advene framework [8], which makes us confident in the potential of T4R as a "scripting language for presentation".

[1] See also `http://n2.talis.com/wiki/RDF_JSON_Specification` for a proposal to represent RDF in JSON, with the same rationale.

In the first section we will present the TAL language. Section 3 will present the rationale and basic features of T4R, focusing on the notion of path to retrieve RDF data. In the next section, we will discuss features of T4R that are more related to the rendering process. Section 5 compares T4R to related works, and the last section concludes and gives some further research directions.

2 TAL

The Template Attribute Language or TAL [9] has been introduced in the Zope web development framework[2] for presenting data in HTML or any XML format. It is a *template* language: the document specifying the rendering of the underlying data is a mock-up of the expected result. TAL puts an emphasis on preserving the integrity of the template with regard to the target format.

This is achieved by encoding the processing instructions in XML attributes with a specific namespace (usually associated to the prefix `tal:`). Standard editors for the target format can then be used to modify the presentation of the template without altering (or being altered by) the processing instructions. Furthermore, only minor changes are required in such editors to provide ad-hoc management of the TAL attributes, this functionality being orthogonal to the other features of the format. This approach has been applied in the Advene project and could also be applied to T4R.

We present in this section a subset of the processing instructions of TAL, in order to give an overview of its capabilities. Simple modifications are performed by TAL attributes `tal:content` and `tal:attributes`, respectively replacing the content and attributes of the XML element by the data specified as their value. Conditional and iterative structures are provided by `tal:condition` (suppressing the XML element if the condition specified as its value is not met), and `tal:repeat` (repeating the XML element for each data matching its value). Finally, `tal:define` can be used to store data in a variable usable in further TAL attributes. A comprehensive example will be given in the next section, in Figure 2.

All TAL attributes use a common syntax for accessing the underlying data: TALES (TAL Expression Syntax). TALES expressions are, in most cases, slash-separated paths. The exact meaning of those paths depends on the underlying data structure, but their similarity to file or URL paths makes them pretty intuitive. Hence the idea of using TAL to query and render RDF data, as will be demonstrated in the following sections.

TAL also has the advantage of being implemented in several popular scripting languages [9]. Hence T4R could easily be ported to those languages (the current implementation is in Python, and uses SimpleTAL[3]).

[2] `http://zope.org/`
[3] `http://www.owlfish.com/software/simpleTAL/`

3 A path to query RDF

The rationale of using TAL for RDF rendering was that TALES paths could easily be mapped to paths in the underlying RDF graph, hence provide an intuitive way of querying RDF data. For example, using the FOAF vocabulary [10], a path retrieving the homepages of the projects currently worked on by the people I know could be represented by the path:

```
knows/currentProject/homepage
```

meaning that, starting from the resource representing myself, T4R would need to traverse in sequence three arcs labelled with `knows`, `currentProject` and `homepage` respectively.

3.1 Namespaces

The motivating example above is a bit over-simplistic. In RDF, arcs (and resources) are not labelled by plain terms, but by URIs, in order to avoid name clashes. We need a way of concisely representing URIs as path elements. This problem is well known and a common solution is to use CURIEs [11]. A CURIE is composed of a namespace prefix and a suffix, separated by a colon. The namespace prefix is associated with a namespace URI, and the CURIE is simply interpreted as the concatenation of the namespace with the suffix. For example, if the namespace URI `http://xmlns.com/foaf/0.1/` was assigned to the prefix `foaf`, then the CURIE `foaf:knows` would correspond to URI `http://xmlns.com/foaf/0.1/knows`.

In T4R, namespaces prefix and URIs are associated by defining special variables (using `tal:define`) of the form `t4rns:prefix`, in a way very similar to XML. Note that it is recommended by [11] that CURIE prefixes should use XML namespaces whenever available. There are several reasons why this is not done in T4R. First, T4R aims at rendering non-XML formats, so we could not rely on XML namespaces in all cases. Second, in XML templates, the namespaces used for querying the graph are rarely the same as the ones used in the output format, so keeping them separate seems to be a better practice. The final reason, though not sufficient in itself, is nevertheless very pragmatic: not all TAL implementations give access to the XML namespace declarations of the template.

In the following, we will assume that the appropriate namespaces have been declared, with their usual prefix (`t4rns:rdf` for the RDF namespace, `t4rns:foaf` for the FOAF vocabulary, etc.).

3.2 Simple Path

Using CURIEs, our intuitive example above, to retrieve the homepages of the current projects of the people I know, becomes:

```
foaf:knows/foaf:currentProject/foaf:homepage
```

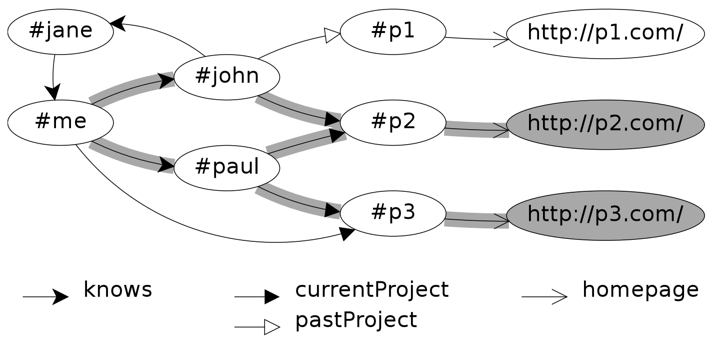


Fig. 1. An example data graph and how the path `foaf:knows/foaf:currentProject/foaf:homepage` applied to `#me` evaluates.

hardy more complicated that our initial proposal. The evaluation of this path on an example dataset is illustrated in Figure 1.

The first interesting thing to notice is that each node may have several values for the same property, hence such a path expression evaluates to a *collection* of RDF nodes, which can be iterated with the `tal:repeat` construct. It is also possible to keep only a single node by appending the T4R keyword `any` to this path. However, this keyword is optional when the result of the path is rendered as an element content of attribute. Hence if the path above was used as is to fill the `href` attribute of a link, it would render as a *one* of the result URIs (the other would then be ignored), keeping the link working[4].

Another thing worth pointing out is that, since RDF data has a graph structure, the path may discover the same node several times (cf. Figure 1). However, in T4R, each node matching the path will appear only once, no matter how many times it was reached through the path.

3.3 More complex paths

Sometimes, we are interested in the *inverse* of a given properties. This is possible by appending `:-` to a CURIE. Hence, the path:

```
foaf:activeProject/foaf:activeProject:-
```

will retrieve the people working on the same projects as myself (and yield only `#paul` in the example of Figure 1).

Another frequent problem when querying RDF data in the open is that some properties from different vocabularies have a similar meaning (it even happens sometimes in the same vocabulary). Since all variants are likely to be used in

[4] This tolerant behaviour, convenient for rapid prototyping of templates, can nevertheless be changed to a stricter one, which is preferable for debugging complex templates.

the data, queries have to take all of them into account. A similar problem occurs when two properties are defined to be inverse of each other, and can therefore be used indifferently (only by changing the orientation of the arc). Managing this variability in common query languages, like SPARQL [12], can be pretty cumbersome. In T4R, the keyword `or` can be used to elegantly solve that problem:

```
foaf:img/or/foaf:depiction/or/foaf:depicts:-
```

will retrieve all the images representing myself (according to the FOAF vocabulary).

One may argue that this problem can (or even should) be solved by inference rather than the presentation layer; indeed, an inference engine will recognize that `foaf:depicts` and `foaf:img` are, respectively, the inverse and a particular case of `foaf:depiction`. Should T4R be backed by such an inference engine (which is a possible use of T4R), the simple path `foaf:depiction` would be equivalent to the more complex path above. However, in practice, inference capabilities are not always available nor easy to add (if for example the data is accessed through a SPARQL endpoint without inference). Aiming to be usable as a lightweight solution, T4R must provide means to cope as well as possible with the absence of inference. The T4R keyword `or` is such a mean.

We have presented until now two T4R keywords (`any` and `or`) that can be used in paths. T4R provides several other keywords useful for presentation (e.g. rendering a URI node as a CURIE), filtering (e.g. keeping only literal nodes from a node list, or literals with a given language) or testing (e.g. if a node is a URI node, if two nodes are in the same namespace). The interested reader may find an exhaustive list at `http://champin.net/t4r/doc/reference`.

3.4 Relative and absolute paths

In all the examples given above, the path was evaluated relatively to an implicit resource (the resource being described by the template). Each CURIE in the path, including the first one, is interpreted as a property.

A path can also be evaluated relatively to the resource(s) stored in a variable (usually resulting from the previous evaluation of another path). In this case, the first item of the path is not a CURIE, but a variable name, for example

```
v/foaf:currentProject
```

Since variable names in T4R can not contain a colon, there is no possible ambiguity with a CURIE[5].

A third kind of paths are absolute paths. Those paths start with a slash, just like file or URL absolute paths. The first CURIE of those paths is not interpreted as a property, but as a resource. For example:

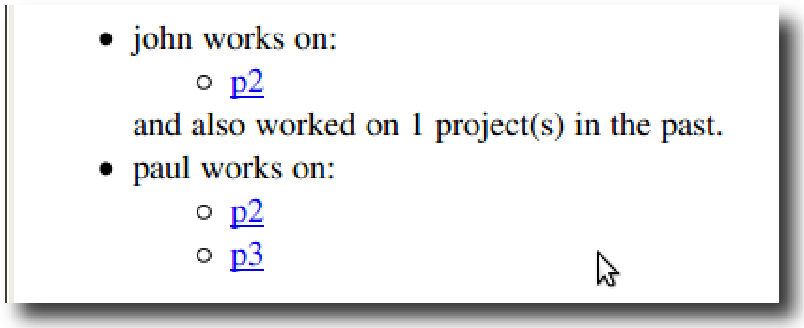
[5] There is no possible ambiguity with a T4R keyword either, because keywords may not appear as the first item of a path, while variables may only appear in first position.

```
/foaf:Person/rdf:type:-
```

will retrieve all the instances of `foaf:Person`.

A simple example of the TAL language and the use of CURIE paths is given in Figure 2. Further example from the online demo are illustrated in Figure 3.

```
<ul tal:define="global t4rns:foaf string:http://xmlns.com/foaf/0.1/">
  <li tal:repeat="pe foaf:knows">
    <span tal:content="pe/id">someone I know</span> works on:
    <ul>
      <li tal:repeat="pr pe/foaf:currentProject">
        <a tal:attributes="href pr/foaf:homepage"
           tal:content="pr/id">a project</a></li>
    </ul>
    <span tal:define="past pe/foaf:pastProject" tal:condition="past">
      and also worked on <span tal:content="past/count">n</span>
      project(s) in the past.
    </span>
  </li>
</ul>
```



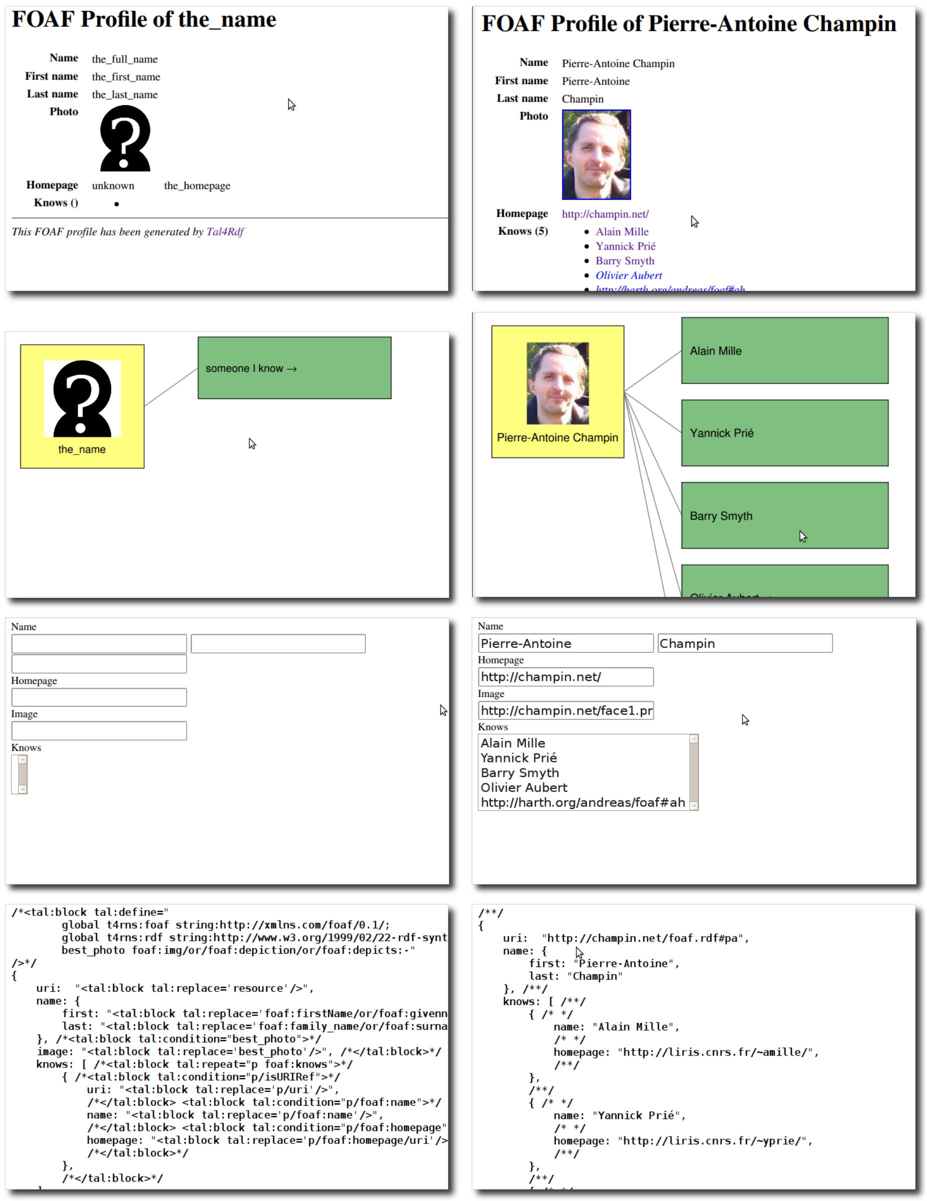
- john works on:
  - p2

  and also worked on 1 project(s) in the past.
- paul works on:
  - p2
  - p3

Fig. 2. A template and its result when applied to the resource `#me` in the graph from Figure 1.

4 T4R templates and data sources

As any typical rendering engine, T4R combines a presentation specification (the template) with a data source (the RDF graph) into an output document. In this section, we will discuss noteworthy features of those three parts.



FOAF Profile of the_name

Name	the_full_name
First name	the_first_name
Last name	the_last_name
Photo	
Homepage	unknown the_homepage
Knows ()	•

This FOAF profile has been generated by Tal4Rdf

FOAF Profile of Pierre-Antoine Champin

Name	Pierre-Antoine Champin
First name	Pierre-Antoine
Last name	Champin
Photo	
Homepage	http://champin.net/
Knows (5)	• Alain Mille • Yannick Prié • Barry Smyth • *Olivier Aubert* • *http://harth.org/andreas/foaf#ah*

Name

Homepage

Image

Knows

Name
Pierre-Antoine Champin
Homepage
http://champin.net/
Image
http://champin.net/face1.pr
Knows
Alain Mille
Yannick Prié
Barry Smyth
Olivier Aubert
http://harth.org/andreas/foaf#ah

```
/*<tal:block tal:define="
        global t4rns:foaf string:http://xmlns.com/foaf/0.1/;
        global t4rns:rdf string:http://www.w3.org/1999/02/22-rdf-synt
        best_photo foaf:img/or/foaf:depiction/or/foaf:depicts:-"
/>*/
{
    uri:  "<tal:block tal:replace='resource'/>",
    name: {
        first: "<tal:block tal:replace='foaf:firstName/or/foaf:givenn
        last: "<tal:block tal:replace='foaf:family_name/or/foaf:surna
    }, /*<tal:block tal:condition="best_photo">*/
    image: "<tal:block tal:replace='best_photo'/>", /*</tal:block>*/
    knows: [ /*<tal:block tal:repeat="p foaf:knows">*/
        { /*<tal:block tal:condition="p/isURIRef">*/
            uri: "<tal:block tal:replace='p/uri'/>",
            /*</tal:block> <tal:block tal:condition="p/foaf:name">*/
            name: "<tal:block tal:replace='p/foaf:name'/>",
            /*</tal:block> <tal:block tal:condition="p/foaf:homepage"
            homepage: "<tal:block tal:replace='p/foaf:homepage/uri'/>
            /*</tal:block>*/
        },
        /*</tal:block>*/
```

```
/**/
{
    uri:  "http://champin.net/foaf.rdf#pa",
    name: {
        first: "Pierre-Antoine",
        last: "Champin"
    }, /**/
    knows: [ /**/
        { /* */
            name: "Alain Mille",
            /* */
            homepage: "http://liris.cnrs.fr/~amille/",
            /**/
        },
        /**/
        { /* */
            name: "Yannick Prié",
            /* */
            homepage: "http://liris.cnrs.fr/~yprie/",
            /**/
        },
        /**/
```

Fig. 3. Four examples from the online demo. The same data is rendered (right column) using HTML, SVG, HTML forms and JSON. The left column shows the unprocessed templates, which are valid documents in their respective formats.

4.1 RDF data sources

The data source of the T4R rendering engine is given as the URI of the resource to be rendered with the template. This URI is actually used for two distinct purposes:

– identify the resource used to resolve relative paths, and
– locate the RDF data.

Since it is not always the case that the URI of a resource gives access to the RDF data about that resource, it is possible to provide T4R with an alternative URL for retrieving RDF data. This URL can point to an RDF document, but other kinds of data sources are possible.

Follow your nose. The "Follow your nose" strategy consists in obtaining information about a resource by retrieving data from its URI, and from other resources known to be related to the former (e.g. with the `rdfs:seeAlso` property). That strategy has been included in our T4R application: using the special URL `fyn:` for the a data source, the engine will retrieve data on demand from all intermediate resources involved in a path, and their related resources. Since this can lead to retrieve a lot of data, the number of queries that can be performed for one rendering can be bounded (this is the case for the online demo).

SPARQL endpoint. With the trend of linked data[6] gaining momentum, an increasing number of data sources are available as SPARQL endpoints. Although our implementation does not yet provide support for SPARQL endpoint, we plan to add this feature in a near future. Furthermore, the design of the implementation has been guided with this aim: a path is not evaluated on the fly, but parsed until the end, then evaluated. That way, a long path can be converted into a small number (ideally one) of SPARQL queries rather than querying the SPARQL endpoint at each step of the path.

Inference-enabled data sources. As pointed out in section 3, T4R makes no assumption about the inference capabilities of the underlying data sources, and purposefully aims at making no such assumption. It is not impossible, however, to deploy T4R in a context where RDF stores or SPARQL endpoints are known to have such inference capabilities, shifting the burden of complex queries from the templates to the inference engine. Although we have not implemented it yet, we have a back-end architecture making such an evolution straightforward.

4.2 Output format

We has stated in section 2 that TAL was designed to produce HTML and XML documents, while we claimed in introduction that T4R is able to produce any textual document. This deserves more explanation.

[6] `http://linkeddata.org/`

Although TAL is mainly based on XML attributes, it also recognizes a special XML element: `tal:block`. This element is a placeholder for TAL attributes, but only its content, not the tag, is rendered in the output document. Its use is not encouraged, since it breaks the validity of the template with respect to the output format, but nevertheless necessary in some situations to produce a valid output document.

The current approach of T4R for producing non-XML text-based output documents is to:

- exclusively use `tal:block` elements in the body of the template,
- enclose it in an artificial XML element before processing, to make it a well-formed XML document,
- remove that artificial XML element after processing.

This solution is not very elegant: processing instructions in the template are quite verbose, requiring both the `tal:block` element *and* one of the TAL attributes. However, it was straightforward to implement and to use, and enlarges, almost for free, the scope of T4R.

Proposing alternative syntaxes for integrating TAL processing instructions in specific non-XML languages is a possibility. However, the burden for the user of learning another syntax may counteract the advantage of that syntax being more integrated to the target language.

4.3 Modularity in templates

Modularity is a key to scalability, hence a desirable feature for any open and web-based technology. T4R offers two levels of modularity: one at the path level, and one at the template level.

Path level modularity is a mere consequence of a standard feature of TAL that we have not presented yet: indirection. We saw that TAL allows the definition of variables. A special use of variables is to evaluate their content as *elements of a path.* Assume the following variable declaration;

```
IMG  string:foaf:img/or/foaf:depiction/or/foaf:depicts:-
```

(note the TAL keyword `string:` indicating that the following text should not be evaluated, but considered as a literal string). This variable can now be dereferenced in any path using a leading question mark, avoiding the need to copy this long path multiple times. For example, the path:

```
foaf:knows/?IMG
```

will retrieve all the images representing the people I know.

Template level modularity, on the other hand, is more specific to T4R. It is implemented with the `renderWith` keyword, which must be followed by a CURIE path. For example, the following path:

```
foaf:knows/any/renderWith/lib:card.html
```

will use the resource template `card.html`, located at the URI associated with prefix `lib`, to render one of the people I know. Note that any CURIE path (or variable indirection) can be used after the `renderWith` keyword, allowing for complex selection of the template based on the resource itself. For example, the following TAL sample:

```
tal:define  = "p foaf:knows/any;
               t p/rdf:type/ex:template_for:-/any/asPath"
tal:content = "p/renderWith/?t"
```

will store a person I know in variable `p`, then retrieve the URI of any template suitable for one of the declared type of that person and store it in `t`[7], then use indirection to render `p` with that template.

5 Related works

The reference in terms of RDF presentation is Fresnel [13], a powerful RDF-based language for expressing presentation knowledge for RDF. Fresnel offers a very high level of modularity, distinguishing *lenses*, that group related information, from *formats*, that organise this information into an abstract box model. How this box model is rendered to concrete syntaxes is not in the scope of Fresnel and left to the implementations. Lenses and formats can be organized in groups, and more or less complex *selectors* can be defined to allow an agent to automatically identify the lenses and formats suitable for a resource. This powerful architecture has already federated several applications, like HayStack[8] and IsaViz[9], being the evolution of their original stylesheet formats [14, 15].

Compared to T4R, the strengths of Fresnel are also its weaknesses. Its model is quite complex and not practical for rapid prototyping of templates. Furthermore, the mapping from the abstract box model to concrete syntaxes being not specified in Fresnel, it is not a “ready to use” solution for application developers. In fact, we believe that T4R could be used to implement this missing step between Fresnel and concrete formats. Even more, provided some ad-hoc mechanisms to implement Fresnel selectors, we believe that (at least some part of) Fresnel could be implemented on top of T4R, following the method outlined in the end of section 4.3, since Fresnel lenses and formats are expressed in RDF. This would give an interesting hybrid rendering engine, allowing presentation knowledge to be gradually and seamlessly migrated from quickly hacked templates to cleanly designed lens and format groups.

[7] The T4R keyword `asPath` is used to convert the URI to an appropriate CURIE, in order to make variable `t` suitable for dereference.
[8] `http://haystack.csail.mit.edu/`
[9] `http://www.w3.org/2001/11/IsaViz/`

6 Conclusion and future works

In this paper, we have presented Tal4Rdf (T4R), a language for rendering RDF data in various XML and non-XML formats. Based on the proven TAL language, this uses an intuitive path-based language for retrieving RDF data and integrates well with output formats, both points providing it with a gentle learning curve and a suitability for rapid development. An open-source implementation and online demo are available at `http://champin.net/t4r/`.

A number of planned or possible evolution have been presented in sections 4 and 5: alternative syntaxes for non-XML formats, integration with SPARQL endpoints and inference-enabled RDF back-ends, support for Fresnel selectors and a possible T4R-based implementation of Fresnel.

Another interesting lead would be a better integration with RDFa [5]. The starting idea is that an HTML page containing RDFa is pretty similar to a T4R HTML template, although RDFa attribute provide the RDF information rather than retrieving it from another source. A first idea would be to convert such a page into a template, in order to reuse its presentation with another set of data. Another idea, suggested by Niklas Lindström, would be for T4R to automatically generate RDFa annotations when generating HTML.

References

1. Bizer, C.: D2R MAP - language specification. (May 2003)
2. Auer, S., Bizer, C., Kobilarov, G., Lehmann, J., Cyganiak, R., Ives, Z.: Dbpedia: A nucleus for a web of open data. Lecture Notes in Computer Science **4825** (2007) 722
3. Connolly, D.: Gleaning resource descriptions from dialects of languages (GRDDL). W3C recommendation, W3C (September 2007)
4. Khare, R.: Microformats: The next (Small) thing on the semantic web? Internet Computing, IEEE **10**(1) (2006) 75, 68
5. Adida, B., Birbeck, M.: RDFa primer. W3C working group note, W3C (October 2008)
6. Buffa, M., Gandon, F.: SweetWiki: semantic web enabled technologies in wiki. Proceedings of the international symposium on Symposium on Wikis (2006) 69–78
7. Krötzsch, M., Vrandečić, D., Völkel, M.: Semantic MediaWiki. In: The Semantic Web - ISWC 2006. Volume 4273 of LNCS., Spinger (2006) 935–942
8. Aubert, O., Prié, Y.: Advene: an open-source framework for integrating and visualising audiovisual metadata. In: Proceedings of the 15th international conference on Multimedia, ACM New York, NY, USA (2007) 1005–1008
9. contributors, W.: Template attribute language - wikipedia, the free encyclopedia (March 2009)
10. Brickley, D., Miller, L.: FOAF vocabulary specification. http://xmlns.com/foaf/spec/ (November 2007)
11. Birbeck, M., McCarron, S.: CURIE syntax 1.0. W3C working draft, W3C (March 2007)
12. Prud'hommeaux, E., Seaborne, A.: SPARQL query language for RDF. W3C recommendation, W3C (2008)

13. Pietriga, E., Bizer, C., Karger, D., Lee, R.: Fresnel: A browser-independent presentation vocabulary for RDF. In: Lecture Notes in Computer Science. Volume 4273., Athens, GA, USA (November 2006) 158
14. Quan, D., Karger, D.: Xenon: An RDF stylesheet ontology, Chila, Japan (May 2005)
15. Pietriga, E.: Semantic web data visualization with graph style sheets. In: Proceedings of the 2006 ACM symposium on Software visualization, ACM New York, NY, USA (2006) 177–178