

**Proceedings of the  
4<sup>th</sup> International Workshop on  
  
Scripting for the  
Semantic Web  
(SFSW 2008)**



Co-located with 5th European Semantic Web Conference  
June 1-5, 2008, Tenerife, Spain.



## **Workshop Co-Chairs' Message**

### **SFSW 2008 - Workshop on Scripting for the Semantic Web**

Scripting languages such as PHP, JavaScript, Ruby, Python, Perl, JSP and ActionScript are playing a central role in current development towards flexible, lightweight web applications following the AJAX and REST design paradigms. These languages are the tools of a generation of web programmers who use them to quickly create server and client-side web applications. Many deployed Semantic Web applications from the FOAF, RSS/ATOM, blog and wiki communities, as well as many innovative mashups from the Web 2.0 and Open Data movements are using scripting languages and it is likely that the process of RDF-izing existing database-backed websites, wikis, weblogs and CMS will largely rely on scripting languages.

The workshop brings together developers of the RDF base infrastructure for scripting languages with practitioners building applications using these languages. Last years Scripting for the Semantic Web workshop in Innsbruck, Austria focused on the role of scripting languages in the process of populating the Web with linked RDF data as well as to showcase applications that consume RDF data from the Web. The special focus of this years workshop is the creation of Semantic Web data through social interactions as well as applications that integrate socially-created data across communities. As in the last two years the workshop includes a Scripting Challenge, which awards a prize to the most innovative semantic web application developed in a scripting language.

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, University of Pennsylvania, USA  
Gunnar Aastrand Grimnes, DFKI, Germany  
Tom Heath, Open University, UK

## **SFSW 2008 Program Committee**

- Benjamin Nowack, semsol, Germany
- Claudia Müller, University of Potsdam, Germany
- Dan Brickley, Semantic Web Vapourware, UK
- Danny Ayers, Talis, UK
- David Aumüller, Universität Leipzig, Germany
- Denny Vrandečić, AIFB, Universität Karlsruhe, Germany
- Edd Dumbill, Useful Information Company, United Kingdom
- Eero Hyvönen, Helsinki University of Technology (TKK), Finland
- Elias Torres, IBM, USA
- Eyal Oren, Free University Amsterdam, Netherlands
- Giovanni Tummarello, DERI, NUI Galway, Ireland
- Gregory Williams, Rensselaer Polytechnic Institute, USA
- Jens Lehmann, Universität Leipzig, Germany
- Laurian Gridinoc, KMi, The Open University, UK
- Leigh Dodds, Ingenta, United Kingdom
- Libby Miller, Joost, United Kingdom
- Masahide Kanzaki, Keio University, Japan
- Matt Biddulph, Dopplr, United Kingdom
- Michael Hausenblas, Joanneum Research, Austria
- Morten Høybye Frederiksen, MFD Consult, Denmark
- Nadeem Shabir, Talis, UK
- Richard Cyganiak, DERI, NUI Galway, Ireland
- Sandro Hawke, W3C/MIT, USA
- Santtu Toivonen, Idean Enterprises, Finland
- Sean Palmer, Independent Developer, United Kingdom
- Sebastian Dietzold, Universität Leipzig, Germany
- Sebastian Schaffert, salzburg research, Austria
- Stefan Dietze, KMi, The Open University, UK
- Uldis Bojars, DERI, NUI Galway, Ireland
- Vlad Tanasescu, KMi, The Open University, UK

## **Table of Contents**

### **XSLT+SPARQL: Scripting the Semantic Web with SPARQL embedded into XSLT stylesheets**

Diego Berrueta, Jose Emilio Labra and Ivan Herman

### **Publishing and Using Ontologies as Mash-Up Services**

Kim Viljanen, Jouni Tuominen and Eero Hyvönen

### **Cooking HTTP content negotiation with Vapour**

Diego Berrueta, Sergio Fernández and Iván Frade

### **The Talia library platform - Rapidly building a digital library on Rails**

Daniel Hahn, Michele Nucci and Michele Barbera

### **Scripting User Contributed Interlinking**

Michael Hausenblas, Wolfgang Halb and Yves Raimond

### **World of WebCraft - Mashing up World of Warcraft and the Web**

Knud Möller

### **Neologism: Easy Vocabulary Publishing**

Cosmin Basca, Stéphane Corlosquet, Richard Cyganiak, Sergio Fernández and Thomas Schandl

### **Microblogging: A Semantic Web and Distributed Approach**

Alexandre Passant, Tuukka Hastrup, Uldis Bojars and John Breslin

### **Using JavaScript RDFa Widgets for model/view separation inside read/write websites**

Sebastian Dietzold, Sebastian Hellmann and Martin Peklo

### **RDF in JSON**

Keith Alexander

### **Semantic Scripting Challenge Submissions**

# XSLT+SPARQL: Scripting the Semantic Web with SPARQL embedded into XSLT stylesheets

Diego Berrueta<sup>1</sup>, Jose E. Labra<sup>2</sup>, and Ivan Herman<sup>3</sup>

<sup>1</sup> Fundación CTIC

Gijón, Spain

`diego.berrueta@fundacionctic.org`

<sup>2</sup> Departamento de Informática

Universidad de Oviedo, Spain

`labra@uniovi.es`

<sup>3</sup> Centre for Mathematics and Computer Sciences (CWI)

Amsterdam, the Netherlands

`Ivan.Herman@cwi.nl`

**Abstract.** Scripting the Semantic Web requires to access and transform RDF data. We present XSLT+SPARQL, a set of extension functions for XSLT which allow stylesheets to directly access RDF data, independently of any serialization syntax, by means of SPARQL queries. Using these functions, XSLT stylesheets can retrieve, query, merge and transform data from the semantic web. We illustrate the functionality of our proposal with an example script which creates XHTML pages from the contents of DBpedia.

## 1 Introduction

The semantic web has adopted RDF as its preferred data model for resource description. However, the document web is based on markup languages like XML and HTML. Transforming data and gluing between the two worlds with the current set of tools is not always easy. In particular, XSLT [16] and the newer XQuery [10] have been around since 1999 and 2007 respectively, and they offer excellent functionality to query and transform XML documents into other formats. Although they do not directly produce RDF data, indeed they can output RDF serialized in RDF/XML or N3.

However, the inverse transformation, from RDF to XML, is insufficiently covered by the current tool set. Most popular web scripting languages, such as Python and PHP, have their own non-standard APIs for accessing RDF data (conversely, there are standardized APIs for XML, such as DOM and SAX). This is an hindrance to the use of these scripting languages with RDF.

Unfortunately, standard languages such as XSLT and XQuery are inappropriate for this task. Even if RDF data can be apparently addressed as XML through its RDF/XML serialization syntax [5], a second analysis reveals that this is only possible in a very controlled environment in which the serialization is specially produced under certain constraints, or normalized in a previous step [2].

We propose XSLT+SPARQL, an extension to the XSLT function set to embed SPARQL SELECT and ASK queries in the XPath selection expressions. This makes it possible to process RDF data directly within the RDF model, i.e., regardless of any particular serialization syntax. We believe XSLT+SPARQL can contribute to the semantic web by providing a new platform for scripting and transforming RDF into XML, which is often the last processing step of many semantic web scripts. XSLT+SPARQL can easily produce XHTML reports, SVG graphs, SOAP messages or, more generally, any valid XSLT output format (XML or text documents). Moreover, developers can exploit the expressivity of XSLT to create complex scripts that combine, filter and transform data retrieved from different web sources or even from remote SPARQL endpoints.

The rest of the paper is organized as follows. Next section examines related work prior to the description of XSLT+SPARQL in Section 3. Implementation is briefly discussed in Section 4, and a use case is presented in Section 5. The discussion in Section 6 concludes the paper.

## 2 Related work

After a number of proposals from different groups [15], W3C finally came with SPARQL [17], a flexible query language for RDF. There are two companion specifications: the query protocol [13] and a XML Schema to represent the results [6]. Many have observed that the latter makes SPARQL results tractable with XSLT stylesheets in order to generate XML (XHTML, RSS...) from RDF data. This approach is indeed useful for simple tasks, where a single SPARQL query is enough to extract all the relevant data from an RDF graph, but it falls short for complex transformations. Nevertheless, this technique has been suggested as a mechanism to implement the “lowering” operation of semantic web services grounding (cf. section 4.2 of [3]).

The most direct precedent to our work is [19]. The authors of the Topia project introduce a multi-stage processing architecture that consumes RDF and XML data with XSLT stylesheets. They use XSLT extension functions to query a Sesame RDF repository [11] using RDQL [18], RQL and SeRQL, three RDF query languages which preceded in time to SPARQL. In this sense, our work is an updated version of theirs, but in addition to the change of the language, there are other differences. Our queries return binding tables (for SELECT queries) or just Boolean results (for ASK queries), while Topia queries return sub-graphs (sets of triples) serialized in XML.

More recently, XSPARQL [2] has been proposed to unify SPARQL and XQuery in a single language that extends both of them. XQuery and XSLT are W3C recommendations with an important overlap in their functionality. XSPARQL is built on top of the XQuery syntax and semantics. On the other hand, XSLT+SPARQL retains the XSLT syntax and processing model and exploits the extensibility of the XPath function set. The two proposals transform between RDF and XML in both directions, and both can read and produce RDF serialized in different syntaxes (Turtle or RDF/XML).

### 3 Description of XSLT+SPARQL

XSLT+SPARQL exploits the extensibility mechanism provided by the host language, to introduce a number of extension functions. They are grouped in two different namespaces.

#### 3.1 Basic query functions

The core functionality of XSLT+SPARQL is provided by just two functions:

1. `sparql:sparql(query [, documentUrl, ...])`
2. `sparql:sparqlEndpoint(query, endpointUrl)`

These functions execute a SELECT or ASK query (CONSTRUCT and DESCRIBE queries are not allowed because they return RDF/XML documents, which are notoriously difficult to handle in XSLT). The first function executes the query locally, while the second one sends a request to a remote SPARQL endpoint and retrieves the results. The documents pointed by the (potentially empty) list of URLs are merged with the ones referred by the “FROM” clauses of the query to create the default graph.

#### 3.2 Advanced functions

Some applications may need to efficiently execute multiple queries against the same dataset, or even to build custom datasets by merging multiple graphs. To address the requirements of these applications, a second set of extension functions is defined in a different namespace:

1. `sparqlModel:parseString(serializedRdf [, serializationSyntax])`
2. `sparqlModel:readModel(documentUrl [, serializationSyntax])`
3. `sparqlModel:readModel(nodeset)`
4. `sparqlModel:mergeModels(firstModel, secondModel, ...)`
5. `sparqlModel:sparqlModel(query, model)`

The first three functions read RDF data from different sources: (1) a string; (2) a document pointed by a URL; or (3) any fragment of the XSLT input tree, the result tree –only in XSLT 2.0– or even the XSLT stylesheet itself. When the input is a string or a web document, an optional parameter selects the RDF serialization syntax to be parsed (N3 [7], Turtle, RDF/XML, GRDDL [14], RDFa [1]). These three functions return a handler for an in-memory local RDF model that can be used in the next functions.

The “mergeModels” function combines any number of input models into a new one. Due to the functional character of XSLT, it is not possible to do “in-place” modifications of a model, therefore this function returns a new model with the result of combining the data. Consequently, scripts that require to merge data from a sequence of sources cannot iterate through the sequence, but have to

be re-casted as recursive templates using well-known patterns from functional programming.

Finally, the fifth function executes a SPARQL query over an in-memory model. Any “FROM” or “FROM NAMED” clause in the query is ignored.

Together, these advanced functions bring great flexibility to the developer, and also a performance boost to execute multiple queries against the same dataset, avoiding the need for repeated parsing of the input documents.

## 4 Implementation

We have implemented the above functions in Java as an extension for Apache Xalan<sup>4</sup>, an open-source XSLT processor, although it should be easy to adapt our code to other XSLT processors. Our implementation uses the Jena framework [12] to manage RDF documents, and to execute queries locally and remotely. We note that a pure XSLT implementation of “sparql:sparqlEndpoint” is also possible using the “document” function of XSLT. However, due to the lack of control on how that function performs HTTP content negotiation, the pure XSLT implementation may be unable to query some endpoints –notably, DBpedia’s endpoint– when certain XSLT processors are used. On the other hand, our Java-based implementation can manage content negotiation to retrieve RDF or XML documents as needed, therefore it can access the Linked Data web [8,9].

A limitation of our current implementation concerns the “sparql:sparql” and “sparqlModel:mergeModels” functions, which do not accept an arbitrary number of arguments, due to the inability of Xalan 2.7 to access Java5 vararg methods using reflectivity. Particularly, the former accepts zero or one document URLs, and the latter merges exactly two models.

## 5 Use case: DBpedia to XHTML

Practical use of XSLT+SPARQL is demonstrated by a sample script (Figure 1) that extracts data from DBpedia [4] and creates a simple report in XHTML. The stylesheets contains only two templates. The first one efficiently accesses DBpedia data through its public SPARQL endpoint (i.e, the script does not retrieve the RDF dataset, but just the results of the query in XML format). The second one generates XHTML mark-up for each row of the results bindings table.

The functionality of XSLT+SPARQL goes far beyond this small example. More complex examples can be found in the project web page<sup>5</sup>.

## 6 Conclusions and future work

We presented XSLT+SPARQL, a simple extension for XSLT to embed SPARQL queries in XPath expressions, making it possible to process RDF data from

<sup>4</sup> <http://xml.apache.org/xalan-j/>

<sup>5</sup> <http://berrueta.net/research/xsltsparql>



```

<xsl:variable name="query">
  PREFIX skos: <http://www.w3.org/2004/02/skos/core#>;
  PREFIX cat: <http://dbpedia.org/resource/Category:>;
  PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>;
  PREFIX foaf: <http://xmlns.com/foaf/0.1/>;
  SELECT ?name ?person ?img
  WHERE { ?person skos:subject    cat:Spanish_actors .
          ?person rdfs:label     ?name .
          ?person foaf:depiction ?img .
          FILTER (lang(?name)="es") }
  ORDER BY ?name
</xsl:variable>

<xsl:template match="/">
  <html> <body>
    <h1>Spanish Actors</h1>
    <ul>
      <xsl:apply-templates select="sparql:sparqlEndpoint(
        $query, 'http://dbpedia.org/sparql')"/>
    </ul>
  </body> </html>
</xsl:template>

<xsl:template match="results:result">
  <li>
    <xsl:value-of select="results:binding[@name='name']"/>
    <img> <xsl:attribute name="src">
      <xsl:value-of select="results:binding[@name='img']"/>
    </xsl:attribute> </img>
  </li>
</xsl:template>

```

**Fig. 1.** An XSLT+SPARQL stylesheet to create a XHTML page with data extracted from DBpedia by querying its SPARQL endpoint.

XSLT stylesheets without bothering with the tricky RDF/XML serialization. XSLT+SPARQL is a new standards-based platform to write declarative scripts for the semantic web. It is straightforward to use for developers experienced in XML and RDF technologies because it does not require to learn a new language or processing model. Furthermore, it can be implemented by re-using current XSLT processors.

The usage of XSLT+SPARQL has been illustrated by means of a sample application to create XHTML reports from DBpedia data. We envisage the usage of XSLT+SPARQL in a wide variety of scenarios. They mainly involve transforming RDF data into other formats which are more suitable for human consumption. However, more complex semantic web agents can be developed as well. For instance, we are working on a semantic web crawler/browser that

collects relevant information about a subject from different linked data sources and creates a comprehensive report in XHTML.

We are also considering new additions to the current set of functions. One area of particular interest would be to add support for RDFS and OWL reasoning.

## References

1. B. Adida and M. Birbeck. RDFa primer 1.0. Working draft, W3C, March 2007.
2. W. Akhtar, J. Kopecky, T. Krennwallner, and A. Polleres. XSPARQL: Traveling between the XML and RDF worlds and avoiding the XSLT pilgrimage. Technical Report 2007-12-14, DERI Galway, 2007.
3. R. Akkiraju and B. Sapkota. Semantic annotations for WSDL and XML schema - usage guide. Working group note, W3C, 2007.
4. S. Auer, C. Bizer, J. Lehmann, G. Kobilarov, R. Cyganiak, and Z. Ives. DBpedia: A nucleus for a web of open data. In *International Semantic Web Conference*, 2007.
5. D. Beckett. RDF/XML syntax specification (revised). Recommendation, W3C, February 2004.
6. D. Beckett and J. Broekstra. SPARQL query results XML format. Recommendation, W3C, January 2008.
7. T. Berners-Lee. Notation 3. Available at <http://www.w3.org/DesignIssues/Notation3.html>, 1998.
8. D. Berrueta and J. Phipps. Best practice recipes for publishing RDF vocabularies. Working draft, W3C, 2007.
9. C. Bizer, R. Cyganiak, and T. Heath. How to publish linked data on the web?, Jul 2007.
10. S. Boag, D. Chamberlin, M. F. Fernández, D. Florescu, J. Robie, and J. Siméon. XQuery 1.0: An XML query language. Recommendation, W3C, 2007.
11. J. Broekstra, A. Kampman, and F. van Harmelen. Sesame: A generic architecture for storing and querying RDF. In *International Semantic Web Conference*, 2002.
12. J. J. Carroll, I. Dickinson, C. Dollin, D. Reynolds, A. Seaborne, and K. Wilkinson. Jena: implementing the semantic web recommendations. In *13th international World Wide Web conference Alternate track papers*, 2004.
13. K. G. Clark. SPARQL protocol for RDF. Recommendation, W3C, January 2008.
14. D. Connolly. Gleaning Resource Descriptions from Dialects of Languages (GRDDL). Recommendation, W3C, September 2007.
15. P. Haase, J. Broekstra, A. Eberhart, and R. Volz. A comparison of RDF query languages. In *Third International Semantic Web Conference, Hiroshima, Japan*, 2004.
16. M. Kay. XSL transformations (XSLT) version 2.0. Recommendation, W3C, 2007.
17. E. Prud'hommeaux and A. Seaborne. SPARQL query language for RDF. Recommendation, W3C, January 2008.
18. A. Seaborne. RDQL - a query language for RDF. Member submission, W3C, January 2004.
19. J. van Ossenbruggen, L. Hardman, and L. Rutledge. Towards smart style: combining RDF semantics with XML document transformations. Technical Report INSE0303, CWI, October 2003.

# Publishing and Using Ontologies as Mash-Up Services

Kim Viljanen, Jouni Tuominen, and Eero Hyvönen

Semantic Computing Research Group (SeCo)  
Helsinki University of Technology and University of Helsinki  
P.O. Box 5500, 02015 TKK, Finland  
`first.last@tkk.fi`, <http://www.seco.tkk.fi>

**Abstract.** The Semantic Web is based on using ontologies for enabling semantically disambiguated data exchange between distributed systems on the web. This requires efficient means for publishing ontologies on the web to ensure the availability, sharing and acceptance of the ontologies. Support services are needed for utilizing ontologies easily and cost-effectively in applications and legacy systems lacking ontology support. To address these vital needs, this paper presents the ONKI ontology service which provides ready-to-use “mash-up” functionalities, such as semantic disambiguation, concept finding and concept fetching as ready-to-use web widgets for adding ontology support to e.g. HTML forms using JavaScript. Two implementations of the ONKI Server are presented: ONKI-SKOS for ontologies presented in the Simple Knowledge Organization System (SKOS) language and ONKI-Geo for geographical ontologies with a map interface. The presented ONKI systems are operational on the web, used in the National Finnish Ontology Service. They have been successfully used in several pilot applications.

## 1 Ontologies as Web 2.0 Services

The Semantic Web<sup>1</sup> introduces a metadata layer on top of the World Wide Web infrastructure for describing its content and services in an explicit, machine “understandable” way using ontologies [1–3]. When such content is available, semantically aware applications for e.g. searching and browsing the distributed content can be created, as demonstrated in e.g. various semantic portals [4–6]. Many ontologies have been created and are available online in RDF(S) and OWL form today. For example, the Swoogle<sup>2</sup> [7] search engine index contains over 10,000 ontologies on the web.

One of the main lessons learned in our work on creating semantic portals [5, 8, 9, 6] is that metadata in data sources, such as museum databases, are often syntactically heterogeneous and contain spelling errors, are semantically ambiguous, and are based on different vocabularies [10]. This results in lots of tedious syntactic correction, semantic disambiguation, and ontology mapping work when making the contents semantically interoperable, and when publishing them on

<sup>1</sup> <http://www.w3.org/2001/sw/>

<sup>2</sup> <http://swoogle.umbc.edu/>

the Semantic Web. A natural solution to this problem would be to enhance legacy cataloguing and content management systems (CMS) with ontological annotation functions so that the quality of the original data could be improved and errors fixed in the content creation phase. However, implementing such ontological functions in existing legacy systems may require lots of work and thus be expensive, which creates a severe practical hindrance for the proliferation of the Semantic Web.

This relates to the more general challenge of the Semantic Web today: ontologies are typically published as files without support for using them in applications. Each application tends to re-implement similar functions for utilizing ontologies, such as semantic autocompletion and disambiguation [11], browsing and finding concepts, and populating ontologies. It is like re-creating map services from scratch in different geographical web applications, rather than using available services such as Google Maps<sup>3</sup>, Yahoo Maps<sup>4</sup>, or Microsoft Live Search Maps<sup>5</sup>. We argue that ontologies should be published as lightweight shared services which can be easily utilized in legacy systems using a mash-up approach in the same spirit as e.g. Google Maps, Yahoo Maps and Freebase<sup>6</sup> are used today. This approach for publishing ontologies means, that generic, shared functionalities are combined with specific applications using lightweight scripting and programming technologies such as Ajax<sup>7</sup>.

In the following, we first outline the requirements of mash-up ontology services and present the implementation of a generic mash-up ONKI Ontology Service and framework which is currently used in the National Ontology Service in Finland<sup>8</sup>. We then present two implementations of ontology specific server implementations conforming to the ONKI Service framework: ONKI-SKOS for general SKOS<sup>9</sup> ontologies and ONKI-Geo [12] for geographical ontologies. After this, utilization of ONKI services in an external application is discussed by presenting two application scenarios. Finally, contributions, results, and lessons learned are summarized, and directions for further research outlined.

## 2 Requirements of a Mash-Up Ontology Service

The national semantic web infrastructure model being built by the FinnONTO project in Finland [13] argues that ontology services are needed for three major user groups: 1) *Ontology developers* need a collaborative ontology development, versioning, and publishing environment for ontologies [14]. 2) *Content indexers* need services for finding the desired annotation concepts and for transporting the corresponding URIs and other data from the ontology service into external

---

<sup>3</sup> <http://maps.google.com/>

<sup>4</sup> <http://maps.yahoo.com/>

<sup>5</sup> <http://maps.live.com>

<sup>6</sup> <http://code.google.com/p/freebase-suggest/>

<sup>7</sup> [http://en.wikipedia.org/wiki/Ajax\\_\(programming\)](http://en.wikipedia.org/wiki/Ajax_(programming))

<sup>8</sup> <http://www.yso.fi>

<sup>9</sup> <http://www.w3.org/2004/02/skos/core/>

applications. 3) *Information searchers* need services for finding and disambiguating keyword meanings, and for transporting the corresponding URIs into search engines and other applications. In this paper we focus on the problem of supporting content indexing in applications and legacy systems.

Ontology servers are intended for managing ontologies, providing support for designing, choosing and accessing ontologies [15–17]. However, compared to previous work on ontology servers, we propose the idea of creating ontology services which can easily be used in applications. This requires the following features:

*Mash-up integration support.* Ontology servers should support runtime integration of the functionalities to applications and legacy systems, especially for annotation and semantic search.

*Semantic autocompletion and disambiguation.* Efficient search functionalities are important when trying to find the semantically correct concepts from large ontologies. Text search boosted up with semantic autocompletion and disambiguation functionalities [11] supports the user in finding the right concept by giving constant feedback of the query, and by helping in disambiguating the intended concept meaning.

*Concept fetching.* When using ontologies in combination with other applications, the idea of “copying” or “transferring” concepts between applications is important. We propose a concept fetching functionality for moving concept URIs from the ontology server to the target application, such as a legacy cataloguing system or CMS. To support legacy systems, indexing terms (concept labels) should be possible to use instead of URIs even though this may create disambiguation and mapping problems e.g. between ontology versions due to potentially less specific identifiers than the URIs.

*Concept collecting.* Usually no single concept describes all the aspects of the entity that is being described with a certain metadata property such as *dc:subject*. Therefore, it should be possible to collect multiple concepts from the ontology server and return these as a combination value in a specific metadata field to the legacy system.

*Domain-specific user interfaces.* The concepts of an ontology are typically visualized as an abstract graphical tree or graph visualization of the currently selected concept with its semantic vicinity [18]. Complementing this, we propose providing domain-specific interfaces, such as a map interface for geographical ontologies, when applicable.

### 3 ONKI Ontology Service

The ONKI Ontology Service is a general ontology library and framework that provides functionalities for accessing the ontologies using ready-to-use mash-up web widgets as well as application interfaces for humans and machines for doing, e.g., content indexing, concept disambiguation, searching and fetching. The service is based on ontology and domain specific implementations of ONKI Servers which conform to the ONKI interfaces. This means that it is possible to

provide a single mash-up web widget to access all ontologies but at the same time provide domain-specific user interfaces and technical implementations optimized for ontologies of different sizes, modelling languages, etc.

The ONKI Widget (Figure 1) is a ready-made user interface widget for using the ONKI Service in content annotation (indexing). It enables the user, e.g. a content annotator, to find the correct ontological concepts and their URIs and then transfer the URIs and the concept labels to their own content management application. Such a simple means for getting the URIs and to use them in applications is crucial for enabling the content creation on the Semantic Web.

In the following, the JavaScript and Direct Web Remoting (DWR)<sup>10</sup> based implementation of the widget is described which is intended to be used for extending HTML forms with ontology functionalities. However, the proposed solution is more general because in the case of other user interface technologies such as Java Swing, the ONKI Web Service<sup>11</sup> interface could be used to implement user interface technology specific implementations of the ONKI Widget.

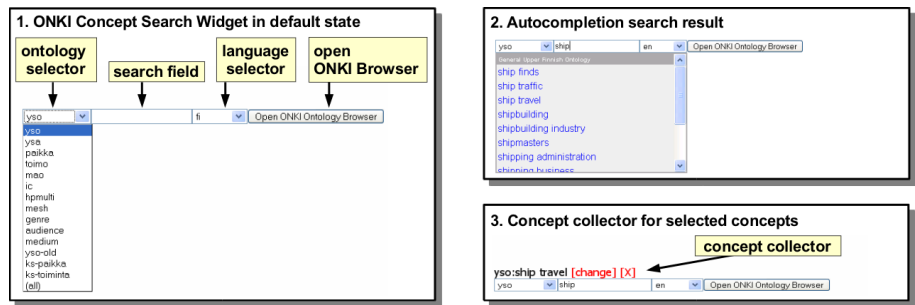


Fig. 1. ONKI Concept Search Widget.

Part 1 of Figure 1 shows the default components of the widget. The ontology selector can be used to change the ontology used in search or to select all ontologies as target for the search. The search field is used for finding concepts using text queries. In part 2 of Figure 1 the user is typing a search string to the autocompletion search field which dynamically performs a query after each input character (here “s-h-i-p-...”) to the ONKI service and returns the concepts whose labels match the string, given the language selection. The results of the query are shown in the web widget’s result list below the input field. In the case of synonym terms, the preferred label of a concept will be presented. For example, when searching for an (outdated) term “birch sugar”, the system returns “birch sugar → xylitol” which means that “xylitol” is the preferred term.

When a desired concept is selected from the result list, it’s URI and label are

<sup>10</sup> <http://getahead.org/dwr>

<sup>11</sup> <http://www.w3.org/TR/ws-arch/>

put in the widget's concept collector (Part 3 in Figure 1) for further usage such as submitting the content to the server application or accessing the collector from an application specific JavaScript program. The idea of the concept collector can be compared to the idea of shopping carts in web stores. In the example, the concept "ship travel" has been put into the concept collector.

The language of concept labels used in matching the query string can be chosen by using the language selector. The choice of languages depends on the ontology selected. For example, for YSO, English and Swedish are supported in addition to Finnish, and the Finnish Geo-ontology<sup>12</sup> can be used in Finnish, Swedish, and in three dialects of Sami spoken in Lapland. It is possible to use all languages simultaneously.

If the user doesn't know what to type in the text search field, the alternative of using a browsing interface is available by using the domain specific ONKI Browser ("Open ONKI Ontology Browser" button). The ONKI Browser can also be used for disambiguating homonym terms, i.e. concepts with identical labels, by aiding the user to inspect the context of the concepts. When the desired concept has been found using the ONKI Browser, the concept's URI and label are fetched into the application by pressing the "Fetch concept" button on the ONKI Browser page corresponding to the concept. Two implementations of the ONKI Browser are presented in the next section of the paper.

The web widget can be integrated into an HTML Form with two lines of JavaScript code. The following code line loads the ONKI JavaScript library and should be added into the HEAD section of the HTML page:

```
<script type="text/javascript"
      src="http://www.yso.fi/onki.js"></script>
```

Using the library, ordinary HTML form input fields can be extended with ONKI functionality by declaring the *onkeyup* event handler for the field. For example, adding the General Finnish Upper Ontology YSO to a example *dc:subject* field is done as follows:

```
<input id="dc:subject" onkeyup="onki['yso'].search()" />
```

As a result, when a page is accessed, the user interface is enhanced with ontology support. The widget provides a default concept collector (Part 3 in Figure 1) which shows the fetched concepts in the widget's user interface and stores them in hidden input fields. When the form is submitted, the values of the hidden input fields can be processed by the target application in the same way as any HTML form submissions.

The ONKI Widget can be customized by configurations and by implementing callback functions. Configuration possibilities include disabling the menus for selecting ontologies and the language, the search field, or the "Open ONKI Browser" button. The widget can also be configured to restrict the search to concepts of certain type or belonging to a specific subtree of an ontology. Additionally, CSS styling can be used for configuring the appearance of the widget. The

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

ONKI Widget’s callback functions enable application specific implementations of e.g. the concept collector or the concept search result list using JavaScript.

The ONKI API includes the following methods:

- *search(query, lang, maxHits, type, parent)* - for searching for ontological concepts. Returns a list of hits.
- *getLabel(URI, lang)* - for fetching a label for a given URI in a given language.
- *getAvailableLanguages()* - for querying for supported languages of an ontology. Returns a list of languages.

By implementing these methods, any system can be added to the ONKI Service to be used via the general ONKI Service functionalities such as the ONKI Widget. This is demonstrated by the case implementations presented below. Thus, the ONKI Service is not tied to a single ONKI Server implementation.

## 4 Two domain specific ONKI Server implementations

Two domain-specific ONKI Servers have been implemented conforming to the general ONKI service functionalities described in the previous section. ONKI-SKOS is intended for lightweight ontologies and ONKI-Geo [12] for geographical ontologies. In the following these two systems are shortly described.

### 4.1 ONKI-SKOS Server for SKOS Vocabularies

ONKI-SKOS is a general ontology service supporting thesaurus-like ontologies especially in content indexing. ONKI-SKOS can be used to browse, search and visualize any vocabulary conforming to the SKOS recommendation, and also RDF(S) and OWL ontologies with additional configuration. ONKI-SKOS does simple reasoning, e.g. transitive closure over class and part-of hierarchies. The implementation has been tested using various ontologies, e.g. the General Finnish Upper Ontology YSO, containing 20,000 concepts.

ONKI-SKOS Browser (Figure 2) is the graphical user interface of ONKI-SKOS. It consists of three main components: 1) *concept search with semantic autocompletion*, 2) *concept hierarchy* and 3) *concept properties*. When typing text to the search field, a query is performed to match the concepts’ labels. The result list shows the matching concepts, which can be selected for further examination. The search can be further narrowed by restricting the search to concepts of a certain type or to a desired subtree of the ontology.

When a concept is selected in ONKI-SKOS Browser, its concept hierarchy is visualized as a tree structure, and its properties are shown as a table. Various configuration properties are specified to enable ONKI-SKOS to process the ontologies as desired. The configurable properties include the ontological properties used in concept hierarchy generation, the properties used to label the concepts, the concept to be shown in the default view and the default concept type used in restricting the searches.





Fig. 2. ONKI-SKOS Browser.

ONKI-SKOS Server is implemented as a Java Servlet using the Jena Semantic Web Framework<sup>13</sup>, the DWR library and the Lucene<sup>14</sup> text search engine.

#### 4.2 ONKI-Geo Server for Geographical Ontologies

ONKI-Geo [12] is an ontology service specialized for geographical data. It was developed for the Finnish Place Ontology SUO (Suomalainen Paikkaontologia) [19] which currently has been populated with 1) place information from the Geographic Names Register (GNR) provided by the National Land Survey of Finland<sup>15</sup> and with 2) place information from the GEOnet Names Server (GNS)<sup>16</sup> maintained by the National Geospatial-Intelligence Agency (NGA) and the U.S. Board on Geographic Names (US BGN). GNR contains about 800,000 multilingual resources of natural and man-made features in Finland, including data such as place type or feature type and the coordinates of a place. The GNS register contains similar information about 4,100,000 places around the world.

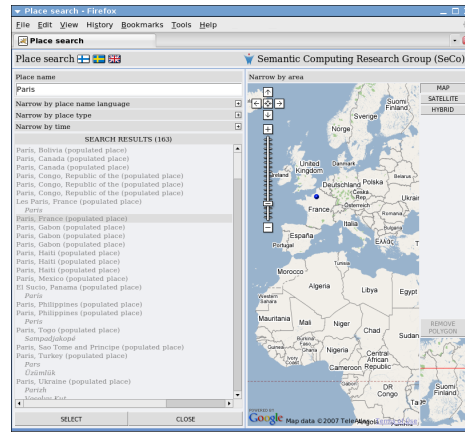
The ONKI-Geo Browser (Figure 3) is intended for annotating resources using unambiguous place identifiers (URIs) or coordinates for arbitrary points or polygons. Using unambiguous place identifiers is useful e.g. due to homonymous place names: there are hundreds of places in Finland with the name "Isosaari" ("Big Island"). The ONKI-Geo Browser contains several facets for narrowing the search to find the intended place instance: A polygon can be drawn in the map interface for making a search on all places in the selected area. The other facets are an ontology of geographic features (e.g., lake, city, etc.), the languages of the place names, and a place name search with autocompletion. The system uses Google Maps widgets for visualizing the places.

<sup>13</sup> <http://jena.sourceforge.net/>

<sup>14</sup> <http://lucene.apache.org/java>

<sup>15</sup> <http://www.maanmittauslaitos.fi/>

<sup>16</sup> <http://earth-info.nga.mil/gns/html/>



**Fig. 3.** ONKI-Geo Browser. Search can be constrained by using the facets on the left or by drawing a polygon on the map. By pushing the “Select” button in the left bottom corner, the concept or selected coordinate information is transferred into the mash-up widget.

## 5 Integrating ONKI with Application Systems

In the following we describe use cases of the ONKI system.

### 5.1 Integrating ONKI with a Cataloguing System

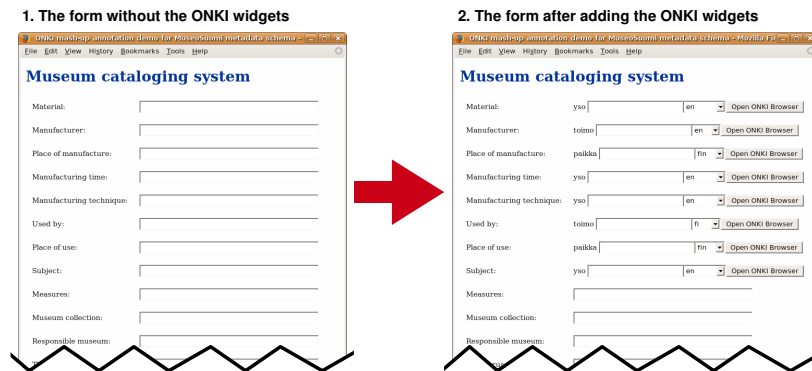
To demonstrate how to add ONKI functionalities to a legacy system, we created a simple web form (part 1 of Figure 4) presenting the MuseumFinland [5] metadata fields [20]. By adding the ONKI Concept Search Widget to the fields, ontologies can be used in annotating museum collection items. Part 2 of Figure 4 depicts the original form after adding the widgets.

After this the form can be used for creating semantic metadata. If the underlying system does not support URIs, the system has to be modified to handle this kind of information. Alternatively the ONKI Widget can be configured to return concept labels instead of URIs.

### 5.2 An Annotation Editor Based on ONKI Ontology Services

SAHA<sup>17</sup> [21, 22] is a generic annotation system supporting distributed collaboration in creating annotations, and hiding the complexity of the annotation schema and the domain ontologies from the annotators. SAHA adapts to different metadata schemas, which makes it suitable for different applications. Support for using ontologies is based on ONKI ontology services (Figure 5). The system

<sup>17</sup> <http://www.seco.tkk.fi/services/saha/>



**Fig. 4.** A museum cataloging system before and after integrating the ONKI widgets.

is being tested in various practical semantic portal projects such as HealthFinland [6] and CultureSampo [23].

The metadata field elements are implemented using the ONKI widget, as discussed above. Depending on the field, different ONKI servers are used as specified in the SAHA configuration. In this case the Finnish General Upper Ontology YSO [24], published as an ONKI service<sup>18</sup>, is used for selecting annotation concepts. SAHA can also make use of the automatic text extraction component POKA<sup>19</sup> in extracting potential annotation concepts from web resources [21], and populating the SAHA concept collector with them.

Annotations created with SAHA are stored in a centralized database, from which they can be retrieved for editing or to be used in applications such as semantic portals. It is possible to view and edit existing annotations by reading the metadata fields in corresponding widget collectors. Furthermore, SAHA supports population of its own annotation ontologies by new resources. In this way, different users creating annotations collaboratively can share new resources created by anyone, e.g., instances of new works of art or other artifacts.

## 6 Discussion

The main contribution of this paper is to present the idea of publishing *ontologies as mash-up services* that can be integrated in a lightweight fashion to legacy systems on the user interface level. To demonstrate the applicability of the idea, we presented the ONKI service and two implementations of the ONKI interface: the general ontology server ONKI-SKOS and the geographical ontology server ONKI-Geo. The two ONKI implementations also demonstrated the idea of creating domain specific user interfaces to better support the usage of different

<sup>18</sup> <http://www.yso.fi/onto/yso/>

<sup>19</sup> <http://www.seco.tkk.fi/tools/poka/>

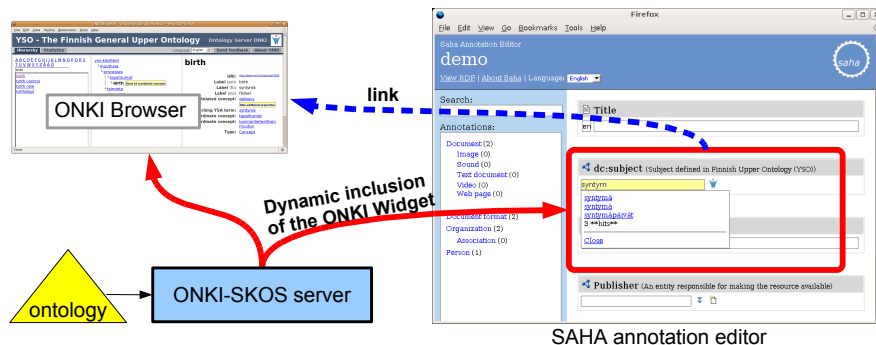


Fig. 5. ONKI integrated with the SAHA annotation editor.

types of ontologies. A practical contribution of the paper was to introduce the idea of concept fetching between applications and the need for concept collecting when using an ontology server for annotation purposes. Finally, semantic auto-completion was proposed and implemented in the user interface components to provide an efficient method for finding and disambiguating concepts.

A lesson learned from implementing the concept fetching functionality as a web browser application was that special tricks are needed to transfer data between browser windows loaded from different domains<sup>20</sup>. Security being an important concern, we suggest that browsers should provide some standardized solution for communication between domains.

The widget is currently being developed for supporting other tasks where ontological concepts need to be searched and fetched, such as ontological content search. Future work includes researching how the autocomplete concept search could help even more in disambiguation the concepts without forcing the user to open the ONKI Browser with all information about the concepts. In future, other user interface environments than HTML and the web browser could be supported, such as Java Swing. The ONKI API does not currently use RDF for returning the content to the Widget to make it easier to handle the content with JavaScript. However, in future RDF might be used. Finally, according to our vision of a national ontology service, the ontologies in such a service should be extensively and mutually interlinked to support creating cross-domain applications. Therefore, the question of how to support developing and using mutually interlinked ontologies should be researched.

<sup>20</sup> Since the ONKI-Browser is located in a different domain than the ONKI-Widget, the communication between them was solved as follows: the Widget opens a new browser window, which contains the ONKI-Browser in an IFRAME. The selected concept URI is returned from the ONKI-Browser by changing the fragment identifier of the window with the IFRAME, which can be accessed by the Widget.

## Acknowledgments

We thank Ville Komulainen for his work on the first version of the ONKI server, and Robin Lindroos and Tomi Kauppinen for collaboration in ONKI-Geo development. Our research is a part of the National Semantic Web Ontology Project in Finland<sup>21</sup> (FinnONTO) 2003–2007 funded by the Finnish Funding Agency for Technology and Innovation (Tekes) and 36 companies and public organizations.

## References

1. Gruber, T.R.: A translation approach to portable ontology specification. *Knowledge Acquisition* **5**(2) (June 1993) 199–220
2. Staab, S., (eds.), R.S.: *Handbook on ontologies*. Springer-Verlag (2004)
3. Fensel, D.: *Ontologies: Silver bullet for knowledge management and electronic commerce (2nd Edition)*. Springer-Verlag (2004)
4. Reynolds, D., Shabajee, P., Cayzer, S.: *Semantic Information Portals*. In: *Proceedings of the 13th International World Wide Web Conference on Alternate track papers & posters*, New York, NY, USA, ACM Press (May 2004)
5. Hyvönen, E., Mäkelä, E., Salminen, M., Valo, A., Viljanen, K., Saarela, S., Junnila, M., Kettula, S.: *Museumfinland—Finnish museums on the semantic web*. *Journal of Web Semantics* **3**(2) (2005) 25
6. Hyvönen, E., Viljanen, K., Suominen, O.: *Healthfinland—Finnish health information on the semantic web*. In: *Proceedings of the 6th International Semantic Web Conference (ISWC 2007)*, Busan, Korea, Springer-Verlag (Nov 2007)
7. Ding, L., Finin, T., Joshi, A., Pan, R., Cost, R.S., Peng, Y., Reddivari, P., Doshi, V., Sachs, J.: *Swoogle: a search and metadata engine for the semantic web*. In: *CIKM '04: Proceedings of the thirteenth ACM international conference on Information and knowledge management*, New York, NY, USA, ACM Press (2004) 652–659
8. Sidoroff, T., Hyvönen, E.: *Semantic e-government portals - a case study*. In: *Proceedings of the ISWC-2005 Workshop Semantic Web Case Studies and Best Practices for eBusiness SWCASE05*. (Nov 2005)
9. Käsälä, T., Hyvönen, E.: *A semantic view-based portal utilizing Learning Object Metadata (August 2006)* 1st Asian Semantic Web Conference (ASWC2006), *Semantic Web Applications and Tools Workshop*.
10. Hyvönen, E., Salminen, M., Kettula, S., Junnila, M.: *A content creation process for the Semantic Web (2004)* *Proceeding of OntoLex 2004: Ontologies and Lexical Resources in Distributed Environments*, May 29, Lisbon, Portugal.
11. Hyvönen, E., Mäkelä, E.: *Semantic autocompletion*. In: *Proceedings of the first Asian Semantic Web Conference (ASWC 2006)*, Beijing, Springer-Verlag, New York (August 4–9 2006)
12. Kauppinen, T., Henriksson, R., Sinkkilä, R., Lindroos, R., Väätäinen, J., Hyvönen, E.: *Ontology-based disambiguation of spatiotemporal locations*. In: *1st international workshop on Identity and Reference on the Semantic Web (IRSW2008)*, 5th European Semantic Web Conference 2008 (ESWC 2008), Tenerife, Spain. (June 1-5 2008) forthcoming.

<sup>21</sup> <http://www.seco.tkk.fi/projects/finnonto/>

13. 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 European Semantic Web Conference ESWC 2008, Springer (June 1-5 2008)
14. Komulainen, V., Valo, A., Hyvönen, E.: A tool for collaborative ontology development for the semantic web. In: Proc. of the International Conference on Dublin Core and Metadata Applications (DC 2005). (Nov 2005)
15. Ahmad, M.N., Colomb, R.M.: Managing ontologies: a comparative study of ontology servers. In: ADC '07: Proceedings of the eighteenth conference on Australasian database, Darlinghurst, Australia, Australia, Australian Computer Society, Inc. (2007) 13–22
16. Ding, Y., Fensel, D.: Ontology library systems: The key to successful ontology reuse. In: Proceedings of SWWS'01, The first Semantic Web Working Symposium, Stanford University, USA. (2001) 93–112
17. Komulainen, V.: Public services for ontology library systems. Master's thesis, University of Helsinki, Department of Computer Science (January 2007)
18. Eklund, P., Roberts, N., Green, S.: Ontorama: Browsing rdf ontologies using a hyperbolic-style browser. In: First International Symposium on Cyber Worlds, CW02, Theory and Practices, IEEE Press. (2002) 405–411
19. Kauppinen, T., Henriksson, R., Väätäinen, J., Deichstetter, C., Hyvönen, E.: Ontology-based modeling and visualization of cultural spatio-temporal knowledge. In: Developments in Artificial Intelligence and the Semantic Web. Proceedings of the 12th Finnish AI Conference STeP 2006. (October 26–27 2006)
20. Viljanen, K., Tuominen, J., Käsälä, T., Hyvönen, E.: Distributed semantic content creation and publication for cultural heritage legacy systems. In: Proceedings of 2008 IEEE International Conference on Distributed Human-Machine Systems, IEEE Press (2008)
21. Valkeapää, O., Alm, O., Hyvönen, E.: Efficient content creation on the semantic web using metadata schemas with domain ontology services (system description). In: Proceedings of the European Semantic Web Conference ESWC 2007, Innsbruck, Austria, Springer (June 4–5 2007)
22. Valkeapää, O., Hyvönen, E.: A browser-based tool for collaborative distributed annotation for the semantic web. In: Proceedings of the Semantic Authoring and Annotation Workshop, 5th International Semantic Web Conference. (November 2006)
23. Hyvönen, E., Ruotsalo, T., Häggström, T., Salminen, M., Junnila, M., Virkkilä, M., Haaramo, M., Mäkelä, E., Kauppinen, T., , Viljanen, K.: Culturesampo—finnish culture on the semantic web: The vision and first results. In Robering, K., ed.: Information Technology for the Virtual Museum. LIT Verlag, Berlin. (Nov 2007)
24. Hyvönen, E., Viljanen, K., Mäkelä, E., Kauppinen, T., Ruotsalo, T., Valkeapää, O., Seppälä, K., Suominen, O., Alm, O., Lindroos, R., Käsälä, T., Henriksson, R., Frosterus, M., Tuominen, J., Sinkkilä, R., Kurki, J.: Elements of a national semantic web infrastructure—case study finland on the semantic web (invited paper). In: Proceedings of the First International Semantic Computing Conference (IEEE ICSC 2007), Irvine, California. (September 2007) IEEE Press.

# Cooking HTTP content negotiation with Vapour

Diego Berrueta<sup>1</sup>, Sergio Fernández<sup>1</sup> and Iván Frade<sup>2</sup>

<sup>1</sup> Fundación CTIC

Gijón, Asturias, Spain

{diego.berrueta,sergio.fernandez}@fundacionctic.org

<http://www.fundacionctic.org/>

<sup>2</sup> Universidad de Oviedo

Oviedo, Asturias, Spain

ivan.frade@gmail.com

<http://www.uniovi.es/>

**Abstract.** The Semantic Web is built upon distributed knowledge published on the Web. But this vision cannot be implemented without some basic publishing rules to make the data readable for machines. Publication of RDF vocabularies must receive special attention due to their important role in the Semantic Web architecture. In this paper we describe a scripting web-based application that validates the compliance of a vocabulary against these publication rules. Practical experimentation allows to illustrate and to discuss some common problems in the implementation of these rules.

## 1 Introduction

The Semantic Web is a big container, a universal medium for data, information and knowledge exchange. However the Semantic Web is not only about putting data on the Web, there are some publishing rules. Tim Berners-Lee outlined four basic principles [2] to publish Linked Data on the Web [5]. These rules describe how URIs must be used as names for things, and how to provide useful information on these things and other related ones. Although there are guidelines to coin adequate URIs for things [11], there is still the need to provide the best representation of the information for each request depending on each kind of client agent, human or software.

Web documents are retrieved using mainly the HTTP [8] protocol. This protocol provides a mechanism known as *content negotiation*. By means of content negotiation, it is possible to serve Web content in the format or language preferred by the requester (if it is available, obviously). Using transparent content negotiation in HTTP [9] has many benefits [12], and it can be implemented using different techniques in the Apache web server, as we describe in more detail in Section 2 of this paper. Section 3 introduces a scripting application that provides help and guidance to implement correctly and to debug HTTP content negotiation. In Section 4 the compliance of some of the most used vocabularies in the Semantic Web is evaluated with respect to the publishing rules. Finally, Section 5 presents some conclusions and future work.

## 2 Content negotiation with Apache: Recipes

Nowadays, the Apache HTTP Server is the most used Web server<sup>3</sup>, and it provides three different approaches to implement content negotiation<sup>4</sup>:

**Type Map:** Explicit handlers are described in a file (`.var`) for each resource. The necessary configuration is quite complicated and tedious, therefore this method is hardly used.

**MultiViews:** Based in the MIME-type and names of the files in a directory, MultiViews serves the most appropriate file in the current directory when the requested resource does not exist. It returns an additional header (`Content-Location`) to indicate the actual location of the file. This method can be extended using the Apache module `mod_mime` to associate handlers to new file extensions. However, this solution has a quite important problem: it only works if the files exist in the same directory.

**Rewrite request:** Probably because the two alternatives above do not provide an easy solution, the most widely used method is one which was not specifically designed to implement content negotiation. This mechanism uses the module `mod_rewrite` in order to rewrite the request according to some ad-hoc rules. As a result, requests (for objects that are not known to be information resources) are redirected using the HTTP 303 status code, to the URI of the appropriate content depending on the format requested. Obviously, some time is lost with the extra HTTP round-trip, but it is negligible for many applications, as well as mandatory according the `httpRange-14` resolution from the TAG<sup>5</sup>.

There is some ongoing work by W3C on *Best Practice Recipes for Publishing RDF Vocabularies* [3], a document which contains several recipes that advice on how to publish RDF/OWL Vocabularies using `mod_rewrite`. This “cookbook” provides step-by-step instructions to publish vocabularies on the Web, and gives example configurations designed to address the most common scenarios.

However, the *Recipes* are not perfect, and there is at least one important issue to be solved<sup>6</sup>. Tim Berners-Lee reported that “the recipe for responding to an accept header only responds to a header which EXACTLY matches [the rule antecedent]”. For those requests which contain values for the Accept header such as `text/*` or `application/rdf+xml;q=0.01`, where wildcards or *q*-values are used, the actual representation served by the rules proposed in the *Recipes* might differ from the expected one. This is a serious problem of the *Recipes*, but it can be easily solved using a script at server-side.

<sup>3</sup> <http://www.netcraft.com/survey/> (retrieved 13/Mar/2008)

<sup>4</sup> <http://httpd.apache.org/docs/2.0/content-negotiation.html>

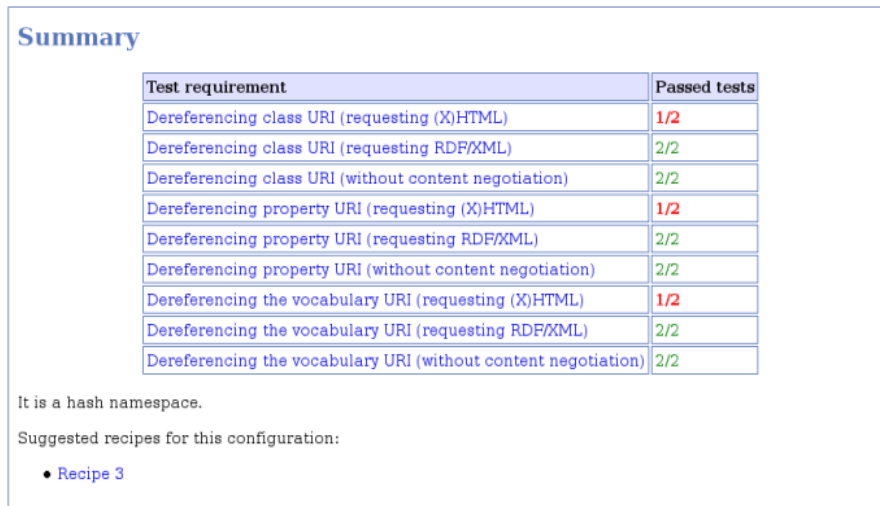
<sup>5</sup> <http://www.w3.org/2001/tag/issues.html#httpRange-14>

<sup>6</sup> <http://www.w3.org/2006/07/SWD/track/issues/58> (retrieved 13/Mar/2008)



### 3 Vapour: a scripting approach to debug content negotiation

The previous section has shown that a correct implementation of content negotiation is not an easy task. Furthermore, manually testing an implementation is not complex, but it is long and cumbersome. Although it can be done with tools such as cURL<sup>7</sup>, this process is not handy, specially for intensive or repetitive tests against a vocabulary.



The screenshot shows a report summary with a table of test requirements and passed tests. Below the table, it notes that the namespace is a hash namespace and suggests recipe 3.

Test requirement	Passed tests
Dereferencing class URI (requesting (X)HTML)	1/2
Dereferencing class URI (requesting RDF/XML)	2/2
Dereferencing class URI (without content negotiation)	2/2
Dereferencing property URI (requesting (X)HTML)	1/2
Dereferencing property URI (requesting RDF/XML)	2/2
Dereferencing property URI (without content negotiation)	2/2
Dereferencing the vocabulary URI (requesting (X)HTML)	1/2
Dereferencing the vocabulary URI (requesting RDF/XML)	2/2
Dereferencing the vocabulary URI (without content negotiation)	2/2

It is a hash namespace.

Suggested recipes for this configuration:

- [Recipe 3](#)

Fig. 1. Example of a report summary made by Vapour.

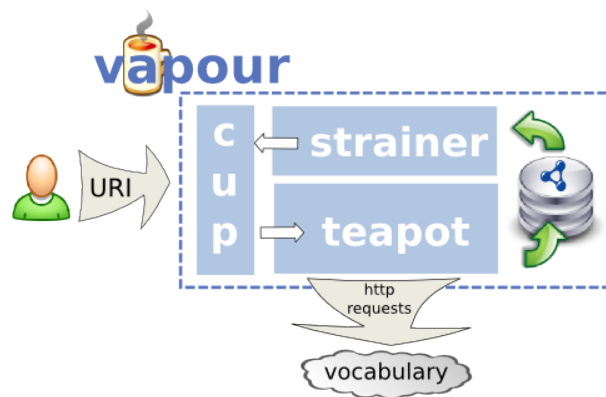
In order to facilitate the task of testing the results of content negotiation on a vocabulary, we developed a web-based application called Vapour<sup>8</sup>. This application provides a service that makes multiple requests to a set of URIs and runs a test suite specifically designed to check the responses of the server against the best practices defined in the *Recipes*. Tests are executed against the vocabulary URI, a class URI and a property URI (the latter two can be automatically detected). Based on the input parameters, the application provides a pointer to the specific recipe, in case the user wants to learn more on how to configure the web server. Vapour stores all assertions into an in-memory RDF store, using a combination of EARL [1], HTTP Vocabulary [10] and an RDF representation of the best practices of the *Recipes*. Thus Vapour can provide the

<sup>7</sup> Richard Cyganiak's explanation of how to use cURL to debug content negotiation, blog post available at: <http://dowhatimean.net/2007/02/debugging-semantic-web-sites-with-curl>

<sup>8</sup> <http://vapour.sourceforge.net/>

reports both in HTML and in RDF, using content negotiation. The HTML view displays a clean and concise pass/fail report of each set of tests (Figure 1), as well as a detailed explanation of its findings that includes a graphical representation of the HTTP dialog. Needless to say, the examples included in the *Recipes* are successfully validated by Vapour.

The application is written in Python, and it uses common Python libraries such as urllib, httplib, web.py and RDFLib. Scripting languages such as Python allow an agile development of applications in a short time with little resources. Source code of Vapour is available on SourceForge<sup>9</sup>, and an online demo of the service is also available<sup>10</sup>.



**Fig. 2.** High level architecture of Vapour.

As depicted in Figure 2, Vapour has a simple and functional design that fulfils the objectives of the project. There are three components:

**cup** is the web front-end. It uses the web.py framework and the Cheetah template engine, and it provides a web interface that allows the user to interact with other components of the application in a simple way. The architecture has been designed to allow other kind of interfaces. For instance, a command line interface is also provided.

**teapot** is the core of the application. It launches HTTP dialogs (with and without content negotiation) to evaluate the response status code and content-type. Teapot requests the URI of the vocabulary, and also the URIs of a class and a property from the vocabulary. All the resulting assertions are inserted into the RDF store.

**strainer** is the module in charge of generating the reports for each test performed by the application. It queries the RDF model using SPARQL to get

<sup>9</sup> <http://sourceforge.net/projects/vapour/>

<sup>10</sup> <http://idi.fundacionctic.org/vapour>

the result and trace of each test, and it produces a report in XHTML or RDF/XML. For the XHTML reports, we also use Cheetah templates.

The service can be deployed as a normal CGI in Apache or using a Python web framework. We reviewed the security of the application avoiding some common problems in this kind of applications, such as limiting requests per client.

## 4 Experimental results

Practical experimentation illustrates some common problems of how content negotiation is implemented, and enables the discussion on these problems. We checked some RDFS and OWL vocabularies published on the web. We chose the most frequently used vocabularies, in terms of number of instances, according to the last scientific study [7]. However, this ranking is aging (2004), so we also included some newer vocabularies, such as SKOS, DOAP and SIOC, which are also popular according to more up-to-date sources<sup>11</sup>.

**Table 1.** Ratio of passed tests / total tests for a list of widely used vocabularies in the semantic web.

Namespace	Accept RDF	Accept HTML	Default response
<a href="http://www.w3.org/1999/02/22-rdf-syntax-ns#">http://www.w3.org/1999/02/22-rdf-syntax-ns#</a>	3/3	N/A	RDF/XML
<a href="http://www.w3.org/2000/01/rdf-schema#">http://www.w3.org/2000/01/rdf-schema#</a>	3/3	N/A	RDF/XML
<a href="http://xmlns.com/foaf/0.1/">http://xmlns.com/foaf/0.1/</a>	3/3	3/3	HTML
<a href="http://purl.org/dc/elements/1.1/">http://purl.org/dc/elements/1.1/</a>	2/2	0/2	RDF/XML
<a href="http://www.w3.org/2003/01/geo/wgs84_pos#">http://www.w3.org/2003/01/geo/wgs84_pos#</a>	3/3	0/3	RDF/XML
<a href="http://rdfs.org/sioc/ns#">http://rdfs.org/sioc/ns#</a>	3/3	0/3	RDF/XML
<a href="http://www.w3.org/2004/02/skos/core#">http://www.w3.org/2004/02/skos/core#</a>	3/3	3/3	RDF/XML
<a href="http://usefulinc.com/ns/doap#">http://usefulinc.com/ns/doap#</a>	3/3	0/3	RDF/XML
<a href="http://purl.org/rss/1.0/">http://purl.org/rss/1.0/</a>	1/3	0/3	HTML
<a href="http://semantic-mediawiki.org/swivt/1.0#">http://semantic-mediawiki.org/swivt/1.0#</a>	0/3	0/3	text/plain

Table 1 summarizes the results of running Vapour against a list of ten popular vocabularies of the semantic web. These results provide an approximation to the quality of the publication of vocabularies on the web. All the vocabularies were retrieved on 12/Mar/2008. For each vocabulary, the vocabulary URI, a class URI and a property URI were tested (except for Dublin Core, which does not have any class). The results show that most vocabularies are correctly published as RDF. However, it is significant that most vocabularies do not correctly provide HTML representations of the resources, even if they are available. Additionally, some vocabularies return an incorrect MIME type, such as `text/plain` or `application/xml`.

<sup>11</sup> See the ranking at <http://pingthesemanticweb.com/stats/namespaces.php> (retrieved 12/Mar/2008) by PingTheSemanticWeb.com [6]

## 5 Conclusions

Content negotiation is a powerful technique. Although the basic mechanism is simple, it is often badly implemented. Vapour is useful to debug and to provide advice on how to solve common problems, as well as to provide quality assurance in the best possible way.

The application presented in this paper is fairly simple, but it actually helps to debug the implementation of content negotiation in web servers. It is particularly interesting that Vapour provides the results also in RDF . Using this machine-readable format, it should be easy to build another service on top of Vapour, and to use these data for other tasks, such as a service to check the compliance of a specific collection of vocabularies published on the Web.

Current best practices (and consequently, Vapour) should probably be updated to cover new methods to publish RDF data, such as RDFa [4] embedded in XHTML pages. In the future, we would like to extend Vapour to cover more generic validations in Linked Open Data scenarios, and to help webmasters to better understand some common implementation issues.

## References

1. S. Abou-Zahra. Evaluation and Report Language (EARL). Working draft, W3C, 2007.
2. T. Berners-Lee. Linked Data Design Issues. Available at <http://www.w3.org/DesignIssues/LinkedData.html>, 2006.
3. D. Berrueta and J. Phipps. Best Practice Recipes for Publishing RDF Vocabularies. Working draft, W3C, 2008.
4. M. Birbeck, S. Pemberton, and B. Adida. RDFa Syntax, a collection of attributes for layering RDF on XML languages. Technical report, W3C, 2006.
5. C. Bizer, R. Cyganiak, and T. Heath. How to Publish Linked Data on the Web. Available at <http://www4.wiwiw.fu-berlin.de/bizer/pub/LinkedDataTutorial/>, 2007.
6. U. Bojars, A. Passant, F. Giasson, and J. Breslin. An Architecture to Discover and Query Decentralized RDF Data. In *3rd Workshop on Scripting for the Semantic Web*, 2007.
7. L. Ding, L. Zhou, T. Finin, and A. Joshi. How the Semantic Web is Being Used: An Analysis of FOAF Documents. In *38th International Conference on System Sciences*, January 2005.
8. J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. RFC 2616: Hypertext Transfer Protocol - HTTP/1.1. RFC, IETF, 1999.
9. K. Holtman and A. Mutz. Transparent Content negotiation in HTTP. RFC, IETF, 1998.
10. J. Koch, C. A. Velasco, and S. Abou-Zahra. HTTP Vocabulary in RDF. Technical report, W3C, 2007.
11. L. Sauermann and R. Cyganiak. Cool URIs for the Semantic Web. Working draft, W3C, 2007.
12. S. Seshan, M. Stemm, and R. Katz. Benefits of Transparent Content Negotiation in HTTP. In *Proceedings of the IEEE Globcom 98 Internet Mini-Conference*, 1998.

# The Talia library platform

## Rapidly building a digital library on Rails

Michele Nucci<sup>1</sup>, Daniel Hahn<sup>2</sup>, and Michele Barbera<sup>2</sup>

<sup>1</sup> Semedia Group - 3mediaLabs  
Università Politecnica delle Marche  
`mik.nucci@gmail.com`

<sup>2</sup> NET7 - Pisa  
`hahn@netseven.it`

**Abstract.** Talia is a web-based distributed digital library and publishing system, designed for scholarly research in philosophy. Talia is based on semantic web technology; it's being developed with the Ruby on Rails web framework.

Rails is a relatively new environment, which allows developers to easily create well-structured web applications. By combining its power with semantic web technology and leveraging on existing solutions like ActiveRDF, it is possible to quickly create a full-featured semantic library platform, from scratch, in a short time.

Talia does not aim at creating new semantic web technology as such, but at providing practical solutions for embedding the existing technology in modern web applications.

This paper focuses on a few select features of Talia to show the possibilities.

## 1 Project

Discovery<sup>3</sup> is a European project to create an extensive online library for philosophers. The four content partners of the project will provide tens of thousands digitised and annotated pages, so that the project will start with a large body of material.

Philosophers usually work in a traditional, print-and-paper based way. However, finding all relevant publications on a topic is difficult and acquiring copies is yet another matter. This is especially true for original manuscripts, which can be notoriously hard to acquire [1].

By creating a comprehensive resource with original manuscripts and secondary writings, we can provide an invaluable tool to the scholarly research community.

Discovery will be based on *Philosource*, a federation of interlinked online libraries. The libraries currently use the *Hyper* platform. *Hyper* was created for the preceding *HyperNietzsche* project (now *NietzscheSource*<sup>4</sup>).

<sup>3</sup> <http://www.discovery-project.eu/>

<sup>4</sup> <http://www.nietzschesource.org/>

The *Hyper* platform was developed specifically for a Nietzsche library. Particular assumptions about Nietzsche are hardcoded into the application, and the codebase is not flexible enough to be a viable long-term solution for the project. It will be superseded by the *Talia* platform described in this paper.

In the humanities, the text itself is the subject of study. Talia aims to provide an integrated library system that offers tools to *work with original content*. This is in marked contrast to citation systems like CiteSeer<sup>5</sup>, CiteULike<sup>6</sup> or even online archives like arXiv<sup>7</sup>. These focus on bibliographical metadata, the actual content is not more than an opaque, downloadable document.

Talia provides a tight integration between a semantic backend store and a powerful interface toolkit, making it unlike existing “infrastructure” library systems, such as BRICKS [2] or Fedora<sup>8</sup>. These provide a backend on which an interface has to be built from scratch. The JeromeDL system [3] is somewhat similar to Talia, but aimed at a different audience.

## 2 Requirements and Technology

Within the Discovery project, Talia does not exist in a vacuum – it’s a means to an end. There are a number of features and components that *must* be implemented in Talia to make the software useful within the project:

- Users expect that the **User interface** contains all features that are already present on the Hyper platform.
- **Metadata and ontology support** is essential. Each group of scholars will create their own *domain ontology* to order their material.
- A **Remote Federation API** has to be implemented to allow automatic bidirectional references between documents in different libraries.
- **Publication and Workflow**. Scholars will be able to publish new results online. Talia must offer a number of workflow models for peer-reviewed publication.

A *speedy development* is also essential, since the existing content has to be migrated in the second year of the project and Talia must be running for the project to succeed.

The rapid development of Talia would not be possible without leveraging existing technology, both from commercial web development and semantic web research.

- **Ruby on Rails**<sup>9</sup> is a web development framework that has been picking up a lot of pace recently. It uses the Model-View-Controller [4] pattern and allows Developers to create full-featured web applications with a minimal amount of code. It’s available under the MIT license [5].

---

<sup>5</sup> <http://citeseer.ist.psu.edu/>

<sup>6</sup> <http://www.citeulike.org/>

<sup>7</sup> <http://arxiv.org/>

<sup>8</sup> <http://www.fedora-commons.org/>

<sup>9</sup> <http://www.rubyonrails.com/>

- **ActiveRDF**<sup>10</sup> [6] is a Ruby toolkit that provides an object-RDF mapping for a number of existing RDF triple stores.
- An **RDF store** is central to the application. During development the Redland RDF<sup>11</sup> engine is used, but Talia can work with any storage that is supported by ActiveRDF. The development team will also provide a setup for the Sesame<sup>12</sup>, store which supports inferencing.
- Of course Talia will also need “normal” web application features, like user sign-on and permission management.

As a general rule, Talia tries to avoid any unnecessary complexity. This is also true for the use of RDF metadata:

- Talia is schema-agnostic. Ontology descriptions may be used by the system; however the software does not rely on their presence. It also doesn’t attempt to enforce any kind of schema rules on the RDF data.
- Talia does not attempt to do any inferencing on the RDF data. If this is needed, it will be the responsibility of the RDF store.
- Talia tries not to use specialised RDF store features, in order to be compatible with any RDF endpoint.

### 3 RDF storage and querying

Talia uses a hybrid RDBMS/RDF store solution in the storage backend. A relational database as a highly reliable, transaction-aware storage for the critical data. It is kept in sync with an RDF data store for advanced semantic features, which may range from SPARQL queries to inferencing, depending on the type of the store. Using a standard RDF store also provides an easy way to interoperate with semantic web software.

The hybrid design is feasible because a Talia library has relatively static content. Data access will mostly be read-only, the few modifications can be easily synchronised without much overhead for the system.

Talia provides a simple API that hides most of the internal workings of the data store. Each document is represented as an object of type `Source`; the object provides access to all properties of the document, no matter if defined as RDF or not. Listing 1.1 shows an example of this API.

**Listing 1.1.** Basic Operations on a document

```
document = TaliaCore::Source.new("http://url.com/my_document")
document.workflow_state = 2 # non-rdf property
document.dcms::title << "My_first_document" # RDF property
author.inverse.dcms::creator # "inverse"
# Replace triple
document.dcms::title.replace("My_first_document", "New_name")
document.dcms::title.remove # remove triples
```

<sup>10</sup> <http://www.activerdf.org/>

<sup>11</sup> <http://librdf.org>

<sup>12</sup> <http://www.openrdf.org/>

The interface borrows heavily from the ActiveRDF interface, and ActiveRDF is used in the backend to connect to the RDF store. However, ActiveRDF was designed mostly as an easy read interface for web applications. The library was substantially refactored to improve the data manipulation capabilities (such as deleting triples). Other modifications were made to make it easier to call the library indirectly as part of a backend, instead of directly as a part of a script.

### 3.1 Queries

Talia provides an unified query interface for both database and RDF meta-data, as shown in listing 1.2. For normal queries, the interface hides most of the complexity and automatically decides whether to use a RDF/SPARQL or an RDBMS/SQL query on the backend.

#### Listing 1.2. Querying for documents

```
# The following will do a query on the RDF store
TaliaCore::Source.find(:all, N::DCNS.Creator => "Daniel")
# The following will LIMIT a query that uses RDF and DB data
TaliaCore::Source.find(:all, N::DCNS.Creator => "Daniel",
                      :workflow_state => 2, :limit => 5)
```

During development we found that the SPARQL [7] query language is not always best suited for web applications, where large result sets are usually broken down into individual *pages*. This is usually done by using the *LIMIT*, *OFFSET* and *COUNT* operators to retrieve the overall size and a subset of the (possibly huge) result set. This method requires, however, that whole query can be executed as a single statement.

SPARQL does not provide an easy way express *OR* statements in a single query (for example “subtype OR supertype”), and its *filter* mechanism is highly inefficient for large result sets. A straightforward *COUNT* implementation is also missing from the standard. Another problem is that in different RDF stores the SPARQL specifications is implemented to various extents, making it difficult to provide a store-agnostic solution.

Early versions of Talia also encountered the problem of reconciling “mixed” queries that use both the database and the RDF store. With the new hybrid design it will be possible to answer each query either from the RDF data or the database. This will allow the backend engine to select the query language best suited for the job.

The built-in query mechanism is optimised for compatibility with a number of RDF stores and RDBMS. If this is too limiting the developer has the choice issue queries (either SQL or SPARQL) directly and access store-specific features.

### 3.2 Ontologies

As mentioned, Talia’s core functionality does not rely on an ontology description. Still, if one is needed, it can easily be loaded into the RDF store and queried from Talia.



Talia provides a `SourceClass` abstraction to represent RDF classes and to navigate the ontology hierarchy, as shown in listing 1.3. The user may also retrieve meta-information from the ontology, supertypes or subtypes of a class.

**Listing 1.3.** Navigating the ontology

```
# Get the first rdf type and get sub- and superclasses
first_class = document.rdf_types.first
sup = first_class.supertypes
```

## 4 Semantic UI templates

`Source` objects can be used in HTML templates to create a web representation a document. Talia uses Rails' standard *rhtml* templates that contain HTML with embedded ruby code. Listing 1.4 show a simple template that renders a HTML snippet with some properties from the RDF store.

**Listing 1.4.** Sample rendering template

```
<p> The current document is <%= document.rdfs::label.first %>
it 's authored by <%= document.dcms::creator.join(",_") %> </p>
```

Talia needs a rich user interface for each document type. Unlike many semantic web applications that use an “auto-generated” generic interface for semantic metadata, Talia needs to provide specialised views depending on the RDF type of a resource.

A *automatic semantic template* engine is built into Talia to do just that. Listing 1.5 shows a simple example; the site developer simply passes the `Source` object to the `source_snippet` UI widget, and the semantic template engine does the rest.

**Listing 1.5.** Rendering a document with a semantic template

```
<% my_document = TaliaCore::Source.new(url ,
                                     N::MYONT::the_type) %>
<%= widget(:source_snippet, :source => my_document) %>
```

When the template engine renders a document, it will look at the document's RDF type and attempt to find a template which has a name that matches the namespace and name of one of those types. In the example the document has the type `myont:the_type`; if the template engine finds a template named `_myont_the_type.rhtml` it will use it to render the document. Otherwise it will fall back to a default template.

The template engine allows the UI templates to be easily created by professional web designers who don't know semantic web concepts. In the final web application, template selection happens automatically and it's very easy to add new templates. By providing a sensible default template, the engine is still able to deal with elements of new and unknown types.

## 5 Conclusions

This paper showed a quick glimpse of Talia's semantic web features and demonstrated how semantic web development can be made a breeze by combining the power of an existing framework with a dynamic RDF store API.

More semantic web features will be included in future version, like semantic links between remote libraries, user-created metadata and integration with the DBin desktop application.

Talia is freely available from its home page<sup>13</sup>, the page also contains some instructions and additional documentation for the software. There's also an online demo of the current development version<sup>14</sup>.

## Acknowledgements

This work has been supported by Discovery, an ECP 2005 CULT 038206 project under the EC eContentplus programme.

The authors wish to thank Eyal Oren and the ActiveRDF team for their work and support.

## References

1. D'Iorio, P.: Nietzsche on new paths: The hypernietzsche project and open scholarship on the web. In Fornari, C., Franzese, S., eds.: Friedrich Nietzsche. Edizioni e interpretazioni. Edizioni ETS, Pisa (2007)
2. Risse, T., Knežević, P., Meghini, C., Hecht, R., Basile, F.: The bricks infrastructure - an overview. In: The International Conference EVA, Moscow (2005)
3. Kruk, S., Woroniecki, T., Gzella, A., Dabrowski, M., McDaniel, B.: Anatomy of a social semantic library. In: European Semantic Web Conference. Volume Sematic Digital Library Tutorial. (2007)
4. Reenskaug, T.: MVC Xerox Parc 1978-79. <http://heim.ifi.uio.no/~trygver/themes/mvc/mvc-index.html> (1979 [accessed March 2008])
5. MIT: MIT License. <http://www.opensource.org/licenses/mit-license.php> ([accessed March 2008])
6. Oren, E., Debru, R., Gerke, S., Haller, A., Decker, S.: ActiveRDF: Object-Oriented Semantic Web Programming. In: 16th International World Wide Web Conference (WWW2007), Banff, Alberta, Canada. (8-12 May, 2007) 817-823
7. : SPARQL Query Language for RDF. <http://www.w3.org/TR/rdf-sparql-query/> (January 2008 [accessed March 2008])

---

<sup>13</sup> <http://trac.talia.discovery-project.eu/>

<sup>14</sup> <http://demo.talia.discovery-project.eu/> - note that this site may not always be online

# Scripting User Contributed Interlinking

Michael Hausenblas<sup>1</sup>, Wolfgang Halb<sup>1</sup>, and Yves Raimond<sup>2</sup>

<sup>1</sup> Institute of Information Systems and Information Management,  
JOANNEUM RESEARCH, Steyrergasse 17, 8010 Graz, Austria

`firstname.lastname@joanneum.at`

<sup>2</sup> Centre for Digital Music,  
Queen Mary, University of London, UK

`yves.raimond@elec.qmul.ac.uk`

**Abstract.** When building a linked-data dataset for humans and machines, a range of issues emerges. In this paper we discuss our findings regarding the implementation of *riese* (<http://riese.joanneum.at>), the RDFized and interlinked version of the Eurostat data. The contribution of our work is twofold: On the one hand we propose a new way of creating semantic links, labelled as User Contributed Interlinking, on the other hand we discuss integration issues regarding Ajax and embedded RDF-metadata.

## 1 Motivation

In early 2007 the Linking Open Data (LOD) community project has been launched within the W3C Semantic Web Education and Outreach (SWEO) group. The LOD community project bootstraps the Semantic Web by publishing datasets in RDF [1] on the Web. By creating large numbers of typed links between datasets [2, 3] Semantic Web application development is fostered.

Several issues emerge when building an LOD dataset; from the schema level—that is how to map from, e.g., a relational schema to an RDF Schema—to the proper and meaningful assignments of URIs to entities. One key success factor is the used interlinking method. Several approaches exist for semantically linking data.

With *riese* (“RDFizing and Interlinking the EuroStat Data Set Effort”) [4] we have contributed to the LOD cloud by adding the Eurostat data. The implementation of *riese* heavily depends on scripting languages such as PHP and JavaScript. In this paper we report on our findings when implementing the *riese* dataset. We introduced a new way of enriching datasets called “User Contributed Interlinking” (UCI), which is a Wiki-style approach enabling users to add semantic (that is: typed) links between data items on a URI-basis.

The paper is structured as follows: In section 2 we briefly introduce the LOD principles and discuss the current state of the LOD datasets. Then, in section 3 we explain the *riese* implementation, including its UCI-interface and Web 2.0 issues. In 4 we discuss the generalisation of the UCI. Finally, we conclude our findings in section 5.

## 2 A Linking Open Data Overview

According to Tim Berners-Lee<sup>3</sup> it is desirable to interlink datasets, thus allowing the discovery of more data. Interlinking is one of the key factors that made the (hypertext) Web so successful. The same principle holds for the Semantic Web; rather than creating (untyped) hyperlinks between different documents, semantic links are used to interlink data items from different sources.

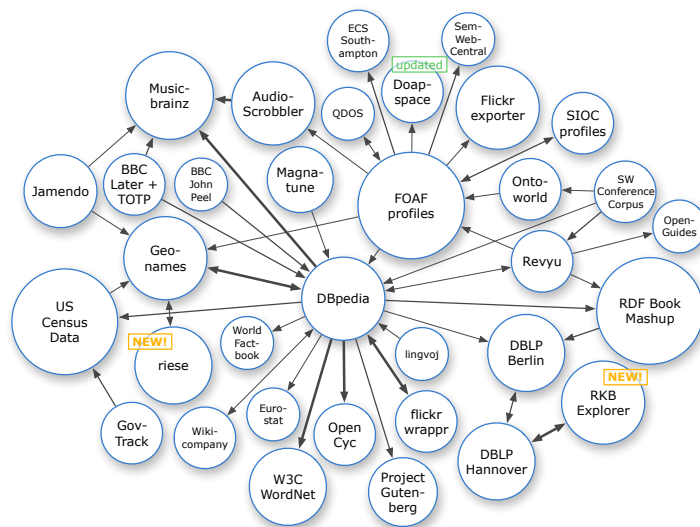


Fig. 1. The Linking Open Data dataset cloud in early 2008.

The Linking Open Data (LOD) project [2] is an open, collaborative effort carried out in the realm of the W3C SWE<sup>4</sup> community projects initiative. Notable landmarks within the LOD community project include the publication of common, real-world datasets such as Wikipedia, Geonames and Musicbrainz. Currently, the project includes over 30 different datasets (cf. Fig. 1, by courtesy of Richard Cyganiak<sup>5</sup>), ranging from rather centralised ones (such as DBpedia) to those that are very distributed (cf. the FOAF-o-sphere).

From one billion triples and 250k links in mid-2007 the LOD dataset has grown to over two billion triples and 3 million links in early 2008, representing a steadily growing, open implementation of the Linked Data principles detailed below. It should be noted that we do not view this dataset as a fixed, delineated

<sup>3</sup> <http://www.w3.org/DesignIssues/LinkedData.html>

<sup>4</sup> <http://www.w3.org/2001/sw/sweo/>

<sup>5</sup> <http://richard.cyganiak.de/2007/10/lod/>

or closed entity, but rather a snapshot of a major data ecosystem within the Semantic Web at this point in time.

*Linked Data Principles* The linked data principles read as follows:

1. All items should be identified using *URI references* (URIs)<sup>6</sup>, which implies that ideally no blank nodes are used<sup>7</sup>;
2. All URIs should be *dereferenceable*—using HTTP URIs allows looking up the items identified through URIs; see also the so called “http-range-14 TAG finding”<sup>8</sup>);
3. When looking up an URI—that is, a property is interpreted as a hyperlink—it leads to more data, which is usually referred to as the follow-your-nose principle [5];
4. Links to other URIs should be included in order to enable the discovery of more data.

*Interlinking* [6] describes how to publish linked data, and further discusses the two basic approaches for creating links to other datasets. Generally speaking, the RDF links can either be set manually or generated by automated linking algorithms for large datasets. For the latter case Raimond et.al. [7] have shown that simple interlinking algorithms produce rather poor results.

Naive approaches trying to perform a simple literal lookup are likely to fail; for instance, when trying to interlink data from the geographical domain with Geonames it is possible to do a simple literal lookup using the search facility provided by Geonames. However, when querying for the city Vienna almost 20 results will be returned as there exist that many cities named Vienna around the world. Advanced approaches such as described in [7] are needed to disambiguate similar matches and finally create appropriate interlinks. Still, there is no guarantee that the automatically generated interlinks are truly relevant. Moreover the automated process is also restricted to predefined datasets implying that only a subset of the data available on the Semantic Web is considered when looking for potential interlinks.

### 3 riese: Scripting LOD for humans and machines

With *riese* (launched in early 2008), we aim at offering a Semantic Web version of the publicly accessible data provided by the Eurostat data source, for both humans and machines. We currently serve some 3.6 million RDF triple in total, including the interlinking to Geonames data. There is ongoing work to cover the total Eurostat data (yielding some 4 billion RDF triple) and extending the interlinks to DBpedia, WordNet and other LOD data sets.

In Fig. 2 the *riese* system architecture is depicted. The data from Eurostat is converted into RDF/XML using SWI-Prolog, and dumped into the file system.

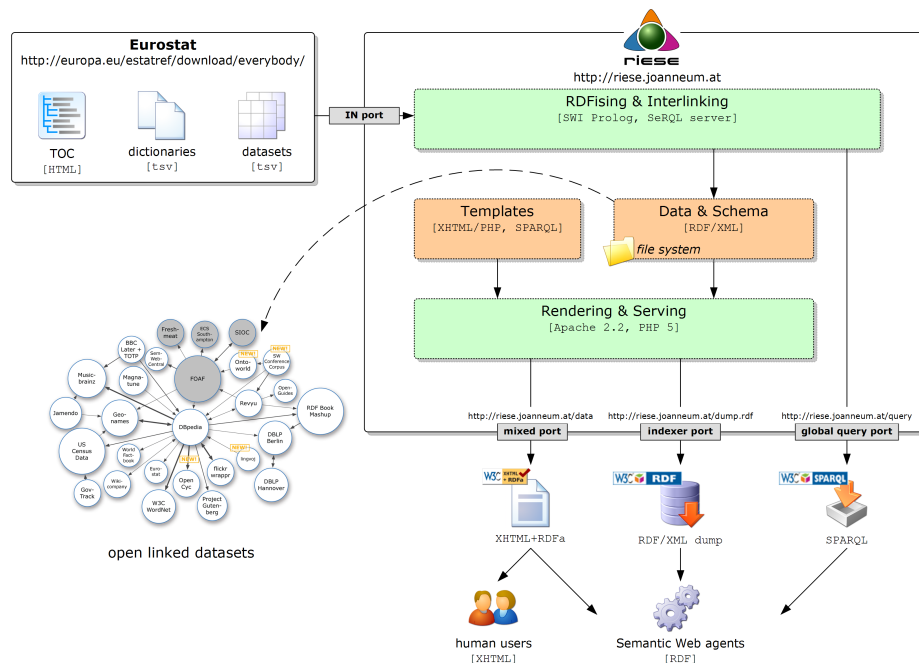
<sup>6</sup> <http://www.w3.org/TR/rdf-concepts/#section-Graph-URIref>

<sup>7</sup> <http://iandavis.com/blog/2007/03/bnodes-out>

<sup>8</sup> <http://www.w3.org/2001/tag/doc/httpRange-14/HttpRange-14.html>

This is to say that for each table—in tab separated values (TSV) format—from the Eurostat download<sup>9</sup> a corresponding RDF/XML (`content.rdf`) file, holding the statistical data, exists. The *riese* core schema is modelled using RDF-Schema [8] and comprises three main classes: `riese:Dataset`, `riese:Item` and `riese:Dimension`. A dataset is the logical container of either more sub-datasets (related via `skos:narrower`) or data items. We refer to [4] for further details on the modelling issues of the schema.

An Apache 2 Server along with a set of PHP scripts is used to render the pages in XHTML+RDFa [9]. The *riese* front-end is very light-weight; some 450 LOC in PHP and ca. 130 LOC in JavaScript were necessary to create a pleasant yet functional Web-based user interface.



**Fig. 2.** The system architecture of *riese*.

Additional to the statistical data available for both humans and machines, *riese* offers another novel feature: we have implemented a User Contributed Interlinking (UCI) interface, discussed in the following.

<sup>9</sup> <http://europa.eu/estatref/download/everybody/>

### 3.1 UCI in riese

The User Contributed Interlinking (UCI) part of riese, the UCI-interface, can be understood as an agent in the sense of [10]. The UCI-interface allows to list, add, and remove user-contributed semantic links from each of the statistical data items<sup>10</sup>.

Operation	Query String
list semantic links of the data item <code>sURI</code>	<code>?src=sURI</code>
add a semantic link to the data item <code>sURI</code>	<code>?src=sURI&amp;property=pURI&amp;target=tURI</code>
remove a semantic link from the data item <code>sURI</code>	<code>?src=sURI&amp;property=pURI&amp;target=tURI&amp;remove</code>

**Table 1.** Supported operations of the UCI-interface.

The operations supported by the current version of the UCI-interface are listed in Table 1. Note that the base service URI `http://ries.e.joanneum.at/interlinking/uci-interface.php` is assumed. With an additional `format` parameter the output format can be controlled. The default format is XHTML, an RDF/XML representation can be obtained using `format=RDF`.

To avoid concurrent editing a simple lock mechanism has been implemented. In case two users simultaneously want to add a semantic link to a data item, an according “please-hold-the-line” message is displayed.

It has to be noted that the UCI data is kept in a separate document—that is, a separate RDF/XML document, `uci-store.rdf`, per data item—in order to allow updates independently from statistical-data updates.

```
1 @PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
2 @PREFIX foaf: <http://xmlns.com/foaf/0.1/> .
3 @PREFIX riesed: <http://ries.e.joanneum.at/data/>.
4
5 <riesed:economy>
6   rdfs:seeAlso <http://www.unece.org/Welcome.html> ;
7   foaf:topic <http://dbpedia.org/resource/Economy> .
```

**Listing 1.1.** An example result from a UCI query.

To obtain, for example, an RDF representation of the UCI data for the data item `http://ries.e.joanneum.at/data/economy`, one would use the query

<sup>10</sup> `http://ries.e.joanneum.at/data/`

string?src=http://riese.joanneum.at/data/economy/&format=RDF. The result (rendered in RDF/N3 for better readability, here) would then be as shown in listing 1.1.

The HTTP-GET-interface of the UCI itself contains some 270 lines of code (LOC) in PHP, extensively making use of the RAP library<sup>11</sup>.

*UCI User Interface* On the client side we have implemented an user interface that controls the UCI-interface using Ajax (the UCI-UI). The Yahoo! User Interface Library (YUI)<sup>12</sup> has been utilised for panels, events, etc. but also for the asynchronous communication. The UCI data is merged into the UCI user interface at rendering time. Fig. 3 shows the UCI user interface “launch pad”: For each data item a user may choose to add semantic links using the “I know more” button, effectively launching the UCI-UI.

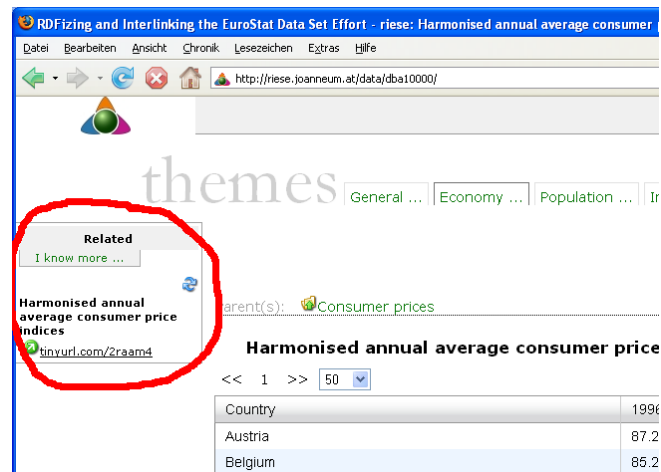


Fig. 3. UCI in riese - I.

In Fig. 4 the main UCI panel is depicted. Users can view, add, and remove semantic links with it. Note how the subject of the RDF statement is implicitly set to the data item from which it has been fired. Currently three semantic link types (properties) are supported (`owl:sameAs`, `rds:seeAlso`, and `foaf:topic`). We decided to control this part of the RDF statement as well strictly to (i) make it easier to use for the average user from the street, and (ii) to avoid issues when following-your-nose. Finally, the object of the RDF statement is the open part of the UCI data. With open we mean that is is up to the user to determine what URI to paste in. However, people are encouraged to use URIs pointing to RDF

<sup>11</sup> <http://www4.wiwiw.fu-berlin.de/bizer/rdfapi/>

<sup>12</sup> <http://developer.yahoo.com/yui/>





Fig. 4. UCI in riese - II.

(or GRDDL-able) resources. The UCI implementation is still in an early stage; further investigations both regarding scalability and usability are under way.

### 3.2 Issues in Web (2.0) Environments

*Issues with embedded metadata and Ajax* As described in [11] issues such as the do-not-repeat-yourself principle, the locality of structured data, or self-containment of descriptions need to be addressed when embedding metadata in (X)HTML. As these are generic issues, they are not limited to a specific methodology or technology, such as microformats<sup>13</sup>, eRDF<sup>14</sup>, or RDFa. For a deeper discussion of these issues the reader is referred to [12].

We have encountered issues with the in-place creation of RDFa in the utilised Ajax framework (YUI). Whenever rendering the metadata—expressed in RDFa, in our case—directly in the DOM, a non-DOM-based extractor is not aware of the RDF, hence unable to make use out of it. Take for example the RDFaDistiller<sup>15</sup>, a REST-based, conforming RDFa-processor. When RDFaDistiller fetches the content from a data item it is not able to access the DOM-only parts, hence they are lost. This seems to be a general problem when using embedded metadata along with dynamic content. We are not aware of a fix allowing a generic solution. We note, however, that for example in a Last Call comment to RDFa this has been recorded as a known issue to be addressed in future versions of this standard<sup>16</sup>.

*Access of Semantic Web data sources* Another integration issue turned out to be the access of RDF-based resources. The use case in riese reads as follows: When new data is available, one way to signal this is to subscribe to a news feed. We

<sup>13</sup> <http://microformats.org/>

<sup>14</sup> <http://research.talis.com/2005/erdf/wiki/Main/RdfInHtml>

<sup>15</sup> <http://www.w3.org/2007/08/pyRdfa/>

<sup>16</sup> <http://www.w3.org/2006/07/SWD/track/issues/114>

chose Atom [13] as the news feed format, as an corresponding RDF vocabulary (AtomOwl [14]) exists.

```
1 <body about="http://riese.joanneum.at/updates" instanceof="awol:Feed">
2 <div rel="awol:title" instanceof="awol:Content">
3 <span property="awol:body">updates</span>
4 </div>
5 <div id="main-updates">
6 <ul rel="awol:entry" instanceof="awol:Entry">
7 <li rel="awol:title" instanceof="awol:Content">
8 <span property="awol:body">Compensation of employees - NACE J-K -
9 Current prices - Millions of euro - SA</span>:
10 <span rel="awol:link" instanceof="awol:Link">
11 <a rel="awol:to" href="http://riese.joanneum.at/data/na075">
12 http://riese.joanneum.at/data/na075</a>
13 </span>
14 </li>
</ul>
```

**Listing 1.2.** An AtomOwl data update example in XHTML+RDFa.

On the riese updates page (<http://riese.joanneum.at/updates/>) the data news feed is made available in AtomOwl. The AtomOwl feed in turn is serialised as XHTML+RDFa; see listing 1.2 for an excerpt of the updates page.

Using AtomOwl over XHTML+RDFa allows both humans and machines to consume the data updates properly. A human user directly accessing the page is able to view the updates, a Semantic Web agent capable of understanding XHTML+RDFa can process the feed entries for its purposes. A real-world example of how to use the AtomOwl-feed is provided in the following. In this experiment we have programmed SPARQLBot<sup>17</sup> to access and query the AtomOwl embedded in the riese updates page. SPARQLBot offers a Web-based interface to define commands, which in turn maps to a SPARQL query (shown in listing 1.3).

```
1 PREFIX aowl: <http://bblfish.net/work/atom-owl/2006-06-06/#>
2 PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
3
4 SELECT DISTINCT ?headline ?feed WHERE {
5 ?feed rdf:type aowl:Feed ;
6       aowl:entry ?entry .
7 ?entry aowl:title ?eTitle .
8 ?eTitle aowl:body ?headline .
9 }
10 LIMIT 10
```

**Listing 1.3.** A SPARQL query for data updates on riese.

<sup>17</sup> <http://semsol.org/semcamp/sparqlbot>

Eventually, the same procedure can be applied to other scenarios, for example, when attempting to consume news feeds in an online news-reader, such as netvibes.com. It can be seen that certain indirections are necessary, however we are confident that with the growing support of Semantic Web technologies—as recently indicated by Yahoo!<sup>18</sup>—the burdens are likely to vanish.

## 4 Towards Generalising User Contributed Interlinking

With the User Contributed Interlinking (UCI) we have proposed a novel approach for creating high-quality interlinks by relying on the users. The UCI approach is motivated by the observation that generic, template-based algorithms (such as described in [7]) are limited regarding the *quality* of the typed links.

For large datasets such as *riese* where the entire European statistics are brought to the Semantic Web it might appear impractical at first sight to manually generate interlinks to other datasets. It is obvious that it is not feasible to have one person dedicated to manually looking for adequate related sources. However, by applying the Wiki-principle we want to initiate a crowdsourcing process that encourages users to contribute to linked datasets with similar enthusiasm as they already show in the case of Wikipedia. It has to be noted that the proposed UCI-feature is in an early stage of development and the first of its kind. The current implementation as it can be found in *riese* is meant to bootstrap the community-involvement in the area of linked datasets. It should be adapted to other datasets as well. Based on the experiences gained with the first release of UCI the system and the related processes will be refined. User acceptance is the critical success factor of UCI and therefore we aim at implementing as many of the best practices of Wikipedia as possible.

Sanger [15] was actively involved in the beginning of Wikipedia and has identified several factors that led to the great success of the platform such as openness and ease of editing. By inviting everybody to contribute we clearly highlight the openness of UCI. In addition we are working on enhancing the user experience by constantly improving the user interface design and keeping the user requirements at an absolute minimum as for instance no registration is required for using the UCI.

One of the disadvantages of common Wikis as identified in [16] is the limitation that “Wiki content is generally not available in a machine-processable format”. With UCI we directly address this issue as the target outcome RDF is machine-processable per se. There are nevertheless still challenges left, such as reaching a critical mass of contributors by providing appropriate incentives or addressing data provenance issues. However, we would like to see a conversion of the strong community-engagement from Web 2.0 to the Semantic Web and contribute to the initiation of this transformation by providing useful tools such as the UCI.

As a next step, we have prototypically implemented a generalised UCI in a demonstrator called IRS (which is for interlinking of resources with semantics).

---

<sup>18</sup> <http://www.ysearchblog.com/archives/000527.html>

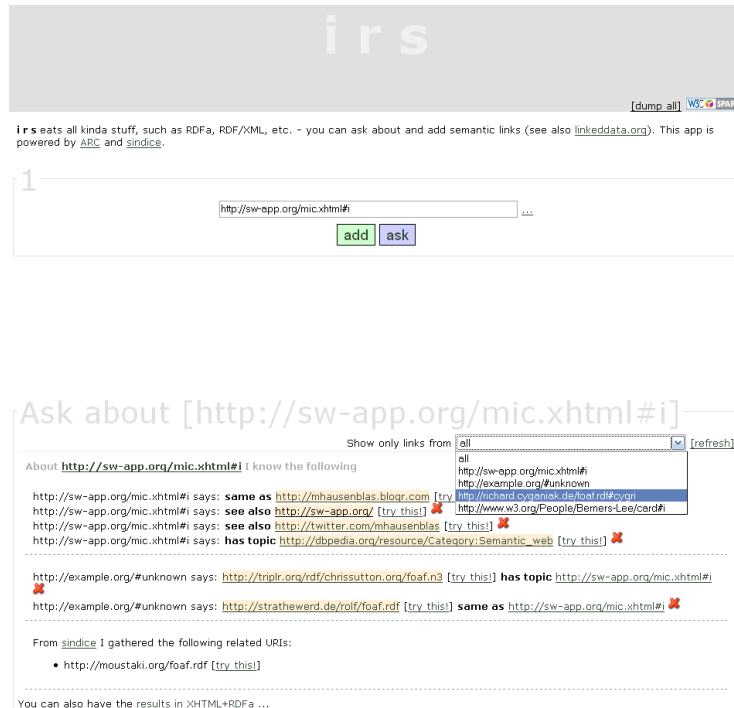


Fig. 5. A demonstrator for a generalised UCI: IRS.

The IRS demonstrator—implemented with ARC<sup>19</sup>—is available for testing purposes at <http://143.224.254.32/irs/>. A screen shot of IRS is shown in Fig. 5; it enables users to create semantic links (currently `owl:sameAs`, `rds:seeAlso`, and `foaf:topic`), to ask about existing links and to preview the (RDF) content. Further, a simple version of provenance tracking is offered: By placing the statements into a named graph (default is `http://example.org/#unknown`), one can track down who stated what. A simple off-the-shelf SPARQL-endpoint is also available in IRS.

## 5 Discussion and Conclusion

While the Semantic Web itself may be regarded as a (backbone) infrastructure, developers of Semantic Web applications have to be aware of issues arising with it.

In this paper we have presented a Wiki-style approach for user contributed (semantic) interlinking (UCI) in general, along with a discussion of tangible

<sup>19</sup> <http://arc.semsol.org/home>

results. First we have implemented the UCI within *riese*, the RDFized and inter-linked version of the European statistics. We have also addressed issues emerging from using Semantic Web technologies in Web 2.0 (Ajax) environments.

With UCI we have showcased an approach potentially increasing the end-user involvement in the Semantic Web. The acceptance of such features by the community is crucial, hence we will keep working on improving the tools in order to provide an enjoyable user experience.

Due to the usage of scripting languages, an efficient and effective development of *riese* (and *IRS*, alike) was made possible. The feature-testing cycle was kept to a minimum; from a developer's perspective it was possible to focus on the important issue: functionality rather than configuration and heavy framework-study.

## Acknowledgements

The research reported in this paper was carried out in two projects: the “Knowledge Space of semantic inference for automatic annotation and retrieval of multimedia content” (K-Space) project<sup>20</sup>, partially funded under the 6th Framework Programme of the European Commission, and the “Understanding Advertising” (UAd) project<sup>21</sup>, funded by the Austrian FIT-IT Programme.

The authors would further like to thank the following people for their lively input, discussions and support in implementation issues: Danny Ayers, Benjamin Nowack, Tom Heath, and Richard Cyganiak.

## References

1. G. Klyne, J. J. Carroll, and B. McBride. RDF/XML Syntax Specification (Revised). <http://www.w3.org/TR/rdf-concepts/>, 2004.
2. C. Bizer, T. Heath, D. Ayers, and Y. Raimond. Interlinking Open Data on the Web (Poster). In *4th European Semantic Web Conference (ESWC2007)*, pages 802–815, 2007.
3. G. Tummarello, R. Delbru, and E. Oren. Sindice.com: Weaving the Open Linked Data. In *The Semantic Web, 6th International Semantic Web Conference, 2nd Asian Semantic Web Conference, ISWC 2007 + ASWC 2007*, pages 552–565, 2007.
4. W. Halb, Y. Raimond, and M. Hausenblas. Building Linked Data For Both Humans and Machines. In *WWW 2008 Workshop: Linked Data on the Web (LDOW2008)*, Beijing, China, 2008.
5. L. Sauer mann, R. Cyganiak, and M. Völkel. Cool URIs for the Semantic Web. W3C Editor's Draft, W3C Semantic Web Education and Outreach Interest Group., 2007.
6. C. Bizer, R. Cyganiak, and T. Heath. How to Publish Linked Data on the Web. <http://sites.wiwiss.fu-berlin.de/suhl/bizer/pub/LinkedDataTutorial/>, 2007.

<sup>20</sup> <http://kspace.qmul.net/>

<sup>21</sup> <http://www.sembase.at/index.php/UAd>

7. Y. Raimond, C. Sutton, and M. Sandler. Automatic Interlinking of Music Datasets on the Semantic Web. In *WWW 2008 Workshop: Linked Data on the Web (LDOW2008)*, Beijing, China, 2008.
8. D. Brickley and R.V. Guha. RDF Vocabulary Description Language 1.0: RDF Schema. W3C Recommendation, RDF Core Working Group, 2004.
9. B. Adida, M. Birbeck, S. McCarron, and S. Pemberton. RDFa in XHTML: Syntax and Processing. W3C Working Draft 21 February 2008, W3C Semantic Web Deployment Working Group, 2007.
10. D. Ayers. Graph Farming. *IEEE Internet Computing*, 12(1):80–83, 2008.
11. B. Adida. hGRDDL: Bridging microformats and RDFa. *Web Semantics: Science, Services and Agents on the World Wide Web*, 6(1):54–60, 2008.
12. M. Hausenblas, W. Slany, and D. Ayers. A Performance and Scalability Metric for Virtual RDF Graphs. In *3rd Workshop on Scripting for the Semantic Web (SFSW07)*, Innsbruck, Austria, 2007.
13. M. Nottingham and R. Sayre. The Atom Syndication Format. RFC 4287, Network Working Group, 2005.
14. D. Ayers and H. Story. AtomOwl Vocabulary Specification . Namespace Document, Atom Owl Working Group, 2006.
15. L. Sanger. The Early History of Nupedia and Wikipedia: A Memoir. In C. DiBona, M. Stone, and D. Cooper, editors, *Open Sources 2.0: The Continuing Evolution*. O'Reilly, 2005.
16. D. E. O'Leary. Wikis: 'From Each According to His Knowledge'. *Computer*, 41(2):34–41, 2008.

# *World of WebCraft* — Mashing up World of Warcraft and the Web

Knud Möller

Digital Enterprise Research Institute, National University of Ireland, Galway  
`knud.moeller@deri.org`

**Abstract.** This short paper presents *World of WebCraft*, a set of tools which together allow players of the MMORPG World of Warcraft to generate photoblog-like Web representations of their in-game avatars. This is achieved by periodically logging information of the location of the avatar during the game, matching this information with in-game screenshots and then uploading them to Flickr, using machine-tags as annotations. Finally, an additional Web application uses the machine-tagged pictures to generate the photoblog. The tools are implemented using a combination of Lua and Ruby (two scripting languages), as well as Objective-C.

## 1 Introduction

So-called MMORPGs (Massively Multiplayer Online Role-playing Game) such as World of Warcraft (WoW) are online games in which players take the role of a fictitious, often mythical character (their *avatar*), explore a vast imaginary world, solve quests, battle enemies, and thus gain riches, knowledge and new skills within the game. Unlike traditional single-player games, an important factor in MMORPGs is the fact that players don't just interact with computer-generated and -controlled characters, but instead with the avatars of other human players who are online at the same time. In this way, players can form groups and tackle situations in which a single player would have failed. It is the interaction with others that lets these online games transcend simple computer games and move into the realm of online social networks (OSN), an area that is more commonly associated with platforms such as FaceBook, LinkedIn or Flickr: “*At its essence WoW is a social network; like minded people come together online to share a common experience and make connections*” [!] [6].

Users often choose to establish online identities in a number of different online social networking sites. Unfortunately, these sites tend to be closed, and their data not interlinked. E.g., this author has accounts on Flickr, FaceBook, LinkedIn, Xing, Twine, YouTube and probably others he has already forgotten about. Semantic Web technologies like FOAF<sup>1</sup> and SIOC<sup>2</sup>, as well as the very

---

<sup>1</sup> <http://www.foaf-project.org/>

<sup>2</sup> <http://www.sioc-project.org/>

recent and prominent Social Graph API<sup>3</sup> by Google have been proposed to cure this problem and connect the various online identities of a person into a whole [4,3]. Ignoring all potential problems which may occur when attempting to integrate data from all these different services together, their nature as *Web* services at least defines a general strategy for doing so: a person's identity on each site can be referenced by their profile identity URL, which can then be related using a vocabulary such as FOAF. In the case of MMORPGs however, this is not the case. While they do take place on the internet, they are not located on the Web. Consequently, a player's avatar also doesn't have a natural Web URL which could function as a reference.

This paper presents a strategy and (partial) implementation to overcome the barrier between the online, but non-Web world of MMORPGs and traditional OSNs. Taking the popular game World of Warcraft and the photo sharing site Flickr as an example, we will show how various technologies play together to generate a *World of WebCraft* — a mashup of World of Warcraft and the Web.

## 2 Architecture

Figure 1 shows a high-level overview of the approach taken for World of WebCraft. In this section, we will first describe the overall idea, and then focus each of the individual components in turn.

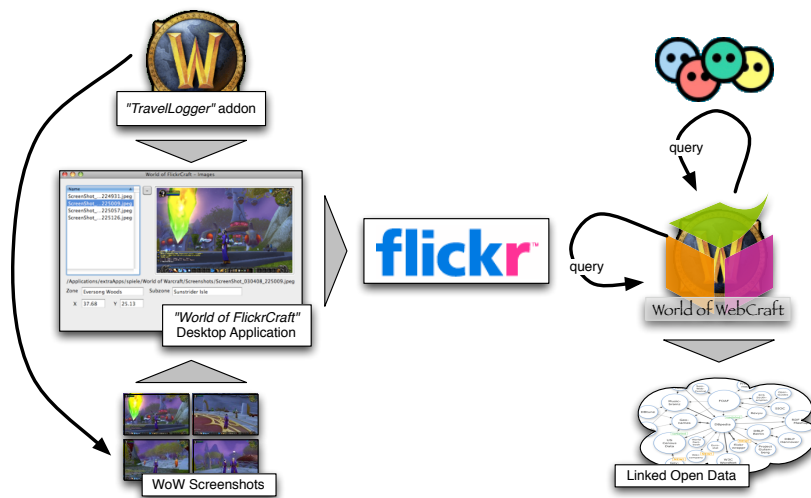


Fig. 1: World of WebCraft High-level Architecture

<sup>3</sup> <http://code.google.com/apis/socialgraph/>



In our setup, players can take screenshots of gameplay, which will automatically be tagged with information such as who the player is, when the screenshot was taken and where in the game world. This is possible by exploiting the dynamic nature of WoW's game engine, which allows it to write plugins (usually referred to as *addons*) to extend the game's functionality and user interface. A plugin called "TravelLogger" will periodically record the player's position in the world. After finishing a gaming session, a player can use the *World of FlickrCraft* desktop application to match up screenshots and log entries, and then upload the tagged screenshots to Flickr. While automatic tagging is useful in itself — it alleviates the player of the tedious task of manual tagging —, the tagged screenshots can now be used to generate a Web presence of the player's avatar through the *World of WebCraft* Web application<sup>4</sup>. By simply supplying their Flickr profile URL, the player enables World of WebCraft to generate a photographic online diary for their avatar — a photoblog for an orc! Alternatively, the player can also provide a link to their FOAF profile, given that it in turn points to their Flickr profile. Finally, World of WebCraft will make the avatar's photoblog accessible on a stable, meaningful URL and provide the diary data in a variety of formats (e.g., as RDF to contribute to the Web of Linked Open Data [1,2]).

## 2.1 *TravelLogger* — Using the World of Warcraft API with Lua

Almost all user interface elements in World of Warcraft (with the obvious exception of the 3D graphics) — windows, buttons, chat panes, etc. — are implemented on top of an engine which is based on the dynamic scripting language *Lua*<sup>5</sup>. Interaction between the UI and the game itself is handled through an events system. Almost anything that can happen in the game world — the player or any other object moves, monsters attack, spells are being cast, items sold and chat messages sent, etc. — will fire an event, which can be registered and acted upon by the Lua engine. WoW producer Blizzard have decided to open this system up for external developers. This makes it possible to write plugins (or *addons*) for WoW which can interact with the game events, extend the user interface, etc. A comprehensive documentation for the WoW API is available through [http://www.wowwiki.com/World\\_of\\_Warcraft\\_API](http://www.wowwiki.com/World_of_Warcraft_API).

The *TravelLogger* addon, which was developed as part of the World of WebCraft project, is a simple, light-weight plugin which makes use of the extensible WoW-engine to create a log of the avatar's movements through the game world. The plugin can be started and stopped from within the game using command `\tlog`. While running, the plugin will call the methods `GetPlayerMapPosition`, `GetZoneText` and `GetSubZoneText` in intervals of five seconds, to query both the precise geographical coordinates of the avatar, as well as a human-readable description of the approximate location (the *zone* and *subzone*). This data will

---

<sup>4</sup> <http://www.kantenwerk.org/wowc/>

<sup>5</sup> <http://www.lua.org/>

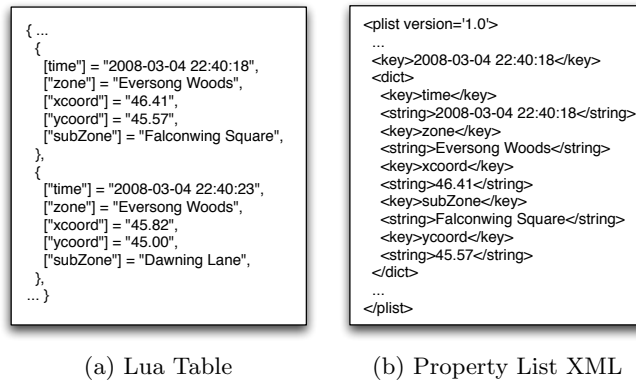


Fig. 2: The Travel Log in Lua and XML format

then be written out to a log file as a Lua table (see Fig. 2a). For security reasons, WoW only allows plugins to write in very specific locations of the file system, which follow the pattern below. Also, it is only possible to write Lua data structures (and not, e.g., a more common format such as XML).

```

WTF/Account/$ACCOUNT_NAME/$SERVER_NAME/
  $AVATAR_NAME/SavedVariables/$PLUGIN_NAME.lua

```

Even though there are now millions of registered WoW players and even more avatars, the `$SERVER_NAME` (which server is the player currently on) and `$AVATAR_NAME` together uniquely identify an avatar and can therefore later be used by World of WebCraft to mint a URI for this particular avatar.

## 2.2 *World of FlickrCraft* — A Ruby-based Screenshot Uploader

WoW is very restrictive in the actions it allows plugins to perform outside the game. It is for example not possible to write arbitrary files, let alone open a network connection. For this reason, if we want to get any data out of the game and onto the Web, we have to do so from outside the game. In our approach, we use a dedicated desktop application called “World of FlickrCraft” (or just FlickrCraft) for OS X to parse and interpret the output of the TravelLogger plugin described in Sect. 2.1, match the log with a number of in-game screenshots specified by the player and upload the screenshots to Flickr.

The application is implemented using a mixture of the Lua, Ruby and Objective-C (ObjC) languages, each chosen for specific tasks. Native OS X application development is based on the *Cocoa* libraries<sup>6</sup>, which are implemented in large parts in ObjC (an dynamic OO extension to C, heavily inspired by SmallTalk).

<sup>6</sup> <http://developer.apple.com/cocoa/>

However, through language bridges, it is also possible to access the full stack of Cocoa libraries through other languages such as Python or Ruby. For the benefit of fast prototyping, we have chosen Ruby over ObjC for the development of FlickrCraft, except in a few border cases.

In the remainder of the section, we will go through the individual steps of the application.

*Parsing the Log File* Since the output of TravelLogger is a Lua data structure, we have chosen Lua itself to parse and transform it to a format more accessible to the Cocoa libraries. From within the application, a Lua script process will be started, which parses the log and generate an XML structure from it (in the *Property List* format used throughout Cocoa), as shown in Fig. 2b. The output of the script is then piped back into FlickrCraft.

*Matching Screenshots with Log Entries* The player can now select a number of screenshots they have taken during the game (which probably show them accomplishing heroic feats or other noteworthy actions) and drag & drop them onto FlickrCraft. Based on the creation date attributes of the image files, the application will then perform a simple algorithm to select the log entry which is closest in time for each image. The image will be thus be associated with a number of tags, which consist of the `zone`, `subZone`, `xcoord` and `ycoord` attributes of the log entry, as well as the server and avatar name, which are extracted from the path of the log file.

*Uploading to Flickr* As a final step, FlickrCraft then uploads the selected screenshots to Flickr, using then *ObjectiveFlickr*<sup>7</sup> wrapper for the Flickr API. Two things are important during this phase: (i) the file creation dates are injected into each image file as the value of the EXIF<sup>8</sup> `DateTimeOriginal` attribute, in order for these dates to be picked up by Flickr after the upload, (ii) the tags are added both as simple tags (using just the values of the attribute value pairs from the log entries), but also as *machine tags* [5], which are Flickr's light-weight implementation of RDF-like metadata. Fig. 3 shows the uploaded screenshot in Flickr, along with both its simple and machine tags. The property names for the latter consist of a (not further specified) `wowc` namespace and the attribute names taken from the log file.

### 2.3 Photoblogs for Avatars with *World of WebCraft*

*World of WebCraft* (or WebCraft) is a Web application, which, by harvesting the machine tags uploaded to Flickr through FlickrCraft, allows users to generate a Web representation of their WoW avatars. The application follows a very simple series of steps: (i) The user is requested to provide their Flickr profile URL. (ii) WebCraft will then query Flickr for any pictures of this user which are tagged

<sup>7</sup> <http://lukhnos.org/objectiveflickr/blog/>

<sup>8</sup> <http://www.exif.org/>

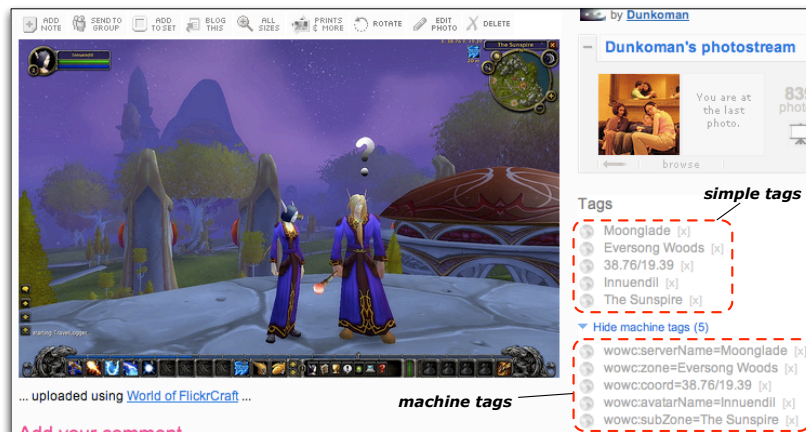


Fig. 3: Simple and Machine Tags in Flickr

with machine tags in the `wowc` namespace. (iii) The information in the tags, as well as the URLs of the images, are stored internally. (iv) For each avatar, a URL following the schema `http://kantenwerk.org/wowc/$SERVERNAME/$AVATARNAME` is minted. (v) Under this URL, the avatar's photoblog will be served in different formats (depending on the request sent to the server): JSON, an Exhibit webpage based on the JSON output, and as an RDF graph.

### 3 Conclusion and Future Work

We have presented an approach for mashing up the popular MMORPG World of Warcraft with the photo sharing service Flickr. Additionally, the World of WebCraft application exposes the data that a player wishes to publicise about their in-game avatars in a variety of formats, thus making it possible to link and mash up WoW with the Web. This opens up WoW's inherent, but largely untapped nature of an online social network.

Using the Exhibit framework, we visualise WoW data and screenshots as a timeline. By making use of Exhibit's map view and the GoogleMaps API, it would be possible to extend World of WebCraft to include a visual overview of where each in-game screenshot has been taken. A good example of how the GoogleMaps API is used on WoW data is `http://mapwow.com/`.

### 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. T. Berners-Lee. Linked data, 2006. <http://www.w3.org/DesignIssues/LinkedData.html>.
2. C. Bizer, R. Cyganiak, and T. Heath. How to publish linked data on the Web, 2007. <http://sites.wiwiw.fu-berlin.de/suhl/bizer/pub/LinkedDataTutorial/>.
3. U. Bojars, J. G. Breslin, A. Finn, and S. Decker. Using the Semantic Web for linking and reusing data across Web 2.0 communities. *Web Semantics*, 6(1):21–28, 2008.
4. J. G. Breslin, A. Harth, U. Bojars, and S. Decker. Towards Semantically-Interlinked Online Communities. In *The 2nd European Semantic Web Conference (ESWC '05), Heraklion, Greece, Proceedings, LNCS 3532*, pages 500–514, May 2005.
5. A. S. Cope. Machine tags, January 2007. <http://www.flickr.com/groups/api/discuss/72157594497877875/>.
6. A. LaFauce. What can social networks learn from World of Warcraft?, March 2008. <http://www.socialtimes.com/2008/01/what-can-social-networks-learn-from-world-of-warcraft/>.

# Neologism: Easy Vocabulary Publishing

Cosmin Basca<sup>1</sup>, Stéphane Corlosquet<sup>1</sup>, Richard Cyganiak<sup>1</sup>, Sergio Fernández<sup>2</sup>  
and Thomas Schandl<sup>1</sup>

<sup>1</sup> Digital Enterprise Research Institute  
National University of Ireland, Galway  
Galway, Ireland

`{firstname.surname}@deri.org`

<sup>2</sup> Fundación CTIC

Gijón, Asturias, Spain

`sergio.fernandez@fundacionctic.org`

**Abstract.** Creating, documenting, publishing and maintaining an RDF Schema vocabulary is a complex, time-consuming task. This makes vocabulary maintainers reluctant to evolve their creations quickly in response to user feedback; it prevents use of RDF for casual, ad-hoc data publication about niche topics; it leads to poorly documented vocabularies, and contributes to poor compliance of vocabularies with best-practice recommendations. Neologism is a web-based vocabulary editor and publishing system that dramatically reduces the time required to create, publish and modify vocabularies. By removing a lot of pain from this process, Neologism will contribute to a generally more interesting, relevant and standards-compliant Semantic Web.

## 1 Introduction

Anyone who wants to publish information as RDF on the Semantic Web first faces the choice which RDF Schema vocabulary or OWL ontology to use. Some areas, such as social networks (FOAF), online communities (SIOC) or general document metadata (DC) are covered by established vocabularies. Outside of these domains, registries like SchemaWeb<sup>3</sup> and search services like Falcons Concept Search<sup>4</sup> assist in the task of finding vocabularies for niche topics, but what they find might be of insufficient quality, or might not cover all required terms, and at present many areas of interest are not covered by any vocabulary at all.

In summary, most efforts to publish information on the Semantic Web first require an effort to create, extend or modify an RDF Schema vocabulary or OWL ontology. But this is a complex and time-consuming task in itself. It involves:

- Creating the formal specification of the vocabulary in RDFS or OWL,
- writing documentation that is clear and helpful for users of the ontology,
- keeping both documents in sync as the vocabulary evolves,

<sup>3</sup> <http://www.schemaweb.info/>

<sup>4</sup> <http://iws.seu.edu.cn/services/falcons/conceptsearch/>

- archiving older versions of the documents,
- defining and maintaining mappings to related vocabularies,
- configuring the web server in accordance with W3C best practices [3].

In this paper we present an online vocabulary editor and publishing system based on Drupal<sup>5</sup>, implemented in PHP and ActionScript, which will support vocabulary authors in the tasks above and thereby dramatically reduce the time required to create, publish and modify vocabularies. The work presented in this paper is in progress.

## 2 The value of vocabularies

We define vocabularies as simple, “lightweight” ontologies, such as FOAF, DC, SIOC and SKOS. They usually comprise less than 50 terms. Expressivity is limited to RDF Schema plus selected OWL features, e.g. inverse functional properties and class disjointness. Their value is in providing common terminology for exchanging information between programs. The actual information is in the RDF instance data that is expressed with the vocabulary’s terms, while in more complex ontologies, the actual information lies in the definitions of the classes and properties. A vocabulary is created by publishing a description of its terms in natural using HTML or formal using RDFS/OWL language. Since classes and properties are identified by URIs, it is considered a good practice to make these URIs resolvable [2, 3]. This enables clients to look up definitions of the vocabulary terms, with the following benefits:

- Information publishers can refer to a specification. This is important to create interoperability around a vocabulary. The top ten most popular vocabularies of 2006<sup>6</sup> all have a such a specification.
- RDF-aware tools such as data browsers (e.g. Tabulator [2]), SPARQL query builders and RDF instance editors can use the formal specification to improve the user experience, e.g. by showing friendlier labels and comments, listing available terms and providing widgets appropriate to a property’s data type.
- Inference can be performed to increase recall when performing queries or lookups against RDF data, which is especially useful when terms are mapped to other vocabularies. Systems that use such techniques are the Tabulator data browser [2] and the Sindice semantic lookup index [6].

## 3 Current approaches to vocabulary publishing

*Vocabulary maintenance with text editors and custom scripts.* Many popular vocabularies such as FOAF and SIOC are maintained by a process involving hand-authoring of RDF and HTML files and custom scripts, e.g. SpecGen<sup>7</sup>. Often, complex custom Apache configurations are employed to follow best practices regarding content negotiation, MIME types and resolvable URIs [3].

<sup>5</sup> <http://drupal.org/>

<sup>6</sup> <http://ebiquity.umbc.edu/resource/html/id/196/>

<sup>7</sup> <http://sioc-project.org/specgen>



**Fig. 1.** A vocabulary page in Neologism, as it appears to an authenticated user.

*Offline ontology editors.* OWL ontology editors such as Protégé [5], TopBraid Composer<sup>8</sup> and SWOOP<sup>9</sup> can be used to create the formal specification of a vocabulary. While being great tools for knowledge engineering professionals, these applications have a steep learning curve and they intimidate casual users. They use a file-based, offline model, where ontology files are stored on the local user's computer. Remote publishing, if supported at all, is an after-thought.

*Web-based systems.* OntoWiki [1] provides basic ontology editing, but its main focus is the display and editing of RDF instance data. MyOntology [7] focuses on collaborative editing in a larger community, in the hope of creating rich knowledge bases, while creation of simple vocabularies typically does not involve many collaborating users. Knoodl<sup>10</sup> is a hosted service with strong community features and an easy-to-use vocabulary editor, but it does not publish created vocabularies with resolvable URIs or according to best-practice guidelines.

*Areas for improvement.* We identify four points where we can simplify the process: (i) Instant web-based publishing instead of file-based offline editing. (ii) Focus on a limited subset of RDFS and OWL. (iii) No instance editing or browsing. (iv) Handling of HTTP details like URI management, content negotiation and redirects within the web-based application.

## 4 Easier vocabulary publishing with Neologism

Neologism<sup>11</sup> is a web-based vocabulary editor and publishing platform designed to address these issues. It is currently being implemented and will soon be released as an open-source project. This section presents Neologism's current state.

*Public interface.* To non-authenticated users on the Web, Neologism presents a very simple interface: a homepage that lists one or more vocabularies, and for each of them a *vocabulary page* containing some general information about the vocabulary (Figure 1), followed by the descriptions of all its classes and properties.

<sup>8</sup> <http://www.topbraidcomposer.com/>

<sup>9</sup> <http://www.mindswap.org/2004/SWOOP/>

<sup>10</sup> <http://knoodl.com/>

<sup>11</sup> <http://neologism.deri.ie/>



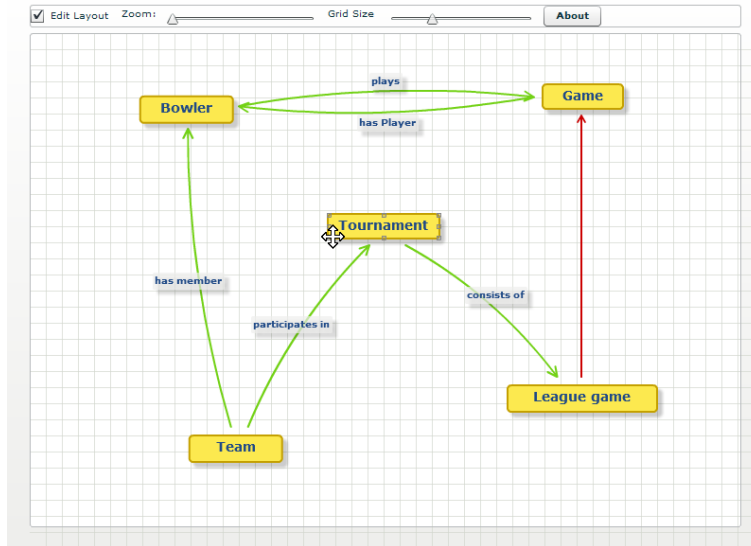
**Fig. 2.** A form for editing a class.

*Editor.* After a vocabulary maintainer logs in, additional links become visible on the vocabulary page and allow adding new terms, as well as editing of existing terms. Terms are created and edited through a web form (Figure 2). The form allows entry of an ID (to become part of the term’s URI), label, comment, subclasses, subproperties, domain, range, disjoint classes, inverse properties, and marking a property as inverse functional. Authenticated users can also create new vocabularies and modify the vocabulary metadata.

*Overview diagram.* The vocabulary page provides access to a diagram that shows the vocabulary’s classes and their relationships (Figure 3). The vocabulary maintainer can arrange the diagram into a sensible layout and then save its current state which will henceforth be shown to all users.

*RDFS output, URIs and content negotiation.* The URIs identifying classes and properties are always generated by appending the hash character and the term’s ID to the URI of the vocabulary page. This makes sure that the vocabulary page is returned when these URIs are resolved. HTTP requests to the vocabulary page are subject to content negotiation. Web browsers will see the HTML variant shown in Figure 1. RDF-aware clients will receive the RDFS/OWL specification, either in RDF/XML or N3 syntax. In a nutshell, Neologism publishes standards-compliant vocabularies on the Web without requiring any additional effort on the part of vocabulary maintainers.

*Implementation.* Neologism is implemented in PHP as a Drupal module. Drupal reduces development time by providing many features for free, such as account management. It also makes integration with a larger Drupal-based site very easy, for example to provide a news blog and discussion forum for each vocabulary. All



**Fig. 3.** The vocabulary overview diagram.

data is stored in a MySQL database. RAP<sup>12</sup> is used to serialize RDF/XML and N3. The PHP Content Negotiation library<sup>13</sup> is used instead of the usual Apache rules to implement content negotiation, and Vapour<sup>14</sup> was used to validate its correctness. The overview diagram is implemented using Adobe Flex and coded in ActionScript; the ObjectHandles and Tweeners libraries are used for animation and object handling.

## 5 Future Work

*Hosted Neologism service.* Currently, vocabulary maintainers must install Neologism on their own webspace. A central hosted service, which could be easily built on the Drupal platform, would remove this barrier.

*Branching and revision tracking.* Neologism does not yet offer revision control. Some desirable features for vocabulary revision control are: archival of all prior versions; grouping of several small edits into a single version to avoid putting the vocabulary into an inconsistent intermediate state; publishing changes as a draft before accepting them as a new version.

<sup>12</sup> <http://www4.wiwiw.fu-berlin.de/bizer/rdfapi/>

<sup>13</sup> <http://ptlis.net/source/php-content-negotiation/>

<sup>14</sup> <http://vapour.sourceforge.net/>

*Plugin system.* We intentionally kept the set of supported class and property annotations small to simplify the user experience, and don't support many possible further annotations, such as OWL cardinality constraints, plural and inverse labels<sup>15</sup>, multilingual labels or associating Fresnel lenses [4] with classes and properties. Such additional annotations could be supported through plugins that are installed by vocabulary maintainers.

*Consistency checking.* Neologism doesn't check the created vocabulary for consistency. This can become an issue when a vocabulary is integrated with several external vocabularies. A solution could be the integration of an external reasoning service that performs consistency checks and is invoked through an API over the Web.

## 6 Conclusion

We have shown a web-based vocabulary publishing system that simplifies the process of creating, publishing and maintaining RDF vocabularies by (i) instant web-based publishing, (ii) focus on a limited subset of RDFS and OWL, (iii) avoiding instance editing or browsing, and (iv) handling URI management and HTTP content negotiation. We hope that the presented system will encourage the creation of new vocabularies and thereby contribute to a generally more interesting, relevant and standards-compliant Semantic Web.

## References

1. S. Auer, S. Dietzold, and T. Riechert. OntoWiki, a tool for social, semantic collaboration. *The Semantic Web - ISWC 2006*, 4273/2006:736–749, 2006.
2. T. Berners-Lee, Y. Chen, L. Chilton, D. Connolly, R. Dhanaraj, J. Hollenbach, A. Lerer, , and D. Sheets. Tabulator: Exploring and Analyzing Linked Data on the Semantic Web. In *The 3rd International Semantic Web User Interaction Workshop (SWUI06)*, 2006.
3. D. Berrueta and J. Phipps. Best Practice Recipes for Publishing RDF Vocabularies. Working Draft, W3C, 2008.
4. C. Bizer, R. Lee, and E. Pietriga. Fresnel, a Browser-Independent Presentation Vocabulary for RDF. In *International Semantic Web Conference 2006*, 2006.
5. H. Knublauch, R. W. Ferguson, N. F. Noy, and M. A. Musen. The Protégé OWL Plugin: An open development environment for semantic web applications. *The Semantic Web ISWC 2004*, 3298/2004:229–243, 2004.
6. E. Oren, R. Delbru, M. Catasta, R. Cyganiak, H. Stenzhorn, and G. Tummarello. Sindice.com: A document-oriented lookup index for open linked data. *International Journal of Metadata, Semantics and Ontologies*, 3(1), 2008.
7. K. Siorpaes and M. Hepp. myOntology: The marriage of ontology engineering and collective intelligence. In *ESWC 2007 Workshop Bridging the Gap between Semantic Web and Web 2.0*, 2007.

---

<sup>15</sup> <http://www.wasab.dk/morten/2004/03/label>

# Microblogging: A Semantic and Distributed Approach

Alexandre Passant<sup>1</sup>, Tuukka Hastrup<sup>2</sup>, Uldis Bojārs<sup>2</sup>, John Breslin<sup>2</sup>

<sup>1</sup> LaLIC, Université Paris-Sorbonne,  
28 rue Serpente, 75006 Paris, France  
`firstname.lastname@paris4.sorbonne.fr`

<sup>2</sup> DERI, National University Of Ireland,  
Galway, Ireland

`firstname.lastname@deri.org`

**Abstract.** While microblogging has quickly gained a lot of interest in the Web 2.0 community, it still has not been leveraged to the Semantic Web unlike blogs and wikis. This paper describes the features, methods and architecture of a distributed Semantic Web microblogging system, as well as the implementation of an initial prototype of this concept that provides ways to leverage microblogging with the Linked Data Web guidelines.

**Key words:** Microblogging, Semantic Web, Web 2.0, SIOC, Data Portability, Linked Data Web

## 1 Introduction

Microblogging is one of the recent social phenomena of Web 2.0. It fills a gap between blogging and instant messaging, allowing people to publish short messages on the web about what they are currently doing. As a simple and agile form of communication in a fluid network of subscriptions, it offers new possibilities regarding lightweight information updates and exchange. Yet, current microblogging services are centralised and confined, and efforts are still to be made to let microblogging be part of the Social Semantic Web [5]. This is in stark contrast to blogs and wikis that can already be considered as components of the Semantic Web after a lot of work leveraging their data and metadata in machine-readable formats with projects like SIOC [6] and systems like Semantic MediaWiki [16].

In this paper, we introduce the main idea and a first implementation of distributed microblogging systems, enabled by Semantic Web technologies and providing machine-processable views of microblogging content and metadata. This way microblogging can become part of the Semantic Web as Linked Data [3]. First, we introduce classical microblogging and some of the issues it raises. Second, we see how Semantic Web can help in getting rid of these issues and what it can offer that traditional services could not achieve. Especially, we see how metadata and data can be represented using Semantic Web technologies to in-

terlink multiple services and related datasets. Third, we describe the functions of our prototype for distributed semantic microblogging and give an overview of the current source code of the system. Finally, we conclude with ideas for future work and with thoughts on connections between this paper and projects like data portability on Web 2.0 on one hand and relationships with existing microblogging services such as Twitter<sup>1</sup> on the other hand.

## 2 Overview of microblogging

### 2.1 Why microblogging?

After blogging that let people put their thoughts online to an open audience, podcasting where people record it and even videoblogging (also known as vlogging) where they deliver messages in video, microblogging enabled anyone to exchange short messages within their community or simply to write in brief to the general public on the Web. This new form of blogging allows individuals to publish brief text *updates* using a multitude of various communication channels such as text messages from cell phones, instant messaging, e-mail and the Web. The simplicity of publishing such short updates in various situations and in a fluid social network based on subscriptions and response posts makes microblogging a groundbreaking communication method that can be seen as a hybrid of blogging, instant messaging and status notifications, and that some already studied from a social point of view [11]. Moreover, this way of publishing can be extended with more advanced communication means like video recording, as in Seismic<sup>2</sup>, which is considered a video microblogging service.

This communication method is also promising for corporate environments in facilitating informal communication, learning and knowledge exchange. Its so far untapped potential can be compared to that of company-internal wikis some years ago. Microblogging can be characterised by rapid (almost real-time) knowledge exchange and fast propagation of new information. For a company, this can mean real-time Q&As and improved informal learning and communication, as well as status notifications, e.g. about upcoming meetings and deliveries. Yet, potential for microblogging in corporate environments still has to be demonstrated with real use cases, which we hope to happen in the next years, as already was the case for blogging, wikis and other Enterprise 2.0 [12] services.

Nevertheless, microblogging is currently mostly used by technically-minded Web users and bloggers. Twitter is one of the largest microblogging services and the value of microblogging is manifested by its popularity - now ranked at website number 635 in the world - and by Google's recent acquisition of Jaiku<sup>3</sup>, another leading microblogging service. Microblog-type publishing can also be setup on personal services, since for example the WordPress blogging software

---

<sup>1</sup> <http://twitter.com>

<sup>2</sup> <http://seismic.com>

<sup>3</sup> <http://jaiku.com>

offers a dedicated template interface (Prologue<sup>4</sup>) that lets people publish this kind of short and real-time updates. However, there is no aggregation for personal microblogs that would take into account the special characteristics of it as a new medium.

## 2.2 Current issues

While microblogging gained a lot of interest on the Web and quickly became one of the main knowledge management schemes in the Web 2.0 world, like blogs or wikis, it also raises various issues.

First, most microblogging services act as closed worlds like, actually, most of Web 2.0 services: only a few of them allow interlinking with other services. For example, merging your latest blog posts or your Flickr pictures with your Twitter updates cannot be done, except using simple HTML links between them, or using RSS. RSS provides syndication, i.e. real-time export of latest updates for a given user, but cannot be used to retrieve the complete update history at any later time. Moreover, those services do not expose their metadata in a way that could be easily reused. Twitter has adopted microformats for describing *follower* (subscriber) lists, but there is no simple way to retrieve metadata about the complete updates of any user (e.g. who did the update and when). One solution would be to combine the RSS feed of latest updates with the XML export of each update. Some scripts could then map them to Semantic Web vocabularies and URIs with potential use of external data, as SWAML [9] does to find people URIs<sup>5</sup>. Yet, the process can be quite complex, and since it is based on RSS, only the latest updates would be available.

In addition to these metadata concerns, the content of the updates does not carry any semantics, making its reuse difficult. Twitter users have adopted certain short-hand conventions in their writing called hash tags<sup>6</sup>, but their semantics are not readily machine-processable thus raising the same ambiguity and heterogeneity problems that tagging causes. For example, the *hash tag* #paris could mean various things (cities, people etc.) depending on the context, and so cannot be automatically processed by computers. This lack of data formalism also makes finding relevant content difficult. While some services provide plain-text search engines, there is no way to answer queries like "What are the latest updates talking about a programming language" or "What is happening now within ten kilometres from here".

Finally, one issue with current services is their centralised architecture. Most services do not act in a client-server way, but require users to post their updates on a given platform, which is the same for publishing and reading data. This means that most of the time, published data belongs to the publishing site, and cannot be automatically reused on multiple microblogging sites, or even re-used locally for other purposes. It can also be a problem to private communities, since

<sup>4</sup> <http://wordpress.com/blog/2008/01/28/introducing-prologue/>

<sup>5</sup> <http://www.wikier.org/blog/using-sindice-to-get-the-best-uri-for-a-person>

<sup>6</sup> <http://hashtags.org/>

users need to rely on an external service where they cannot completely control privacy and security.

We believe that the Semantic Web is an elegant solution to opening these data from proprietary silos and to providing machine-processable data and metadata to microblogging as well as to delivering an open and distributed environment for microblogging, as we will expose in the next section.

### 3 Architecture of a semantic microblogging service

#### 3.1 Metadata modelling

In order to model the metadata of a microblogging service, we rely on two widely used ontologies on the Social Semantic Web: FOAF [7] and SIOC.

As expected, the former is used to model microbloggers and their properties (name, email etc.). Using FOAF allows the reuse of an existing URI for a person that wants to start microblogging, instead of creating a new identity URI. Moreover, since some Web 2.0 services already offer FOAF export, either directly as LiveJournal<sup>7</sup> or thanks to external services as Flickr [13], people can reuse their existing URI from these services without having to dig in RDF modelling. Yet, in case the person needs for any reason to create a new instance of `foaf:Person`, Linked Data principles allow them to identify uniquely with an already existing profile via a `owl:sameAs` link.

While FOAF aims to model the people aspect, SIOC is used to define the related user contexts, providing a way to identify a user account on a given microblogging service. The SIOC model provides for one person subscribing to various services, i.e. a single `foaf:Person` can be connected to various `sioct:User`. This employs the distributed architecture of the Semantic Web to enable people to consolidate their identity across a network of services.

In addition to the account aspect, SIOC is used to model the microblogging service itself and the updates of any user. In order to do so, we extended the SIOC types module<sup>8</sup> [4] with two new types: `sioct:MicroblogPost` and `sioct:Microblog`, as respective subclasses of `sioct:Item` and `sioct:Container`, thus allowing a Microblog to contain (using `sioct:container_of`) instances of `MicroblogPost`. This also provides for modelling a microblog post with the same SIOC and FOAF properties as blog posts and wiki pages. Moreover, having such a class hierarchy of SIOC types allows people to access a dataset containing a set of `sioct:BlogPost`, `sioct:WikiArticle` and `sioct:MicroblogPost` with a single SPARQL query for all the data while leaving open the option to refine their search by restricting the type of the items to be retrieved.

Thus, modelling metadata in a machine-readable format is the first step in getting rid of proprietary data silos for microblogging content as it becomes possible to merge it with other Web 2.0 content that has been described in

<sup>7</sup> <http://community.livejournal.com/ljfoaf>

<sup>8</sup> <http://rdfs.org/sioc/types>

RDF. We can also rely on the connection between `sioc:User` and `foaf:Person` to find relevant content, e.g. answer queries like *"List all my activity on the Web during the first week of January"*, something that could not be done with non-semantic Web 2.0 data. To a large extent, this combination of FOAF and SIOC can be used as a solution to the data portability issues<sup>9</sup> since it relies on machine readable and interlinked data models to represent people, user accounts and data, as shown in Fig. 1

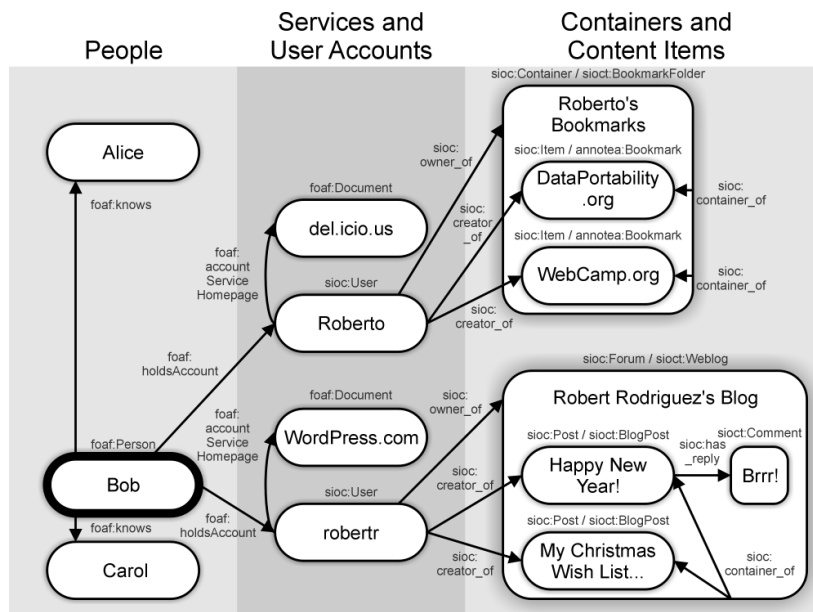


Fig. 1. FOAF and SIOC for data portability

### 3.2 Data modelling

While microblog posts are by nature relatively light in content, it is interesting to identify some of the data they contain, which is one of the problematic areas for current systems as mentioned earlier. While hash tags can be useful, there is a need to describe some content more formally because of the problems of plain text-only descriptions. Instead of plain text or tags, we think that using URIs and RDF to model this data can be useful for two reasons: (1) we rely on existing, unambiguous resource definitions to model the content and (2) we open microblogging entries to the Linked Data Web in the case these URIs are available on the Web and in the better case, already linked to other content, providing a path to the Giant Global Graph [2].

<sup>9</sup> <http://dataportability.org>



Thus, there is a need to (semi-)automatically extract those URIs or concepts from plain text or to let users annotate it similarly to what they can already do on Twitter with hash tags, but with more powerful processing that can extract and define URIs based on those tags. For example, rather than writing *"Visiting #Eiffel\_Tower in #Paris"*, someone could microblog *"Visiting #dbp:Eiffel\_Tower in #geo:Paris\_France"* so that the processor would be able to extract the two hash tags and thanks to a predefined prefix mapping process, query DBpedia [1] and GeoNames<sup>10</sup> to retrieve URIs of the related concepts. Thus, the updates would be automatically linked to existing URIs rather than to simple and meaningless – from a software agent point of view – text strings.

Such a way to extract data and to interlink with existing URIs makes content more easily searchable on the Semantic Web. Indeed, thanks to lookup services such as Sindice [15] that crawl the web for RDF data and links between documents, one could be suggested to look at the update above when searching for *"Eiffel Tower"*.

### 3.3 Distributed content

We want the microblogging system to be open and distributed, following the spirit of the Web architecture. We envision a multitude of publishing services and aggregation servers interacting with each other. A publishing service makes the posts of one or more authors available on the Web in RDF. When a new post is available, the service pings one or more aggregation servers, defined by the user, with the URI of the post that is then retrieved by the servers. As with blogs today, we expect some people to deploy their own publishing services while others use public ones, as well as aggregation servers that can be public or dedicated to private communities of interest.

An aggregation server receives pings from publishers and retrieves posts it deems relevant for further use. The relevancy depends on the nature of the aggregation function of each server. Some servers may have a strict list of sources they aggregate while others try to provide inclusive views on the global activity on the Web. In any case, the system is open for new aggregators to provide new views. An open question is how publishers decide which aggregators to ping and whether publishers should let aggregators subscribe to them.

### 3.4 Distributed aggregation

Aggregators function as super-peers in the network, taking the burden of following publishers off the readers and making it simpler for publishers to announce new posts. Pinging is essential to meeting the timeliness requirement without excessive polling. In this sense it is a push technique, but since the posts are already published, pinging results in a simpler interface and makes it cheaper for aggregators to disregard posts from irrelevant sources.

<sup>10</sup> <http://geonames.org>

If an aggregator disconnects and returns to the network, it may have missed pings. In this situation, it may make sense to poll known publishers for new content. This would be a typical situation for personal aggregators. Further, aggregators would be able to crawl and readers to browse more posts as long as the Linked Data principles are followed.

We can even envision intelligent readers, that will accept new posts only if they are linked to relevant URIs. For example, we could setup a "Travel microblogging" server that will accept only posts that contain links to one or more URIs from the GeoNames dataset.

### 3.5 Users own their data

As a consequence of the distributed nature of the system, one feature of our architecture is that people can really own their data. By self-hosting a publishing service and then publishing to a microblog aggregator server, they keep all their updates even if one service closes. Moreover, by hosting their data, people can reuse it in other applications, including future microblogging servers they want to publish to and any Semantic Web applications. They can also mash it up with other RDF data they own or that is publicly available on the Web, or in case of corporate microblogging, in their organisation.

We think that this feature is really important, especially from a user rights and data portability point of view on the Web, following some thoughts that have been expressed in "A Bill of Rights for Users of the Social Web" [14]. Combining the distributed architecture and this data ownership and reuse aspect, Fig. 2 provides an overview on the complete architecture of the process.

### 3.6 Security and privacy issues

The open and distributed nature of the architecture complicates the authentication requirements in some use cases. It is easy to publish posts in someone else's name or fill a public aggregator with spam. Moreover, aggregators may need to authenticate to publishers if the posts are for a restricted audience only.

One solution is to require publishers to register using OpenID on an aggregator server. The server delivers each registrant an API key (a password) for publishing their content on that server. Relying on OpenID allows servers to automatically discover the FOAF profile and the URI of a user<sup>11</sup> as long as the OpenID provider can offer FOAF autodiscovery<sup>12</sup>.

Combined with the use of the `foaf:openid` property that was recently introduced in the FOAF specifications, this is a way to provide a lightweight authentication and security layer, since the server can ensure that someone publishing on it is really the person identified by the FOAF URI. Of course, one can deliver false information in a fake FOAF profile, thus additional strategies

<sup>11</sup> <http://apassant.net/blog/2007/09/23/retrieving-foaf-profile-from-openid/>

<sup>12</sup> <http://wiki.foaf-project.org/Autodiscovery>

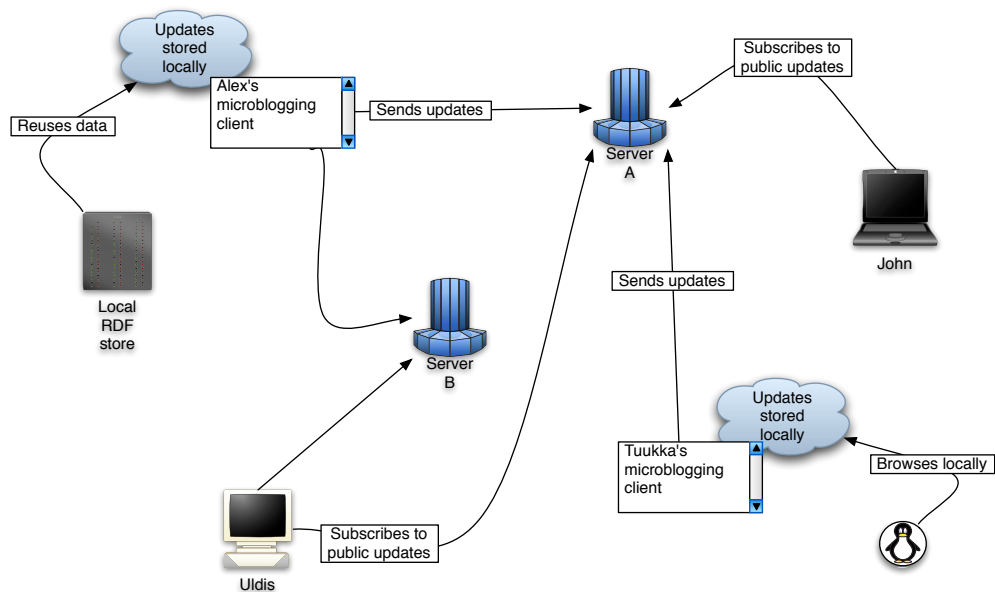


Fig. 2. Global architecture of distributed semantic microblogging

such as a network of trust between community members or graph signing<sup>13</sup> with public-key cryptography (PGP) should be considered.

## 4 A prototype: SMOB

In order to demonstrate our thoughts, we have implemented SMOB<sup>14</sup>, a prototype of the publishing client and server web services for semantic microblogging.

### 4.1 Publishing Content

The publishing client is accessed as a web page that contains a small form field for content. Submitting the form creates the post and makes it available on the Web in RDF. Further, there are checkboxes for choosing which of the configured aggregation servers are pinged about the new content, so that within the same client, a user can decide that some update will ping a server while another update will ping another server.

The publishing client is configured for its location, the list of servers it can ping and the `foaf:Person` URIs of the author and related file. An existing FOAF URI can be reused for this service, thus providing it a new `sioc:User` account for this URI.

<sup>13</sup> <http://usefulinc.com/foaf/signingFoafFiles>

<sup>14</sup> <http://code.google.com/p/smob/>

## 4.2 Reading content





Based on pings received from clients, the server loads all posts into its triple store using SPARUL. SPARUL<sup>15</sup> (or SPARQL/Update) is an update language for RDF graphs and currently implemented in Jena, OpenLink Virtuoso as well as partially in ARC2 within its SPARQL+ support. The server uses the LOAD instruction to load all statements for any incoming URL of an RDF file (i.e. of a microblogging item) into the triple store. Then, people can browse and read the posts using a web interface that implements faceted browsing as shown on Fig. 3. The posts are available as a sortable and groupable list, as an ordered list, a timeline and on a map. We rely on Exhibit [10] to provide this interface, and facets are created using metadata (author and date) but also data extracted from the semantic hash tags as described before. Currently, we have implemented two of those facets: (1) locations, which are mapped with the Google Maps view of Exhibit thus providing a user-friendly geolocation interface for microblogging (Fig. 4), and (2) topics, which can be based on DBpedia URIs.

**Semantic MicroBlogging - demo timeline**

**BLOCS** · LIGNE DE TEMPS · CARTE

24 MicroBlogPost

Trier par : [date](#), [puis par...](#) ·  Grouper selon le tri

-  Tuukka Hastrup  
testing new client  
2008-03-14T18:09:21+00:00
-  Alexandre Passant  
test  
2008-03-14T03:26:19-07:00
-  Alexandre Passant  
one more test with new config file  
2008-03-14T03:22:34-07:00
-  Alexandre Passant  
test  
2008-03-14T03:19:16-07:00

**Date**

- 1 2008-02-05
- 19 2008-02-06
- 4 2008-03-14

**Name**

- 20 Alexandre Passant
- 4 Tuukka Hastrup

Fig. 3. Latest updates rendered in Exhibit

## 4.3 Code overview

Our client is a simple 57-line PHP page that presents a submission form and handles the received content. The content is wrapped in an RDF-XML document using SIOC PHP Export API<sup>16</sup> and saved as a file locally. The URI of the file

<sup>15</sup> <http://jena.hp1.hp.com/~afs/SPARQL-Update.html>

<sup>16</sup> <http://wiki.sioc-project.org/index.php/PHPEXportAPI>

is then sent to the server(s) as a HTTP GET ping using CURL. The SIOC PHP Export API implementation is 678 lines of PHP but fairly generic, and is already used on other prototypes such as the DotClear weblogging exporter and the currently in-development SIOC plugin for VBulletin forums. Using the API offers a way to update with zero-cost to new version of the ontology in case it changes.

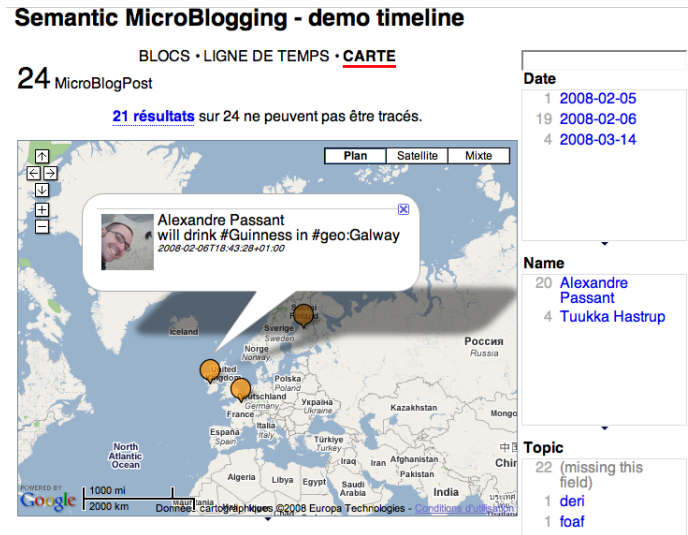


Fig. 4. Map view of latest updates with Exhibit

The server uses ARC2<sup>17</sup> to store the data, since it supports the LOAD instruction in SPARQL+<sup>18</sup>, and relies on a single SPARQL query to render the JSON file needed for Exhibit (assuming properties are single-valued):

```
SELECT ?post ?date ?content ?maker ?name ?depiction
WHERE {
  ?post rdf:type sioc:MicroblogPost ;
  foaf:maker ?maker ;
  sioc:content ?content ;
  dct:created ?date .
  ?maker foaf:name ?name .
  { ?maker foaf:img ?depiction } union
  { ?maker foaf:depiction ?depiction }
} ORDER BY DESC(?date) LIMIT 20
```

The preprocessor for hash tags uses simple regular expressions and mappings between prefixes, and URIs and services are mapped internally. It is less than

<sup>17</sup> <http://arc.semsol.org>

<sup>18</sup> <http://arc.semsol.org/docs/v2/sparql+>

100 lines of code, excluded libraries, and can be easily deployed in shared hosting environments, so that people can set-up their own service for their community.

## 5 Conclusions and future work

In this paper, we introduced the architecture and a first implementation of a distributed semantic microblogging platform. While existing approaches to convert microblogging services to RDF already exist for Twitter<sup>19</sup> or Jaiku<sup>20</sup>, our approach relies on a complete open and distributed view, using some standards of the Social Semantic Web. Moreover, some parts of our work, as the hash tag processing could be adopted to services such as Twitter to enable some semantics in existing tools.

Some issues still remain to be resolved, such as data privacy, private aggregation communities, and building personalised views and aggregation services for public updates. For this latter point, we can imagine personal aggregators based on the `foaf:knows` list of a user to automatically accept or reject new updates. More generally, and since this field is quite new to the Semantic Web, we think that microblogging can be one of the next steps of semantically-enhanced blogging systems [8].

## Acknowledgements

This material is based upon work supported by Science Foundation Ireland under grant number SFI/02/CE1/I131. The authors would like to thank Richard Cyganiak of DERI Galway for his valuable comments about this paper.

## References

1. Sören Auer, C. Bizer, G. Kobilarov, Jens Lehmann, R. Cyganiak, and Z. Ives. DBpedia: A Nucleus for a Web of Open Data. *6th International Semantic Web Conference, Busan, Korea, 2007*.
2. Tim Berners-Lee. Giant Global Graph. <http://dig.csail.mit.edu/breadcrumbs/node/215>, November 2007.
3. Chris Bizer, Richard Cyganiak, and Tom Heath. How to Publish Linked Data on the Web. <http://sites.wiwiss.fu-berlin.de/suhl/bizer/pub/LinkedDataTutorial/>, 20 July 2007.
4. Uldis Bojārs, John Breslin, Aidan Finn, and Stefan Decker. Using the Semantic Web for Linking and Reusing Data Across Web 2.0 Communities. *The Journal of Web Semantics, Special Issue on the Semantic Web and Web 2.0 (Forthcoming)*, 2008.

<sup>19</sup> <http://sioc-project.org/node/262>

<sup>20</sup> <http://sioku.sioc-project.org/>

5. John G. Breslin and Stefan Decker. Semantic Web 2.0: Creating Social Semantic Information Spaces. In *Tutorial in the 15th International World Wide Web Conference (WWW 2006)*, Edinburgh, Scotland, May 2006.
6. John G. Breslin, Andreas Harth, Uldis Bojars, and Stefan Decker. Towards Semantically-Interlinked Online Communities. In *Proceedings of the Second European Semantic Web Conference, ESWC 2005, May 29–June 1, 2005*, Heraklion, Crete, Greece, 2005.
7. Dan Brickley and Libby Miller. FOAF Vocabulary Specification. Namespace Document 2 Sept 2004, FOAF Project, 2004. <http://xmlns.com/foaf/0.1/>.
8. Steve Cayzer. What next for semantic blogging.from visions to applications. OCG Verlag, 2006.
9. Sergio Fernández, Diego Berrueta, and Jose E. Labra. Mailing lists meet the semantic web. In Dominik Flejter, editor, *Proceedings of SAW2007 Workshop*, pages 45–52, 2007.
10. David Huynh, David Karger, and Rob Miller. Exhibit: Lightweight structured data publishing. In *16th International World Wide Web Conference*, Banff, Alberta, Canada, 2007. ACM.
11. Akshai Java, Xiaodan Song, Tim Finin, and Belle Tseng. Why we twitter: Understanding microblogging usage and communities. Jul 2007. Simple description of Twitter.
12. Andrew P. McAfee. Enterprise 2.0: The dawn of emergent collaboration. *MIT Sloan Management Review*, 47(3):21–28, 2006.
13. Alexandre Passant. :me owl:sameas flickr:33669349@n00 . In *Linked Data on the Web (LDOW2008)*, 2008.
14. Joseph Smarr, Marc Canter, Robert Scoble, and Michael Arrington. A bill of rights for users of the social web. <http://opensocialweb.org/2007/09/05/bill-of-rights/>, 4 September 2007.
15. Giovanni Tummarello, Renaud Delbru, and Eyal Oren. Sindice.com: Weaving the open linked data. In *ISWC/ASWC*, pages 552–565, 2007.
16. Max Völkel and Sebastian Schaffert, editors. *SemWiki2006, First Workshop on Semantic Wikis - From Wiki to Semantics, Proceedings, co-located with the ESWC2006, Budva, Montenegro, June 12, 2006*, volume 206 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2006.

# Using JavaScript RDFa Widgets for Model/View Separation inside Read/Write Websites

Sebastian Dietzold, Sebastian Hellmann and Martin Peklo

Universität Leipzig, Department of Computer Science  
Johannisgasse 26, D-04103 Leipzig, Germany,  
{dietzold|hellmann}@informatik.uni-leipzig.de

**Abstract.** As more and more websites start to embed RDFa content in their web application view, the need arises to provide a more extensive way for viewing and editing this semantic content independently from the remainder of the application. We present a JavaScript API that allows the independent creation of editing widgets for embedded RDFa, which adds a new edge to Web development in the context of the Semantic Web. As an addition, the API also provides sound update methods that allow on-the-spot model synchronization between client and server.

## 1 Introduction

Future web applications will use more and more client side application logic to obtain fast and responsive user interfaces. While it is currently possible to use semantic technologies at the backend, web application developers can benefit greatly from a way to preserve semantics between the server and the client. Especially functionalities such as extensive editing, including type verification, can be implemented more efficiently (i.e faster and without additional requests) on the client side. By using RDFa [1] or Microformats [3], semantically enriched data can be transferred back and forth and e.g. JavaScript enables the modification on the client side. To return semantic data from the client side back to the server, developers need APIs to modify both the semantic model and the corresponding view from the RDF model in a consistent and independent way.

We present an RDF-API for JavaScript (`rdfapi-js`<sup>1</sup>), which uses an RDFa parser to create a JavaScript object as an in-memory representation of the RDF model from the website. The API gives developers the methods to modify the in-memory model, roll out the changes to the (X)HTML view and report them back to the server. Furthermore, RDFa widgets are used to create custom editing functions based on the local model in the (X)HTML representation. These widgets give a degree of independence to developers, since it is now possible to add appropriate editing widgets based on the semantic content of the RDFa enhanced website without interfering with the development of the view of the application. This greatly improves maintainability and customization. A useful

---

<sup>1</sup> `rdfapi-js` is available at <http://powl.svn.sf.net/viewvc/powl/trunk/rdfapi-js/>



implication we would also like to mention is the redundancy of creating separate views for editing, because updating can often be solved with the proposed widgets.

## 2 API Overview

The RDF-API for JavaScript implements the RDF JSON specification proposal from the Talis n2 Wiki<sup>2</sup> for data representation and allows for RDF manipulation methods. The RDF model is extracted by a modified version of the parser from the RDFa Javascript implementation<sup>3</sup> provided by the W3C.

The added methods can be grouped as follows:

*Statement modification:* These methods are used to modify the content of the model. We have implemented methods for adding one or more statements to a model and for deleting one or more statements from the model.

*Namespace management:* Since RDFa incorporates the use of namespaces, methods on in-memory models for adding, deleting and applying namespaces are needed. The Talis RDF JSON specification proposal does not include namespace declarations, so we have added this to our object notification. We have implemented methods for adding and deleting namespaces as well as for converting models from qualified names to full URIs (and back) according to the current namespace declarations.

*Model comparison and check:* These methods are used to check the consistency of a given model and to compare two models and produce a statement diff for the update service. At this time, we do not care about bnodes in the diff, e.g. by using blank node enrichment [6] or minimum self-contained graphs [5]. This could be added in the future.

We complete the rdfapi-js by adding a small set of administration methods for counting, (X)HTML output and debugging.

## 3 Usage in a Semantic Wiki

We use rdfapi-js in our semantic wiki application OntoWiki [2]. OntoWiki is able to handle plugins for the visualization of resources from specific RDF schema (e.g. a FOAF person). These plugins are simple templates which have access to the attributes of the resource that is to be rendered. Without rdfapi-js, plugin developers have the choice to develop either a special edit template or let users edit the attributes with the generic table-edit view. With rdfapi-js, a new option is to enable view templates for inline editing just by adding RDFa markup.

---

<sup>2</sup> [http://n2.talis.com/wiki/RDF\\_JSON\\_Specification](http://n2.talis.com/wiki/RDF_JSON_Specification)

<sup>3</sup> <http://www.w3.org/2006/07/SWD/RDFa/impl/js/>

The following steps describe an exemplary editing process:

1. The complete page with RDFa markup and the link to the rdfapi-js and a control script are created on the server and transferred to the client.
2. The client receives the page and loads the rdfapi-js as well as the application control script through a `script`-link.
3. rdfapi-js parses the page and creates an in-memory model from the RDFa markup.
4. An onclick handler is added next to every RDFa Literal found in the page.
5. By clicking on the event handler, an editing widget is displayed as an overlay to the existing page (see Fig.1).



**Fig. 1.** An RDFa enhanced vCard: plain (left), with highlighted statements and widget (right)

6. The user submits the new value to the widget. The widget modifies the in-memory model by using statement modification methods of rdfapi-js. These API methods apply the changes to the in-memory model as well as modify the (X)HTML view.
7. The modified in-memory model triggers a new submit button which enables the user to submit the change request to the server. If pressed by the user, a statement difference is calculated and transferred to the server via an asynchronous HTTP request to an update service. The current implementation sends two JSON models to the server, one with the statements which have to be deleted, another with the statements which have to be created on the server. The execution of these atomic add / delete actions is left to the server update interface.

Widgets are not limited to the editing of plain literals. According to the in-memory model, different widgets can be loaded (e.g. for dates or geo locations). The idea of such widgets is to achieve independence from the page template so that a growing widget library can enhance every application which uses rdfapi-js.

## 4 Conclusion and Future Work

The presented API includes methods for customized client-side edit widgets to modify embedded RDFa. It tackles the synchronization of a locally changed

RDF model with the model on the server. The main advantage is the separation of edit functions from the data view. Widgets can be created for different datatypes and properties. These widgets add new possibilities for flexible user interaction. The API is included in OntoWiki, but can be used in other Semantic Web applications as well. Without making a promising statement, we have already considered to propose the API as a full alternative to an editing view. However, the problems that have to be overcome are numerous such as multi-user synchronization, security and user validation. Although solutions for these problems have to be provided, they are clearly out of the scope of this API, since these problems have to be solved on the server side.

Future work includes the improvement of the diff computation algorithm to support bnodes and the enhancement of the widget library with a set of widgets for common attributes. The (X)HTML rollout of in-memory changes is currently limited to plain literals and resource relations. This limitation has to be resolved in order to support more complex RDFa markup. To generalize the editing process, future work should include support for SPARUL [4] queries in addition to transferring JSON model objects for change requests.

## References

1. Ben Adida, Mark Birbeck, Shane McCarron, and Steven Pemberton. RDFa in XHTML: Syntax and Processing. W3C Working Draft, W3C, 2008. <http://www.w3.org/TR/rdfa-syntax/>.
2. Sören Auer, Sebastian Dietzold, and Thomas Riechert. OntoWiki - A Tool for Social, Semantic Collaboration. In Isabel F. Cruz, Stefan Decker, Dean Allemang, Chris Preist, Daniel Schwabe, Peter Mika, Michael Uschold, and Lora Aroyo, editors, *The Semantic Web - ISWC 2006, 5th International Semantic Web Conference, ISWC 2006, Athens, GA, USA, November 5-9, 2006, Proceedings*, volume 4273 of *Lecture Notes in Computer Science*, pages 736–749. Springer, 2006.
3. Rohit Khare and Tantek Çelik. Microformats: a pragmatic path to the semantic web. In Les Carr, David De Roure, Arun Iyengar, Carole A. Goble, and Michael Dahlin, editors, *Proceedings of the 15th international conference on World Wide Web, WWW 2006, Edinburgh, Scotland, UK, May 23-26, 2006*, pages 865–866. ACM, 2006.
4. Andy Seaborne and Geetha Manjunath. SPARQL/Update – A language for updating RDF graphs. Technical report, Hewlett-Packard, January 2008. V4.
5. Giovanni Tummarello, Christian Morbidoni, Reto Bachmann-Gmür, and Orri Erling. RDFSyc: Efficient Remote Synchronization of RDF Models. In Karl Aberer, Key-Sun Choi, Natasha Fridman Noy, Dean Allemang, Kyung-Il Lee, Lyndon J. B. Nixon, Jennifer Golbeck, Peter Mika, Diana Maynard, Riichiro Mizoguchi, Guus Schreiber, and Philippe Cudré-Mauroux, editors, *The Semantic Web, 6th International Semantic Web Conference, 2nd Asian Semantic Web Conference, ISWC 2007 + ASWC 2007, Busan, Korea, November 11-15, 2007*, volume 4825 of *Lecture Notes in Computer Science*, pages 537–551. Springer, 2007.
6. Max Völkel, Carlos F. Enguix, Sebastian Ryszard Kruk, Anna V. Zhdanova, Robert Stevens, and York Sure. SemVersion - Versioning RDF and Ontologies. Technical report, AIFB, University of Karlsruhe, 2005.

# RDF/JSON: A Specification for serialising RDF in JSON

Keith Alexander

Talis Information Ltd, Knights Court, Solihull Parkway, Birmingham Business Park, B37 7YB,  
United Kingdom  
[keith.alexander@talis.com](mailto:keith.alexander@talis.com)

**Abstract.** This paper outlines the design of a data structure for RDF serialisable in JSON, and describes why it is significant for developing with RDF and scripting languages.

## What is JSON

JSON (JavaScript Object Notation) is billed as a “lightweight data-interchange format”[1]. It is a subset of Javascript, using its “object literal notation” and as such, can simply be evaled in javascript, and parsed with little effort in most other languages. As a data-interchange format, it has become increasingly popular as an alternative to XML. Its advocates generally cite its human-readability and the ease of parsing as the main advantages. However, the key feature, I believe, is that JSON data structures translate transparently into the native data structures universal to almost all programming languages used today.

## RDF in JSON?

At Talis, we provide a Semantic Web Technologies Platform, of which RDF is a core component. We want to be able to provide the option of requesting RDF output from our services (for example, from SPARQL CONSTRUCTs and DESCRIBEs) as JSON in order to make the data accessible in scripting language environments without the overhead of an RDF/XML parser.

There are of course, other ‘light-weight’ serialisations, such as Turtle[2], for which parsers exist in the most of the popular web scripting languages. Turtle, in fact, probably has the advantage of greater human-readability over any JSON or XML serialisation. However, consumers of Turtle still need to parse it into triples, and represent those triples in some kind of data structure. This is the niche we see for RDF in JSON: a well-defined and commonly understood data structure for representing RDF data that can be transparently serialised in JSON and passed easily between server and client.

We found several existing approaches to the problem of representing RDF data in JSON:

## Approaches to Representing RDF in JSON

### Flat Triples Approach

These approaches represent the components of RDF triples in a readily understandable, and easily serialisable fashion. The disadvantage is that it is difficult to access the data without an RDF model API of some kind to iterate over the triples and pick out a property for you.

### ARC[3] v.1's RDF Parser Output

```
{
  "data" : [
    {
      "s" : { "type" : "uri" , "uri" : "http://example.org/about" } ,
      "p" : "http://purl.org/dc/elements/1.1/creator",
      "o" : { "type" : "literal", "val" : "Anna Wilder" }
    } ,
    {
      "s" : { "type" : "uri" , "uri" : "http://example.org/about" } ,
      "p" : "http://purl.org/dc/elements/1.1/title",
      "o" : { "type" : "literal", "val" : "Anna's Homepage", "lang" :
"en" }
    }
  ]
}
```

### Resource-oriented Approach

This approach groups properties of resources together.

*JDIL[4]*

```
{
  "@namespaces": {
    "dc": "http://purl.org/dc/elements/1.1/",
    "rss": "http://purl.org/rss/1.0/",
    "georss": "http://www.georss.org/georss/"
  },
  "@type": "rss:channel",
  "rss:items": [
    { "@type": "rss:item",
      "rss:title": "A visit to Astoria",
      "rss:description": "sample description",
      "dc:coverage": {
        "@id": "a0",
        "dc:title": "Astoria, Oregon, US",
        "georss:point": "46.18806 -123.83"
      }
    }
  ]
}
```

This is a more transparently RDF-like format, and it looks fairly natural to read and write by hand. However, several things make it difficult for scripting with:

- Namespace prefixes are used. This makes it pretty to read and write for humans, but difficult for scripting consumption, since the script will in most cases not be able to know beforehand which prefixes will be used for which namespaces, and hence will have to resolve all 'Qnames'[5] to full URIs.
- Resources can be nested - this makes it hard to find individual resources, as you do not know where they are stored in the hierarchy
- Resource identifiers are JSON object properties - so to find a particular resource, you must iterate over all the resources, checking for an "@id" key

## Talis's RDF/JSON specification

RDF/JSON represents a set of RDF triples as a series of nested data structures. Each unique subject in the set of triples is represented as a key in JSON object (also known as associative array, dictionary or hash table). The value of each key is a object whose keys are the URIs of the properties associated with each subject. The value of each property key is an array of objects representing the value of each property.

Blank node subjects are named using the Turtle a string conforming to the nodeID production in Turtle. For example: `_ :A1`

A triple (subject S, predicate P, object O) is encoded in the following structure:

```
{ "S" : { "P" : [ O ] } }
```

The object of the triple O is represented as a further JSON object with the following keys:

### **type**

one of 'uri', 'literal' or 'bnode' (**required** and must be lowercase)

### **value**

the lexical value of the object (**required**, full URIs should be used, not qnames)

### **lang**

the language of a literal value (optional but if supplied it must not be empty)

### **datatype**

the datatype URI of the literal value (optional)

The 'lang' and 'datatype' keys should only be used if the value of the 'type' key is "literal".

For example, the following triple:

```
<http://example.org/about> <http://purl.org/dc/elements/1.1/title>  
"Anna's Homepage" .
```

can be encoded in RDF/JSON as:

```
{  
  "http://example.org/about" :  
    {  
      "http://purl.org/dc/elements/1.1/title": [ { "type" : "literal" ,  
"value" : "Anna's Homepage." } ]  
    }  
}
```

## Usage Examples

### *Accessing a Resource in a Graph*

```
var resource = data['http://example.org/about'];  
// resource is an object containing  
// all the properties belonging to http://example.org/about
```

### *Accessing the title of a Resource*

```
var title = data['http://example.org/about']['http://purl.org/dc/  
elements/1.1/title'][0]['value'];
```

## Iterating over a Graph

```
for(var uri in data){
  for(var property in data[uri]){
    for(var i=0; i<data[uri][property].length; i++ ){
      var s = uri;
      var p = property;
      var o = data[uri][property][i]['value'];
      var o_type = data[uri][property][i]['type'];
      var o_lang = data[uri][property][i]['lang'];
      var o_datatype = data[uri][property][i]['datatype'];
    }
  }
}
```

## Design Constraints

We decided what we needed was a JSON structure for RDF that:

- expresses the whole RDF model (we didn't want information loss just because a consumer had requested JSON rather than XML or turtle)
- requires as little processing / control structures for common tasks, as possible.

As I mentioned above, the most compelling advantage of JSON is that it translates directly into universal data structures. So what we tried to do is come up with the most useful default structure for RDF data.

## Design Decisions

An important goal was that the structure should be as consistent and predictable as possible, making it easier for scripts to handle any data encoded as RDF/JSON.

- **Resource-centric structure.** There are various options for how to arrange RDF data into a structure. The most common, from those that we found, are "triple" structures, where the basic unit is the triple, and "resource" structures, where a resource's properties are gathered up in one unit. Technically, JDIL's structure does not strictly enforce this unification of a resource's properties: since the resource identifier is a property of, rather than a key to, the resource object, it is possible to spread the resource description across multiple object structures. It would, of course, also be possible to structure the data according to one of the other basic components of RDF - by property URI, or Object, for example - or to introduce named graphs as the top level grouping for the data, and there are certainly use cases for which such structures would be better suited. However a resource-centric structure makes things easier for most common tasks (such as rendering descriptions in HTML, and accessing particular properties of a given resource.
- **No nesting of resources.** Although nesting resources can sometimes seem intuitive when authoring by hand, it opens the serialisation up to a huge amount of variability, making it more tedious for a script consuming the data to find a particular resource in the graph.
- **Resource Descriptions should be identified by their URIs.** In contrast to the JDIL format, URIs are not "properties" of a resource description, but the key to the resource description (which consists of a hash object of properties and their values). In conjunction with no nesting, this means that you access the properties of any given resource in the rdf/json object simply by

using the URI (or node ID) as a key. eg: `data[ 'http://example.org/#foo' ]`. It also enforces that all known properties of a resource are contained in the same

- **No prefixes.** While declaring prefixes to stand for namespace URIs seems natural enough in a hand-authoring environment, and it can make the resulting document easier to read, we realised it would also make it more difficult for consumers to use, since, in most situations the consumer will not be able to know which prefixes will be used for which
- **Property URIs are keys to an array of values** - even if there is only one 'object' in the array. If you knew in advance that a particular property only had one value, it might be simpler to access it directly, rather than as the single item of an array. However, the (we think) more common case is not knowing whether there is one or more than one values of a property, in which case it is simpler to always deal with an array of objects.

## Implementations

Our RDF/JSON specification already has several implementations, including, at the time of writing:

- [ARC PHP RDF toolkit http://arc.semsol.org/](http://arc.semsol.org/) (2008)
- [Drupal RDF Module http://drupal.org/handbook/modules/rdf](http://drupal.org/handbook/modules/rdf) (2008)
- Raptor [http://librdf.org/raptor/RELEASE.html#rel1\\_4\\_17](http://librdf.org/raptor/RELEASE.html#rel1_4_17) (2008)
- Triplr <http://triplr.org> (2008)

## Benefits of Wider Adoption

Since JSON is not just a text format, but a "serialised data structure", RDF libraries can support it, not just as format to parse from or output to, but also internally, as a data structure that can be passed to, and returned by functions and methods, as ARC, and the Drupal RDF module do.

There are a few advantages to converging on a common data structure for RDF in code. One is that it becomes easier to pass data between components; if I use library A to spider some RDF from the web, and I need to use another library B to do some OWL reasoning, say, ordinarily, I would have to reserialise the RDF data with library A so that I can pass it to library B, but if both libraries use the same data structure, I can pass the data directly, without the parsing, re-serialising, and parsing again. This in turn may mean that we don't need such monolithic RDF API libraries, and can encourage the development of more modular components.

Of course, our proposed RDF/JSON structure is not optimal for every task, but (given the advantages of interoperability and familiarity) in such cases where the internal structure would mainly be arbitrarily different, I would like to encourage RDF library developers to support our RDF/JSON structure internally as well as externally.

The other big advantage for the RDF developer standardising on a common data structure is that there is less mental overhead once you are familiar with that structure. You can use the same algorithms for processing RDF in PHP, Javascript, Ruby, Perl and Python, and between library components that use that structure too.

By using and supporting RDF/JSON, semantic web developers can encourage a code ecosystem that will make programming with RDF easier across languages and libraries to wider audience of developers.



## References

1. JSON <http://json.org/> (2008)
2. Turtle <http://www.dajobe.org/2004/01/turtle/> (2007)
3. ARC PHP RDF toolkit <http://arc.semsol.org/> (2008)
4. JDIL <http://www.jdil.org/> (2007)
5. Namespaces in XML <http://www.w3.org/TR/REC-xml-names/#dt-qualname> (2006)
6. Drupal RDF Module <http://drupal.org/handbook/modules/rdf> (2008)
7. Raptor [http://librdf.org/raptor/RELEASE.html#rel1\\_4\\_17](http://librdf.org/raptor/RELEASE.html#rel1_4_17) (2008)
8. Triplr <http://triplr.org> (2008)
9. RDF/JSON Specification [http://n2.talis.com/wiki/RDF\\_JSON\\_Specification](http://n2.talis.com/wiki/RDF_JSON_Specification) (2008)
10. RDF/JSON Brainstorming [http://n2.talis.com/wiki/RDF\\_JSON\\_Brainstorming](http://n2.talis.com/wiki/RDF_JSON_Brainstorming) (2008)
11. Beckett, Dave: <XML/> without the X: The return of Template:Textual markup <http://www.dajobe.org/talks/200705-textual/> (2008)
12. Feigenbaum, Lee: Using RDF On the Web - a Vision [http://thefigtrees.net/lee/blog/2007/01/using\\_rdf\\_on\\_the\\_web\\_a\\_vision.html](http://thefigtrees.net/lee/blog/2007/01/using_rdf_on_the_web_a_vision.html) (2007)
13. Feigenbaum, Lee: Using RDF on the Web - a Survey [http://thefigtrees.net/lee/blog/2007/01/using\\_rdf\\_on\\_the\\_web\\_a\\_survey.html](http://thefigtrees.net/lee/blog/2007/01/using_rdf_on_the_web_a_survey.html) (2007)
14. Willison, Simon: Why JSON isn't just for Javascript <http://simonwillison.net/2006/Dec/20/json/> (2006)
15. Cyganiak, Richard: RDF/JSON <http://dowhatimean.net/2006/05/rdfjson> (2006)
16. Torres, E., Feigenbaum, Lee., Clark, K. G.: Serializing SPARQL Query Results in JSON <http://www.w3.org/TR/rdf-sparql-json-res/> (2007)
17. Wilson, G.: URF: A Semantic Framework alternative to RDF, XML, and JSON. <http://www.urf.name/> (2008)

# Semantic Scripting Challenge Submissions

**RDF2FS -- A Unix File System RDF Store**

Michael Sintek and Gunnar Grimnes.

**A Distributed Semantic Microblogging Platform**

Alexandre Passant, Tuukka Hastrup, Uldis Bojars and John Breslin.

**SPARQLBot - The Semantic Web Command Line**

Benjamin Nowack.

**MOAW -- URI's Everywhere**

Laurian Gridinoc and Mathieu d'Aquin.

**Ruby Semantic Web Pipes**

Daniel Hahn and Michele Barbera.

**The challenge prize is kindly provided by:**



# RDF2FS – A Unix File System RDF Store

Michael Sintek and Gunnar Aastrand Grimnes

DFKI GmbH, Knowledge Management Department  
Kaiserslautern, Germany  
`firstname.lastname@dfki.de`

## 1 Motivation

RDF2FS is a script that translates an arbitrary RDF graph into a directory structure in a (Unix) file system, which enables Semantic Web applications without the need of having a dedicated RDF store. There are multiple reasons why such an approach makes sense:

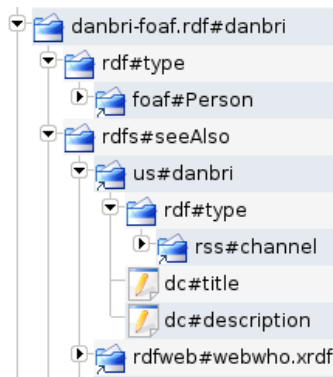
1. To access the RDF store (incl. simple queries), one can use the (scripting and any other) language of one's choice, as long as it has proper file system support (incl. handling of symbolic links).
2. The RDF store can be browsed (and edited) with a normal file browser (see screenshot).
3. Subversion showed that people are more comfortable with storing data in a file system than in a dedicated database.
4. Our approach can also have a nice educational effect: “normal” file system users and also “hackers” will be able to understand RDF more easily, without ever seeing an RDF file.
5. If RDF is used as metadata for documents, one usually has two places where metadata is stored: in the hierarchical directory structure where the documents are placed, and in the RDF metadata graph. This means that documents have to be “annotated” twice, by filing the documents into the intended folders, and by creating the RDF metadata with a separate tool. Our approach allows this to be done at one place and with only one tool that everyone is using anyway: a file browser.

## 2 Mapping Sketch

RDF2FS maps an RDF graph to a target root directory  $r$  as follows:

- All resources  $s$  in the graph become sub-directories of the root directory:  $r/s$ .
- For each triple  $\langle s, p, o \rangle$ :
  - if  $p$  is literal-valued, we create the file  $r/s/p$  (if it does not yet exist) and append  $o$  to this file (i.e., each line of  $o$  is a value for the property  $p$  for subject  $s$ )

- if  $p$  is object-valued, we create a directory  $r/s/p$ , and the object  $o$  becomes a symbolic link  $r/s/p/o \rightarrow ../../o$ , i.e., it links back to that resource on the root level (properties that are both literal- and object-valued are not yet supported; this could easily be done with some naming convention, but would make queries slightly more difficult)
- The names of the files and directories are not the full resource URIs but of the format *namespace-abbrev#localname*. RDF2FS keeps a list of commonly used namespace prefixes, such as *rdfs*, *foaf*, *dc*, etc., and will generate new mappings for unknown namespaces.



### 3 Querying the Store

The appendix shows that all of the simple queries are easily supported using standard POSIX file system utilities (such as `find`, `grep`, ...), including all combinations of statement queries, path expressions, and conjunctive queries.

### 4 Future Work

In a future version, RDF2FS should support to translate back from a file system RDF store to an RDF file. Real files that correspond to resources and that should be stored in these resource directories (e.g., in some ".” file), are not yet supported. Furthermore, mounting an RDF file via FUSE instead of translating to an existing file system would allow more efficient storage (esp. in case of file systems that do not handle many small files well), better handling of queries, and checking for illegal operations.

### 5 Demo Details

The mapping is implemented twice, as a Java program and a Python script. Either version will translate a set of RDF files into a bash-script that will create the necessary directories, files, and symbolic links. Please download <http://www.dfki.uni-kl.de/~sintek/SFSW2008/RDF2FS.tgz> and follow the instructions in `README`.

## Appendix: Bash Queries

```
# Literal-valued properties

# triple queries with one variable:

# s p ?o -> cat s/p
echo '** danbri-foaf.rdf#danbri foaf#name ?o'
cat danbri-foaf.rdf#danbri/foaf#name

# ?s p o -> grep -l o */p | $FIRST
# (where $FIRST is an awk call that selects the first part of a path)
echo -e '\n** ?s foaf#name "Dan Brickley"'
grep -l "Dan Brickley" */foaf#name | $FIRST

# s ?p o -> grep -l o s/* | $SECOND
# (where $SECOND is an awk call that selects the second part of a path)
echo -e '\n** danbri-foaf.rdf#danbri ?p "Dan Brickley"'
grep -l "Dan Brickley" danbri-foaf.rdf#danbri/* | $SECOND

# triple queries with two variables:

# ?s ?p o -> grep -l o */*
echo -e '\n** ?s ?p "Dan Brickley"'
grep -l "Dan Brickley" */*

# ?s p ?o -> ls -l */p
# plus retrieval of literals (simply with cat)
echo -e '\n** ?s foaf#name ?o'
for f in $(ls -l */foaf#name ); do
    echo $( echo $f | $FIRST ) \"$( cat $f )\"
done

# s ?p ?o -> file pattern s/* plus retrieval of literals (and resources)
echo -e '\n** danbri-foaf.rdf#danbri ?p ?o'
for f in danbri-foaf.rdf#danbri/*; do
    if [ -f $f ]; then
        echo $( echo $f | $SECOND ) \"$( cat $f )\"
    else # resources are just the files (links) in $f
        echo $( echo $f | $SECOND ) $( ls $f )
    fi
done

# Object-valued properties

# triple queries with one variable:
# s p ?o -> ls s/p
echo -e '\n** danbri-foaf.rdf#danbri/foaf#knows ?o'
ls danbri-foaf.rdf#danbri/foaf#knows
```

```

... (left as exercise to the reader :-) )

# Path expressions:

# s p1 _ p2 ?o -> cat s/p1/*/p2
echo -e '\n** path: danbri-foaf.rdf#danbri foaf#knows ?_ foaf#name ?o'
cat danbri-foaf.rdf#danbri/foaf#knows/*/foaf#name

# ?s p1 _ p2 o -> grep -l o */p1/*/p2
echo -e '\n** path: ?s foaf#knows ?_ foaf#name "Libby Miller"'
grep -l "Libby Miller" */foaf#knows/*/foaf#name | $FIRST

# Conjunction:

# ?s p1 o1 AND ?s p2 o2
# ->
# using intersection (realized with sort and uniq)
# grep -l o1 */p1 | $FIRST | sort -u > tmp1
# grep -l o2 */p2 | $FIRST | sort -u > tmp2
# sort -m tmp1 tmp2 | uniq -d # = intersection

echo -e '\n** ?s foaf#plan "Save the world" AND ?s uranai#bloodtype "A+''

grep -l "Save the world" */foaf#plan | $FIRST | sort -u > tmp1
grep -l "A+" */uranai#bloodtype | $FIRST | sort -u > tmp2
sort -m tmp1 tmp2 | uniq -d
rm tmp1 tmp2

```

*Output of this script:*

```

** danbri-foaf.rdf#danbri foaf#name ?o
Dan Brickley

** ?s foaf#name "Dan Brickley"
anon-64848a97%3A1187e661172%3A-7ffb
danbri-foaf.rdf#danbri

** danbri-foaf.rdf#danbri ?p "Dan Brickley"
foaf#name

** ?s ?p "Dan Brickley"
anon-64848a97%3A1187e661172%3A-7ffb/foaf#name
danbri-foaf.rdf#danbri/foaf#name

** ?s foaf#name ?o
anon-64848a97%3A1187e661172%3A-7fde "Pastor N Pizzor"
...
anon-64848a97%3A1187e661172%3A-7ffb "Dan Brickley"
card#i "Tim Berners-Lee"

```

danbri-foaf.rdf#danbri "Dan Brickley"

```
** danbri-foaf.rdf#danbri ?p ?o
contact#nearestAirport anon-64848a97%3A1187e661172%3A-7ffc
foaf#aimChatID "danbri_2002"
foaf#archnemesis anon-64848a97%3A1187e661172%3A-7ffb
foaf#dateOfBirth "1972-01-09"
foaf#holdsAccount anon-64848a97%3A1187e661172%3A-7ffd anon-64848a97%3A1187e661172%3A-7ffe ...
foaf#homepage danbri.org#
foaf#img ns01#Image1.gif
foaf#jabberID "danbri@jabber.org"
foaf#knows anon-64848a97%3A1187e661172%3A-7fe6 ...
foaf#mbox mailto#danbri%40apocalypse.org mailto#danbri%40danbri.org ...
...
wot#keyid "B573B63A"
```

```
** danbri-foaf.rdf#danbri/foaf#knows ?o
anon-64848a97%3A1187e661172%3A-7fe6
...
card#i
```

```
** path: danbri-foaf.rdf#danbri foaf#knows ?_ foaf#name ?o
Damian Steer
...
Tim Berners-Lee
```

```
** path: ?s foaf#knows ?_ foaf#name "Libby Miller"
danbri-foaf.rdf#danbri
```

```
** ?s foaf#plan "Save the world" AND ?s vocab#uranaibloodtype "A+"
danbri-foaf.rdf#danbri
```

# A Distributed Semantic Microblogging Platform

Alexandre Passant<sup>1</sup>, Tuukka Hastrup<sup>2</sup>, Uldis Bojārs<sup>2</sup>, John Breslin<sup>2</sup>

<sup>1</sup> LaLIC, Université Paris-Sorbonne,  
28 rue Serpente, 75006 Paris, France  
`firstname.lastname@paris4.sorbonne.fr`

<sup>2</sup> DERI, National University Of Ireland,  
Galway, Ireland  
`firstname.lastname@deri.org`

**Abstract.** The application showcases the ideas of a distributed, Semantic-Web enabled microblogging architecture, providing a way to leverage this new Web 2.0 practice to the Semantic Web.

**Key words:** Microblogging, SIOC, Data Portability, Linked Data Web

Microblogging is one of the recent social phenomena of Web 2.0 but unlike blogs or wikis has not yet been leveraged to the Semantic Web. To achieve this goal, we designed a semantically-enabled distributed architecture for semantic microblogging, which relies on an open world of publishing clients and aggregation servers that exchange data modelled in RDF.

When users write microblog posts within their clients, RDF files are created on the client webservers, describing the posts using FOAF [3] and SIOC [2], and pushed live to a number of aggregation servers. Thus, the user really owns his data and can reuse it locally for other purposes, either browsing or merging with other RDF data, while aggregation servers are mainly dedicated to providing a browsing interface for shared communities. To model updates, we extended the SIOC types module [1] with a `MicroblogPost` class, as well as `Microblog` to model the service itself.

Thanks to the use of existing libraries, the code of both the client and the server is really light<sup>1</sup>. The client uses the SIOC PHP API<sup>2</sup> to create the RDF files from an HTML form submission, and is only 57 lines of code. This file is pushed to some aggregation servers (chosen from the list of servers stored in the client configuration file) using CURL. Regarding the server, we rely on ARC2<sup>3</sup> which provides a lightweight environment for developing RDF-based applications in PHP. The server uses the SPARUL `LOAD` instruction to store received updates in the server backend store, and a single SPARQL query to render a view of public updates. To make the interface fancier, we use Exhibit [4] to display a faceted view of these latest updates. These facets include date and author but also some user-defined data. Indeed, the server features a preprocessor

---

<sup>1</sup> <http://code.google.com/p/smob/>

<sup>2</sup> <http://wiki.sioc-project.org/index.php/PHPEXportAPI>

<sup>3</sup> <http://arc.semsol.org>



that allows users to use some *semantic hashtags* in their updates. The current implementation includes a GeoNames<sup>4</sup> mapping, allowing users to use tags like `#geo:paris_france` to retrieve the URI of the related resource, thus providing a way to leverage location-based microblogging to the Linked Data Web. Consequently, this mapping permits the use of the geographical rendering part of Exhibit, as shown on Fig. 1 Other simple topics can be extracted with a similar processor and can also be linked to DBPedia with a given prefix.

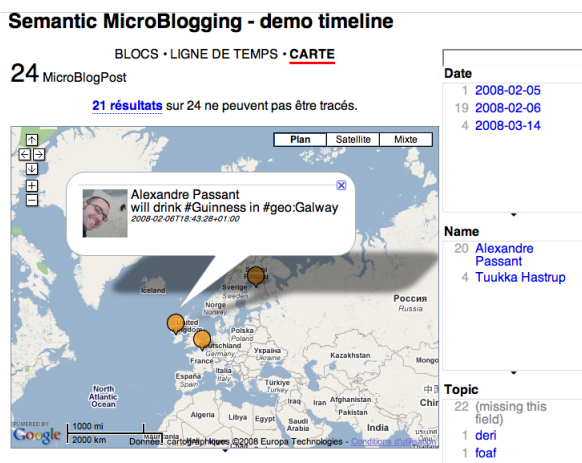


Fig. 1. Geographical faceted browsing of updates with Exhibit

This material is based upon work supported by Science Foundation Ireland under grant number SFI/02/CE1/I131.

## References

1. Uldis Bojārs, John Breslin, Aidan Finn, and Stefan Decker. Using the Semantic Web for Linking and Reusing Data Across Web 2.0 Communities. *The Journal of Web Semantics, Special Issue on the Semantic Web and Web 2.0 (Forthcoming)*, 2008.
2. John G. Breslin, Andreas Harth, Uldis Bojars, and Stefan Decker. Towards Semantically-Interlinked Online Communities. In *Proceedings of the Second European Semantic Web Conference, ESWC 2005, May 29–June 1, 2005*, Heraklion, Crete, Greece, 2005.
3. Dan Brickley and Libby Miller. FOAF Vocabulary Specification. Namespace Document 2 Sept 2004, FOAF Project, 2004. <http://xmlns.com/foaf/0.1/>.
4. David Huynh, David Karger, and Rob Miller. Exhibit: Lightweight structured data publishing. In *16th International World Wide Web Conference*, Banff, Alberta, Canada, 2007. ACM.

<sup>4</sup> <http://geonames.org>

# SPARQLBot: The Semantic Web Command Line (Scripting Challenge Submission)

Benjamin Nowack

semsol, Bielefelder Str. 5, 40468 Düsseldorf, Germany  
bnowack@semsol.com

**Abstract.** SPARQLBot is an RDF-driven agent that loads structured information from the Web and reacts to user-defined questions and commands via an IRC interface. The bot is implemented using a small number of PHP scripts and ARC, an open-source PHP/MySQL-based RDF system for storage and query functionality.

## 1 Motivation

SPARQLBot was developed during a single-day coding session to demonstrate a) the potential of tools that support structured and linked Web data, b) the developer-friendliness of SPARQL, and c) how very little custom code can be used to build useful applications. It consists of three core user interface components: An HTML-based command editor that simplifies the definition of custom bot operations, a long-running process that listens to command calls on an IRC channel<sup>1</sup>, and a standard SPARQL endpoint<sup>2</sup> for simplified debugging of SPARQL Queries and HTTP-based data access.

## 2 Implementation

SPARQLBot is built on top of two open-source toolkits. For core RDF functionality (RDF storage, parsing, microformats extraction, querying, etc.), it uses ARC<sup>3</sup>, a light-weight RDF toolkit written in PHP, and the user-facing applications are implemented as Trice<sup>4</sup> modules. Trice is a Web development framework that extends ARC with session management, HTML forms processing, IRC access, themes, and similar standard Web CMS functionality.

---

<sup>1</sup> <http://semsol.org/semcamp/sparqlbot>, to be moved to <http://sparqlbot.semsol.org/> soon

<sup>2</sup> [#sparqlbot](#) on <http://irc.freenode.net/>

<sup>3</sup> <http://semsol.org/semcamp/sparqlbot/sparql>, to be moved to <http://sparqlbot.semsol.org/bot/sparql> soon

<sup>4</sup> <http://arc.semsol.org/>

<sup>5</sup> <http://trice.semsol.org/>

## 2 Benjamin Nowack

The SPARQLBot-specific code consists of only four custom PHP Classes (~25KB / 800 LOC altogether). A generic RequestHandler dispatches HTTP requests to the three user-facing Sub-Handlers (the IRC ProcessHandler that implements the bot, a Command Editor, and the SPARQL endpoint). Only very few commands had to be built directly into the bot (e.g. "quit", or "smush"). As ARC supports LOAD, INSERT, and DELETE via SPARQL, both read and write operations can be defined using the command editor.

## 3 Examples: XFN Lookups

XFN<sup>6</sup>, the "Xhtml Friends Network" is a widely deployed microformat to specify a person's social network in XHTML pages. SPARQLBot's RDF toolkit can convert XFN markup to RDF triples and then make them accessible to SPARQL queries. The code below shows the command's essential parts which can be defined using an online form.

### Command Pattern (a regular expression):

```
(.*)'s? (contact|friend|me)s
```

### Triggered SPARQL Query (*\$i* can be used for command matches):

```
SELECT DISTINCT ?name WHERE {  
  {?res foaf:nick "$1"} UNION {?res foaf:name "$1"}  
  ?res xfn:$2 [ foaf:name ?name ] .  
}
```

### Result Template (*\$var* can be used for result bindings):

```
$nick, I found {$name, }
```

### Example Conversation:

```
<bengee> sparqlbot, load http://twitter.com/bengee  
<sparqlbot> 290 triples loaded in 4.9s seconds  
<bengee> sparqlbot, smush  
<sparqlbot> OK  
<bengee> sparqlbot, Benjamin Nowack's contacts  
<sparqlbot> bengee, I found Danny Ayers, Tom,  
Gregory Williams, Arto Bendiken, Paul Miller, John  
Breslin,Uldis Bojars, Alexandre Passant, ...
```

<sup>6</sup> <http://gmpg.org/xfn/>

<sup>7</sup> <http://microformats.org/>

# MOAW – URI’s Everywhere

Laurian Gridinoc and Mathieu d’Aquin

Knowledge Media Institute (KMi)  
The Open University, Milton Keynes, United Kingdom  
{l.gridinoc, m.daquin}@open.ac.uk

The manipulation of Semantic Web data is a relatively difficult and unfriendly task, generally carried out only by experts or technology enthusiasts. URIs of semantic resources for example are not really easy to manipulate, as rich and friendly interfaces allowing to understand and reuse them are still required. Therefore, the current status of the integration of such technologies with more traditional Web interfaces hampers their adoption by a broader public. Of course, many initiatives are currently focusing either on building systems using Semantic Web technologies in a way transparent to the user (e.g. [Revyu.com](http://Revyu.com)<sup>1</sup>) or in integrating semantic data in the traditional activities of the Web 2.0 user (e.g. [MOAT](http://MOAT2)<sup>2</sup>).

MOAW (pronounce like mauve: `||mɔ̃v||`) intends to provide a simple and lightweight contribution to such initiatives. MOAW can be seen as a *URI suggestion tool*, building on the “auto-completion” feature made popular by Web 2.0 websites and Google keyword suggestion. Basically, MOAW can be attached to any text field (HTML `input` element) so that, while typing, suggestions of URIs would appear that can be selected to replace the corresponding word (see figure 1). The URIs suggested by MOAW are discovered thanks to Watson. [Watson](http://Watson)<sup>3</sup> can be seen as a search engine for the Semantic Web, crawling, indexing and providing access to Semantic Web resources for applications.

One important strength of MOAW is that, thanks to the use of flexible technologies, it can be easily applied to any online Web form without having to edit the corresponding webpage: URIs coming from virtually anywhere on the Web can be reused virtually anywhere on the Web. In addition, as can be seen in figure 1, MOAW not only displays the suggested URIs, but also provides, thanks to Watson, a rich description of each URI so that it can be properly understood and selected by the user.

**Underlying technologies:** In practice MOAW takes the form of a *bookmarklet*: a link is dragged onto the bookmarks of the client browser and, whenever this bookmark is clicked, the Javascript code of MOAW is “injected” into the current webpage. This procedure for “installing” MOAW is therefore relatively simple and provides an homogeneous way to extend the capability of any web form, in potentially any browser.<sup>4</sup>

---

<sup>1</sup> <http://revyu.com>

<sup>2</sup> <http://moat-project.org/>

<sup>3</sup> <http://watson.kmi.open.ac.uk>

<sup>4</sup> Note that the current prototype version only works with Firefox.



Fig. 1. MOAW suggesting URIs for “tom” on the Revyu.com website.

Once activated, MOAW will load the necessary scripts and libraries, and attach the auto-completion feature to any “input” field in the current web-page. This feature is based on the JQuery auto-complete plugin<sup>5</sup>, that has been customized to search and display URIs of entities from Watson. Searching and retrieving rich description of URIs is realized through the Watson Javascript library, which is based on AJAX for the communication with the Watson server. Finally, the description of each URI (type, relations, link to Watson for “More...” information) is displayed thanks to a JQuery tooltip plugin.<sup>6</sup>

**What to do with it?** The original motivation for the development of MOAW was the possibility to easily bring URIs for the purpose of editing semantic data in the (next) version of Tabulator.<sup>7</sup> However, MOAW is a nice little tool that can be used anywhere a web form is present and it makes sense to fill it with URIs. It can be envisaged, for example, to use it for tagging resources with URIs instead of terms in various systems (blogs, collaborative bookmarks, etc.) Another interesting scenario can be, whenever HTML is edited online, to facilitate the integration of RDFa annotations.

The current prototype corresponds to an early, untested implementation. We intend to improve it both on the level of robustness and of the features it provides. The source code can be retrieved from the web page of the project (<http://watson.kmi.open.ac.uk/MOAW>) and is open to any form of contribution.

<sup>5</sup> <http://www.bassistance.de/jquery-plugins/jquery-plugin-autocomplete/>

<sup>6</sup> <http://bassistance.de/jquery-plugins/jquery-plugin-tooltip/>

<sup>7</sup> <http://eprints.ecs.soton.ac.uk/14773/>

# Ruby Semantic Web Pipes

## Semantic Web Pipes just got easier

Daniel Hahn<sup>2</sup> and Michele Barbera<sup>2</sup>

NET7 Internet Open Solutions - Pisa  
(hahn|barbera)@netseven.it

*Ruby SemPipes* are based on the *Semantic Web Pipes* proposed by Giovanni Tummarello, Christian Morbidoni et. al. [1]. Semantic web pipes provide a mechanism similar to Yahoo! Pipes<sup>1</sup>, but work on semantic web data. They allow users to filter and recombine semantic web data on the fly (more information can be found in the cited work). The original Semantic Pipes also have some advanced features that are not touched here, such as the revocation of triples.

*Ruby SemPipes*<sup>2</sup>. are an intentionally simple reimplementaion of the Semantic Web Pipes. They provide an easy “pipe description language”. Pipes can easily be created and modified from a script or application, the definitions are painless to read and they don’t require the pipe developer to manually edit XML files or SPARQL queries.

To make things simple, *Ruby SemPipes* treat all RDF graphs as enumerations of triples. A pipe itself is also an enumeration over it’s result set and any enumeration of triples (including other pipes) can be used as an input for a pipe.

Using Eyal Oren’s *ActiveRDF* [2] library, it will be quite easy to provide input adapters that connect to “real” RDF endpoints and perform SPARQL queries. These adapters could then be used as input to a pipe. Conversely, output adapters could re-write the result of a pipe to a common storage format, such as RDF/XML.

The first implementation of *Ruby SemPipes* contains a *merge*, *filter* and *rewrite* operator; a *smushing* operator is in the pipeline. The *filter* and *rewrite* operators allow a simple but powerful filtering (or rewriting) of triples using regular expression. In addition to the operators, pipes may also contain *actions* which allow the developer to manipulate triples programmatically. All built-in operators are themselves pipes written in the same way as user-created pipes.

Each pipe takes on ore more sets of RDF triples as an input; the pipe itself will operate on the union (including duplicates) of the input sets – the *merge* operator will actually just remove duplicates from the input.

### Listing 1.1. Sample Pipe setup

```
module SemPipe
  # The first pipe
  define _pipe :new_pipe do
    merged_triples = merge(input)
```

<sup>1</sup> <http://pipes.yahoo.com/>

<sup>2</sup> The first experimental version is available via public svn from [http://svn.talia.discovery-project.eu/talia-intern/playground/ruby\\_pipes/trunk](http://svn.talia.discovery-project.eu/talia-intern/playground/ruby_pipes/trunk)

```

# Rewrite Tim Berners-Lee entries
rewritten_triples = rewrite(merged_triples) do
  rewrite :all,
    "<http://dbpedia.org/resource/Tim_Berners-Lee>",
    "<http://www.w3.org/People/Berners-Lee/card#i>"
end

# Remove all triples dealing with Alice
self.output = filter(rewritten_triples) do
  reject :all "<http://strangeworld.com/Alice"
end

# Define another pipe
define_pipe :another_pipe do
  temp = new_pipe(input)
  self.output = action(temp) do |input|
    input.to_a << [@subject, @predicate, @object]
  end
end

# Use the pipe
AnotherPipe.new(load_some_input).each { |triple| puts triple }
end

```

Listing 1.1 shows a sample setup of two *Ruby SemPipes*. The first pipe merges all input sets, then rewrites Tim Berners-Lee's dbpedia URL with his own URL and finally filters out all entries dealing with Alice. Pipes can be easily connected using variables; the special variable `input` is used in the pipe description to denote the pipe's input. The `self.output` variable is used to assign a value to the current pipe's result set.

The second pipe shows that other pipes can be used just as easily as the built-in operators. The pipe also contains a generic action that adds a (complete random) triple to the graph.

## Acknowledgements

This work has been supported by Discovery, an ECP 2005 CULT 038206 project under the EC eContentplus programme. Of course we also like to acknowledge the original work of Christian and Giovanni.

## References

1. Morbidoni, C., Polleres, A., Phuoc, D.L., Tummarello, G.: Semantic web pipes. Technical report, DERI (2007)
2. Oren, E., Debru, R., Gerke, S., Haller, A., Decker, S.: ActiveRDF: Object-Oriented Semantic Web Programming. In: 16th International World Wide Web Conference (WWW2007), Banff, Alberta, Canada. (8-12 May, 2007) 817–823