

**Proceedings of the
5th International Workshop on

Scripting and
Development for the
Semantic Web
(SFSW 2009)**



Co-located with 6th European Semantic Web Conference
May 31 - June 4, 2008, Heraklion, Greece.



Workshop Chairs' Message
SFSW 2009 - Workshop on Scripting and Development
for the Semantic Web

On the current Semantic Web there is an ever increasing need for lightweight, flexible solutions for doing publishing, presentation, transformation, integration and general manipulation of data for supporting and making use of the increasing number of deployed open linked datasets and publicly available semantic applications. Communication and architectural standards such as AJAX, REST, JSON already cater to this need of flexible, lightweight solutions, and they are well supported by scripting languages such as PHP, JavaScript, Ruby, Python, Perl, JSP and ActionScript.

This workshop is concerned with the exchange of tools, experiences and technologies for the development of such lightweight tools, especially focusing on the use of scripting languages. Last year's workshop focused on the creation of Semantic Web data through social interactions as well as applications that integrate socially-created data across communities. Keeping in step with the increasing number of semantically enabled web-sites for public consumption, this year's focus is bringing the semantic web applications to the main-stream: everything from improving the user experience for browsing and accessing data, through integrating with existing non-semantic services, to quickly and cheaply porting such services to using a Semantic Web architecture. The workshop will follow the tradition and include a scripting challenge which will award an industry sponsored prize to the most innovative scripting application.

We would like to thank the organizers of ESWC conference for supporting the workshop. We especially thank all members of the SFSW program committee for providing their expertise and giving elaborate feedback to the authors and Talis for providing the Scripting Challenge prize. Last but not least, we hope that you will enjoy the workshop and the whole conference.

Chris Bizer, Freie Universität Berlin, Germany
Sören Auer, Universität Leipzig, Germany
Gunnar Aastrand Grimnes, DFKI, Germany

SFSW 2009 Program Committee

- Benjamin Nowack, semsol, Germany
- Bernhard Schandl, Universität Wien, Austria
- Bernhard Haslhofer, Universität Wien, Austria
- Claudia Müller, University of Potsdam, Germany
- Dan Brickley, The FOAF Project, UK
- Danny Ayers, Talis, UK
- David Aumüller, Universität Leipzig, Germany
- Eero Hyvönen, Helsinki University of Technology (TKK), Finland
- Eyal Oren, Free University Amsterdam, Netherlands
- Georgi Kobilarov, Freie Universität Berlin, Germany
- Giovanni Tummarello, DERI, NUI Galway, Ireland
- Gregory Williams, Rensselaer Polytechnic Institute, USA
- Harald Sack, HPI, Universität Potsdam, Germany
- Jens Lehmann, Universität Leipzig, Germany
- Knud Möller, DERI, NUI Galway, Ireland
- Leigh Dodds, Ingenta, United Kingdom
- Libby Miller, Joost, United Kingdom
- Markus Luczak-Rösch, Freie Universität Berlin, Germany
- Masahide Kanzaki, Keio University, Japan
- Michael Hausenblas, DERI, NUI Galway, Ireland
- Morten Høybye Frederiksen, MFD Consult, Denmark
- Richard Cyganiak, DERI, NUI Galway, Ireland
- Santtu Toivonen, Idean Enterprises, Finland
- Sebastian Dietzold, Universität Leipzig, Germany
- Stefan Dietze, KMi, The Open University, UK
- Tom Heath, Talis, UK
- Uldis Bojars, DERI, NUI Galway, Ireland
- Vlad Tanasescu, KMi, The Open University, UK
- Yves Raimond, BBC, UK

Table of Contents

Full Papers

Pierre-Antoine Champin

Tal4Rdf: lightweight presentation for the Semantic Web

Rob Styles, Nadeem Shabir and Jeni Tennison

A Pattern for Domain Specific Editing Interfaces - Using Embedded RDFa and HTML Manipulation Tools

Eugenio Tacchini, Andreas Schultz and Christian Bizer

Experiments with Wikipedia Cross-Language Data Fusion

Laura Dragan, Knud Möller, Siegfried Handschuh, Oszkar Ambrus and Sebastian Trueg

Converging Web and Desktop Data with Konduit

Short Papers

Jouni Tuominen, Tomi Kauppinen, Kim Viljanen and Eero Hyvönen

Ontology-Based Query Expansion Widget for Information Retrieval

Christoph Lange

Krextor -- An Extensible XML->RDF Extraction Framework

Stéphane Corlosquet, Richard Cyganiak, Axel Polleres and Stefan Decker

RDFa in Drupal: Bringing Cheese to the Web of Data

Mariano Rico, David Camacho and Oscar Corcho

Macros vs. scripting in VPOET

Norman Gray, Tony Linde and Kona Andrews

SKUA -- retrofitting semantics

Tal4Rdf: lightweight presentation for the Semantic Web^{*}

Pierre-Antoine Champin^{1,2}

¹LIRIS, Université de Lyon, CNRS, UMR5205,
Université Claude Bernard Lyon 1, F-69622, Villeurbanne, France

²CLARITY: Centre for Sensor Web Technologies,
CSI, University College Dublin, Ireland
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, allows to easily generate any XML or textual output format, using available and proven technologies. 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 RDF from legacy databases [1] or existing Web content [2, 3], to tag published content with machine-readable metadata [4, 5], or to ease the collaborative and flexible authoring of RDF data [6, 7]. Furthermore, the Linked Data initiative¹ advocates the interconnection of this growing amount of RDF data. It is expected that processing, adaptation, aggregation of data from multiple sources become common place on the Semantic Web. This creates a need for tools able to present open RDF data to the end user. By “open”, we mean that those tools can not know in advance the precise structure of the data, since RDF is by nature extensible. Another requirement 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, see also [8]).

Tal4Rdf (T4R), a lightweight template language for RDF, aims at providing the underlying technology to help fulfill those requirements. It allows to render RDF data in any XML or textual format, has an open-source implementation and an interactive demo both available at <http://champin.net/t4r/>. It is based on TAL (Template Attribute Language), an existing template language that we have already reused

^{*} This work is supported by Science Foundation Ireland under grant 07/CE/I1147 and the French National Research Agency (ANR) under project CinéLab (ANR-06-ANR-AM-025).

¹ <http://linkeddata.org/>

successfully in the Advene framework [9], which makes us confident in the potential of T4R as a “scripting language for presentation”.

In the next 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 [10] has been introduced in the Zope web development framework² 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 Table 1 a subset of the processing instructions of TAL, in order to give an overview of its capabilities. 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 [11]. Hence T4R could easily be ported to those languages (the current implementation is in Python, and uses SimpleTAL³).

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 providing an intuitive way of querying RDF data. For example, using the FOAF vocabulary [12], 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`

² <http://zope.org/>

³ <http://www.owlfish.com/software/simpleTAL/>

<code><tag tal:content="x/y/z">...</code>	Replace the content of the tag by the evaluation of <code>x/y/z</code> .
<code><tag tal:attributes="at x/y/z">...</code>	Add or replace attribute <code>at</code> in the tag, with the evaluation of <code>x/y/z</code> as its value.
<code><tag tal:condition="x/y/z">...</code>	Remove the tag and its content if <code>x/y/z</code> evaluates to False.
<code><tag tal:repeat="i x/y/z">...</code>	Assuming that <code>x/y/z</code> evaluates to a collection, variable <code>i</code> will iterate over it, and the tag will be repeated for each value of <code>i</code> .
<code><tag tal:define="v x/y/z">...</code>	Creates a variable <code>v</code> with the evaluation of <code>x/y/z</code> as its value.

Table 1. A summary of TAL processing instructions

meaning that, starting from the resource representing myself, T4R would need to traverse in sequence three arcs labelled with `knows`, `currentProject` and `homepage` respectively. The rest of this section describes the actual path syntax used in T4R, starting from this example, and explains the rationale for its design. For a full BNF grammar of the path syntax, see [15].

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 [13]. 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 [13] 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
```

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

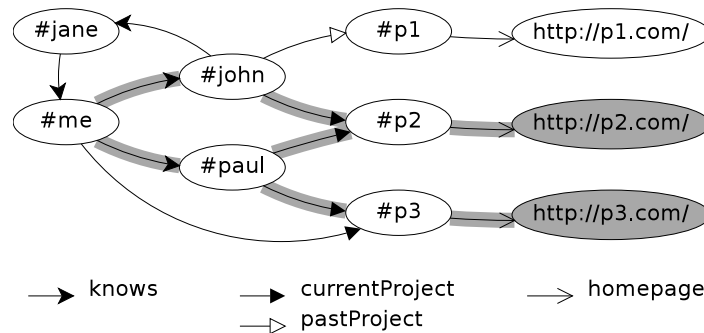


Fig. 1. An example data graph; the highlighted arcs and resources represent the evaluation, starting at #me, of the path `foaf:knows/foaf:currentProject/foaf:homepage`.

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 operator `any` to this path (see 3.4). However, it is not required 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 *one* of the result URIs (the other would then be ignored), keeping the link working⁴.

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:

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


```
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 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 [14], 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.

3.4 Operators

We have already encountered a T4R operator: `any`. Although all are not listed here because of space limitations (the interested reader is referred to [15] for an exhaustive list), let us just describe the three categories of operators:

- Node operators (such as `any`) transform a collection of nodes into another collection of nodes, hence may appear inside a path.
- Data operators transform a collection of nodes into a data values, hence may only appear at the end of a path. Figure 2 illustrate the use of operators `id` (shortening a URI to its last component) and `count` (counting the number of elements in a node collection).
- Binary operators always appear between two paths; an example will be provided in Section 4.3.

3.5 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 CURIE⁵.

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 such a path is not interpreted as a property, but as a resource. For example:

```
/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>
```

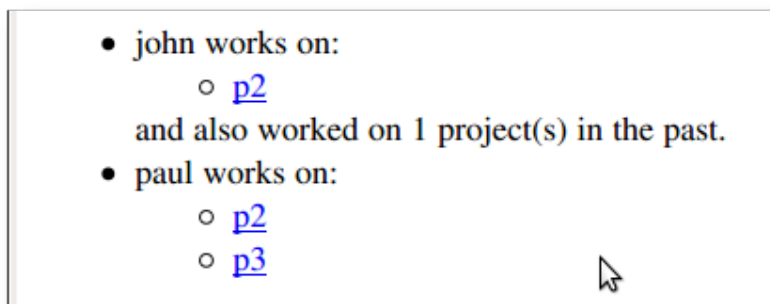


Fig. 2. A template and its result when applied to the resource `#me` in the graph from Figure 1.

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

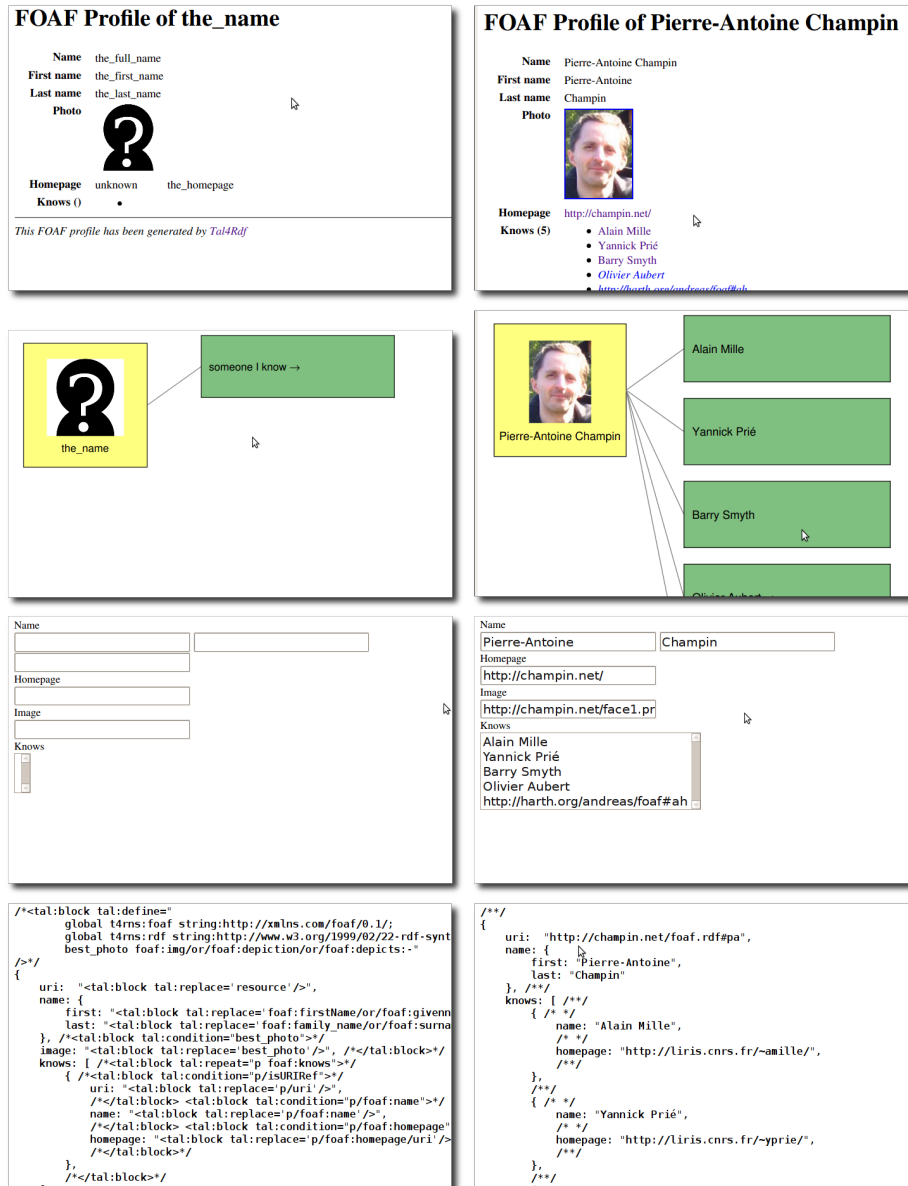


Fig. 3. Four examples from the online demo. The same data is rendered (right column) using HTML, SVG, HTML forms and JSON. The left column shows the unprocessed templates, which are valid documents in their respective formats.

4 T4R templates and data sources

As any typical rendering engine, a T4R processor 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.

4.1 RDF data sources

The RDF data source is provided to the T4R rendering engine 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 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 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 have 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.

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 prefix `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` binary operator, 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. Since templates can be located anywhere on the Web⁶, the number of retrieved templates can be bounded (like retrieved graphs with the “Follow your nose” strategy) to prevent overloading of the rendering engine.

Note also that any CURIE path (or variable indirection) can be used after the `renderWith` operator, 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 idea of using paths to query RDF is not new: see for example ARQ⁸, nSparql [16], or RDF Template [17]. While the first two are general purpose query languages, RDF Template aims at rendering RDF data to any XML document, making it very similar in purpose to T4R. All those languages are more expressive than our path language, which is constrained by the syntax of TALES, our underlying technology. While this limitation could easily be lifted in theory, one of the rationale of T4R is to rely as much as possible on the existing base of TAL, especially to make it straightforward to implement on existing TAL libraries. Hopefully, this will allow T4R to reach a broader acceptance than RDF Template did (the project doesn’t seem to be active anymore). Furthermore, this constraint on T4R has the virtue of keeping the templates relatively simple, enforcing the separation between presentation and application logics. We believe that complex queries should be stored *outside* the templates and invoked by them (a planned feature for T4R) rather than mixed with presentation structure.

The reference in terms of RDF presentation is now Fresnel [18], 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

⁶ Templates do not have privileged access to the system, so we do not consider this feature to be a major security issue. It is nevertheless possible that a malicious template consume an excessive amount of CPU time, so limiting the CPU resource granted to external templates is a planned feature.

⁷ The T4R operator `asPath` is used to convert the URI to an appropriate CURIE, in order to make variable `t` suitable for dereference.

⁸ http://jena.sourceforge.net/ARQ/property_paths.html

suitable for a resource. This powerful architecture has already federated several applications, like HayStack⁹ and IsaViz¹⁰, being the evolution of their original stylesheet formats [19, 20].

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.

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 external queries and 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.

The author would like to thank the reviewers for their constructive remarks, including references to some related works.

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

⁹ <http://haystack.csail.mit.edu/>

¹⁰ <http://www.w3.org/2001/11/IsaViz/>

3. Connolly, D.: Gleaning resource descriptions from dialects of languages (GRDDL). W3C recommendation, W3C (September 2007) <http://www.w3.org/TR/grddl/>.
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) <http://www.w3.org/TR/xhtml1-rdfa-primer/>.
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., Springer (2006) 935–942
8. Community, N.: RDF JSON specification. http://n2.talis.com/wiki/RDF_JSON_Specification (May 2009)
9. 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
10. Zope: TAL specification 1.4. <http://wiki.zope.org/ZPT/TALSpecification14> (May 2009)
11. Wikipedia contributors: Template attribute language - wikipedia, the free encyclopedia (March 2009)
12. Brickley, D., Miller, L.: FOAF vocabulary specification. <http://xmlns.com/foaf/spec/> (November 2007)
13. Birbeck, M., McCarron, S.: CURIE syntax 1.0. W3C working draft, W3C (March 2007) <http://www.w3.org/TR/2007/WD-curie-20070307/>.
14. Prud'hommeaux, E., Seaborne, A.: SPARQL query language for RDF. W3C recommendation, W3C (2008) <http://www.w3.org/TR/rdf-sparql-query/>.
15. Champin, P.: Tal4Rdf v0.4 reference manual. Documentation, SILEX - LIRIS (May 2009) <http://champin.net/t4r/doc/reference>.
16. Perez, J., Arenas, M., Gutierrez, C.: nSPARQL: a navigational language for RDF. In: *Proceedings of the 7th International Conference on The Semantic Web, Springer (2008)* 66–81
17. Davis, I.: RDF template language 1.0. <http://www.semanticplanet.com/2003/08/rdf/spec> (March 2008)
18. 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
19. Quan, D., Karger, D.: Xenon: An RDF stylesheet ontology, Chiba, Japan (May 2005)
20. 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

A Pattern for Domain Specific Editing Interfaces Using Embedded RDFa and HTML Manipulation Tools.

Rob Styles¹, Nadeem Shabir¹, and Jeni Tennison²

¹ Talis rob.styles@talis.com

² Talis nadeem.shabir@talis.com

³ Jeni Tennison Consulting jeni@jenitennison.com

Abstract. Many applications have the need to provide end users with editing capabilities. Often the nature of the domain and the user's workflow require a specialised editing interface. This paper describes the approach taken to building a specialised editing interface for academic resource lists and extracts the core aspects to allow others to apply the same pattern to their own applications. The solution described uses commonly available HTML manipulation tools and `rdfQuery`, a javascript RDFa library, to maintain an RDF model embedded in the page. This allows the application to provide a document style editing model over RDF data.

1 A Tool for Managing Academic Resource Lists

Talis Aspire is a SaaS (Software as a Service) application for managing lists of academic resources [1]. These resource lists are a key part of course content given to students to help them study more effectively. It provides a reading interface for students as well as a powerful editing interface for the lists' maintainers. This paper focusses on the approach used to implement the editing interface for list maintainers and how that can be seen as a pattern for other domain specific editing interfaces.

The view of the list when reading (Fig. 1) shows the list in the format students are used to receiving, similar in structure to a document. This can be cleanly printed for offline reference and each list, section and item have Cool URIs [10] to allow for bookmarking. Each item then has additional information and functionality behind simple links.

The editing view (Fig. 2) is used by academics and other faculty staff. Common tasks when editing lists are to add or remove particular items, to group items together into sections and to annotate the items to provide additional guidance to the students. The most common mental model of the resource list is that of a document, with the associated mental models that come from editing documents using typical office software.

This lead us to have a number of requirements for the editing interface.

The screenshot shows the Talis Aspire interface for a student view of an Academic Resource List. At the top, the Broadminster University logo and navigation links (Home, My Bookmarks, My Lists) are visible. The main heading is "Financial Accounting and Reporting". Below this, there are links for "Edit", "Used by ABF203", and "Group by: Section | Type". A "Table of contents" section is shown with a "[show]" link and "(94 items)". The list is categorized into sections: "Recommended Text (2 items)", "New section (0 items)", "Essentials of nursing research" (Book), and "Supplementary Resources (11 items)". Each item in the list includes a title, a link to "Get this item | Availability, buying options and notes", and a resource type (Book or Web Page).

Fig. 1. Talis Aspire: Student View of an Academic Resource List

Firstly, the mental model of a document requires that a number of edits can occur before the user saves the document with an explicit 'save' action. This matches the model of typical office software.

Secondly, the document being edited should be visually the same as the document when read, allowing the user to see what the result of their actions will be.

Thirdly, the editing interface should be a dynamic experience, supporting efficient editing with easy-to-use mechanisms for completing the common tasks. In practice this would mean an AJAX interface.

Fourth, the application allows for concurrent editing, something not in most people's mental model of a document, so we wanted to make saves resilient, where possible, to concurrent edits.

Fifth, the underlying data is stored as RDF and may be annotated and augmented by data the application does not know about. Edits should not be destructive to this data.

It is in this context that we set about designing a technical approach that met all five goals and was as simple as possible to implement.

HTML representations of underlying data are usually specialised interfaces designed to show the data in the best way for a human with a particular task to perform. An editing interaction designed for the user must integrate the editing function with this HTML representation and reflect changes the user makes in

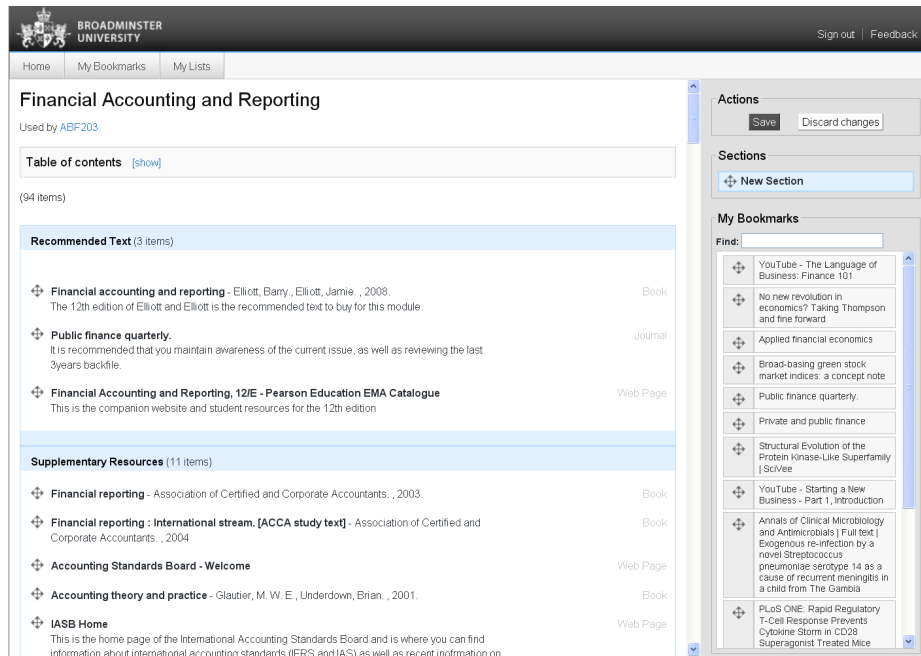


Fig. 2. Talis Aspire: Editing View of an Academic Resource List

that representation. In situations where the data is posted back to the server to update the underlying model the server can simply re-render the HTML based on the new model. If a more dynamic editing interface is to be provided using DHTML techniques then the HTML DOM must be manipulated directly and a number of changes are submitted to the server in one go.

There are a number of mature libraries that allow for manipulation of the DOM based on user input, including support for text editing, drag-and-drop, enumerated options and more. Those the team felt familiar with are Prototype⁴, JQuery⁵, Scriptaculous⁶ and YUI⁷. Any one of these libraries could provide the basis of the editing facilities we aimed to provide.

The approach we developed uses client-side javascript to change a read-only page into an editing mode. On switching into edit mode the first thing that happens is that the page is parsed to extract a copy of the initial model, which has been embedded in the page using RDFa [5]. This is kept as the starting point from which to derive a list of changes. After the initial model has been captured we use normal DHTML and AJAX libraries, in our case we used Prototype

⁴ <http://www.prototypejs.org/>

⁵ <http://jquery.com/>

⁶ <http://script.aculo.us/>

⁷ <http://developer.yahoo.com/yui/>

and Scriptaculous, to provide text editing, drag-and-drop re-ordering, removal of elements and drag-and-drop of new items onto a list.

We arrived at our approach independently and later found a great deal of inspiration in the work done on Ontowiki by Dietzold et al [6]. This work attempted to provide generic editing tools for RDFa within a wiki page. The notion of a suite of generic editing tools for RDFa is very appealing, but the user experience requirements for our application called for a specialised editing interface. The intention was to provide editing of structured data while maintaining the list view that users are accustomed to and would be expecting.

2 The Solution

The data underpinning Talis Aspire is stored as RDF in a Talis Platform Store [7], making use of several ontologies including AIISO⁸, Resource List⁹, Bibliontology¹⁰, SIOC¹¹ and FOAF¹².

Our solution for editing embeds RDFa within the page in such a way that manipulation of the HTML DOM results in consistent and coherent changes to the embedded model.

Server-side the pages are rendered by PHP from data stored natively as RDF in a Talis Platform store. The application does content negotiation to return HTML or rdf/xml to the client. If the response is HTML then RDFa is included in the page markup.

The HTML manipulation during editing uses a combination of custom code for edit dialogs and scriptaculous and prototype to provide drag-and-drop support.

The extraction of the model from RDFa within the page is performed by rdfQuery¹³ which also uses JQuery for some page parsing tasks. Other libraries, such as Ubiquity¹⁴ could be used as well, but our familiarity with jquery and prototyping experience with rdfQuery led us to select that.

Figure 3 shows a very much simplified diagram of the thin layer of application code for Talis Aspire, above a Talis Platform Store, accessed via the internet by both staff and students using standard browsers.

The key to the simplicity of this approach was to recognise that all we needed for an editing session was a pair of before and after models. These models can then be used to generate a changeset¹⁵ that persists the changes to underlying storage. The structure of changesets is such that they also provide an ideal mechanism for creating an auditable change history.

⁸ <http://purl.org/vocab/aiiso>

⁹ <http://purl.org/vocab/resourcelist>

¹⁰ <http://bibliontology.com/>

¹¹ <http://sioc-project.org/>

¹² <http://www.foaf-project.org/>

¹³ <http://code.google.com/p/rdfquery/>

¹⁴ <http://code.google.com/p/ubiquity-rdfa/>

¹⁵ http://n2.talis.com/wiki/Changeset_Protocol

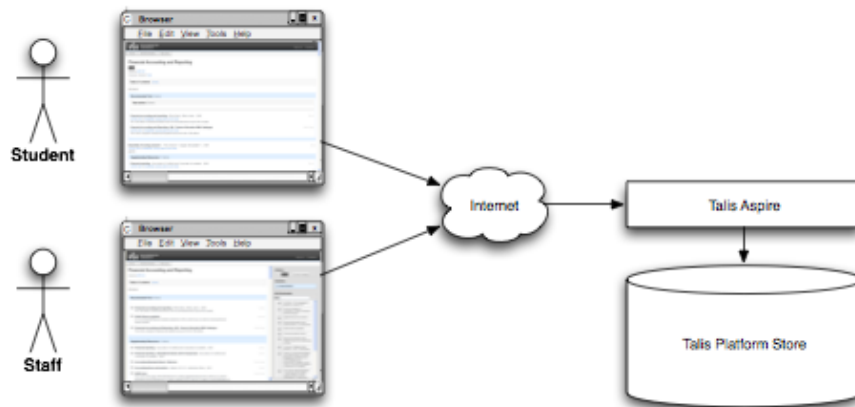


Fig. 3. Talis Aspire: Simplified Application Architecture

The changesets provide the equivalent of a Diff in source code configuration management. It is easy to conceive how this could be used to provide a feature to compare versions at some time in the future.

The client side code for saving a list is rather trivial, needing only to extract the final model and package that along with the initial model for submission to the server:

```

function saveListChanges() {
    showModalLayer('Saving list changes - please wait...');
    reindexSequences();
    var newModel = toRDFXML();
    var params = 'oldmodel=' + oldModel + '&newModel=' + newModel;

    new Ajax.Request(listUri,
    {
        method: 'post',
        parameters: {old_model: oldModel, new_model: newModel},
        contentType: 'application/x-www-form-urlencoded; charset=UTF-8',
        onSuccess: function(transport)
        {
            redirect(listUri);
        },
        on409: function(transport)
        {
            alert('Someone else has changed the data...');
            redirect(listUri);
        },
    },
  
```

```
    onFailure: function(transport)
    {
        alert('There was a problem... ' + transport.responseText);
        redirect(listUri);
    }
});
}
```

Server-side we have code that uses ARC2¹⁶ and Moriarty¹⁷ to generate the changesets from the before and after models and submit to the underlying store. This code is, again, small and relatively trivial. One of the reasons that the changesets are easy to produce, reconcile and apply is that we do not use blank nodes. Blank nodes would require more knowledge of the model and use of inverse functional properties. This would increase complexity.

Next we look at how this solution meets our stated goals.

2.1 Goal One: Explicit Save

Batching several edits from a single editing session together maintains a mental model for the user that is consistent with that of a document. This was a key factor in our user interaction design. Because edits are all held on the client until the user explicitly submits them the application behaves as expected. The client code is able to do this very simply as the model is embedded in the HTML, the same actions that change the DOM also change the model.

2.2 Goal Two: WYSIWYG Editing

The use of the existing HTML, the same HTML as is rendered for the reader view, gives us the same look and feel while editing as while reading. A few subtle changes are made; links disabled and some editing interface elements discreetly added.

By embedding the model within the HTML at read time we negate the need for a separately rendered view for editing, requiring only that the client supports javascript.

2.3 Goal Three: Dynamic Interface

We had to worry very little about finding or writing RDFa aware tools because RDFa is embedded using attributes and non-RDFa aware tools simply ignore attributes they don't recognise. The interface is provided by commonly available HTML manipulation libraries. This allowed us to provide a richer interface than we would otherwise have been able to build.

¹⁶ <http://arc.semsol.org/>

¹⁷ <http://code.google.com/p/moriarty/>

2.4 Goal Four: Concurrent Editing

By having the client maintain both the before and after models for an editing session it becomes possible to detect when different users have made changes concurrently. It is not possible to detect this when submitting the after editing model alone without the introduction of timestamps. This collision detection allows the application to make appropriate choices about how it deals with the conflict.

As the changeset is directly analogous to the diff patches generated by configuration management tools it is clear that non-conflicting changes to the same model can be reconciled and applied, while conflicting changes can be identified and the three states, before editing, edit one and edit two, can be offered to the user for the conflict to be resolved. We currently apply non-conflicting changes and reject conflicting changes with the first set of changes submitted winning.

An alternative approach would have been to implement locking on lists during editing. The benefits of concurrent editing for non-conflicting changes, and the costs of implementing a lock cleanup and reclaim mechanism where lists have been left locked unintentionally made this unappealing.

2.5 Goal Five: Extensible Data

Because the solution uses the changeset protocol to update only the resources and properties it knows about, rather than replacing resources and all of their properties, data that the application doesn't know about is left untouched. This, again, comes from having both the before and after models available to calculate a delta rather than simply performing a replacement.

2.6 Complication: RDF Sequences

The ordering and re-ordering of elements in the list posed a particular problem. We model the order of items on lists using RDF Sequences.

```
<http://lists.broadminsteruniversity.org/lists/abf203>
  sioc:name "Financial Accounting and Reporting" ;
  resource:contains <http://lists.broadminsteruniversity.org/items/abf203-1>,
  [...snip...] ;
  a rdf:Seq, resource:List ;
  rdf:_1 <http://lists.broadminsteruniversity.org/sections/abf203-1> ;
  rdf:_2 <http://lists.broadminsteruniversity.org/items/abf203-9> ;
  rdf:_3 <http://lists.broadminsteruniversity.org/sections/abf203-2> ;
  rdf:_4 <http://lists.broadminsteruniversity.org/sections/abf203-3> ;
  rdf:_5 <http://lists.broadminsteruniversity.org/sections/abf203-16> ;
```

We render these in the DOM using the explicit ordering predicates of `rdf:_1`, `rdf:_2` etc. The obvious implication of this is that now the re-ordering of the DOM is no longer enough to create the equivalent re-ordering in the model. We solved this by triggering a function to re-order the `rdf:Seq` predicates before extracting the post-editing model.

```

function reindexSequences()
{
var containerCuries = new Array();

$$('span[class="sequenceNumber"][rev="rdf:_0"]').each(function(thingToReindex) {
containerCuries[containerCuries.length] = thingToReindex.readAttribute('resource');
});

containerCuries.uniq().each(function(containerToReindex) {
updateSectionSequences(containerToReindex);
});
}

```

One would expect that sequences can be rendered in the RDFa using the sequence dependant `rdf:li` element as shown by Hausenblas [8]. Unlike the RDF/XML specification [9], however, the RDFa specification [4] does not state that `rdf:li` should be treated as a special case. As `rdfQuery` supports the spec accurately it doesn't interpret them specially.

A change to the specification to bring it inline with RDF/XML would allow the model to be re-ordered without changes to the predicates as the sequence is implied by the order of occurrence of the elements within the DOM.

This is one of the complications that results from the immaturity of the domain and the degree to which the specifications have yet to evolve and consolidate.

2.7 Complication: Importance of @Rev Attribute

The model uses a number of paired forward and inverse properties. These are used to provide optimisations for querying and convenience when navigating the data as linked data. When an item is removed from a list it is therefore necessary to remove not only the links from the item itself to other elements but also the links from other elements to the item. One example of this is the reference from the list to all of the items it contains, regardless of the level of nesting - this is provided so that the list and all its contents can be retrieved by a single sparql query.

The way we achieved consistency between the DOM editing and the model is through the use of the `@rev` attribute. This allows us to co-locate statements that reference the item as an object with the item itself in the DOM. This means that removing the item from the DOM also removes the statements that referred to it. This technique removed the need for us to write code to detect statements that had lost their children by the removal of the resource they referenced.

Alternatively we could have made the decision not to render these triples in the RDFa and to use reasoning in the changeset generator or other code to correctly manipulate the model. This would make the RDFa smaller and possibly easier to manipulate at the cost of requiring consuming apps to infer the additional properties from the schema.

As stores become better at reasoning and inference we should see the need for these additional predicates lessen.

2.8 Complication: Ignoring Some Changes to the Graph

The final complication we had to deal with is that some aspects of the model are used widely in the graph. The list itself, its sections and its items are a tree structure. Many items can refer to the same underlying academic resource, however. Within the HTML DOM the academic resource will be rendered as child elements of the list item that refers to it. If several list items refer to the same academic resource then the resource will be listed several times.

The problem arises when we wish to delete an item from a list. What we want to delete is the item on the list, i.e. the reference to the underlying academic resource and not the academic resource itself. When the same resource is referenced many times within the same list this does not present an issue as the triples describing the academic resource within the model remain even after one of the items is deleted.

When the academic resource is referenced only once on the list the removal of the item from the HTML DOM also results in the removal of academic resource and would result in a changeset being created to delete the academic resource from the underlying store. As academic resources can be referenced by many lists this deletion would result in other lists referencing a deleted academic resource.

Our solution is to never delete academic resources. This requires introducing knowledge of the model to the code that calculates differences between the before and after models. This is acceptable to us as specialised editing interfaces such as this one are expected to understand the model and limit themselves to changing only those things they are responsible for.

2.9 Complication: Performance

The Talis Aspire product works with lists that may contain references to several thousand resources, leading to tens of thousands of triples being embedded within a single HTML+RDFa page. Parsing the RDFa in such a page is a substantial task, and the initial beta version of rdfQuery that we used took tens of seconds to process the very largest of these pages, especially in Microsoft Internet Explorer. This proved to be a useful use-case for the optimisation of rdfQuery, which now parses these pages in just over a second in most cases, which is adequate for our purposes.

2.10 Additional Benefit: Re-Use of Data

One of the most attractive things about this approach was that it supported and was supported by our desire to publish RDFa in the pages. With very little out there to consume RDFa and the fact that we already supported content negotiation there was little justification to invest the effort in publishing the RDFa

within the HTML. This approach provided both the simplest implementation for editing as well as a sound reason for including RDFa – which we expect to have additional benefits in future.

2.11 Additional Benefit: Re-Use of Code

Very little bespoke code was written for management of editing. Almost all of the bespoke code was written to support specific user interactions during editing. Had RDFQuery supported RDF sequences we would have had to write only the code to post the before and after models to the server.

3 Future Work

There are a number of opportunities to simplify the pattern still further as the RDFa spec evolves and as RDF stores and libraries become more capable. We aim to track the situation and make changes as they arise and offer benefit.

There are also opportunities to extend the functionality of the client-side code using the capabilities of rdfQuery more effectively. We currently use it to simply extract the model from the page before and after editing, but it is capable of much much more. It would be possible to use rdfQuery to generate other views of the model from the RDFa, such as a list of sections within the editing interface, counts of items and indicators where the same resource is referenced multiple times. These kinds of interface additions are made much easier by having access to the machine readable model in the client-side code.

4 Conclusion

Embedding RDFa within pages provides benefits for the publication of structured data. Beyond that it can also be used to provide a coherent representation of the model that supports editing using tools that are not aware of RDF. The use of these tools can make the provision of specialised, browser-based editing interfaces substantially easier to implement than traditional approaches that involve maintaining a model independently of the HTML DOM.

The client-side, dynamic nature of this approach provides several other benefits beyond implementation simplicity, including a reduction in server interactions and the ability to detect and reconcile concurrent edits.

References

- [1] Clarke, C.: A Resource List Management Tool for Undergraduate Students based on Linked Open Data Principles. In Proceedings of the 6th European Semantic Web Conference, Heraklion, Greece, 2009.

- [2] Halb, W., Raimond, Y., Hausenblas, M.: Building Linked Data For Both Humans and Machines Workshop for Linked Data on the Web (2008) <http://events.linkedata.org/ldow2008/papers/06-halb-raimond-building-linked-data.pdf>
- [3] Tennison, J.: RDFa and HTML5: UK Government Experience <http://broadcast.oreilly.com/2008/09/rdfa-and-html5-uk-government-e.html>
- [4] Adida, B., Birkbeck, M., McCarron, S., Pemberton, S.: RDFa in XHTML: Syntax and Processing <http://www.w3.org/TR/rdfa-syntax/>
- [5] Adida, B., Birkbeck, M.: RDFa Primer: Bridging the Human and Data Webs <http://www.w3.org/TR/xhtml-rdfa-primer/>
- [6] Dietzold, S., Hellmann, S., Peklo M.: Using JavaScript RDFa Widgets for model/view separation inside read/write websites In Proceedings of the 4th Workshop on Scripting for the Semantic Web, Tenerife, Spain, 2008.
- [7] Leavesley, J. and Davis, I.: Talis Platform: Harnessing Sophisticated Mass Collaboration on a Global Scale. http://www.talis.com/platform/resources/assets/harnessing_sophisticated_mass.pdf
- [8] Hausenblas, M.: Writing Functional Code with RDFa <http://www.devx.com/semantic/Article/39016>
- [9] Beckett, D.: RDF/XML Syntax Specification (Revised) <http://www.w3.org/TR/rdf-syntax-grammar/>
- [10] Berners-Lee, T.: Cool URIs don't change <http://www.w3.org/Provider/Style/URI>

Experiments with Wikipedia Cross-Language Data Fusion

Eugenio Tacchini¹, Andreas Schultz² and Christian Bizer²

¹ Università degli Studi di Milano,
Dipartimento di Tecnologie dell'Informazione,
Via Bramante 65-26013 Crema (CR), Italy
eugenio.tacchini@unimi.it

² Freie Universität Berlin
Web-based Systems Group
Garystr. 21 - D-14195 Berlin, Germany
aschultz@mi.fu-berlin.de, chris@bizer.de

Abstract. There are currently Wikipedia editions in 264 different languages. Each of these editions contains infoboxes that provide structured data about the topic of the article in which an infobox is contained. The content of infoboxes about the same topic in different Wikipedia editions varies in completeness, coverage and quality. This paper examines the hypothesis that by extracting infobox data from multiple Wikipedia editions and by fusing the extracted data among editions it should be possible to complement data from one edition with previously missing values from other editions and to increase the overall quality of the extracted dataset by choosing property values that are most likely correct in case of inconsistencies among editions. We will present a software framework for fusing RDF datasets based on different conflict resolution strategies. We will apply the framework to fuse infobox data that has been extracted from the English, German, Italian and French editions of Wikipedia and will discuss the accuracy of the conflict resolution strategies that were used in this experiment.

Keywords: Wikipedia, DBpedia, Web of data, data fusion, information quality evaluation

1 Introduction

Different Wikipedia language versions can describe the same type of objects in different ways, using different infobox templates¹, providing different and often conflicting information about the same topic. As an example, the English version of Wikipedia currently states that the city of Munich has a population of 1,356,594

¹ http://en.wikipedia.org/wiki/Category:Infobox_templates

people while according to the German version the population of the same city is 1,315,476.

Handling these differences by applying data fusion algorithms can increase the information quality [1] of the resulting knowledge base if compared to the knowledge base derived from single Wikipedia editions: in the previous example it would be desirable for a user who enquires about the population of Munich to get the value provided by the German edition, which is more up-to-date, instead of the value provided by the English one. To get these results we need good heuristics which help recognize the correct dataset (Wikipedia edition) to choose from.

This paper examines the hypothesis that by extracting infobox data from multiple Wikipedia editions and by fusing the extracted data among editions it should be possible to complement data from one edition with previously missing values from other editions and to increase the overall quality of the extracted dataset by choosing property values that are most likely correct in case of inconsistencies among editions.

The work is structured as follows: we review related work in section 2; we give an overview of the DBpedia information extraction architecture, which we used to extract infobox data from different Wikipedia editions, in section 3; section 4 describes the data fusion framework that was used to merge data between editions. Section 5 presents the results of our experiments with applying different conflict resolution strategies to merge data between Wikipedia editions and estimates the accuracy of the fused datasets by comparing them to external “trusted” data.

2 Related Work

Data fusion is the process of merging multiple records representing the same real-world object into a single, consistent, and clean representation [3]. Beside of identity resolution, data fusion involves choosing the values that are most likely correct out of conflicting values within different data sets by applying conflict resolution strategies.

Data fusion is mainly addressed in the database research field. An overview of the field is given by Bleiholder and Naumann in [3].

In order to develop conflict resolution strategies for the Wikipedia use case, we reviewed existing work on Wikipedia information quality assessment. In [5] two metrics are used as a simple measure for the reputation of an article: Rigor (total number of edits of an article) and Diversity (total number of unique editors); the authors also verify that these measures positively change if an article gets a press citation. In [6] a rich citation-based trust evaluation is implemented. In [7] a list of quality metrics such as number of registered user edits, article length, currency, number of unique editors are applied to compute the quality of an article; as an experiment, the computed quality is used trying to recognize the *featured* Wikipedia articles. In [8] a Bayesian network model is used to compute the trust of an article, based on who edited the article (unregistered user, registered user or administrators) and on the status of the article (normal, to be cleaned, featured).

3 Infobox Data Extraction

The DBpedia project² extracts structured information from Wikipedia and makes this information available on the Web of Data [2]. Over 2.6 million Wikipedia entities are currently described in RDF [9], published according to the Linked Data principles [10, 11] and queryable via SPARQL [12].

Besides free text, Wikipedia articles contain structured information in the form of links, geo-coordinates, categories, links between different language versions of an article and infobox-templates which contain facts in a table like fashion. The aim of the DBpedia extraction framework³ is to parse this information and to transform it into a consistent structural form, namely RDF. Wikipedia data dumps offered by the Wikimedia Foundation serve as the main data source in the extraction process.

We have used the DBpedia information extraction framework to extract infobox data from English, German, Italian and French editions of Wikipedia. The extraction framework also solves several problems that would otherwise hinder the fusion of the different language versions.

At first a common ontology has to be established for all participating sources. For this purpose we used the already existing DBpedia ontology⁴ and we applied the mapping based approach of the extraction framework to the Wikipedia versions which we needed for our experiments. The main idea is to map infobox-templates coming from different Wikipedia editions which describe the same concept to the same class of the ontology and to map template properties to ontology properties.

The next step is to establish unique URIs of resources among the different Wikipedia editions; this step can be seen as the linking or duplicate detection step done in data integration. The employed approach to generate DBpedia URIs is to take the unique article name and prepending the DBpedia specific namespace (<http://dbpedia.org/resource/>) to it; however, article names can differ among language editions, so the Wikipedia interlanguage links⁵ are exploited to identify every resource by the URI of its English-edition equivalent (if existent) in order to achieve unique identifiers.

An advantage of the afore-mentioned mapping-based approach is that it handles a third problem regarding data fusion, the representation of literals. Literals can be found in all kinds of formats and units, strongly depending on the language edition. The canonization of these values is part of the DBpedia extraction framework and facilitates an easier handling in the fusion process.

For our experiments the following language versions of Wikipedia were extracted and mapped to the common ontology: German, Italian and French, while the English version was already mapped to the DBpedia ontology and extracted⁶.

² <http://wiki.dbpedia.org/>

³ <http://wiki.dbpedia.org/Documentation>

⁴ <http://wiki.dbpedia.org/Ontology?v=1cwu>

⁵ http://en.wikipedia.org/wiki/Help:Interlanguage_links

⁶ <http://download.wikimedia.org/backup-index.html> Versions of the Wikipedia dumps: en 08.10.2008, de 11.10.2008, it 30.10.2008, fr 17.10.2008

4 Data Fusion Framework

We have developed a framework for fusing RDF data from local and remote RDF data sources. The framework conducts two basic steps: 1. query each source to get the required data, and 2. apply a strategy to merge the data from the different sources. Strategies apply different heuristics and thus lead to different outcomes. Provenance information in the resulting dataset is preserved by distributing the outcome to different named graphs [13], one for each source.

For our experiments we developed several strategies for the complementation and conflict resolution of the source data, ranging from a simple union with duplicate elimination to quality based conflict resolution. Table 1 and 2 summarize the strategies. The strategies are partitioned in augmentation and quality related strategies. The goal of the former is solely a quantitative one, whereas the latter, choosing on the base of a quality evaluation process, focuses on increasing the quality of the resulting dataset, albeit they often also augment it. It should be noted that in every single execution of a strategy the data for one property of exactly one entity of the specified class is processed.

Table 1. Augmentation based strategies

Onevalue	This strategy chooses the first value it comes across only. A check order of the sources can be defined.
Union	All values from all sources are taken for the resulting dataset.

Table 2. Quality based strategies

Democratic	The choice is based on the number of sources which share the same value for the same object instance/property couple. It is also possible to assign a weight for each source.
Geographic	The choice is based on the provenance information of the examined entity.
Edits number	The choice is based on the number of edits a page has received since its creation.
Filtered edits number	Same as above but the edits marked as "Minor" by the users are not taken into consideration.
Unique editors number	The choice is based on the number of different users who edited a page since its creation.
Accesses number	The choice is based on the number of visits a page has received since its creation or in general since a starting date
Last update date time	The choice is based on the date and time of the most recent edit a page has received

We will explain the quality based strategies in more detail as follows:

Democratic: This strategy is useful if many sources exist and/or the data of these sources overlap to a high degree. All candidate values are handled like in a majority

decision: the value that gets the most votes - in this case, appears in the most sources – will be chosen. Additionally the user can define a weight for each source, that affects the ranking.

Geographic provenance strategy: The idea behind this strategy is the assumption that the information of concepts that are localized - like cities for example - is better maintained by people who are located near this concept. The term “location” could also be expanded in a broader or more abstract sense like “intellectual proximity” or “cultural proximity”. For this strategy it has to be clearly defined how to categorize the entities and the sources, so the information is chosen by the source that falls into the same category of the entity. An example is to categorize cities by their "country property" (e.g. locatedIn) and choose the information from the source of the same country, in our case the suitable DBpedia language edition.

Wikipedia based quality strategies: The Wikimedia Foundation and other institutions offer metadata⁷ for each page that include various statistics gathered about the changes that occur. The idea is to use these statistics to compute a quality ranking of the data from different sources, in our case, for the different language versions of an article in DBpedia. So this is a Wikipedia/DBpedia specific strategy. Table 2 shows all the implemented approaches for computing scores from this metadata which could be alternatively chosen; for the first four cases holds: the higher the number, the higher the score; for the last one, pages having more recent updates get higher score.

The developed data fusion framework provides for applying different fusion strategies to different properties of an entity. All aspects of the fusion process can be defined in a XML configuration file. The different configuration options are explained in the following.

The data sources are defined under the element *source* as shown in the example below:

```
<source id="dbpedia-en" type="sparql-endpoint "
augment="true">

  <url>http://localhost/sparql</url>

  <graph>dbpedia-en</graph>

</source>
```

⁷ sources of the Wikipedia quality indicators:

- Accesses numbers: <http://wikistatics.falsikon.de/dumps.htm> (July, August and September)

- Other indicators:

<http://download.wikimedia.org/itwiki/20081030/itwiki-20081030-stub-meta-history.xml.gz>

<http://download.wikimedia.org/enwiki/20081008/enwiki-20081008-stub-meta-history.xml.gz>

<http://download.wikimedia.org/dewiki/20081206/dewiki-20081206-stub-meta-history.xml.gz>

<http://download.wikimedia.org/frwiki/20081201/frwiki-20081201-stub-meta-history.xml.gz>

The *id* attribute is the unique name of the source; the *type* characterizes the access method that, in this version, is limited to SPARQL-endpoints. The optional augment *attribute*, if set for one source, makes sure that entities from other sources not present in the augmented one will be ignored. This attribute was set for the English DBpedia dataset for all our experiments: an entity from a non-English dataset not available in the English dataset was therefore ignored.

Besides attributes source-elements have two sub-elements: *url* and *graph*, which define the URL of the SPARQL-endpoint and optionally the named graph containing the data.

An optional default setup of fusion strategies is possible under the element *strategy-config* and can be used to set default configurations for each strategy for later reuse. Such a definition of a strategy element has the following structure:

```
<strategy-config>
  <strategy type="single-value" name="democratic">
    <args>
      <arg id="dbpedia-en" value="3" />
      <arg id="dbpedia-de" value="2" />
      <arg id="dbpedia-it" value="2" />
      <arg id="dbpedia-fr" value="2" />
    </args>
  </strategy>
  ...
</strategy-config>
```

Types can be set to *single-value* or *set-value*, which practically means that for a specific property only one value per entity should be chosen or a set of values. Birth date of a person is an example for the single value case, whereas band members would be a candidate for the set-value case. An optional *args* element defines the strategy arguments in an associative array fashion and is used to set up the strategy. Fusion strategies are applied to properties of specific classes:

```
<class URI="http://dbpedia.org/ontology/Film" >
  <property URI=" http://dbpedia.org/ontology /runtime">
    <strategy type="single-value" name="democratic" />
  </property>
</class>
```

In this case no arguments are supplied to the strategy and in this way the default configuration - only if defined beforehand - is used.

5 Experiments

In order to test our framework, we applied it to different classes of objects extracted from Wikipedia infoboxes, selecting specific properties for each class. We evaluated the information quality of our resulting dataset comparing it with the information extracted from sources external to Wikipedia which we assume to be accurate. As our goal was to improve the English dataset (which is the one currently used by DBpedia to answer queries), the same evaluation was also performed on this dataset only (without applying data fusion); in this way we could verify if the fusion process impacted positively on the information quality level. This is the general approach we used for the experiments, in order to easily get the results, for some classes additional or different steps were done. Three of the experiments we did are described in details in the following paragraphs.

5.1 Dataset augmentation using a simple UNION operator

The first experiment focused on the use of the union operator applied to object properties. We chose to extract the starring property of Wikipedia articles about movies. The strategy was thus to just merge starring information coming from different Wikipedia editions in order to produce a resulting movies-actors dataset which was more complete than the one provided by the English edition only.

We extracted, using the DBpedia extraction framework, the value of the starring property for all the articles that used the “Infobox Film” template in the English version. Analogue templates are used by the German (Infobox Film), Italian (Film) and French (Infobox Cinéma (film)) versions; all the infobox templates included the starring property and this allowed for extracting its value from the four different language versions.

At the end of the process we managed to extract starring information for 30,390 movies and 118,897 movie-actor triples for the English version of Wikipedia, 7,347 movies and 42,858 movie-actor triples from the German version, 6,759 movies and

31,022 movie-actor triples from the Italian version, 1,171 movies and 3,739 movie-actor triples from the French version.

We then used our framework to produce a new dataset of movies which includes the starring values from all four starting dataset and we get a dataset composed by 143,654 movie-actor triples, augmenting the English dataset by 20.82%.

We then created a dataset composed only of the movie-actor triples added to the English dataset and compared this dataset with the IMDB database⁸, which provides, among other data, for each movie, the list of actors who played a role in it.

In order to link DBpedia extracted movies and actors with the corresponding IMDB entries we used movie titles and actor names. In this example the linking process couldn't be accurately done like in the following experiments because movie titles and actor names in the IMDB dataset are not unique and are expressed in a format that differs from the one of DBpedia.

After this linking procedure we got 11,224 movie-actor triples and 61% of them are positively verified by the IMDB database check. The result of the experiment is positive because we expanded the dataset and most of the movie-actor triples were correct.

5.2 Data fusion using different information quality indicators

The second experiment we did focused on the use of the Wikipedia-based information quality indicators implemented in the framework. We took into consideration Wikipedia articles about minor planets; in particular we extracted the values of the orbital eccentricity property. This is a property whose values we could check from the MPC Orbit (MPCORB) Database⁹, a public database which contains orbital elements for more than 200,000 minor planets. The strategy was thus to fuse information coming from different Wikipedia editions using some of the implemented quality indicators proposed in literature in order to produce a resulting planets dataset whose information quality was higher than the one provided by the English edition only, i.e. whose orbital eccentricity values were closer to the ones provided by the MPCORB database.

We extracted, using the DBpedia extraction framework, the value of the eccentricity property for all the articles belonging to the “planets” class i.e. the articles that uses the “Infobox Planet” template in the English version. Analogue, though not identical, templates are used by the German (Infobox Asteroid), Italian (Corpo celeste) and French (Infobox Planète mineure) versions; all the infobox templates included the eccentricity property and this allowed to extract its value for the four different language versions.

At the end of the process we managed to extract eccentricity information for 11,682 planets from the English version of Wikipedia, 11,251 planets from the Italian version, 2,502 planets from the German version and 267 planets from the French version.

⁸ <ftp://ftp.fu-berlin.de/pub/misc/movies/database/>, data retrieved 2009 Feb. 23

⁹ <http://www.cfa.harvard.edu/iau/MPCORB.html>, vers. 2009 Feb. 5

The subset of planets we took into consideration for the experiment was composed of all the planets included in both the English and MPCORB dataset and at least in one of the other datasets. In order to link DBpedia extracted planets with the MPCORB planets we used the name of the planet, which is unique. The final dataset was composed by 11,033 planets.

We then built an “ideal” selection of planets, choosing, for each planet, the data coming from the language version whose eccentricity value is closest to the one provided by the MPCORB dataset; this ideal selection was composed of 9,937 planets extracted from the English version, 962 from the Italian version, 127 from the German version and 7 from the French version. Using this selection it was possible to improve the quality of the data (measured as the sum of the absolute differences between the eccentricity value provided by DBpedia and the MPCORB’s eccentricity value) by 17.47% in respect to a selection which just chose all values from the English version.

We then tried five different data quality indicators in order to see which one performed better and thus were able to create a selection which is as close as possible to the “ideal” selection; the results are shown in Table 3.

Table 3. Second experiment, performance of the information quality indicators tested

I.Q. indicator	Percentage of planets correctly selected
Edits number	10.16%
Filtered edits number	69.49%
Unique editors number	11.28%
Accesses number	19.43%
Last update date time	42.38%

The evaluation of the articles using the number of filtered edits is the one that performed better; 69.49% could be considered a good results but in this case, in which in more than 90% of the articles the information quality is higher in the English version (see “ideal” selection above), the final performance is worse in comparison to an approach which chooses all values from the English version so the final result for this experiment can't be considered positive. The information quality indicators proposed in most of the literature seem to work not very well, at least for this class of objects; one of the reason for poor performances of two of the edits-related indicators could be that we assumed that each edit operation added the same value to an article but, depending on author, size/type of content and other parameters the operation can increase (or, in some cases, decrease) the quality of an article at various levels. There are some parameters which can help us from this point of view (e.g. the minor parameter that we use for the filtered edits indicator) but we also have to take into consideration that the decision on marking an edit operation as minor is left to its author so in some cases the attribute could be unreliable.

5.3 Data fusion based on geographic provenance

The third experiment we did focused on the use of a promising information quality indicator: the geographic provenance. Our hypothesis was that for the class of objects that have a geographic provenance or localization (e.g. cities or people), data should be more accurate if taken from the Wikipedia version of the country they are related to. We took into consideration Wikipedia articles about cities; in particular we extracted the population data of Italian cities. We chose to focus on Italian cities because a public and up-to-date database providing data about cities population is available from the ISTAT (the national statistical institute of Italy) Web site¹⁰.

We extracted, using the DBpedia extraction framework, the value of the population property for all the articles that used the “Infobox CityIT” template in the English version. The analogue template used for the Italian version (the only non-English version considered in this experiment) was “Comune”.

In order to link DBpedia cities with ISTAT database's cities we used the ISTAT code, which is a unique identifier assigned to Italian cities; we got that code from the Geonames database dump¹¹ through the DBpedia - Geonames links dataset¹².

At the end of the process we managed to extract population information for 7,095 Italian cities from the English version of Wikipedia and 7,055 Italian cities from the Italian version.

The subset of cities we took into consideration for the experiment was composed of all the cities included in the English dataset and also in both the ISTAT and the Italian dataset. The final dataset was composed of 6,499 cities.

Following our initial hypothesis, we argued that Wikipedia articles about Italian cities were more accurate in the Italian Wikipedia Version. We thus compared population data of both the English and the Italian datasets with the data provided by ISTAT and these were the results: for 59% of the cities Italian data was more accurate (closer to ISTAT data); for 13% of the cities the quality was the same in both the datasets and for the remaining cities (28%) English data was more accurate. The final result for this experiment can be considered positive because the information quality of the resulting dataset is better in respect to an approach which chooses all values from the English version. This result confirmed our initial hypothesis; for articles with strong geographic localization characteristics as cities data should be more accurate if taken from the Wikipedia version of the provenance country.

¹⁰ <http://demo.istat.it/bilmens2008gen/index02.html>, data retrieved 2009 Feb. 13

¹¹ <http://download.geonames.org/export/dump/IT.zip>, data retrieved 2009 Feb. 13

¹² http://downloads.dbpedia.org/3.2/links/links_geonames_en.nt.bz2, data retrieved 2009 Feb.

6 Conclusions and Future Work

We presented the first version of a framework which is able to perform data fusion among different RDF datasets and which provides several conflict resolution strategies. We tested the framework in the Wikipedia/DBpedia domain, fusing data extracted from different Wikipedia language versions and we demonstrated that in some cases it is possible to increase the quality of the extracted information compared to extracting from the English Wikipedia edition only.

The results of the experiments were not always positive. As the quality of data within the English Wikipedia edition is already relatively high, it was difficult to improve data from the English edition with data from other editions. On the other hand, as the datasets that were extracted from other editions were relatively sparse, the solution proposed should work much better for augmenting a non-English Wikipedia version with the information extracted from the English version.

The geographic provenance of a DBpedia object is a promising indicator for quality evaluation, so one of the directions for future works will be an improvement of its implementation. An improvement of the other indicators is also desirable, especially in the direction of allowing to express the score with an higher level of granularity: as an example consider the possibility to have the last update date referred not to the whole page but to a fragment of it (a row of an infobox), this would give us the possibility to evaluate the currency of a single property instead of the currency of the page. We also have to proceed in the direction of fusing more Wikipedia editions; we tested our framework with four editions but adding other language versions can add more (potentially good) sources and in this way improve the information quality of the final dataset.

References

1. Bizer, C.: Quality-Driven Information Filtering in the Context of Web-Based Information Systems. PhD thesis, Freie Universität Berlin (2007)
2. Auer, S., Bizer, C., Lehmann, J., Kobilarov, G., Cyganiak, R., Ives, Z.. Dbpedia: A nucleus for a web of open data. Proceedings of ISWC07 (2007)
3. Bleiholder, J. and Naumann, F. 2008. Data fusion. ACM Comput. Surv. 41, 1 (Dec. 2008), 1-41. DOI= <http://doi.acm.org/10.1145/1456650.1456651>
4. Naumann, F., Bilke, A., Bleiholder, J, Weis, M.: Data Fusion in Three Steps: Resolving Schema, Tuple, and Value Inconsistencies. IEEE Data Engineering Bulletin 29(2):21-31 (2006)
5. Lih, A.: Wikipedia as Participatory Journalism: Reliable Sources? Metrics for Evaluating Collaborative Media as a News Source. Proceedings of the Fifth International Symposium on Online Journalism (2004)
6. McGuinness, D. L., Zeng, H., Pinheiro da Silva, P., Ding, L., Narayanan, D., Bhaowal, M.: Investigations into trust for collaborative information repositories. Workshop on the Models of Trust for the Web (MTW'06) (2006)

7. Stvilia, B., Twidale, M. B., Smith, L. C., Gasser, L.: Assessing information quality of a community-based encyclopedia. In: Proceedings of the International Conference on Information Quality - ICIQ 2005. Cambridge, MA. 442-454 (2005)
8. Zeng, H., Alhoussaini, M.A., Ding, L., Fikes, R., McGuinness, D.L.: Computing trust from revision history. Intl. Conf. On Privacy, Security and Trust (2006)
9. Beckett, D.: RDF/XML Syntax Specification (Revised). W3C Recommendation. <http://www.w3.org/TR/rdf-syntax-grammar/> (2004)
10. Berners-Lee, T.: Linked data. <http://www.w3.org/DesignIssues/LinkedData.html> (2006)
11. Bizer, C., Cyganiak, R., Heath, T.: How to publish linked data on the web, <http://sites.wiwiwiss.fu-berlin.de/suhl/bizer/pub/LinkedDataTutorial/> (2007)
12. Prud'hommeaux, E., Seaborne, A.: SPARQL Query Language for RDF. W3C Recommendation. <http://www.w3.org/TR/2008/REC-rdf-sparql-query-20080115/> (2008)
13. Carroll, J., Bizer, C., Hayes, P., Stickler, P.: Named Graphs. Journal of Web Semantics, Vol. 3, Issue 4, p. 247-267 (2005)

Converging Web and Desktop Data with Konduit

Laura Drăgan¹, Knud Möller¹, Siegfried Handschuh¹, Oszkár Ambrus¹, and
Sebastian Trüg²

¹ Digital Enterprise Research Institute, National University of Ireland, Galway
`firstname.lastname@deri.org`

² Mandriva S.A., France
`strueg@mandriva.com`

Abstract. In this paper we present Konduit, a desktop-based platform for visual scripting with RDF data. Based on the idea of the semantic desktop, non-technical users can create, manipulate and mash-up RDF data with Konduit, and thus generate simple applications or workflows, which are aimed to simplify their everyday work by automating repetitive tasks. The platform allows to combine data from both web and desktop and integrate it with existing desktop functionality, thus bringing us closer to a convergence of Web and desktop.

1 Introduction

With the Semantic Web gaining momentum, more and more structured data becomes available online. The majority of applications that use this data today are concerned with aspects like search and browsing. However, a greater benefit of structured data is its potential for reuse: being able to integrate existing web data in a workflow relieves users from the investment of creating this data themselves. On the other hand, when it comes to working with data, users still rely on desktop-based applications which are embedded in a familiar environment. Web-based applications either simply do not exist, or have shortcomings in terms of usability. They can only access web data, and do not integrate with data that users might already have on their own desktop, let alone with other applications. Even considering that it may be beneficial for users to publish some desktop data online, releasing all their data on the web may raise significant privacy issues. Instead, what is needed is a way of accessing structured web data from the desktop, integrate it with existing desktop-data and applications and work with both in a unified way.

The Semantic Desktop through projects such as Nepomuk now opens up new possibilities of solving this problem of integrating data and functionality from both web and desktop. On the Semantic Desktop, data is lifted from application-specific formats to a universal format (RDF) in such a way that it can be interlinked across application boundaries. This allows new ways of organizing data, but also new views on and uses of arbitrary desktop data. What is more, because desktop data is now available in a web format, it can also be interlinked and

processed together with genuine web data. While the unified data model makes this scenario easier than it previously was, implementing it would ordinarily still require an experienced developer, who would use a full-edged programming language to create applications that manipulate and visualize RDF data. With current tools, casual or naive users would not be able to perform such tasks.

In this paper, we present an approach for mashing up RDF data, which can originate from either the web or, through the Semantic Desktop, from arbitrary desktop applications. While the individual components that make up our approach are not new in themselves, we believe that the combination is new and opens up possibilities that have not been available before.

2 Background

Our work is based on and influenced by several existing technologies, such as the Semantic Desktop, Unix pipes, scripting languages, visual programming & scripting and dataflow programming. In the following we will describe these technologies.

Our approach assumes that we are working on a **semantic desktop** [1], rather than a conventional one. As discussed earlier, this means that data in application-specific formats has been lifted to a uniform data format, such as RDF or, in the case of the Nepomuk project [3], to an extension such as NRL³ (in the remainder of this paper, we mean a semantic representation language when we say RDF). Representing desktop data in a uniform format means that it can be interlinked and processed in a uniform way across the desktop, but also that it can be interlinked and processed with web data in the same way.

For our application the implementation of choice of the Semantic Desktop is Nepomuk-KDE⁴, developed during the Nepomuk project as part of the K desktop environment. However, also more mainstream products, such as Spotlight technology of Mac OS X are a step towards a unified view on all desktop data.

The concept of **pipes** has been a central part of UNIX and its derivatives since 1973, when it was introduced by M. Doug McIlroy. The basic idea of pipes is that individual processes or programs can be chained into a sequence by connecting them through the operating systems standard streams, so that the *stdout* of one process feeds into its successor's *stdin*. In this way, tasks which require the functionality from different applications or data from different sources can elegantly be combined into a single workflow.

Scripting languages such as Perl and Unix shell allow rapid application development and a higher level of programming. They represent a very different style of programming as compared to system programming languages like C or Java, mainly because they are designed for “gluing” applications [8]. The libraries provided by most scripting languages are highly extensible, new components being added as the need for them arises. Being weakly typed is another defining characteristic of scripting languages that Konduit employs.

³ <http://www.semanticdesktop.org/ontologies/nrl/> (26/02/2009)

⁴ <http://nepomuk.kde.org> (26/02/2009)

As a form of end-user programming, **visual programming** (VP) is targeted at non-experts who want to be able to automate simple processes and repetitive tasks, without having to learn the complexities of a full-fledged programming language. In visual programming users construct the program not by writing source code, but instead by arranging and linking visual representations of components such as data sources, filters, loops, etc. In other words, “a visual programming system is a computer system whose execution can be specified without scripting” [5] — “scripting” here in the traditional sense of writing lines of source code.

Recently, VP has gained some popularity in the form of Yahoo Pipes⁵. In allusion to UNIX Pipes, Yahoo Pipes allows the user to visually compose workflows (or pipes) from various ready-made components. Inputs and outputs are mostly news feed-like lists of items. Being a Web application, Yahoo Pipes is limited in that it operates on Web data only, in formats such as RSS or Atom. Another application that supports a wider range of (Semantic) Web data standards and also tightly integrates with the SPARQL query language is SparqlMotion⁶. Because of the simplicity and typical small-scale scope of software like Yahoo Pipes, SparqlMotion and also Konduit, they are often being tagged with the term *visual scripting* instead of VP.

Closely related to our approach are the Semantic Web Pipes [6], which apply the Yahoo Pipes look and feel and functionality directly to Semantic Web data. Also here, SPARQL is an integral component to define the functionality of the individual building blocks. A crucial difference between SparqlMotion and Semantic Web Pipes on the one hand and Konduit on the other is that they have a clear focus on Web data and do not integrate desktop data or application functionality.

The concept of designing workflows by chaining a set of components through their inputs and outputs is related to a form of programming called **dataflow programming** (e.g., [7]). Unlike the more conventional paradigm of imperative programming, a program in dataflow programming does not consist of a set of instructions which will essentially be performed in sequence, but instead of a number of interconnected “black boxes” with predefined inputs and outputs. The program runs by letting the data “flow” through the connections. As soon as all inputs of a particular component are valid, a component is executed.

2.1 Related Work

Apart from those mentioned above, there are a number of other systems which are related to Konduit. WebScripter [9] is an application that allows users to create reports in a spreadsheet-like environment from distributed data in the DAML (DARPA Agent Markup Language) format. Unlike our approach, WebScripter is based on the now more or less extinct DAML, and offers neither a

⁵ <http://pipes.yahoo.com/> (26/02/2009)

⁶ <http://composing-the-semantic-web.blogspot.com/2007/11/sparqlmotion-visual-semantic-web.html> (26/02/2009)

visual composition environment nor the option to connect to desktop functionality. Potluck [4] is a web-based platform for visually mixing structured data from different sources together, even if the data does not conform to the same vocabulary or formatting conventions. An important restriction is the fact that only data from sites which are hosted using the Exhibit⁷ platform can be merged. Potluck is geared towards data integration, and therefore does not offer any of the workflow capabilities we implement in Konduit.

3 Konduit Components and Workflows

With Konduit we want to allow casual users to build simple programs in order to perform and automate everyday tasks on RDF data. Konduit provides a collection of useful components ready for immediate use. The components offer individual units of functionality and are represented visually as blocks. They are connected through input and output slots, and in this way the flow of the program is defined. In order to keep simple the task of connecting components, the only data that flows through the workflow is RDF. This condition insures that each component always fulfils the minimal requirement for dealing with its input. Obviously, components may be specialized with respect to the actual vocabulary on which they can operate and will decide at runtime if and how it deals with the incoming RDF. By neither allowing different kinds of data (e.g., text, numbers, lists, images, etc.), nor typing the RDF data with respect to the vocabularies they use, we stay very close to the original UNIX pipes concept, where data is always an untyped bytestream on the one of the standard streams *stdin* or *stdout*, and where it is up to each process or program how to handle it (see Fig. 1). Konduit is implemented as a desktop-based application for the

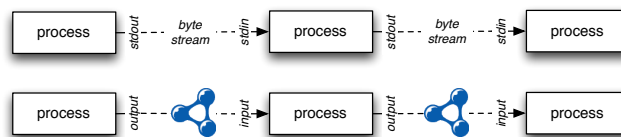


Fig. 1: Unix pipes and RDF pipes

Linux desktop environment KDE4, and is based on Plasma⁸. The architecture is plugin-based, so that each component is realised as a plugin into the Konduit platform. Technically, Konduit plugins are also so-called “Plasma applets”. Therefore designing and implementing new ones is quite straightforward (from the point of view of a KDE developer); and although all existing Konduit plugins have been written in Qt/C++, to write new ones can be done using the Ruby,

⁷ <http://simile.mit.edu/exhibit/>

⁸ <http://plasma.kde.org/> (26/02/2009)

Python or Java bindings of Qt. We expect that new plugins will be developed by external power users, as the need for them arises. As Plasma applets, the Konduit plugins can be loaded and used as independent applications directly on the desktop, without being restricted to the Konduit workspace. The workspace is not unlike a drawing canvas, on which the components can be dropped from a lateral toolbar. On this “drawing area” the user can connect the input slots to output slots of different components, move the blocks around, set their parameters and in this way build small applications.

Konduit makes use of the semantic desktop features that come as part of Nepomuk implementation in KDE4, especially the Soprano RDF framework⁹. Soprano is also used to store the saved workflows and black boxes as RDF in a repository (with the given connections and values for configuration parameters).

3.1 Components

Formally, a component is defined by the following parameters: (i) a set of RDF input slots I , (ii) a set of RDF output slots O , (iii) a set of parameters P which allow for user input in the workflow, (iv) a unit of functionality F , which works on the input I and generates the output O . The parameters P influence the behaviour of F .

Definition 1. *Component* = (I, O, P, F)

The number of input and output slots is not fixed and can be 0 or more. Depending on the number of slots, components can be grouped in three categories: sources, sinks, and ordinary components. *Sources* are components that do not have any inputs. They supply the workflow with data. Because the data graphs can be merged, there can be more than one source for any workflow. Typical examples of sources are connectors to RDF stores, file (URL) input components, or converters from other, non-RDF formats. *Sinks* are components that do not have any outputs. They represent the final point(s) of any workflow. Examples of sink components are application adaptors, serializers (file output components) and visualizers. Unlike in dataflow programming where a component is run as soon as all inputs are valid, the Konduit workflows are activated from a sink component, usually by clicking on an activation button.

Ordinary components, can be further classified according to the kind of functionality F they contain.

- **Merger** - combines the input graphs into a single output graph
- **Duplexer** - duplicates the input graph to two outputs.
- **Transformer** - applies a transformation on the input graph and outputs the resulting graph.

An important aspect of our approach is directly tied to the fact that all inputs and outputs are RDF graphs. As a result, any workflow can itself become

⁹ <http://soprano.sourceforge.net/> (26/02/2009)

a component, meaning that workflows can be built recursively. In this way, it is possible to create a library of specialised components (which we call *blackboxes*), based on the combination of basic components. We will pick this idea up again in Sect. 3.2.

Sources. Sources are a special type of components that do not have any input slots. There is always at least a source at the start of any workflow.

There is a dedicated source component for reading data from the local Nepomuk RDF repository. This source extracts the desktop data according to a SPARQL construct query given as parameter. The Nepomuk source element has a variant that is meant to help the user create the SPARQL query in a friendlier way, by the means of a smart wizard, with autocompletion and suggestions. Another basic source component is the file input source, which takes

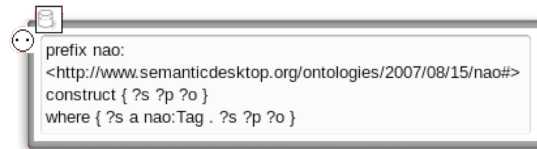


Fig. 2: Konduit Nepomuk Source that finds all the data about the tags from the local Nepomuk repository.

a URL as a parameter. The URL can point to a file (network is transparent so the path can be local or remote) or to a SPARQL endpoint (see Fig. 3). This component takes as parameter the expected serialization of the graph. For parsing it uses the parsers made available by the Soprano library. There are several

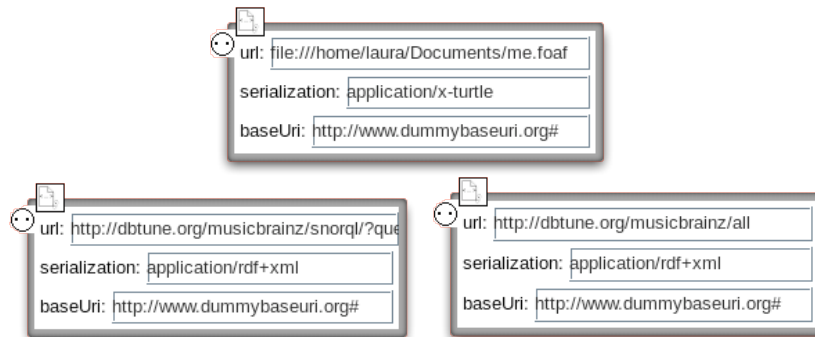


Fig. 3: The three uses of the Konduit File Input Source.

components that transform non-RDF data to RDF. The literal input takes any text given as parameter and transforms it to a RDF graph containing exactly one triple:

```
<http://www.konduit.org/elements/LiteralValue/data>  
  <http://www.w3.org/2000/01/rdf-schema#comment>  
    "string data"^^<http://www.w3.org/2001/XMLSchema#string>
```

The literal file input creates the same kind of triple, using as the string value the content of the file given as parameter.

Transformers. The most basic and simple transformer component is the filter element. It changes the input graph according to a SPARQL construct query given as parameter. The filter element can be saved with fixed queries and thus create specialized converters from one vocabulary to another. Another useful transformer is the duplicate remover component, which as the name suggests, outputs each unique triple from the input graph exactly once and discards all the duplicates.

Visualizers. The visualizer components display the RDF data received as input in various forms. So far there are only two components of this type: the data dump sink which shows the graph as quadruples in a separate window; and the data table sink which creates tables for each class of resource found in the input graph, each table having on each row one data for one instance in the graph. The columns are given by the properties of the class shown in each table.

Application adaptors. Application adaptors call the functionality of an external application or protocol with the data given in the input graph.

One such adaptor is the mailer element. It takes as input several graphs of data: one of foaf:Persons with mbox and name, one with the list of files to attach to the sent emails, a template for the message and a subject.

Another adaptor is the scripter element which passes the input RDF graph as input to a script available on the desktop. There is no restriction regarding the nature of the script or the language in which it is written, as long as it is executable, it takes RDF as input and it outputs RDF as well. The serialization for the input and output must be the same and it can be specified as a parameter.

3.2 Workflows

A workflow is defined by specifying (i) a set of components C , (ii) a function f defined from the set of all the inputs of the components of C to the set of all the outputs of the components of C and the *nil* output. The function f shows how the components of C are connected. The inputs that are not connected have a *nil* value of f ; the outputs that do not represent a value of f are not connected.

Definition 2. $Workflow = (C, f)$ where $f: inputs(C) \rightarrow outputs(C) \cup \{ nil \}$

Workflows can be saved and reused. Saving a workflow implies saving all the components that have at least one connection to the workflow, as well as their existing connections, parameters and layout. There is no restriction that the components should be completely connected, so there can be input or output slots that remain open. A saved workflow can be reopened and modified by adding to it or removing components, or changing connection or parameters and thus obtaining different workflows with minimum effort.

Even the simple workflows can have numerous components, the more complex ones having tens of components can become too big to manage in the workspace provided by the application. To aid the user with handling large and complex workflows, we added modularization to Konduit. Workflows can thus be saved as reusable components, which we call *blackboxes* and which are added to the library of available elements. Blackboxes can be used afterwards in more complex workflows. This can be done recursively as more and more complexity is added. The inputs and outputs of blackboxes must be marked in the original workflow by special input and output components (as illustrated in Fig. 4).

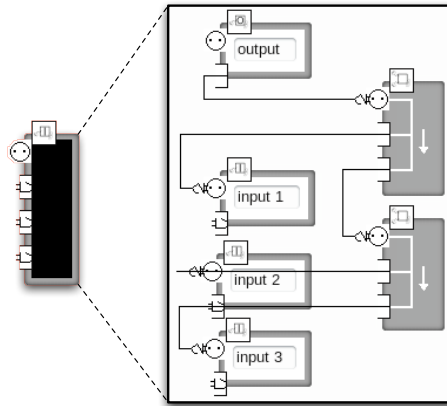


Fig. 4: Looking inside a tri-merger black-box created by concatenating two merger elements.

4 Use Case

The following example illustrates what Konduit can do for the user of a *semantic desktop*.

John is a music enthusiast. He enjoys having his music collection organized, and if he likes an artist he will try to find that artist's entire discography. Whenever he discovers a new singer he creates a file with that singer's discography and

marks which albums or songs he owns and which he does not. This task requires usually many searches - on the web as well as on John's own computer. Some of the common problems he runs into are: on the web the information he needs is spread across several web pages which need to be found; on his computer the music files are spread over several folders, and he would have to manually check each file to mark it as owned.

This example highlights a number of important aspects that our approach addresses, and illustrates how a tool such a Konduit can be used:

- Accessing and processing desktop data: John uses the semantic desktop offered by Nepomuk on KDE4 so he has his music library metadata stored in the Nepomuk repository and can therefore be processed by Konduit.
- Accessing and processing web data: Services¹⁰ expose their data as RDF, which means that our system can use it.
- Merging desktop and web data: Since both kinds of data sources use a unified data model, Konduit can simply mash both together.
- Using desktop functionality: Since our approach is desktop-based, we can easily access and integrate the functionality of arbitrary desktop applications or run local scripts that are normally executable on the desktop (with the restriction of taking as input and outputting RDF).

Three main parts of the workflow stand out: preparation of the data found online, preparation of the data found locally on John's desktop and the generation of the file. For the first two parts we create sub-workflows which we save as blackboxes and use them in the final workflow. Both blackboxes take as input the name of the artist and output album and track data, one from the desktop and the other from the web.

Desktop data. To access the local music metadata we need a Nepomuk source component. It will return the graph of all songs found in the Nepomuk repository, with title, artist, album, track number and the URL of the file storing the song. This graph needs to be filtered so that only the songs by the specified author remain. For it we use a filter element.

Web data. We use the SPARQL endpoint provided by the musicbrainz service¹¹ to retrieve music information. To connect to it we need a file input source with a query that takes data about artists, albums and tracks. The graph returned by the source has to be filtered by the artist name. This is done with a filter component that has also the function of a vocabulary converter, as it takes in data described using the Music Ontology¹² and creates triples containing the same data described with the Xesam ontology¹³.

¹⁰ such as <http://dbtune.org/musicbrainz/>

¹¹ <http://dbtune.org/musicbrainz/sparql>

¹² <http://purl.org/ontology/mo/>

¹³ <http://xesam.org/main/XesamOntology>

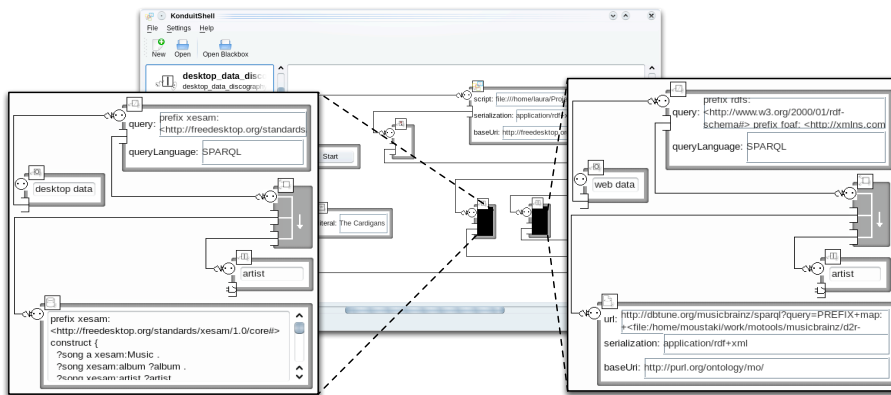


Fig. 5: The entire discography generator workflow.

Running the script. The scripter will take as input a graph constructed from the two subgraphs: one containing the data about the artist extracted from the web and the other the data about the artist available on the desktop. Both graphs contain xesam data. The merged outputs are first passed through a duplicate remover component to eliminate the redundant triples. The script takes the resulting graph of albums and tracks for the given artist and generates a file containing the discography. The RDF output of the script contains the path to the generated file, and is used by a File Open component to display the discography in the system default browser. The final workflow is depicted in Fig. 5 and the generated discography file in Fig. 6. A more detailed description of the workflow, including the SPARQL queries that are used and the script can be found at [2]

5 Discussion and Future Work

In this section, we will discuss a number of issues related to our conceptual approach in general, as well as to our Konduit implementation in particular.

We have argued that we restrict the kind of data that can flow within a workflow to be only RDF. By only allowing one kind of data, we keep the model simple and elegant. However, in reality we will often want to deal with other kinds of data (text, URLs, images, etc). At the moment, we handle these cases through component parameters, but this solution often feels rather awkward. We plan to study further whether adding support for other types than RDF will justify the increase in complexity.

Currently we do not have support for control flow components (loops, boolean gates, etc). On the one hand, including such features would certainly make our approach much more versatile and powerful and may be an interesting line of development for the future.

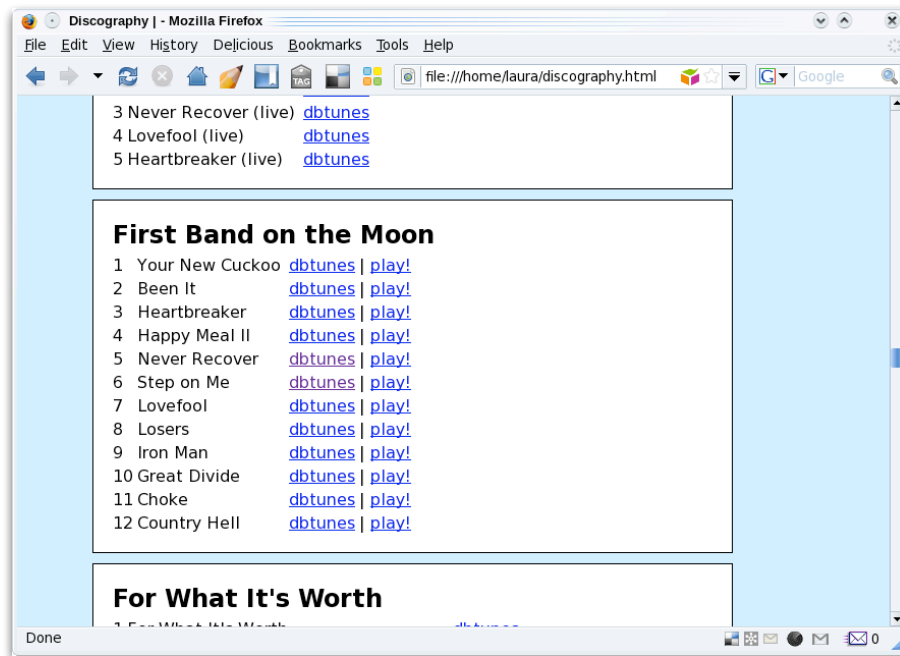


Fig. 6: The generated discography page for *The Cardigans*.

Some of the basic components available for Konduit require previous knowledge of writing SPARQL queries. Since the queries given as parameters to the source and filter elements can influence the performance of the entire workflow, we recognize the need for a smart query editor that is suitable for naive users. Our solution to support end users in creating queries is based on autocompletion, however, in order to make the system more accessible, we think it will be necessary to introduce a different kind of interface, which would abstract away from the actual syntax altogether and model the query on a higher level. Such an interface would possibly still be of a graphical nature, but without simply replicating the SPARQL syntax visually. Alternatively or additionally, a natural language interface would be promising direction for further research.

6 Conclusion

We have presented an approach for enabling casual, non-technical users to build simple applications and workflows from structured data. To simplify the building process, we have chosen a visual scripting approach, which is inspired by software such as Yahoo Pipes. We expect that users will benefit mostly from our approach if they operate in a Semantic Desktop-like environment, where they will have access to the data and functionality they are used to and have to work with on

a daily basis. However, our approach and implementation also enable users to integrate data and functionality from their desktops with data from the Web, thus representing a step towards the convergence of those two domains.

Acknowledgements

The work presented in this paper was supported (in part) by the L on project supported by Science Foundation Ireland under Grant No. SFI/02/CE1/I131 and (in part) by the European project NEPOMUK No FP6-027705.

References

1. S. Decker and M. R. Frank. The networked semantic desktop. In C. Bussler, S. Decker, D. Schwabe, and O. Pastor, editors, *WWW Workshop on Application Design, Development and Implementation Issues in the Semantic Web*, May 2004.
2. L. Dragan and K. M oller. Creating discographies with Konduit, 2009. <http://smile.deri.ie/konduit/discography>.
3. T. Groza, S. Handschuh, K. M oller, G. Grimnes, L. Sauer mann, E. Minack, C. Message, M. Jazayeri, G. Reif, and R. Gudjonsdottir. The NEPOMUK project — on the way to the social semantic desktop. In T. Pellegrini and S. Schaffert, editors, *Proceedings of I-Semantics' 07*, pages pp. 201–211. JUCS, 2007.
4. D. F. Huynh, R. C. Miller, and D. R. Karger. Potluck: Semi-ontology alignment for casual users. In K. Aberer, K.-S. Choi, N. Noy, D. Allemang, K.-I. Lee, L. Nixon, J. Golbeck, P. Mika, D. Maynard, R. Mizoguchi, G. Schreiber, and P. Cudr e-Maroux, editors, *6th International Semantic Web Conference and 2nd Asian Semantic Web Conference, ISWC+ASWC2007, Busan, Korea*, volume 4825 of *LNCS*, pages 903–910, Heidelberg, November 2007. Springer.
5. T. Menzies. Visual programming, knowledge engineering, and software engineering. In *Proc. 8th Int. Conf. Software Engineering and Knowledge Engineering, SEKE*. ACM Press, 1996.
6. C. Morbidoni, D. L. Phuoc, A. Polleres, and G. Tummarello. Previewing semantic web pipes. In S. Bechhofer, editor, *Proceedings of the 5th European Semantic Web Conference (ESWC2008), Tenerife, Spain*, volume 5021 of *LNCS*, pages 843–848. Springer, June 2008.
7. L. Orman. A multilevel design architecture for decision support systems. *SIGMIS Database*, 15(3):3–10, 1984.
8. J. K. Ousterhout. Scripting: Higher Level Programming for the 21st Century. In *IEEE Computer Magazine*, March 1998. <http://home.pacbell.net/ouster/scripting.html>.
9. B. Yan, M. R. Frank, P. Szekely, R. Neches, and J. Lopez. WebScripter: Grass-roots ontology alignment via end-user report creation. In D. Fensel, K. Sycara, and J. Mylopoulos, editors, *2nd International Semantic Web Conference, ISWC2003, Sanibel Island, FL, USA*, volume 2870 of *LNCS*, pages 676–689, Heidelberg, November 2003. Springer.

Ontology-Based Query Expansion Widget for Information Retrieval

Jouni Tuominen, Tomi Kauppinen, Kim Viljanen, and Eero Hyvönen

Semantic Computing Research Group (SeCo)
Helsinki University of Technology (TKK) and University of Helsinki
<http://www.seco.tkk.fi/>
firstname.lastname@tkk.fi

Abstract. In this paper we present an ontology-based query expansion widget which utilizes the ontologies published in the ONKI Ontology Service. The widget can be integrated into a web page, e.g. a search system of a museum catalogue, enhancing the page by providing a query expansion functionality. We have tested the system with general, domain-specific and spatio-temporal ontologies.

1 Introduction

In information retrieval systems the relevancy of search results depends on the user's ability to represent her information needs in a query [1]. If the vocabularies used by the user and the system are not the same ones, or if the shared vocabulary is used in different levels of specificity, the search results are usually poor. Query expansion has been proposed to solve these issues and to improve information retrieval by expanding the query with terms related to the original query terms. Query expansion can be based on corpus, e.g. analyzing co-occurrences of terms, or on knowledge models, such as thesauri [2] or ontologies [1]. Methods based on knowledge models are especially useful in cases of short, incomplete query expressions with few terms found in the search index [1, 2].

We have implemented a web widget providing query expansion functionality to web-based systems as an easily integrable service with no need to change the underlying system. The widget uses ontologies to expand the query terms with semantically related concepts. The widget extends the previously developed ONKI Selector widget, which is used for selecting concepts especially for annotation purposes [3].

The user does not have to be familiar with the ontologies used in content annotations by utilizing the autocompletion search feature of the widget, as the system suggests matching concepts as the user is writing the query string. Also, to help the user to disambiguate concepts the ONKI Ontology Browsers [4] can be used to get a better understanding of the semantics of the concepts, e.g. by providing a concept hierarchy visualization.

The query expansion widget supports Semantic web and legacy systems¹, i.e. either the concept URIs or the concept labels can be used in queries. In

¹ By legacy systems we mean systems that do not use URIs as identifiers.

legacy systems cross-language search can be performed, if the used ontology contains concept labels in several languages. In addition to the widget, the query expansion service can also be utilized via JavaScript and Web Service APIs. The query expansion widget and the APIs are available for public use as part of the ONKI Ontology Service² [4]. The JavaScript code needed for integrating the widget into a search system can be generated by using the ONKI Widget Generator³.

The contribution of this paper is to present an approach to perform query expansion in systems cost-effectively, not to evaluate how the chosen query expansion methods improve information retrieval in the systems.

2 Ontologies used for Query Expansion

The ONKI query expansion widget can be used with any ontology published in the ONKI Ontology Service. The service contains some 60 ontologies at the time of writing. Users are encouraged to submit their own ontologies to be published in the service by using the Your ONKI Service⁴. In the following, we describe how we have used different types of ontologies for query expansion.

2.1 Query Expansion with General and Domain-specific Ontologies

For expanding general and domain-specific concepts in queries we have used The Finnish Collaborative Holistic Ontology KOKO⁵ which consists of The Finnish General Upper Ontology YSO [5] and several domain-specific ontologies expanding it. To improve poor search results caused by using vocabularies in different levels of specificity in queries and in the search index we have used the transitive is-a relation (*rdfs:subClassOf*⁶) for expanding the query concepts with their subclasses. So for example, when selecting a query concept *publications*, the query is expanded with concepts *magazines*, *books*, *reports* and so on.

Using other relations in addition or instead of the is-a relation in query expansion might be beneficial. When considering general associative relations, caution should be exercised as their use in query expansion can lead to uncontrolled expansion of result sets, and thus to potential loss in precision [6, 7]. In case of a legacy system (not handling URIs, using labels instead) the use of alternative labels of concepts (synonyms) may improve the search. The relations used in the query expansion of an ontology can be configured when publishing the ontology in the ONKI Ontology Service.

² <http://www.yso.fi/>

³ <http://www.yso.fi/onkiselector/>

⁴ <http://www.yso.fi/upload/>

⁵ <http://www.seco.tkk.fi/ontologies/koko/>

⁶ Defined in the RDFS Recommendation, <http://www.w3.org/TR/rdf-schema/>

2.2 Query Expansion with the Spatio-temporal Ontology SAPO

A spatial query can explicitly contain spatial terms (e.g. Helsinki) and spatial relations (e.g. near), but implicitly it can include even more spatial terms that could be used in query expansion [8]. For example, in a query “museums near Helsinki” not only Helsinki is a relevant spatial term, but also its neighboring municipalities. Spatial terms – i.e. geographical places – do not exist just in space but also in time [9, 10]. This is especially true for museum collections where objects have references to places from different times. This sets a requirement to utilize also relations between historical places and more contemporary places in query expansion. To provide these mappings we used a spatio-temporal ontology SAPO (The Finnish Spatio-temporal Ontology) [11].

In SAPO regional overlap mappings are expressed as depicted in Figure 1, where example Turtle RDF⁷ statements⁸ express that the region of the latest temporal part of place *sapo:Joensuu* — i.e. the one valid from the beginning of year 2009 — overlaps the region of the temporal part of *sapo:Eno* of years 1871–2008. The temporal part of the place simply means the place during a certain time-period such that different temporal parts might have different extensions (i.e. borders) [11].

```
sapo:Joensuu(2009-)
  sapo:begin
    "2009-01-01" ;
  sapo:overlaps
    sapo:Eno(1871-2008) ,
    sapo:Pyhaselka(1925-2008) ,
    sapo:Joensuu(2005-2008) .

sapo:Joensuu
  sapo:unionof
    sapo:Joensuu(1848-1953) ,
    sapo:Joensuu(1954-2004) ,
    sapo:Joensuu(2005-2008) ,
    sapo:Joensuu(2009-) ;
  sapo:overlapsAtSomeTime
    sapo:Eno ,
    sapo:Pyhaselka ,
    sapo:Tuurpovaara ,
    sapo:Pielisensuu ,
    sapo:Kiihtelysvaara .
```

Fig. 1. Overlap mappings between temporal parts of places.

Fig. 2. A place is a union of its temporal parts. Moreover, places may have overlapped other places *at some time*.

For example, the place *sapo:Joensuu* is a union of four temporal parts, defined in the example depicted in Figure 2. However, annotations of items likely utilize places rather than their temporal parts. For this reason the model uses property *sapo:overlapsAtSomeTime* to explicate that e.g. a place *sapo:Joensuu* has — at some point in the history — overlapped together five different places (*sapo:Eno* and four others). In other words, e.g. at least one temporal part of *sapo:Joensuu* has overlapped at least one temporal part of *sapo:Eno*. We have used this more generic property *sapo:overlapsAtSomeTime* between places for query expansion.

⁷ <http://www.dajobe.org/2004/01/turtle/>

⁸ The example uses the following prefix - *sapo*: <http://www.yso.fi/onto/sapo/>

3 A Use Case of the Query Expansion Widget

We have created a demonstration search interface⁹ consisting of the original Kantapuu.fi search form¹⁰ and integrated ONKI widgets for query expansion. Kantapuu.fi is a web user interface for browsing and searching for collections of Finnish museums of forestry, using simple matching algorithm of free text query terms with the item index terms. The ontologies used in the query expansion are the same ones as used in annotation of the items¹¹, namely The Finnish General Upper Ontology YSO, Ontology for Museum Domain MAO¹² and Agiforest Ontology AFO¹³. For expanding geographical places the Finnish Spatio-temporal Ontology SAPO is used.

When a desired query concept is selected from the results of the autocompletion search of the widget or by using the ONKI Ontology Browser, the concept is expanded. The resulting query expression is the disjunction of the original query concept and the concepts expanding it, formed using the Boolean operation OR. The query expression is placed into a hidden input field, which is sent to the original Kantapuu.fi search page when the HTML form is submitted.

An example query is depicted in Figure 3, where the user is interested in old publications from place Joensuu. User has used the autocompletion feature of the widget to input to the *keywords* field a query term “publicat”, which has been autocompleted to the concept *publications*, which has been further expanded to its subclasses (their Finnish labels). Similarly, the place *Joensuu* has been added to the field *place of usage* and expanded with the places it overlaps.

The result set of the search contains four items, from which two are magazines used in place Eno and the rest two are cabinets for books used in place Joensuu. Without using the query expansion the result set would have been empty, as the place *Eno* and the concept *books* were not in the original query.

4 Discussion

When implementing the demonstration search interface for the Kantapuu.fi system with ONKI widgets we faced some challenges. If a query concept has lots of subconcepts, the expanded query string may become inconveniently long, as the concept URIs/labels of the subconcepts are added to the query. This may cause problems because the used HTTP server, database system or other software components may set limits to the length of the query string. With lengthy queries the system may not function properly or the response times of the system may increase.

⁹ <http://www.yso.fi/kantapuu-qe/>

¹⁰ <http://www.kantapuu.fi/>, follow the navigation link “Kuvahaku”.

¹¹ To be precise, the ontologies are based on thesauri that have been used in annotation of the items.

¹² <http://www.seco.tkk.fi/ontologies/mao/>

¹³ <http://www.seco.tkk.fi/ontologies/afo/>

1. Search query

Concept "publications" expanded to Finnish equivalents of: "magazines", "books" etc.

Place "Joensuu" expanded to overlapping places: "Eno", "Tuupovaara" etc.

Perform the search.

Full text search:

Keywords: publications X QE: books, reports, ...

publicat en Open ONKI Browser

Place of use: Joensuu X

joens en Open ONKI Browser

Materials: en Open ONKI Browser

Time of use:

Name:

Museum: All

Show results: as text as thumbnails

Search

2. Search results

Items annotated with: Keywords - "books" Place of use - "Joensuu"

Items annotated with: Keywords - "magazines" Place of use - "Eno"

E93108:120
Nimi: Siirtokirjasto
Erityisnimi: Kämppekirjasto
Asiasanat: kirjastot, erikoiskirjastot, työpaikkakirjastot, kirjat, kaapit, metsättyö, kämppät, ajankäyttö, vapaa-aika, vapaa-ajantoinnit, harrastukset, lukeminen, lukutaito

E93108:119
Nimi: Siirtokirjasto
Erityisnimi: Kämppekirjasto
Asiasanat: kirjastot, erikoiskirjastot, työpaikkakirjastot, kirjat, kaapit, metsättyö, kämppät, ajankäyttö, vapaa-aika, vapaa-ajantoinnit, harrastukset, lukeminen, lukutaito

IM56:9
Nimi: Aikakauslehti
Erityisnimi: Sinimusta
Asiasanat: julkaisu, aikakauslehdet, fascismi, fasismi, marxilaisuus, politiikka, Lapuanliike, Sinimusta, viikkolehdistö

IM56:7
Nimi: Aikakauslehti
Erityisnimi: Sinimusta
Asiasanat: julkaisu, aikakauslehdet, fascismi, fasismi, marxilaisuus, politiikka, Lapuanliike

Fig. 3. Kantapuu.fi system with integrated ONKI widgets.

Future work includes user testing for finding out if users consider the query expansion of the concepts and places useful. Also, systematic evaluation of the search systems used would be essential to find out if the query expansion improves the information retrieval, and specifically which semantic relations improve the results the most. The user interface of the query expansion widget needs further developing, e.g., the user should be able to select/unselect the suggested query expansion concepts.

Acknowledgements

We thank Ville Komulainen for his work on the original ONKI server and Leena Paaskoski and Leila Issakainen for cooperation on integrating the ONKI query expansion widgets into the Kantapuu.fi system. This work has been partially

funded by Lusto The Finnish Forest Museum¹⁴ and partially by the IST funded EU project SMARTMUSEUM¹⁵ (FP7-216923). The work is a part of the National Semantic Web Ontology project in Finland¹⁶ (FinnONTO) and its follow-up project Semantic Web 2.0¹⁷ (FinnONTO 2.0, 2008-2010), funded mainly by the National Technology and Innovation Agency (Tekes) and a consortium of 38 private, public and non-governmental organizations.

References

1. Voorhees, E.M.: Query expansion using lexical-semantic relations. In: Proceedings of the 17th annual international ACM SIGIR conference on Research and development in information retrieval, Dublin, Ireland (July 3-6 1994) 61–69
2. Wang, Y.C., Vandendorpe, J., Evens, M.: Relational thesauri in information retrieval. *Journal of the American Society for Information Science* **36**(1) (1985) 15–27
3. Viljanen, K., Tuominen, J., Hyvönen, E.: Publishing and using ontologies as mash-up services. In: Proceedings of the 4th Workshop on Scripting for the Semantic Web (SFSW 2008), 5th European Semantic Web Conference 2008 (ESWC 2008), Tenerife, Spain (June 1-5 2008)
4. Viljanen, K., Tuominen, J., Hyvönen, E.: Ontology libraries for production use: The Finnish ontology library service ONKI. In: Proceedings of the 6th European Semantic Web Conference (ESWC 2009). (May 31 - June 4 2009)
5. Hyvönen, E., Viljanen, K., Tuominen, J., Seppälä, K.: Building a national semantic web ontology and ontology service infrastructure—the FinnONTO approach. In: Proceedings of the 5th European Semantic Web Conference (ESWC 2008). (June 1-5 2008)
6. Tudhope, D., Alani, H., Jones, C.: Augmenting thesaurus relationships: Possibilities for retrieval. *Journal of Digital Information* **1**(8) (2001)
7. Hollink, L., Schreiber, G., Wielinga, B.: Patterns of semantic relations to improve image content search. *Journal of Web Semantics* **5**(3) (2007) 195–203
8. Fu, G., Jones, C.B., Abdelmoty, A.I.: Ontology-based spatial query expansion in information retrieval. In: In Lecture Notes in Computer Science, Volume 3761, On the Move to Meaningful Internet Systems: ODBASE 2005. (2005) 1466–1482
9. Kauppinen, T., Hyvönen, E.: Modeling and reasoning about changes in ontology time series. In Kishore, R., Ramesh, R., Sharman, R., eds.: *Ontologies: A Handbook of Principles, Concepts and Applications in Information Systems*. Integrated Series in Information Systems, New York, NY, Springer-Verlag, New York (NY) (January 15 2007) 319–338
10. Jones, C., Abdelmoty, A., Fu, G.: Maintaining ontologies for geographical information retrieval on the web. Volume 2888., Sicily, Italy, Springer Verlag (November 2003) 934–951
11. Kauppinen, T., Väättäinen, J., Hyvönen, E.: Creating and using geospatial ontology time series in a semantic cultural heritage portal. In: S. Bechhofer et al.(Eds.): Proceedings of the 5th European Semantic Web Conference 2008 ESWC 2008, LNCS 5021, Tenerife, Spain. (June 1-5 2008) 110–123

¹⁴ <http://www.lusto.fi>

¹⁵ <http://smartmuseum.eu/>

¹⁶ <http://www.seco.tkk.fi/projects/finnonto/>

¹⁷ <http://www.seco.tkk.fi/projects/sw20/>

Krextor – An Extensible XML→RDF Extraction Framework

Christoph Lange

Computer Science, Jacobs University Bremen, ch.lange@jacobs-university.de

Abstract. The semantics of an XML-based language can be specified by mapping an XML schema to an ontology, thus enabling the wide range of XML applications to contribute to the Semantic Web. The Krextor XML→RDF extraction framework proposes a practical solution to this problem, novel in its extensibility. We present its architecture and show how it can easily be extended to support additional input and output languages.

1 Introduction: Semantic Markup and RDF

In the well-known and hotly debated layer cake architecture of the Semantic Web (see [9] for a survey of its different incarnations), XML has always been placed below RDF. This must not be misunderstood in a way that XML should only be used for encoding higher-layer formalisms like RDF or OWL in a standardized way (e. g. RDF/XML [5]). Other XML-based languages can also be given a semantics, and by that, we are not just considering XML’s inherent semantics of a tree structure, identifiers, and cross-references. Semantic XML languages are widely used for domain-specific and general-purpose applications. The key difference of an XML document compared to an RDF graph is its *sequential order* – suited for the way humans *read*. Style languages like CSS and XSL further help presenting XML to non-technical readers. Schema languages allow for defining the syntax of new XML languages for any domain. A well-engineered XML schema¹ can lead to a much more concise and intuitive knowledge representation than an RDF graph, as it does not force authors to break all statements down to triples; compare the direct XML serialisation of OWL [18] to its RDF/XML serialisation. Given adequate editors, both authoring XML and RDF can be made easy for domain experts – but our own experience in the domain of mathematics shows that many more domain experts are familiar with domain-specific markup languages than with Semantic Web standards.

The semantics of XML languages is usually defined in a human-readable specification and hard-coded into applications. Therefore, XML has often been blamed for not being sufficiently semantic. We argue against that and bridge the semantic gap by improving the extraction of RDF from XML. XML languages

¹ The lowercase term “schema” is used in the general sense here, not referring to the particular schema language XML Schema.

having a well-defined *formal semantics* in terms of RDF already exist. Obvious examples are XML representations for RDF and ontologies themselves. Then, there is RDFa [1] for embedding RDF into the widely-supported but non-semantic XHTML. Less formal alternatives, such as microformats, can also be considered semantic if an RDF semantics has been specified for them and there is a way of obtaining RDF from such documents, e.g. by a GRDDL link from the document to the implementation of a translator to RDF. For data-centric XML languages, e.g. XML representations of relational databases, it is also straightforward to specify an RDF semantics. Finally, there are *semantic markup languages* – XML languages with a formal semantics that have explicitly been designed for knowledge representation. Consider, for example, OMDoc (Open Mathematical Documents), which is developed in our group [12]. While the formal core of OMDoc (symbol declarations, modular theories, proof objects) has a model- and proof-theoretic semantics that is much more expressive than Semantic Web ontologies (see [20]), we have specified an OWL-DL semantics for large parts of OMDoc’s semi-formal structural aspects (document structure, mathematical statements, structured proofs; see [13]).

A widened connection between the XML and RDF layers of the layer cake has most notably been suggested by Patel-Schneider and Siméon, who developed a unified model theory for both [19]. However, the benefit of that approach is rather theoretical, as it makes impractically restrictive assumptions about the XML structure (see [17] for details). Moreover, XML and RDF have evolved in parallel for years and gained a wider tool support each. Therefore, we take the more practical approach of extracting RDF from XML on the level of syntax and thus giving XML languages an RDF semantics by providing 1. rules that translate XML to RDF and, if needed, 2. an ontology providing the vocabulary for the extracted RDF.

2 The Krextor XML→RDF Extraction Framework

The Krextor XML→RDF extraction framework originated from the need to manage OMDoc documents in a Semantic Web application [14,13]. Having modeled an OWL-DL ontology for OMDoc, an OMDoc→RDF extraction was needed, which we hard-coded in XSLT from scratch, after an older, hard-coded Java implementation had proven to be too unflexible to maintain. The RDF was output in the RXR notation (Regular XML RDF [4]), from which it was parsed by a Java library. Later, the same was required for OpenMath content dictionaries, a language similar to OMDoc. This led to the decision to create a generic XSLT-based framework (cf. fig. 1) that allows developers to define translations (“extraction modules”) from any XML language to RDF more easily than in pure XSLT, as will be shown in the following.

A generic module provides convenience templates and functions for defining extraction rules in a way that abstracts from the concrete output format and instead defining the semantics of XML structures on a high level, in terms of resources and properties. Krextor’s generic “representation” of XML is a transient

one; the generic module is just a step in the pipeline, grouping extracted data into triples and forwarding them to the selected output module. Supported output formats, besides RXR, are: RDF/XML [5], the text notation Turtle [6], and, thanks to the Saxon XSLT processor [11], a direct interface to Java, for a more efficient integration into applications. In RDF/XML and Turtle output, the triples are grouped by common subjects and predicates. This is achieved by first obtaining RXR and then transforming it to the notation desired using XSLT grouping – a compromise between efficiency and a clean separation of concerns. Syntactic sugar, offered by some RDF notations, has only partly been implemented. At the moment, there is no support for author-defined namespace prefixes, “anonymous” blank nodes (bnodes) without identifiers, and RDF containers or collections in the output. Semantically, that does not make a difference. After all, our target “audience” are not humans, who usually do not want to read raw RDF, but applications that further process the RDF and conveniently prepare it for users – as, e. g., the semantic wiki SWiM does [13]. Nevertheless, some syntactic sugar remains on our agenda, as it facilitates testing Krextor during development.

Krextor is available as a collection of XSLT style sheets, with an optional Java wrapper for direct integration into applications. For scripting and debugging, there is a shell script frontend, reading XML from the standard input and writing RDF in the desired notation to the standard output.

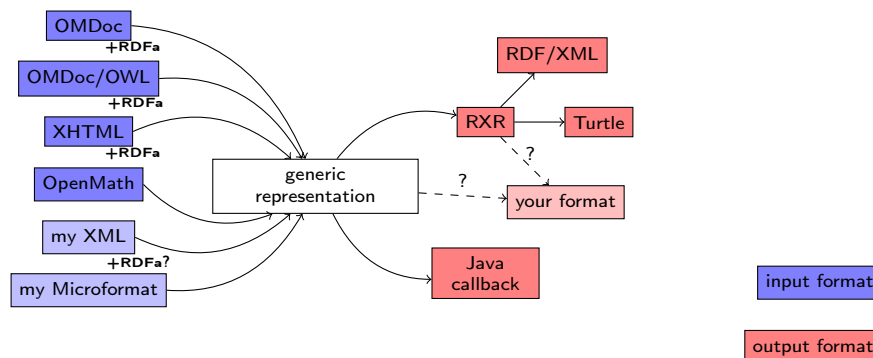


Fig. 1. Krextor’s extraction process and modules

Besides the input formats mentioned so far, Krextor also supports RDFa – embedded in XHTML or other host languages, such as Open Document², and in the following section, we will show how it can be extended to microformats. Moreover, we are working on a translation from OMDoc to OWL, implemented as a Krextor extraction module, which allows for authoring Semantic Web ontologies with integrated documentation and in a more modular way (see [16] for details).

² See <http://rdfa.info/2008/03/13/rdfa-support-coming-in-odf-12/> and stay tuned for Open Document 1.2 ©

Thus, Krextor can even be used as a bridge from the XML layer into the ontology layer of the Semantic Web cake. To add an input format, one has to provide XSLT templates that map XML structures of the input to calls of Krextor’s generic templates, as shown in the following section. We follow the paradigm of making easy things easy and hard things possible – an inexpensive claim, actually, but considerable efforts have been made to implement convenience templates and functions for common extraction tasks, which are easier to use than plain XSLT. There are predefined templates for creating a resource that is instance of some class and for adding literal- or URI-valued properties to the current resource. Several ways of generating resource URIs are provided, including fragment URIs of the form “document’s URI” # “fragment’s xml:id”, but an extraction module can also implement its own URI generator(s). (The latter has been done for OMDoc, which uses a *document/theory/symbol* URI pattern [16].) The information that the developer has to provide explicitly is kept at a minimum level: When no subject URI for a resource is given, Krextor auto-generates one using the desired generation function. When no object value for a property is given, it is taken from the currently processed XML attribute or element. As an alternative for very simple formats, where XML elements directly map to ontology classes and properties, a declarative mapping can be given as annotated literal XML. For complex input formats like the above-mentioned OMDoc, the full computational power of XSLT can be used, at the expense of readability. A new output module merely has to implement one template for low-level RDF generation, accepting the parameters subject (URI or bnode ID), subject type (URI or bnode), predicate (URI), object, object type (URI, bnode ID, or literal), language, and datatype. More complex output modules can be realized by post-processing output from existing output modules.

3 Use Cases and Applications

One application area of Krextor is the semantic wiki SWiM [13]. Mathematical documents can be imported and edited in their original formats, which allows for building on existing tool support. An RDF outline is only extracted from them after storing them in the database; the RDF plus the background knowledge from the ontologies then powers semantic services – currently navigation, querying, and problem-solving assistance [13,15]. OMDoc particularly needs complex extraction rules: Its mathematical symbols have their own URI schema, and it can mix formal and informal knowledge. The RDF graph extracted from a full-featured OMDoc document consists of two parallel trees, one tree of the mathematical structure, and one of the rhetorical structure, interwoven via an annotation ontology. Despite this complexity, 21 out of the 44 templates in the extraction module for OMDoc have completely been implemented using Krextor’s convenience templates only. 15 make use of additional XPath constructs, 5 use additional, more complex XSLT constructs, and 3 use both. OMDoc as a frontend for OWL ontologies, as mentioned above and detailed in [16], will eventually be integrated into SWiM. The extraction of OWL from special OMDoc documents has also

been implemented using Krextor. In these documents, ontologies are modeled as mathematical theories, resources are declared as symbols having definitions, axioms, and theorems. Many of these mathematical statements are modeled in a way that is more familiar to people with a logics background: the range and domain of a property is, e. g., represented by a single relation type declared for the property symbol [16]. The OMDoc→OWL module makes considerably more use of XPath and XSLT than the above-mentioned module that obtains the structure of OMDoc documents as RDF, but still it paid off to implement it within Krextor, as part of the required functionality could be shared with the former module.

We exemplify Krextor’s extensibility, a major design goal, by an extraction module for a simple language, the hCalendar microformat [7], using the RDF Calendar vocabulary [8]. The extraction rules for an event and its start date are given in listing 2, which is considerably shorter than an equivalent implementation in plain XSLT. The first template matches any element of class “vevent” and creates an instance of the *ical:Vevent* class from it. When a child link annotated as the URI of the event is present, its target is used to identify the event; otherwise, a bnode is created for the event. The second template matches any element of class “dtstart” and adds an *ical:dtstart* property of datatype *xsd:date* to the current resource. Krextor’s convenience templates automatically take care of recursing to child elements, keeping track of the current resource, and reading the values of properties if they are given in a reasonable place, such as the text content of an element. Given the following sample input, Turtle output can be

```
<stylesheet version="2.0">
  <!-- we generate resource URIs ourselves -->
  <param name="autogenerate-fragment-uris" select="()"/>
  <template match="*[@class='vevent']">
    <!-- Take URL property if given, otherwise create a bnode -->
    <variable name="subject" select="a[@class='url']/@href"/>
    <call-template name="krextor:create-resource">
      <with-param name="subject" select="$subject"/>
      <with-param name="blank-node" select="not($subject)"/>
      <with-param name="type" select="'&ical;Vevent'"/>
    </call-template></template>
  <template match="*[@class='dtstart']">
    <call-template name="krextor:add-literal-property">
      <with-param name="property" select="'&ical;dtstart'"/>
      <with-param name="datatype" select="'&xsd;date'"/>
    </call-template></template> <!-- ... and so on ... -->
```

Fig. 2. A hCalendar extraction module

obtained e.g. by calling `krextor hcalendar..turtle infile.xhtml` on the command line:

```
<div class="vevent">
  <a class="url" href="http://www.eswc2009.org">ESWC</a>
  starts on <span class="dtstart">2009-05-31</span>.</div>
```

```
<http://www.eswc2009.org>
  a <http://www.w3.org/2002/12/cal/ical#Vevent> ;
  <http://www.w3.org/2002/12/cal/ical#dtstart>
    "2009-05-31"^^<http://www.w3.org/2001/XMLSchema#date> .
```

4 Related Work and Conclusion

Swignition's [10] architecture is very similar to **Krextor**'s. For end-users and web developers, it offers much richer features, including support for many microformats, other legacy ways of embedding RDF into HTML, and GRDDL links. For knowledge engineers or developers who quickly want to define an RDF translation from a new XML language, **Krextor** performs better, being extensible by additional input formats with much less lines of code than the **Swignition** Perl library. So far, GRDDL is only "supported" by **Krextor** in the obvious sense that it facilitates the XSLT-based implementation of an XML→RDF translation that can then be linked to a schema or to documents using GRDDL; automatically choosing the right extraction module by interpreting GRDDL annotations in the input document is not yet supported. Both systems approach integration into semantic applications differently: **Swignition** comes with a TCP/IP interface, whereas **Krextor** features a Java API and benefits from the wide support for XSLT. The authors of **XSDL** [17] have done substantial theoretical elaboration on a semantics-preserving translation of XML into RDF and provide a concise declarative syntax mapping XML to OWL-DL. To the best of our knowledge, **XSDL** has not been implemented, though. As its syntax uses XML and XPath, we consider it feasible to prove the theoretical results the authors have obtained for **Krextor** as well by rewriting **XSDL** definitions into equivalent **Krextor** extraction modules. This would also make **XSDL** usable as a convenient input language for **Krextor**, making extraction modules look less like XSLT and XPath. **XSPARQL** [2] mixes SPARQL into XQuery, resulting in a query language that fully breaks the boundaries between XML and RDF. It avoids the necessity of first converting from one representation into the other. However, persistently storing the result of such a conversion is often desired in applications, whereas the current implementation of **XSPARQL** focuses on one-time queries.

Conclusion: The **Krextor** framework supports many XML→RDF conversion tasks and can easily be extended by additional input and output formats and integrated into semantic applications. Thereby, we have opened new paths from the XML layer of the Semantic Web architecture to the RDF and higher layers. When designing new ontologies, knowledge engineers can now take the creative challenge of developing a convenient XML syntax for domain-specific knowledge and provide a **Krextor** extraction module that translates this XML to RDF in terms of these ontologies. We will continue using **Krextor** for mathematical markup but are also interested in proving its extensibility on other semantic markup languages. A future research direction that we want to explore is adding

extraction rules as annotations to XML schema languages like RELAX NG [21], thereby unifying two tasks that belong together but have been separated so far: specifying the syntax and the semantics of an XML language.

Acknowledgments: The author would like to thank Richard Cyganiak for providing him with a Java-based RDFa test suite, and several reviewers for constructive feedback.

References

1. B. Adida, M. Birbeck, S. McCarron, and S. Pemberton. RDFa in XHTML: Syntax and processing. Recommendation, W3C, 2008.
2. W. Akhtar, J. Kopecký, T. Krennwallner, and A. Polleres. XSPARQL: Traveling between the XML and RDF worlds – and avoiding the XSLT pilgrimage. In Bechhofer et al. [3].
3. S. Bechhofer, M. Hauswirth, J. Hoffmann, and M. Koubarakis, editors. *5th European Semantic Web Conference*, volume 5021 of *LNCIS*. Springer, 2008.
4. D. Beckett. Modernising semantic web markup. In *XML Europe*, 2004.
5. D. Beckett. RDF/XML syntax specification. Recommendation, W3C, 2004.
6. D. Beckett. Turtle – terse RDF triple language, 2007.
7. T. Çelik. hCalendar. Microformat specification, Technorati, 2008.
8. D. Connolly and L. Miller. RDF calendar. Interest Group Note, W3C, 2005.
9. A. Gerber, A. v. Merwe, and A. Barnard. A functional semantic web architecture. In Bechhofer et al. [3].
10. T. A. Inkster. Swignition. <http://buzzword.org.uk/swignition/>, 2009.
11. M. Kay. Saxonica: XSLT and XQuery processing. <http://www.saxonica.com>.
12. M. Kohlhase. OMDoc – An open markup format for mathematical documents. Number 4180 in *LNAI*. Springer, 2006.
13. C. Lange. SWiM – a semantic wiki for math. knowledge. In Bechhofer et al. [3].
14. C. Lange. Krextor. <http://kwarc.info/projects/krextor/>, 2009.
15. C. Lange, T. Hastrup, and S. Corlosquet. Arguing on issues with mathematical knowledge items in a semantic wiki. In J. Baumeister and M. Atzmüller, editors, *LWA (Lernen, Wissensentdeckung und Adaptivität)*, volume 448, 2008.
16. C. Lange and M. Kohlhase. A mathematical approach to ontology authoring and documentation. In *Mathematical Knowledge Management*, *LNAI*. Springer, 2009. <https://svn.omdoc.org/repos/omdoc/trunk/doc/blue/foaf/mkm09.pdf>.
17. S. Liu, J. Mei, A. Yue, and Z. Lin. XSDL: Making XML semantics explicit. In C. Bussler, V. Tannen, and I. Fundulaki, editors, *SWDB*, volume 3372, 2004.
18. B. Motik and P. F. Patel-Schneider. OWL web ontology language: XML serialization. Working draft, W3C, Dec. 2008.
19. P. F. Patel-Schneider and J. Siméon. The Yin/Yang web: A unified model for XML syntax and RDF semantics. *IEEE TKDE*, 15(4), 2003.
20. F. Rabe. *Representing Logics and Logic Translations*. PhD thesis, Jacobs University Bremen, 2008.
21. RELAX NG. <http://www.relaxng.org/>.

RDFa in Drupal: Bringing Cheese to the Web of Data

Stéphane Corlosquet, Richard Cyganiak, Axel Polleres and Stefan Decker

Digital Enterprise Research Institute
National University of Ireland, Galway
Galway, Ireland
`{firstname.surname}@deri.org`

Abstract. A large number of websites are driven by content management systems (CMS), which manage not only textual content but also structured data related to the site’s topic. Exposing this information to the Web of Data has so far required considerable expertise in RDF modelling and programming. We present a plugin for the popular CMS Drupal that enables high-quality RDF output with minimal effort from site administrators. This has the potential of greatly increasing the amount and topical range of information available on the Web of Data.

1 Introduction

Semantic Web technologies have matured to the point where they are increasingly being deployed on the Web. Large amounts of RDF data can now be accessed over the Web as *Linked Data*. This data is used by a variety of clients, such as RDF data mashups that integrate information from various sources, search engines that allow structured queries across multiple sites and datasets, and data browsers that present a site’s content in new ways.

But the traditional Web still dwarfs this emerging Web of Data. Thus, the task of “RDFizing” existing websites, which contain structured information such as events, personal profiles, ratings, tags, and locations, is important and of high potential benefit. The recently finished RDFa[1] standard supports this by allowing RDF to be embedded into existing HTML pages.

The Web features some huge websites with millions of pages and users. But a lot of the Web’s interest and richness is in the “long tail”, in smaller special-interest websites, such as cheese reviews, which will be our example for this demonstration. Our goal is to make domain data from such sites available as RDF. This is challenging for several reasons:

No dedicated development staff. Smaller websites usually run off-the-shelf software, such as CMS, wikis, or forums. Site operators cannot build the RDF support themselves, it has to be provided by the software or plugins.

Per-site schemas. The domain schema differs from site to site. The mapping from the site’s schema to RDF vocabularies or ontologies cannot be pre-defined by a software developer; it must be defined by the site operator.

No ontologists. Site administrators will have little interest in learning the details of RDF and description logics. The process of configuring RDF support has to be simple and straightforward, or else it won't be used.

We show a practical and easy-to-use system that overcomes these challenges and allows the publication of high-quality RDF data from websites that run on off-the-shelf content management systems. It targets the popular Drupal CMS.

2 Drupal and the Content Construction Kit

We will start by briefly introducing Drupal¹ and some of its terminology. Drupal is a popular open-source content management system (CMS). It is among the top three open-source CMS products in terms of market share[5]. Drupal facilitates the creation of websites by handling many aspects of site maintenance, such as data workflow, access control, user accounts, and the encoding and storage of data in the database.

A *site administrator* initially sets up the site by installing the core Drupal Web Application and choosing from a large collection of *modules* that add specific functionality to the site. Site administrators need a fair bit of technical knowledge to choose and configure modules, but usually do not write code; this is done by *module developers* instead. After the site has been set up, Drupal allows *non-technical users* to add content and handle routine maintenance of the site.

Each item of content in Drupal is called a *node*. Nodes usually correspond to the pages of a site. Nodes can be created, edited and deleted by content authors. Some modules extend the nodes, for example a comment module adds blog-style comment boxes to each node.

Another example is the *Content Construction Kit (CCK)*, one of the most popular modules used on Drupal sites. It allows the site administrator to define types of nodes, called *content types*, and to define *fields* for each content type. Fields can be of different kinds such as plain text fields, dates, email addresses, file uploads, or references to other nodes. When defining content types and fields, the site administrator provides the following information:

- label, ID, and description for content types and fields,
- fields can be optional or required,
- fields can have a maximum cardinality,
- fields that reference other nodes can be restricted to nodes of a certain type.

For example, for a cheese review website, the site administrator might define content types such as *Cheese*, *Review*, and *Country of Origin*. The *Cheese* type might have fields such as *description*, *picture*, and *source of milk*.

Thus, site administrators use the CCK to define a site-specific content model, which is then used by content authors to populate the site. The focus of the work we are presenting here is to expose this CCK content as RDF on the Web.

¹ <http://drupal.org/>

3 Weaving Drupal into the Web of Data

Given a Drupal CCK content model consisting of content types, fields, and nodes that instantiate the types, what would be a good way of representing it in RDF? We consider the following features desirable for the RDF output which are in line with the Linked data principles and best practices [3, 4]:

Resolvable HTTP URIs for all resources, to take advantage of existing tools that can consume Linked Data style RDF content.

Re-use of published ontology terms. To support sites of arbitrary subject matter, we cannot in any way pre-define the RDF classes and properties that should be used to express the data. The site administrator has to select them when setting up the content model. But to enable queries across sites, it is necessary that the sites use the same (or mapped) vocabularies. This requires that both sites import vocabulary terms from somewhere else.

Expressing Drupal constraints in OWL. Constraints that are defined on the types and fields (domains, ranges, cardinalities, disjointness) should be automatically published as RDF Schema or OWL expressions.

Auto-generate terms where necessary. Re-use of published ontology terms is important for interoperability, but not always possible or practical, as there might be no existing ontology term matching a type or field, or finding them is too hard.

Safe vocabulary re-use. Mixing the content model constraints with constraints of a published ontology might have unintended semantic effects, especially since most site administrators will not be familiar with the details of OWL semantics. For example, a carelessly applied cardinality constraint could affect the definition of a shared vocabulary term, rendering data published elsewhere inconsistent. The system must prevent such effects as far as possible.

These features strike a balance between preserving as much information as possible from the original content model, keeping the barrier to entry low, and enabling interoperability between multiple data publishers and consumers.

3.1 Site Vocabularies for Basic RDF Output

When the module is first enabled, it defaults to auto-generating RDF classes and properties for all content types and fields. Thereby it provides zero-effort RDFa output for a Drupal site, as long as no mappings to well-known public vocabularies are required.

An RDFS/OWL site vocabulary document that describes the auto-generated terms is automatically generated. The definitions contain label, description, and constraints taken from the CCK type/field definitions. The HTML views of all nodes contain RDFa markup for the type and all shown fields, using the auto-generated classes and properties.

3.2 Mapping the Site Data Model to Existing Ontologies

To map the site data model to existing ontologies, the site administrator first imports the ontology or vocabulary. We assume that it has been created using a tool such as Protégé², OpenVocab³, or Neologism⁴, and published somewhere on the Web in RDF Schema or OWL format.

For every content type and field, the site administrator can choose a class or property it should be mapped to. Mappings are expressed as subclass/subproperty relationships. For instance, if the field `description` on type `cheese` is mapped to Dublin Core's `dc:description`, then a triple `site:cheese_Description rdfs:subPropertyOf dc:description` would be added to the site vocabulary.

This subclassing is a simple way of minimizing unintended conflicts between the semantics of local and public terms. Per OWL semantics, constraints imposed on the local term by the content model will not apply to the public term. This ensures *safe vocabulary re-use*[2].

It must be stressed that this mapping step is optional, and the main benefit of the Web of Data – exposing site data for re-use by third parties – is realized by the default mapping.

4 The User Experience

This section describes our cheese review site from a user point of view, and shows an example of what can be done to reuse the RDFa data.

Cheese and reviews. Figure 1 shows a cheese entry and its user review. Using the Content Construction Kit, we defined a type (1) `cheese` with fields for the name of the cheese, the source of the milk, the country of origin, a picture and a description and (2) `cheese_review` with fields for the title of the review, a reference to the `cheese` being reviewed and the review.



Fig. 1. A cheese with a review. Users can create new cheese entries and add reviews.

² <http://protege.stanford.edu/>

³ <http://open.vocab.org/>

⁴ <http://neologism.derii.ie/>

Content type and field mapping. The module adds a “Manage RDF mappings” page to the CCK interface as shown in Figure 2 (left). For a given content type, it offers to map the type to an RDF class and each field to an RDF property. We have mapped the Cheese type and its fields to previously imported RDF terms. In order to ease the mapping process and prevent confusion between classes and properties, the module will only display RDF classes or RDF properties where appropriate. Moreover an AJAX autocomplete search through the imported terms allows the user to quickly identify the most relevant terms for each mapping. These measures help to make the mapping process a fairly straightforward task that does not require deep understanding of the Semantic Web principles.

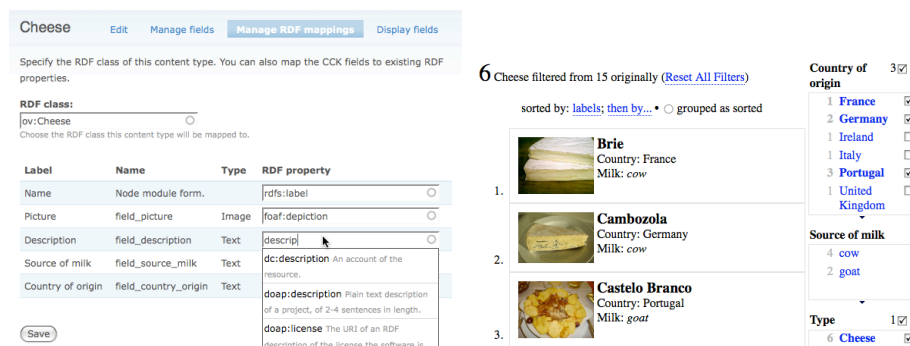


Fig. 2. RDF mappings management (left) and Exhibit view (right).

Exhibit view. Next we show a simple example of using the site’s RDFa data. Exhibit⁵ now supports RDFa natively and it is easy to setup a basic faceted browsing interface on top of an RDFa page. In our example, Exhibit allows filtering of the cheese types via several facets, as shown in Figure 2 (right).

5 Conclusion and Future Work

The presented system differs from existing approaches in several ways. *SIOC exporters* and similar fixed-schema RDF generators cannot deal with the case where the site schema and its mapping into RDF terms are defined by the site administrator at setup time. *Database-level RDF exporters*, such as Triplify⁶, require understanding of database technologies and of the application’s internal database schema. *Semantic MediaWiki*⁷ and similar systems address a different use case: all content authors, rather than just the site administrator, collaboratively define the structure of the site. We address the common case where the site’s basic structure should not be changed by individual content authors.

⁵ <http://simile.mit.edu/exhibit/>

⁶ <http://triplify.org/>

⁷ http://semantic-mediawiki.org/wiki/Semantic_MediaWiki

The presented system is a working prototype⁸. The module used for the prototype and discussed in this paper is available on drupal.org⁹. Further planned improvements include the use of the semantics of imported ontologies (e.g. domain, range and disjointness constraints) to restrict the selection of available classes and properties in the RDF mapping UI.

Another issue remains for the larger RDF community to solve: a complex part of the process of generating high-quality RDF is the task of finding and choosing good vocabularies and ontologies. There are some services that support this task, but they are not sufficient and this task remains difficult for non-experts. This is a major problem that the community needs to address in order for the Web of Data to succeed.

6 Acknowledgements

The work presented in this paper has been funded in part by Science Foundation Ireland under Grant No. SFI/08/CE/I1380 (Lion-2), the European FP6 project inContext (IST-034718) and the European Union under Grant No. 215032 (OKKAM).

References

1. B. Adida, M. Birbeck, S. McCarron, and S. Pemberton (eds.). Rdfa in xhtml: Syntax and processing, Oct. 2008. W3C Recommendation, available at <http://www.w3.org/TR/rdfa-syntax/>.
2. A. P. Aidan Hogan, Andreas Harth. Saor: Authoritative reasoning for the web. In *ASWC 2008*, pages 76–90, 2008.
3. D. Berrueta and J. P. (eds.). Best practice recipes for publishing rdf vocabularies, Aug. 2008. W3C Working Group Note, available at <http://www.w3.org/TR/swbp-vocab-pub/>.
4. C. Bizer, T. Heath, K. Idehen, and T. Berners-Lee, editors. *Linked Data on the Web (LDOW2008)*, Apr. 2008.
5. R. Shreves. Open source cms market share. White paper, Water & Stone. <http://waterandstone.com/downloads/2008OpenSourceCMSMarketSurvey.pdf>.

⁸ A demo site is online at <http://drupal.deri.ie/cheese/>.

⁹ <http://drupal.org/project/rdfcck>

Macros vs. scripting in VPOET *

Mariano Rico¹, David Camacho¹, Óscar Corcho²

¹ Computer Science Department, Universidad Autónoma de Madrid, Spain [†]
{[mariano.rico](mailto:mariano.rico@uam.es), [david.camacho](mailto:david.camacho@uam.es)}@uam.es

² Ontology Engineering Group, Departamento de Inteligencia Artificial,
Universidad Politécnica de Madrid, Spain
ocorcho@fi.upm.es

Abstract. We present our experience on the provision and use of macros for the management of semantic data in semantically-enabled web applications. Macros can be considered as a lightweight version of scripting languages, mostly oriented to end users instead of to developers. We have enabled the use of macros in a wiki-based application named VPOET, oriented to web designers, and have confirmed through evaluations that macros satisfy a wide audience.

1 Introduction

VPOET ³ [1] is a semantic web application aimed at web designers (i.e. client-side web specialists) without any knowledge in Semantic Web. It allows them to create web templates to display semantic data (output templates) or to request information from users that is then converted into semantic data (input templates).

These templates are shared and reused by the web designers community, and can be used by third parties easily. Any developer can request VPOET templates by means of simple HTTP messages (GET and POST), created in any programming language, sent to a specific VPOET servlet. The response to these messages is a piece of client code (HTML, CSS, Javascript). A typical request is “render the semantic data at URL Z by using the output template X created by designer Y”, which can be codified as a HTTP GET message by means of the following URL:

```
http://URL-to-servlet/VPoetRequestServlet?action=renderOutput  
&designID=X&provider=Y&source=Z.
```

An additional argument could specify a specific individual in the data source. In this case, only the individual is rendered. This is a real example using semantic data provided by W3C:

*We would like to thank Roberto García for his collaboration concerning Rhizomic.

[†]This work has been partially funded by the Spanish Ministry of Science and Innovation under the project HADA (TIN2007-64718), METEORIC (TIN2008-02081), and DEDICON (TIC-4425)

³See <http://ishtar.ii.uam.es/fortunata/Wiki.jsp?page=VPOET>

```
http://ishtar.ii.uam.es/fortunata/servlet/VPoetRequestServlet?
action=renderOutput&
designID=FOAFOutputConditionalGraphics&
provider=mra68&
source=http://www.w3.org/People/Berners-Lee/card&
indvID=http://www.w3.org/People/Berners-Lee/card#i
```

Unlike other templates systems based on scripting languages (shown in next section), VPOET provides web designers with simple macros to embed in the template source code. A 20 min. tutorial ⁴ is enough to start creating templates. The experimental evaluation [2] shows that web designers, in a wide skills range, from amateur to professionals, are satisfied with these macros. We argue that macros is a well known concept for web designers, while the current systems are oriented to developers.

2 Scripting Templates. A Comparative Use Case

This section describes briefly some representative semantic web applications that handle templates to present semantic data, such as semantic wikis, semantic browsers, and semantic portals. We have selected one representative application or infrastructure from each group: Semantic Media Wiki, Fresnel (used by the Longwell browser), and Rhizomer (used by the Rhizomik portal) respectively.

Semantic Media Wiki allows users create templates employing an *ad hoc* syntax with parsing functions ⁵. The left part of table 2 shows the source code of a template ⁶, and the right part of the figure shows the renderization of semantic data by using this template. The creator of a template must know the wiki syntax, basics of programming, and basics of ontology components.

Fresnel [3] is a template infrastructure for semantic data, used by a faceted semantic web browser named Longwell. However, as table 3 (top) shows, the Fresnel syntax ⁷ requires skills in semantic web technologies that severely limit the number of designers available.

Rhizomer [4] is an infrastructure to browse and edit semantic data. The presentation of RDF data is achieved by means of Extensible Stylesheet Language Transformations (XSLT). As one can see in table 3 (middle), web designers require additional skills in XSLT programming.

Table 1 summarizes the features of these templates and the competencies required to the creators of templates for the semantic applications considered. The current state of the art is oriented to developers with competencies in both Semantic Web Technologies and Web Client technologies. Therefore, it does not provide web designers with authoring tools to create attractive and reusable templates for semantic data. There must be a balance between expressiveness, to address RDF features (e.g. multi-valued properties, or properties with no value) and complexity, in order to reduce the required level of competencies.

⁴See <http://ishtar.ii.uam.es/fortunata/Wiki.jsp?page=VPOETTutorial>

⁵A template source code and usage example can be found at http://en.wikipedia.org/wiki/Template:Infobox_Settlement

⁶Edit page <http://semanticweb.org/wiki/Template:Person> to see the template source code between `<includeonly>` and `</includeonly>`. Note: the code previous to this block shows an usage example.

⁷See <http://www.w3.org/2005/04/fresnel-info/manual/>

	SMW	Fresnel	Rhizomik	VPOET
Programming Language	Wiki syntax, Programming, HTML/CSS	OWL, Fresnel ont., CSS	XSLT, OWL/XML	HTML, CSS, Javascript, macros
Allows				
Template reusing	No	Yes	Yes	Yes
Conditional rendering	Yes	Yes	Yes	Yes
Images	No	No	Yes	Yes
Dynamics (Web 2.0)	No	No	Yes	Yes
Oriented to	Average programmers	Semantic Web developers + CSS	Semantic Web developers + XSLT	Web designers + Macros
Competencies				
Sem. Web Techs.	Low	High	Very high	Low
Web Client Techs.	Low	High	Very high	Low-Very high
Other	Wiki syntax	High	XSLT	
Pros	Medium requirements	Pure OWL	Generic solution	Low requirements
Cons	Error prone syntax	Too complex for a web designer	Too complex for a web designer	

Table 1. Features of some frameworks to create a template

```

| cellpadding="0" cellspacing="5" style="position:relative; margin: 0 0 0.5em 1em;
border-collapse: collapse; border: 1px solid #aaa; background: #fff; float: right;
clear: right; width: 20em"
| colspan="2" style="background: #86ba0c; color: white" |<span style="font-size: 80%;
float: right; ">{{#ask: [{{{FULLPAGENAME}}]}}
| format=vcard
| ?Name
| ?Affiliation=organization
| ?Email
| ?Foaf:phone=workphone
| ?Homepage
| searchlabel=vCard
}}</span> [{{Name:{{{Name|{{{PAGENAME}}}}}}]]
| -
| {{#ifeq:{{{Picture}}}}|{{Tablelongrow|Value=[[Image:{{{Picture
}}]]|150px|{{{Name|{{{PAGENAME}}}}}}|Color=white}}}}
| -
| {{#ifeq:{{{Email}}}}|{{Tablelongrow|Value={{Mailbox|{{{Email
}}}}|Color=#e4f8b6}}}}
| -
| {{#ifeq:{{{Affiliation}}}}|{{Tablerow|Label=Affiliation:|
Value=[member of::affiliation:{{{Affiliation}}]}}}}
| -
| {{#ifeq:{{{Homepage}}}}|{{Tablerow|Label=Homepage:|
Value=[Homepage:http://{{{Homepage}}}|{{{Homepage label
|{{{Homepage}}}}]}}}}
| -
| {{#ifeq:{{{Phone}}}}|{{Tablerow|Label=Phone:|
Value=[[foaf:phone:{{{Phone}}]}}}}
| -
|<!-- *** Events *** -->
| {{Tablelongrow|Align=Left|Font size=80%|
Value={{#ask: [{{has PC member::{{{PAGENAME}}}}]}}
| format=list
| sort=start date | order=desc
| sep=","
| intro=PC member of: _ }}|Color=#e4f8b6}}
| -
| {{Tablelongrow|Align=Left|Font size=80%|
Value={{#ask: [{{has OC member::{{{PAGENAME}}}}]}}
| format=list
| sort=start date | order=desc
| sep=","
| intro=OC member of: _ }}|Color=#e4f8b6}}
| -
| {{#ifeq:{{{FOAF}}}}|{{Tablerow|Label=See also:|
Value=[see also:{{{FOAF}}}|FOAF]]|Color=white}}
| }|Category:Person]]

```



Table 2. Template in Semantic Media Wiki. Left: template code. Right: rendering an example.

```

:foafGroup rdf:type fresnel:Group ;
fresnel:stylesheetLink <http://www.example.org/example.css> ;
fresnel:containerStyle "background-color: white;"
~fresnel:stylingInstructions ;


:foafPersonFormat rdf:type fresnel:Format ;
fresnel:classFormatDomain foaf:Person ;
fresnel:resourceStyle "background-color: gray;"
~fresnel:stylingInstructions ;
fresnel:group :foafGroup .

:nameFormat rdf:type fresnel:Format ;
fresnel:propertyFormatDomain foaf:name ;
fresnel:propertyStyle "border-top: solid black;"
~fresnel:stylingInstructions ;
fresnel:labelStyle "font-weight: bold;"
~fresnel:stylingInstructions ;
fresnel:group :foafGroup .

:urlFormat rdf:type fresnel:Format ;
fresnel:propertyFormatDomain foaf:homepage ;
fresnel:propertyFormatDomain foaf:mailbox ;
fresnel:value fresnel:externalLink ;
fresnel:propertyStyle "border-top: solid black;"
~fresnel:stylingInstructions ;
fresnel:labelStyle "font-weight: bold;"
~fresnel:stylingInstructions ;
fresnel:group :foafGroup .

:depictFormat rdf:type fresnel:Format ;
fresnel:propertyFormatDomain foaf:depiction ;
fresnel:label fresnel:none ;
fresnel:value fresnel:image ;
fresnel:propertyStyle "border-top: solid black;"
~fresnel:stylingInstructions ;
fresnel:group :foafGroup .


```



```

<?xml version="1.0"?>
<?xml-stylesheet href="/xslt/xsl" type="text/xsl" media="screen"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
<xsl:import href="rdf2html-functions.xsl"/>
<xsl:param name="language">en/<xsl:param>
<xsl:output media-type="text/xhtml" version="1.0" encoding="UTF-8"
indent="yes"/>
<xsl:strip-space elements="*/>
<xsl:template match="*/>
<xsl:apply-templates select="rdf:RDF"/>
</xsl:template>
<xsl:template match="rdf:RDF">
<html lang="en" xmlns="http://www.w3.org/1999/xhtml">
<meta http-equiv="Content-Type" content="text/html;
charset=UTF-8"/>
<title>Rhizomik - ReDeFer - RDF2HTML</title>
<link href="http://rhizomik.net/style/rhizomik.css" type="text
rel="stylesheet" />
<link href="http://rhizomik.net/style/rhizomer.css" type="text
rel="stylesheet" />
</head>
<body>
<!-- div id="browser">
<a href="javascript:history.back()">back</a> - go to... -
<a href="javascript:history.forward()">forward</a>
</div -->
<xsl:if test="count(child:*)=0">
<p>No data retrieved.</p>
</xsl:if>
<xsl:for-each select="child:*">
<xsl:sort select="@rdf:about" order="ascending"/>
<xsl:call-template name="rdfDescription"/>
</xsl:for-each>
... 224 lines more ...
</xsl:stylesheet>

```



```

<table border="0" cellpadding="0" cellspacing="0">
<tr>
<td></td>
<td background="OmemoBaseURL/attach/Mra68Graphics/xample_body_upper_pat.gif"></td>
<td>
</td>
</tr>
<tr>
<td background="OmemoBaseURL/attach/Mra68Graphics/
xample_body_left_patt.gif"></td>
<td border="0" cellpadding="0" cellspacing="0">
<tr><td colspan="2">OmemoConditionalVizFor(title, mra68,
SimpleFOAFOutput.title)OmemoConditionalVizFor(name, mra68,
SimpleFOAFOutput.name)</td></tr>
<tr><td colspan="2">OmemoConditionalVizFor(givenname, mra68,
SimpleFOAFOutput.givenname)</td></tr>
<tr><td colspan="2">OmemoConditionalVizFor(family_name, mra68,
SimpleFOAFOutput.family_name)</td></tr>
<tr><td colspan="2">OmemoConditionalVizFor(homepage, mra68,
SimpleFOAFOutput.homepage)</td></tr>
<tr><td colspan="2">OmemoConditionalVizFor(depiction, mra68,
SimpleFOAFOutput.depiction)</td></tr>
<tr><td colspan="2">OmemoConditionalVizFor(knows, mra68,
SimpleFOAFOutput.knows)</td></tr>
</table>
<td background="OmemoBaseURL/attach/Mra68Graphics/
xample_body_right_patt.gif"></td>
</tr>
<tr>
<td width="17" style="font-size: 2px"> </td>
<td background="OmemoBaseURL/attach/Mra68Graphics/
xample_body_botton_pat.gif"></td>
<td width="17" style="font-size: 2px"> </td>
</tr>
</table>

```

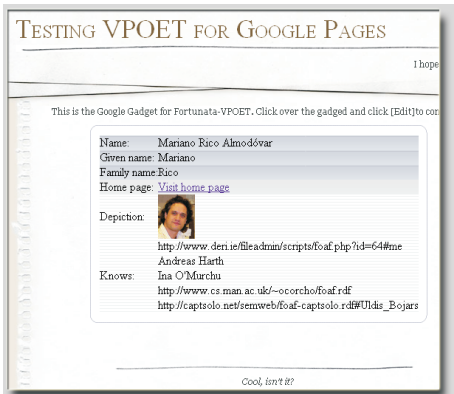


Table 3. Template examples: Fresnel (top), Rhizomic (middle) and VPOET (bottom).

Macro	Arguments	Explanation
<code>OmemoGetP</code>	<code>propName</code>	It is replaced by the property value <code>propName</code>
<code>OmemoBaseURL</code>	No arguments	It is replaced by the URL of the server where VPOET is running
<code>OmemoConditionalVizFor</code>	<code>propName</code> , <code>designerID</code> , <code>designID</code>	It renders the property <code>propName</code> only if it has a value, using the template indicated by (<code>designerID</code> , <code>designID</code>)
<code>OmemoGetLink</code>	<code>relationName</code> , <code>designerID</code> , <code>designID</code>	It is replaced by a link capable of displaying components of the type pointed by the relation <code>relationName</code> using the template indicated by (<code>designerID</code> , <code>designID</code>)

Table 4. Main macros available for web designers in VPOET.

VPOET reduces the requirements needed to create templates, lowering the adoption barrier for web designers. VPOET “speaks” the web designers language, i.e. client side languages such as HTML, CSS and javascript, not requiring competencies in semantic web technologies or additional programming languages. Web designers use simple macros (see table 4) embedded in the source code, with the additional benefit of being capable of detecting errors in the macros in “development time”(checks in every modification of the template) and in “runtime”(checks in every template usage). The information about the structural components of a given ontology component are provided to web designers by OMEMO⁸, another application that generates simplified versions of a given ontology, specifically oriented to web designers. By reading the information provided by OMEMO, web designers can know the sub-components of a given ontology component, requiring only basics of semantic web technologies such as class, property, value or relation.

Table 3 (bottom) shows an example of VPOET template source code (left part), and the rendering of that code inside a Google Page by using VPOET templates in a Google Gadget named GG-VPOET⁹. In this example one can see the macro `OmemoConditionalVizFor`, which renders a given property only if the property has a value. The rendering is transferred to a specified template, which uses the macro `OmemoGetP` to render the property value. As properties use to be multi-valuated, a mechanism based in PPS (pre-condition, post-condition, and separator) has been added to the macro `OmemoGetP`. For example, let us assume a semantic data source with individuals having the property `FOAF:depiction` with values $\{url_1, url_2, \dots, url_N\}$. If a web designer wants to create a VPOET template, like the one shown in bottom of table 3, which renders multiple values of this property with HTMLcode like this:

```
<BR>
....

```

she can use the conditional rendering provided by `OmemoConditionalVizFor` typing:
`OmemoConditionalVizFor(urlprop, mra68, SimpleFOAFOutput.depiction)`

⁸See <http://ishtar.ii.uam.es/fortunata/Wiki.jsp?page=OMEMO>

⁹See <http://ishtar.ii.uam.es/fortunata/Wiki.jsp?page=VPOETGoogleGadget>

which renders the property by means of the template `SimpleFOAFOutput.depiction` (created by designer `mra68`). The code of the template uses `OmemoGetP` in its 3 arguments flavor `OmemoGetP(pre, post, sep)` to code the template like this:
`OmemoGetP(,
)`

Another common requirement are “nested properties”, e.g. displaying only `FOAF:name` for `FOAF:knows` objects. This can be achieved by means of the macro `OmemoGetP` in its 5 arguments flavor `OmemoGetP(relation, propPref, pre, post, sep)`. A simple (multi-values separated by `
`) codification could be:
`OmemoGetP(knows, name,,
)`

But, if additionally you want to associate a link to the value, you have to use the macro `OmemoGetLink`. The final template code would be like this:

```
<A href ="OmemoGetLink(knows)">OmemoGetP(knows, name,,<BR>) </A>
```

This macro is replaced in runtime by a link capable of displaying ontology components of the type pointed by the relation `knows`, i.e. a `FOAF:Person`. In the current implementation, clicking the link would “change to a new page”, but it can be modified to place the client code in a new DOM object in the same page by using AJAX.

3 Conclusions and further work

Our experience in VPOET shows that, in the context of templates creation, macros provides web designers with a friendly environment, requiring lower competencies than other equivalent frameworks such as Semantic Media Wiki, Rhizomer, or Fresnel. An additional advantage of VPOET templates is that these templates can be used easily by other developers skilled in any programming language due to the simplicity of the HTTP messages involved.

An initial small set of macros were obtained from the analysis of the semantic templates in Semantic Media Wiki. The experiments [2] carried out with fifteen real users confirmed that a that small number of macros satisfied the needs of most web designers. These users suggested, by filling detailed questionnaires, new macros or additional parameters for the existing ones.

Our current work is focused on providing web designers with new features such as creation of containers to handle individuals sets in a more attractive way. The current implementation lists individuals in a simple sequence. Probably, this will require new macros or modifications to the existing ones.

References

1. Rico, M., Camacho, D., Corcho, Ó.: VPOET: Using a Distributed Collaborative Platform for Semantic Web Applications. In proc. IDC2008. SCI 162, pp. 167–176. Springer (2008)
2. Rico, M., Macías, J.A., Camacho, D., Corcho, Ó.: Enabling Web Designers to Handle Semantic Data in Web Applications. Submitted to Journal of Web Semantics (2009)
3. Pietriga, E., Bizer, C., Karger, D., Lee, R.: Fresnel: A Browser-Independent Presentation Vocabulary for RDF. In proc. ISWC. LNCS 4273, pp. 158–171. Springer (2006)
4. García, R., Gil, R.: Improving Human–Semantic Web Interaction: The Rhizomer Experience. In proc. SWAP’06. CEUR-WS 201, pp. 57–64. (2006)

SKUA – retrofitting semantics

Norman Gray^{1,2}, Tony Linde¹ and Kona Andrews³

¹ Department of Physics and Astronomy, University of Leicester, UK,

² Department of Physics and Astronomy, University of Glasgow, UK

³ Institute for Astronomy, University of Edinburgh, UK

Abstract. The Semantic Web promises much for software developers, but because its claimed benefits are rather abstract, there is little obvious incentive to master its unfamiliar technology. In contrast, many ‘Social Web’ applications seem rather trivial, and not obviously useful for astronomy.

The SKUA project (Semantic Knowledge Underpinning Astronomy) is implementing a service which will realise the benefits of both these web technologies. This RESTful web service gives application authors ready access to simple persistence, simple (social) sharing, and lightweight semantics, at a low software-engineering cost. The SKUA service allows applications to persist assertions (such as bookmarks and ratings), and share them between users. On top of this, it provides lightweight, astronomy-specific, semantics to enhance the usefulness and retrieval of the users’ data.

1 Introduction

For all its current fashionability, we can identify at least two reasons why the Semantic Web excites little interest among astronomical software developers. Firstly, there is so far no well-known ‘killer app’ for the semantic web, and the use-cases sometimes brandished in support of the Semantic Web’s promise – involving machines booking hospital appointments, or comparing prices ([1], and see <http://www.w3.org/2001/sw/>) – are not obviously relevant to astronomical applications development. Secondly, even when a potential application is dimly discernable – and everyone can agree it must *somehow* be useful for a machine to ‘know’ that a black hole is a type of compact object – there are multiple barriers of novel terminology, standards and technology to be overcome before an idea can be turned into a useful software product. This can be a significant technology hurdle for an application developer who may be rationally sceptical about the practical utility of semantic web technologies.

In the SKUA project (<http://myskua.org>) we are developing an infrastructure which addresses both of these concerns. The SKUA infrastructure provides a mechanism for persisting and sharing a flexible range of application state, including annotations (of which we give examples below), in a way which lets applications transparently take advantage of lightweight semantic knowledge within the SKUA system. That is, we are helping application developers painlessly ‘retrofit’

lightweight semantics to their existing applications at those points where they already persist some data, or could do. By combining the social aspects of the annotation sharing and the lightweight semantics, the SKUA infrastructure can be regarded as a simple ‘Web 3.0’ application, to the extent that that term represents the anticipated melding of Web 2.0 applications with Semantic Web technologies.

2 The SKUA infrastructure

The SKUA infrastructure consists of a network of assertion services, each node of which is an RDF triple store and SPARQL endpoint [2], currently implemented using Jena; these are referred to as SACs, or ‘Semantic Annotation Collections’, and can be either on separate servers or logically distinct entities on a single server. These are the objects to which applications write per-user state information – ‘assertions’ – such as annotations (‘this paper is good’), or preferences (‘I’m interested in pulsars’). The annotations can then be retrieved using a SPARQL query by the same application, by another instance of the same application, or by a cooperating application.

The infrastructure also allows these assertions to be shared between users, in such a way that an application’s SPARQL query against its ‘local’ service is forwarded to the services it federates to (see Fig. 1). This is naïve federation, in which the SAC simply forwards the query to its peers, which potentially forward it in turn, with the results merged before being returned to the caller; thus the final result is the union of the query results to the various nodes, rather than the result of the query over the union of the nodes. Though limited, we believe this model is reasonable in this case, since the information in the various nodes is likely to be both simple and relatively homogeneous. Thus if, in Fig. 1, user ‘u1’ shares the assertion that ‘paper X is good’, then when an application belonging to user ‘u3’ looks for good papers, it picks up the corresponding assertion by ‘u1’. This query federation will be permitted only if the user making the assertion explicitly allows it (which is important in the case where the assertion is something like ‘the author of paper Y is clearly mad’).

Our permissions model is very simple. Each SAC is a personal utility, conceptually more like a USB stick with a PIN than a service with a username/password pair. Each SAC is configured with a list of the SACs to which it should forward queries, and a list of the SACs from which it should accept forwarded requests. Here, the SACs identify themselves when making a delegated query, and do not do so with any delegated credentials from the user on whose behalf they are making the query. This model is easy to implement, we believe it is easy for humans to reason with, and since SACs and users have a close relationship, a user’s SAC is an adequate proxy for the user themselves. This federation model supports both a tree-like and a peer-to-peer network, or anything in between, while allowing a client application to ignore this structure and query only the user’s personal SAC. The trust model is simple-minded, and keeping private my opinion about ‘the author of paper Y’ depends on my friends not federating

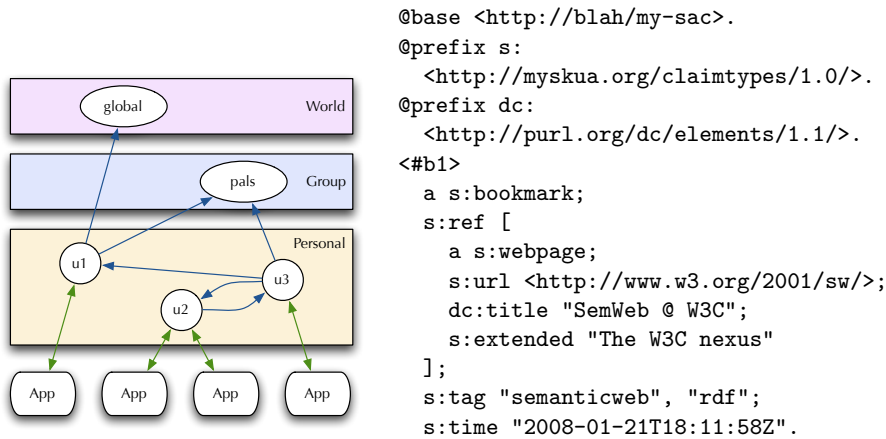


Fig. 1. SKUA’s sharing architecture: on the left we show the relationships, both peer-to-peer and hierarchical, between annotation stores, with double-headed arrows indicating read-write relationships with applications, and the single-headed arrows indicating the federation of queries between services; and on the right we illustrate a potential annotation type, in this case a URL bookmark, using the Turtle notation for RDF [3].

carelessly. User interface details will help couple the user’s mental model to the actual model, but only experience can tell us if the trust model is fundamentally too simple in fact.

Since federation consists only of passing on a SPARQL query, a SAC can federate to any SPARQL endpoint. We have not yet discovered how useful this will be in practice.

Although we have observed that the nodes have astronomy-specific knowledge built in, this is only due to astronomy-specific TBox information uploaded at configuration time, and though this project is specifically motivated by astronomy, the architecture is nonetheless general.

2.1 Interfaces

The SKUA SACs are updated and queried via a RESTful API.

The various annotations are modelled as fragments of RDF which are each named by a URL; these are referred to as ‘claims’ within a SAC. Although the project has defined a lightweight ontology for claims, the RDF which composes a claim is unrestricted. Claims are created by POSTing the RDF to the SAC URL, which responds with a freshly-minted URL naming the claim, which can of course be retrieved in the obvious fashion, with a GET. The contents of the SAC can also be queried by POSTing a SPARQL query to the SAC URL. Individual claims can be replaced by PUTting fresh RDF to the claim URL. There is no cross-reference between the various claims – at least in the applications we have envisaged so far – so little scope for linked-data cross-linking.

The network of federations, and the type of reasoning available (using any of the reasoners available in Jena, or none), is controlled by SAC metadata, also in the form of RDF. This is set when the SAC is created, and may be later adjusted using the Talis Changeset Protocol (http://n2.talis.com/wiki/Changeset_Protocol).

The API is described in a WADL specification available at (<http://myskua.org/doc/qsac/>). Independently of any (debatable) use of this specification for generating client code, we find it useful for generating the interface documentation and generating support for regression tests.

2.2 Implementation

The SAC is implemented using Jena (<http://jena.sourceforge.net>) and SISC (<http://sisc-scheme.org/>), and runs as a web service either standalone (using Jetty), or within a Tomcat container. Essentially all of the application logic is written in Scheme, which allows for rapid development and which, being almost entirely functional, is well-suited for web applications.

The SKUA software is available at <http://skua.googlecode.com>. The current version, at the time of writing, supports updating, persistence, querying and federation; vocabulary-aware querying is available but undocumented; easier sharing and security are in development; and the design of a more sophisticated authorisation model awaits deployment experience.

3 Example applications

An important aim of the SKUA project is to develop applications which use the project's infrastructure, both as a way of validating the approach, and for their intrinsic usefulness. As well, we are cooperating with the developers of existing applications to support them in adding SKUA interfaces where appropriate.

In particular, we are developing *Spacebook* [4], as an adaptation of the my-Experiment code-base ([5], see also <http://myexperiment.org/>). This allows scientists to share digital objects of various kinds, supporting the development of communities. Spacebook builds on this by adding integration with AstroGrid's Taverna workflows, and lets users tag resources using the SKUA infrastructure.

As well, we have adapted the AstroGrid registry browser, VOExplorer [6]. The International Virtual Observatory Alliance (IVOA, <http://www.ivoa.net>) is a consortium of virtual observatory projects, defining and deploying consistent interfaces for accessing astronomical data services. These service resources – image archives and catalogues – are registered in an IVOA registry, and VOExplorer is one of a small number of user-facing applications which allow astronomers to browse the registry, and search within it, including the free-text keyword fields included in the curation metadata.

For each Registry entry, VOExplorer displays title, description, curation and other information, and provides a simple interface for the user to specify a highlight colour, notes about the resource, an alternative title, and tags (see Fig. 2).

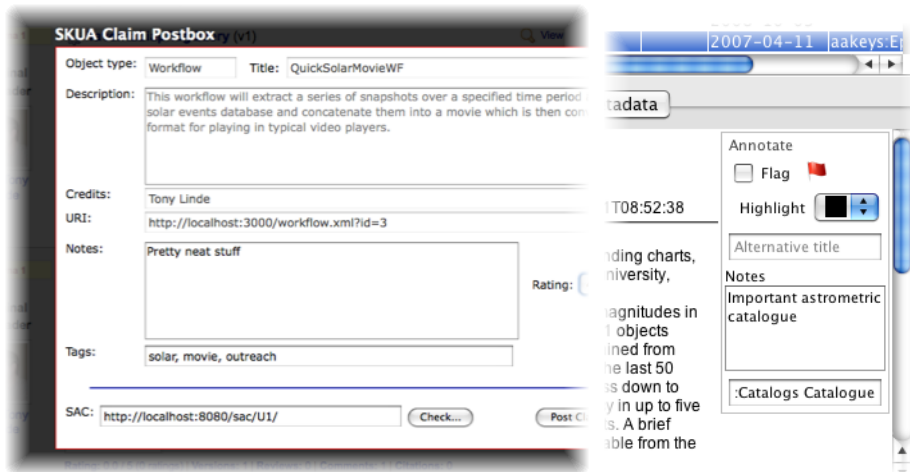


Fig. 2. Annotation panels for Spacebook (left) and VOExplorer (right)

In its original, default, mode, the application persists this information to a local file, but it can also be configured to persist the information to a SKUA SAC; this is not yet the default because SACs have not yet been deployed sufficiently broadly to make this useful to most users.

Users can tag resources using any tags they please, but if they attach keywords from one of the existing IVOA vocabularies [7] a subsequent search on the SKUA store is able to take advantage of the lightweight semantics associated with these keywords. For example, if a user annotates a resource with `aakeys:Ephemerides`, they can later make a SPARQL query for terms which have `AstrometryAndCelestialMechanics` as a broader term, and in doing so pick up resources tagged with `Astrometry`, `CelestialMechanics`, `Eclipses`, `Ephemerides`, `Occultations`, `ReferenceSystems` or `Time`.

The Paperscope application (<http://paperscope.sourceforge.net/>) is a utility for searching and browsing ADS (<http://adswww.harvard.edu/>), which is the principal bibliographic database for astronomy and astrophysics. Like VOExplorer, Paperscope has a simple tagging interface, and like VOExplorer, it was originally limited to a single machine. We have started work on extending the application to use the SKUA RDF nodes as a simple persistence service, using the existing UI and interaction model.

Both the VOExplorer and Paperscope applications were provided with tagging support rather as an afterthought, and in both cases this was barely developed because the tagging could not be shared. Replacing the simple file-handling code with the barely-more-complicated SKUA interface, without changing the user interfaces at all, means that the applications can immediately share annotations and take advantage of the lightweight vocabulary reasoning which the SAC provides. It is in this sense that we claim that the semantic technologies have been *retrofitted* to the applications, giving them an immediate injection of

semantic functionality with minor investment in implementation code, and so allowing the authors to experiment with the user-oriented functionality which this semantic technology prompts.

We emphasise that we are not expecting users to write SPARQL queries for themselves, but instead expect applications to issue them on the user's behalf, based on simple query templates. To support this extra functionality, application developers need make no major commitments to semantic web technologies, and need only manage HTTP transactions using (readily templatable) RDF such as that in Fig 1, and basic SPARQL queries.

4 Conclusion

We have described a simple architecture for storing and sharing simple RDF annotations of external resources, using a RESTful interface to a SPARQL endpoint. The interface is such that application developers have a low barrier to entry, and need make few technology commitments before reaping the benefit of simple semantic enhancement of their applications. We are deploying support for the architecture in a number of existing applications.

Acknowledgements

The SKUA project is funded by the UK's Joint Information Systems Committee (<http://www.jisc.ac.uk>).

References

1. Berners-Lee, T., Hendler, J., Lassila, O.: The Semantic Web. *Scientific American* (May 2001)
2. Prud'hommeaux, E., Seaborne, A.: SPARQL query language for RDF. W3C Candidate Recommendation (June 2007)
3. Beckett, D.: Turtle - terse RDF triple language. W3C Team Submission (January 2008)
4. Linde, T., Gray, N., Andrews, K.: Spacebook: resource sharing for astronomers using SKUA technology. In Bohlender, D., Dowler, P., Durand, D., eds.: *Astronomical Data Analysis Software & Systems, XVIII, PASP* (2009)
5. De Roure, D., Goble, C.: myExperiment – a web 2.0 virtual research environment. In: *International Workshop on Virtual Research Environments and Collaborative Work Environments*, Edinburgh. (2007)
6. Tedds, J.A., Winstanley, N., Lawrence, A., Walton, N., Auden, E., Dalla, S.: VO-Explorer: Visualising data discovery in the virtual observatory. In Argyle, R.W., Bunclark, P.S., Lewis, J.R., eds.: *Astronomical Data Analysis Software and Systems, XVII. Volume 394*. (2007) 159
7. Gray, A.J.G., Gray, N., Hessman, F.V., Martinez, A.P.: Vocabularies in the virtual observatory. IVOA Proposed Recommendation (2008)