Extending the Linked Data API with RDFa

Steve Battle[1], James Leigh[2], David Wood[2]

[1] Gloze Ltd, UK
steven.a.battle@gmail.com
[2] 3 Round Stones, USA
{James, David}@3roundstones.com

Linked data is about connecting the web of data to the web of documents. Two key use-cases where this structured data plays a significant commercial role on the web are marketing via Search Engine Optimization, and viral marketing through the social web. In both cases, this relies on the incorporation of metadata within the document, typically as microdata or RDF in attributes (RDFa). The Linked Data API enables a linked data source to be viewed as structured, machine-readable (RDF, XML, or JSON) data, or in the form of human readable (HTML) documents. In this paper we discuss an extension to this API, based on Elda and Callimachus, supporting embedded RDFa. RDFa 1.0 is a W3C recommendation for embedding RDF data within HTML documents, and therefore should become an integral feature of the Linked Data API.

Introduction

Many existing consumers of structured web data are not designed to request documents and associated metadata as separate representations. Search engines such as Google, Bing and Yahoo rely on embedded markup as defined by Schema.org to improve the presentation of search results. Google Rich Snippets support both microdata and RDFa [1] and allow businesses to add organizational information, product offers and reviews. These features provide new hooks for Search Engine Optimization, and allow users to perform focused search for the products and services they need. For alternative styles of social-merchandizing, Facebook's Open Graph Protocol allows and suitably marked-up page to become an object in a social graph. For example, a product page may be 'liked' by a user and shared with their friends. Open Graph 2.0 extends this idea by adding new verbs, so that in addition to 'liking' something, you might also 'want' or 'own' it. These possibilities are all driven by web pages with embedded metadata, expressed in microdata or RDFa. We aim to support these use-cases by providing an easy-to-use RDFa extension to the Linked Data API [2].

Platform

There are a number of implementations of the Linked Data API. The Elda platform from Epimorphics Ltd [3] provides the necessary hooks for the provision of the custom viewers and formatters used in this work. To support RDFa functionality we employ components from the Callimachus framework developed by '3 Round Stones' [4]. The Callimachus project enables the rapid-prototyping of semantic-web applications using an RDFa based template engine. This means that not only is it possible to add structured RDF data to a web-page automatically, but we can also use an RDFa template to produce the SPARQL query needed to generate that structured data. Similar approaches are taken in Oort [5] and other RDFa Templating proposals [6]. The standard Callimachus distribution supports not only view templates for RDF data, but also templates for creating and editing that data. In the context of the Linked

Data API we use only Callimachus view templates. Both Elda and Callimachus are written in Java and are free open-source software.

RDFa as a template language

We illustrate the use of RDFa templates with the Edubase UK government data at data.gov.uk. A linked data version of Edubase [7] was created by Epimorphics and is available as a standard example within Elda. Views are defined in a configuration file that identifies the required properties. One of the aims of the Linked-Data API is that web-developers should not need to write SPARQL queries. Built-in viewers must construct a graph that describes the resources of interest (schools). The properties of interest are defined in the view configuration. To define a custom view, Fragment 1 specifies a custom object; the RDFaViewer, an RDFa template, and an optional stylesheet that may be applied to the template. It also defines a custom RDFa formatter that takes the results generated from the viewer and renders XHTML with embedded RDFa.

```
# The RDFa Viewer
spec:schoolsRDFa a lda:Viewer ;
      lda:name "schoolRDFa" ;
      extras:className 'org.gloze.lda.rdfa.RDFaViewerFactory' ;
      extras:template 'templates/schools.xhtml' ;
      extras:layout 'xsltsheets/schools-rdfa-layout.xsl' .

# The RDFa Formatter
spec:api a lda:API ;
      lda:formatter [a extras:RDFaFormatter ;
            lda:name 'rdfa' ;
            lda:className 'org.gloze.lda.rdfa.RDFaFormatterFactory' ;
            lda:mimeType 'application/xhtml+xml'
      ]
```

Fragment 1: RDFa configuration in Elda

If RDFa is not the default option for the requested content, it may be explicitly requested by adding the query parameter, _format=rdfa, to the request. identifying the format by name. The custom view may be explicitly identified with the additional parameter _view=schoolRDFa. Different views may be selected dependent upon the kind of resource selected. Each view may be associated with a different XHTML template.

An RDFa template is an XHTML file, already containing valid RDFa markup. As this is a template the selected resources are currently unknown; variables are introduced as resource placeholders. The value of the *about* attribute must be a URI or a CURIE; variables are strings prefixed with a '?' and are valid relative (to the current base) URLs. Variables can be used for RDFa attributes that reference resources, *about* and *resource*, and also for standard XHTML attributes, co-opted by RDFa, *href* and *src*, that may also participate in triples. Relationships, *rel* or *rev*, and *property* may not be variable. The variable, *?this*, has a special status as it is pre-bound to the resource(s) being viewed. The values of properties are similarly unknown, however RDFa syntax permits these so-called hanging properties. At runtime we substitute variables and insert property values as required. The XML in Fragment 2 illustrates the use of an RDFa template.

The custom viewer object is able to process the template with an RDFa parser, generating a corresponding RDF pattern. This is further processed to produce a SPARQL query. The generated pattern has an underlying tree structure, corresponding with the XML tree from which it is derived.

```
<?xml version="1.0" encoding="UTF-8" ?>
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
      xmlns:school="http://education.data.gov.uk/def/school/">
  <head><title>Schools</title></head>
  <body>
      <div about="?this" typeof="school:School">
            <h2 property="rdfs:label" />
            <div rel="school:typeOfEstablishment"
             resource=”?type”>
                  Type of Establishment
                  <span property="rdfs:label" />
            </div>
            <div>
                  School Capacity
                  <span property="school:schoolCapacity"/>
            </div>
      </div>
  </body>
</html>
```

Fragment 2: RDFa template

As we move deeper into the tree, triples are *chained* together such that the subject of the nested triple matches either the subject or object of the parent triple. This extends any existing query result (adding another column) using a left-join that retains all the solutions on the left-hand side of the join, even if we fail to match the right-hand side. For example, failure to match the rel="school:typeOfEstablishment" should not cause the entire query to fail. As we move down the tree, each sub-tree is independent of it's siblings; for example, given the school, the type of establishment is independent of the school capacity.

The syntax for a left-join in SPARQL is the OPTIONAL clause; every time we move deeper into the tree, we open an OPTIONAL block. Sibling branches of the query are independent, we can solve them in isolation and return the UNION of the separate results. Properties and implicit resources are un-named in the RDFa. Human readable names are created by appending the subject and property names without the namespace, and dropping '?this'. For example, the rdfs:label of ?type becomes _type_label; prefixed with an underscore to distinguish it from user-defined variables. Should name clashes occur, an integer count is appended to the variable name. Putting these elements together we produce a query of the form shown in Fragment 3.

RDFa Viewer

The downside of this query form is that to preserve the left-to-right ordering of the UNION – important for building the response in a single left to right pass – the Callimachus query of Fragment 3 cannot incorporate any ordering (ORDER BY) clauses. Similarly, because a UNION generates an indeterminate number of results, it is impossible, in advance, to select the cut point at which to LIMIT the result set, which is necessary for pagination of the output. In the context of the Linked Data API it becomes possible to insert the query used to select resources into the above (where nested queries are supported) enabling the data description to be built using a single query to the endpoint (Where nested queries are not supported, a lazy rewriting of the query, binding ?*this* with each result from the selector in turn, suffices).

```
PREFIX :<http://www.w3.org/1999/xhtml>
PREFIX rdfs:<http://www.w3.org/2000/01/rdf-schema#>
PREFIX school:<http://education.data.gov.uk/def/school/>
SELECT REDUCED * WHERE {
 {
  ?this a school:School .
  OPTIONAL {
   {
    ?this rdfs:label ?_label .
   }
   UNION
   {
    ?this school:typeOfEstablishment ?type .
    OPTIONAL {
     ?type rdfs:label ?_type_label .
    }
   }
   UNION
   {
    ?this school:schoolCapacity ?_schoolCapacity .
   }
}}}
```

Fragment 3: SPARQL derived from RDFa

This approach benefits from the flexible nature of the Callimachus RDFa template combined with the pre-selection of resources, supporting pagination and pre-sorting.

RDFa Formatter

The viewer has the job of handling the incoming request and selecting the appropriate information. The role of the formatter is to handle the return path, ultimately generating the request response. To keep the implementation modular, the formatter should not need to embody knowledge of RDFa syntax to populate the template. RDFa processing should be performed only by the RDFa parser within the viewer. However, the formatter still needs to be able to plug the results into the relevant places in the document. A side-effect of the RDFa parsing is the generation of additional metadata that indicates the origin of each user-defined and automatically generated variable in the source template. For

each variable the formatter needs to know the path as a sequence of node positions from the root '/' to the node in which that variable first appears. The variable type distinguishes between literals, resources, and blank-nodes (introduced by RDFa *typeof*). Fragment 4 shows an example of this template metadata derived from the template of Fragment 2.

```
@prefix var: <http://localhost:8080/elda/templates/schools.xhtml?> .
@prefix rdfa: <http://www.gloze.org/vocabularies/lda#> .
var:this a rdfa:Resource; rdfa:origin "/1/2/1".
var:_label a rdfa:Literal; rdfa:origin "/1/2/1/2/1".
var:_type a rdfa:Resource; rdfa:origin "/1/2/1/3".
var:_type_label a rdfa:Literal; rdfa:origin "/1/2/1/3/2".
var:_schoolCapacity a rdfa:Literal; rdfa:origin "/1/2/1/9/2".
```

Fragment 4: RDFa template metadata

This origin metadata enables result-set bindings to be associated with the XML element in which it should appear. Where multiple bindings for a variable or property are found, the containing XML is repeated for each binding. Similarly, if a variable is unbound or a property is unmatched, then the containing XML is simply removed. Where multiple resources of type *school:School* are selected the outermost *div* section is repeated accordingly. This declarative style of template provides the web developer greater control over the way the RDF properties are embedded within the XHTML document than the standard view configuration. Additional formatting and styling can be introduced without resorting to procedural formatting with XSLT. An XSLT style-sheet may be used in conjunction with the template to add common page elements such as scripts, styles, headers and footers.

Conclusions

RDFa extensions allow the Linked Data API to handle a range of important use-cases including Search Engine Optimization and Social Graphs, where metadata must be embedded within the requested document. The proof-of-concept described in this paper, based on Elda and Callimachus, demonstrates that it is possible to handle RDFa within the existing Linked-Data API architecture. In addition, declarative RDFa templates provide the web developer with greater control over page formatting without having to resort to procedural XSLT formatting.

References

[1] Ben Adida, Mark Birbeck, Shane McCarron, Steven Pemberton, "RDFa in XHTML: Syntax and Processing", http://www.w3.org/TR/rdfa-syntax/

[2] Dave Reynolds, Jeni Tennison, Leigh Dodds, et al, "Linked Data API", http://code.google.com/p/linked-data-api/

[3] Dave Reynolds, Ian Dickinson, Chris Dollin, et al, "Elda – An implementation of the Linked Data API", http://code.google.com/p/elda/

[4] James Leigh, David Wood, "The Callimachus Project" http://www.callimachusproject.org/

[5] "Oort" http://code.google.com/p/oort/

[6] Kjetil Kjernsmo, "RDFa Templating" http://www.kjetil.kjernsmo.net/software/rat/

[7] "Edubase", http://education.data.gov.uk/doc/school