The Callimachus Project: RDFa as a Web Template Language

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

[1] Sysemia Ltd, UK
steve.battle@sysemia.co.uk
[2] 3RoundStones, USA
{david, james, luke}@3roundstones.com

Abstract. The uptake of semantic technology depends on the availability of simple tools that enable Web developers to build complete applications, with particular emphasis on the last mile, the user interface. RDFa is a vocabulary for adding semantic annotations to standard XHTML and HTML5 documents. This allows the page to contain machine-readable content that is easier to find and mashable with other content. This paper describes Callimachus, an Open Source project that turns this idea around, using RDFa as a template language for the bulk generation of human-readable Web pages from machine-readable RDF data. Most existing template engines generate Web pages by combining the template with query results from a relational database. In the Callimachus template engine, XHTML+RDFa templates pass through an RDFa parser and are then compiled into SPARQL for evaluation against an RDF database. The result set is combined with the template to produce a Web page populated with RDF data, retaining the embedded RDFa. This paper evaluates the benefits and shortcomings of RDFa as a template language, and examines the relationship between Callimachus and the adoption of Web standards such as RDFa.

Keywords: RDF, RDFa, template engine, Linked Data.

1 Introduction

The Callimachus project [1] enables the rapid-prototyping of Semantic Web applications by providing an RDFa based template engine. This means that not only is it possible to add metadata annotation to a Web page, but it is also possible to add partially incomplete annotation that is automatically completed with information from a triple-store. Similar approaches are taken in Oort [2] and other RDFa Templating proposals [3]. Oort can be used to access RDF graphs as objects through a Python toolkit, as opposed to Web based tools and languages. Some of the other RDFa Templating proposals are more similar to Callimachus, leveraging HTML markup and queries against a SPARQL endpoint, differing primarily in implementation details.

Furthermore, Callimachus publishes content as Linked Data, allowing remote data access and query. It is written in Java and is available under an Apache 2.0 Open

Source license. It currently uses Sesame[1], OWLIM[2] or Mulgara[3] as an RDF database, and the Alibaba RESTful, subject-oriented programming library to provide an RDF store abstraction supporting transactional updates. This enables Callimachus to support not only view templates for RDF data, but also templates for creating and editing that data. These are essential capabilities for creating end-to-end applications. The other major benefit of Callimachus comes from its adherence to Web standards such as HTML, Javascript, and RDFa. Use of these Web standards makes Callimachus entirely platform-independent, relying solely on the architecture of the Web. However, the uptake of a new language like RDFa can be difficult, as people found to be true for XSLT, so much of Callimachus' adoption will rely on the adoption of RDFa. As usage and acceptance of RDFa increases, so will the legitimacy of Callimachus.

2 Background

RDFa [4] is an extended vocabulary for XHTML that supports fine-grained RDF annotations. RDFa offers Web authors a template language with which they can make HTML documents out of raw RDF data. HTML markup was designed specifically for enabling document layout, rather than for exposing the content for automatic processing. In addition, HTML, a mature Web standard, allows for the use of Javascript and CSS, which are also standard mechanisms for adding capability and format to Web pages. However, making a template language out of HTML requires that it be annotated somehow. RDFa, being a newly minted standard, was the logical choice to combine with HTML to create this template language. Combing RDFa with HTML enables the bulk creation of human-readable Web documents from machine-readable data as RDF.

Traditional screen-scraping techniques suffer from being ad-hoc and unreliable. Approaches like Microformats [5], Microdata [6], and RDFa overlay additional semantic information within the Web document making more of the content machine-readable. Microformats utilize existing XHTML `class` and `rel` attributes (in anchors and link tags) to add semantic content. Microformats overload the existing HTML class, normally used for selecting CSS styles. While not a formal standards process, adding new microformats requires a community process. Microdata adds name/value pairs from custom vocabularies so it is much easier to extend. The namespace of an individual microdata property, identified by an `itemprop` attribute, is defined by the most recent `itemtype` property in the DOM scope. From an RDF perspective this makes it very verbose to mix namespaces in a resource description. An `itemprop` may contain more than one property, but if a property does not belong within the namespace currently in scope, its full URL must be used (mixed `itemtype`s are even more problematic). RDFa was designed as an extended vocabulary for XHTML but is now also compatible with HTML [7]. It builds on

[1] http://openrdf.org
[2] http://www.ontotext.com/owlim
[3] http://mulgara.org

XML namespace conventions, using CURIEs to support a mixture of namespaces within a single description. It introduces a richer vocabulary for semantic markup. All of these approaches allow the addition of semantic data directly to a Web page.

3 RDFa as a template language

RDFa templating capabilities are demonstrated in this paper through a simple application for managing information about an online bookstore service and the products on offer. Service offers are designed to be consumed by semantically aware clients rather than just presented in a traditional online store-front (which is straightforward to build in the same Callimachus framework). To this end, we re-use an existing ontology for describing service offers; the GoodRelations [8] ontology for eCommerce. The catalogue contains book metadata describing (in RDF turtle) the type of product, the author and title, and a link for additional information. The eCl@ss type and properties are defined by the eCl@ss classification scheme and the codes for specific products may be found online [9]. Along with publication as Linked Data, re-use of standard ontologies and industry standard classification schemes is a key enabler for third-party data aggregation. Listing 1 shows example RDF information about books that will be made available on a Web page.

Fragment 1. RDF book data (turtle)

```
@prefix eco:
<http://www.ebusiness-unibw.org/ontologies/eclass/7.0/#>.

<ebooks97>
  # eCl@ss "Fiction book": 24-35-07-01 [ACE398005]
  a eco:C_ACE398005-gen ;

  # author [AAP839001]
  eco:P_AAP839001 "Abbott, Edwin" ;

  # Title [AAQ202001]
  eco:P_BAG165002
  "Flatland: a romance of many dimensions" ;

  # Additional link address [AAQ326001]
  eco:P_AAQ326001 <http://www.gutenberg.org/ebooks/97>.
```

The offer data is distinct from the book data. It represents an n-ary relationship between the bookstore service, the product on offer, and other details about the offer including the price. These properties are defined in the `GoodRelations` ontology. Fragment 2, below, represents an offer to sell the above book at 1 Euro described using ISO standard currency codes. An offer may include multiple products (think of

a “two for the price of one” offer) and different price specifications for different currencies.

Fragment 2. RDF service offer (turtle)

```
@prefix gr: <http://purl.org/goodrelations/v1#> .

<bookstore> a gr:BusinessEntity ; gr:offers [
    a gr:Offering ;
    gr:includes <ebooks97> ;
    gr:hasPriceSpecification [
        gr:hasCurrency "EUR";
        gr:hasCurrencyValue "1"^^xsd:float
]   ].
```

This represents the data that we aim to present in a *view* template, and capture with a *create* (or *edit*) template. Let us first consider how we would express this metadadata in XHTML with RDFa. The body fragment below is the target output for the RDFa template engine. We assume that the `gr` (GoodRelations), and `eco` (e-commerce, or eCl@ss) namespace prefixes are defined as above and are in scope. As it is possible for an offer to include multiple products, the HTML has nested tables so that a single price is associated with all included products, rather than with each individual product.

This example places a number of requirements on the use of RDFa as a template language. First, it makes reference to multiple external resources. The outermost `about` attribute establishes the subject of the entire page, the bookstore; however the page also refers to products (specifically ebooks97, defined as a relative URI with respect to the document base) which have an independent existence, and navigable via Linked Data, from the main bookstore. RDFa also permits us to include blank-nodes; the XHTML refers to the (safe) CURIE `[_:euros]` to emphasize the grouping of the price and currency code. Unlike Microdata, RDFa does not require that all property values be visible. The codified string “EUR” may be better displayed as the Euro symbol, in which case the value of `hasCurrency` can be hidden in the `content` attribute, where it can be read by a semantically aware client. The XHTML shown in fragment 3 below, shows an example of the target XHTML with embedded RDF in attributes.

The names of resources are not known at template design time, so we need to introduce some kind of variable scheme. It is important that, like the target document, the template is valid XHTML + RDFa. 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. This use of variables within RDFa certainly pushes the design envelope for RDFa, but is nevertheless conformant with the RDFa standard, and furthermore produces valid RDF when processed by an RDFa

parser. Consider also that the resulting RDF graph, including *variable* URIs can be made *meaningful* in an open-world by asserting additional triples that create bindings with owl:sameAs.

Fragment 3. The XHTML + RDFa target output.

```
<body about="bookstore" typeof="gr:BusinessEntity">
 <table>
  <div rel="gr:offers">
   <tr typeof="gr:Offering">
    <td><table rel="gr:includes">
     <tr about="ebooks97">
      <td property="eco:P_AAP839001">Abbott, Edwin</td>
      <td property="eco:P_BAG165002"
       >Flatland: a romance of many dimensions</td>
     </tr>
    </table></td>
    <td rel="gr:hasPriceSpecification"
     resource="[_:euros]"> €
     <span property="gr:hasCurrencyValue">1</span>
     <span property="gr:hasCurrency" content="EUR"/>
    </td>
   </tr>
  </div>
 </table>
</body>
```

At template design-time, the values for the various properties are unknown. In the simplest case values are inserted as text-nodes wherever the relevant `property` appears in the markup. RDFa was not created as a query language and there is no way to indicate whether or not a property should be mandatory or optional. The weakest assumption is to display whatever data we can find, assuming that properties are optional. However, if the designer opts to include a property value, then matching resources must be compatible with that value. In this case the property is a mandatory condition on its subject. This can be used to good effect in the example, we may retain the currency "EUR" in the template to select for matching Euro price-specifications. Note that this feature is independent of whether the content is represented as a text-node or in a content attribute, as in the example.

To produce the template we substitute resources for variables (`?this`, `?product`, `?currency`) and remove (optional) property values. These properties are not undefined, but simply empty. The variable, `?this`, has a special status as it is pre-bound to the URL of the resource that is currently being viewed, in this case the bookstore. The template for the bookstore example is shown in fragment 4, below.

Fragment 4. The RDFa template.

```
<body about="?this" typeof="gr:BusinessEntity">
 <table>
  <div rel="gr:offers">
   <tr typeof="gr:Offering">
    <td><table rel="gr:includes">
     <tr about="?product">
      <td property="eco:P_AAP839001"/>
      <td property="eco:P_BAG165002"/>
     </tr>
    </table></td>
    <td rel="gr:hasPriceSpecification"
     resource="?currency"> €
     <span property="gr:hasCurrencyValue"/>
     <span property="gr:hasCurrency" content="EUR"/>
    </td>
   </tr>
  </div>
 </table>
</body>
```

An effective template language must provide fine control over which elements are included and repeated in the output. Most template languages support some kind of iteration allowing sections of markup to be repeated. We saw earlier that an offering may include multiple products; that is, the `?product` variable may have multiple bindings for the same offering. This means that the `tr` (table-row) element and all its children are repeated for every bound product. Conversely, if there were no bindings for `?product` then the `tr` element, and its children, would not appear at all. Similarly, a single business entity would have multiple matching offers. Because we omitted a variable to represent offerings, it is not immediately clear which fragment of markup should be repeated. However, the RDFa specification states that the `typeof="gr:Offering"` (with no `about` attribute) implicitly introduces a blank node on that `tr` element. This means that multiple offers will repeat that element, and if there are no offers there will be no rows at all. We therefore have a simple nested loop, with an outer loop that iterates over offers, and an inner loop that iterates over products.

This example introduces many of the key concepts of template languages, namely variables, text substitution, conditionals, and iteration.

4 Page Generation

The process of generating a page from RDF data and an XHTML+RDFa template comprises two basic steps:
1) The compilation of the RDFa template into a SPARQL query, and
2) the population of the template from the SPARQL result set.

4.1 Compilation to SPARQL

The RDF content necessary for producing a SPARQL query is derived from the template using an RDFa parser. Callimachus uses a streaming parser, so the output is a sequence of RDFa events that represent a) namespace or base declarations b) the opening (or closing) of a new subject context, and c) RDF triples. The parser output below shows just the triple events received from the RDFa parser when parsing the RDFa template. They are indented to indicate the presence of a subject context; note that all triples within a contiguous block at a given indentation, share the same subject. This follows from the rigid tree-structure of the RDFa as it shadows the XML DOM. In this form we can clearly see the empty property values implied by the template, notice also the variables, preserved as relative URLs.

Fragment 5. RDF derived from the parsed template.

```
<?this> <rdf:type> gr:BusinessEntity
    <?this> gr:offers _:n7
        _:n7 <rdf:type> gr:Offering
        _:n7 gr:includes <?product>
            <?product> eco:P_AAP839001 ""
            <?product> eco:P_BAG165002 ""
        _:n7 gr:hasPriceSpecification <?price>
            <?price> gr:hasCurrencyValue ""
            <?price> gr:hasCurrency "EUR"
```

The RDF derived from the parsed template is illustrated in fragment 5, above. In this form, it is easier to see the underlying tree structure of the RDFa. As we move deeper into the tree, triples are *chained* together such that the subject of the nested triples matches either the subject or object of the parent triple. As we move down the tree, each sub-tree is independent of it's siblings; for example, the `?product` is independent of the `?price`. RDFa is only able to generate tree-structured graphs (even if a URI constant were introduced at different points in the tree, the fact that it *is* constant breaks the dependency). This makes for efficient queries without cycles.

Moving deeper into the tree will correspond to extending an existing result; adding another column. Following the principle of *anything is better than nothing*, chained triples are not represented by a full-join, but by 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. 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. The exception to this is mandatory (conditional) triples that must be joined with the existing result set so that if the join fails, these non-matching results are eliminated. This means that the conditional, `<?price> gr:hasCurrency "EUR"`, triple must be joined with the parent it is chained to. Where different parts of a query are independent, we can solve them in isolation and return the UNION of the separate results; therefore every time we move down the tree, we open a UNION block.

Fragment 6. SPARQL compiled from the parsed RDF.

```
PREFIX gr:<http://purl.org/goodrelations/v1#>
PREFIX eco:<http://www.ebusiness-
unibw.org/ontologies/eclass/7.0/#>
SELECT REDUCED *
WHERE {{
  ?this a gr:BusinessEntity .
  OPTIONAL {{
    ?this gr:offers ?_offers .
    ?_offers a gr:Offering .
    OPTIONAL {{
      ?_offers gr:includes ?product .
      OPTIONAL {{
        ?product eco:P_AAP839001 ?_product_P_AAP839001 .
       }
       UNION {
        ?product eco:P_BAG165002 ?_product_P_BAG165002 .
     }}}
     UNION
     {
      ?_offers gr:hasPriceSpecification ?price .
      ?price gr:hasCurrency "EUR" .
      OPTIONAL {
       ?price gr:hasCurrencyValue ?_price_currencyValue .
}}}}}}}
```

While the RDFa includes user defined variable names, there still remain lots of anonymous variables that need to be plugged back into the template later on. This includes all the empty property values, and anywhere a blank node (such as _:n7) is generated in the parsed RDFa. There is no choice but to automatically generate names for these anonymous variables. The user is able to interrogate the template for the generated SPARQL query, so it was an important design choice that the final query be human readable. Generated names are prefixed with an underscore character to distinguish them from user-named variables. The name is derived from the subject and property names (stripping the namespace prefix from the property predicate for brevity). For predicates of the commonly used forms, “hasX” or “inX”, only the suffix “X” is used. Similarly for a reversed predicate, “XBy”, the suffix “By” is removed if present, or otherwise added. If the subject name is “?this” it is ignored. Finally, the two parts of the name are conjoined with an underscore. We can see how simple naming heuristics of this kind improve readability in the example where `?price` and `gr:hasCurrencyValue` are combined to produce the new name, `?_price_currencyValue.` Finally, in cases where these naming rules result in a naming collision, a numeric count is appended to the name to distinguish them.

The production of SPARQL is streamed, enabling the output of the RDFa parser to be piped directly to the input of the SPARQL compiler. For the most part, the order of triples in the RDFa parser output is the same as those in the SPARQL stream. There is, however, a need to look-ahead in the event stream in special cases. The query should be more efficient if joins are evaluated before left-joins. The conditional, `<?price> gr:hasCurrency "EUR"`, triple is promoted in the stream so that it appears before it’s optional sibling.

4.2 Population of template with SPARQL result set

The final form of the query is a SPARQL SELECT query, generating a tabular result set. Page generation combines two streams, the result set and the template XML stream. Streaming is important because even with a potentially explosive result set the server should not have to keep an indefinitely large result document in memory. To achieve this we have to assume that the SPARQL UNION preserves the left-to-right ordering of the result set, rather than some other arbitrary interleaving. This is true of Sesame, Mulgara, and Jena ARQ, subject to how subsequent operations process the result set (e.g. an ORDER-BY clause would, by definition, disrupt this natural order). This assumption allows us to regard the result set as a tree, with each result defining a single path in the tree, and with these results arranged in a pre-order traversal of the target document. This means that results towards the beginning of the document appear first, while those towards the end appear last.

Broadly speaking, an outer loop consumes the XML template, while an inner loop consumes the result-stream.

Fragment 7. SPARQL Annotation comments tying variables to their XML origins

```
# @origin this /1/2
# @origin _offers /1/2/2/2/1/1 _
# @origin product /1/2/2/2/1/1/1/1/1
# @origin _product_P_AAP839001 /1/2/2/2/1/1/1/1/1/1 !
# @origin _product_P_AAQ326001 /1/2/2/2/1/1/1/1/1/2/1 eco:P_AAQ326001
# @origin _product_P_BAG165002 /1/2/2/2/1/1/1/1/1/2/1/1 !
# @origin price /1/2/2/2/1/1/2
# @origin _price_currencyValue /1/2/2/2/1/1/2/1 !
```

The key to re-populating the RDFa template lies in knowing how particular variable bindings are plugged back into the source template. This information is generated in the initial parse of the RDFa and is saved in the form of annotations that are added to the compiled SPARQL. Examples of these annotation comments are shown in fragment 7, above. This means that the Page generation stage does not need to be RDFa aware. As the template engine reaches a node in the tree, it uses the path to that node to pull assignments from the current result. For example, on reaching the `body` element at "/1/2" it assigns the current value of `this` from the result to that node. For completeness, it needs to ensure is that an XML element is only output if *all* variables mapped to that element are bound. This is trivially true if the current element is regular markup with no RDFa properties. But, for example, it knows not to include the `tr` (table-row) element at "/1/2/2/2/1/1" if there is no current binding for _offers. It can do this entirely without knowledge of RDFa, ensuring that all of the RDFa logic is encapsulated in the RDFa parser.

Only when all the bindings from the current result have been assigned, can the next result be consumed. It must be checked for consistency with node assignments in the current context. The template engine has to effectively backtrack to a point in the tree where the current result is consistent with the assignments.

5 Evaluation

We developed a series of surveys to investigate both objective and subjective measures of the Callimachus interface in an effort to evaluate the effectiveness of the Callimachus template engine for displaying structured data content. However, evaluating Callimachus in a vacuum would be to neglect other commonly used alternative interfaces. This comparison is a crucial aspect of understanding exactly where value can be derived from in a certain model.

User interfaces for RDF generally take one of three forms: Showing some form of "raw" triples, using "ball-and-stick" diagrams to visually represent RDF graphs or using hypertext to represent sections of graphs. We thus needed to define proxies for these three styles of interfaces in order to compare the general approaches. We chose RDF Gravity [14] to represent ball-and-stick diagrams (showing resources as balls and relationships as sticks) and the RDF Turtle syntax as a common representation of triples. The three options outlined here (Callimachus, RDF Gravity, and Turtle) were each assigned their own identical survey, and were primarily used to objectively evaluate the ability of a user to extract information from a novel data set.

However, it is important to recognize that this only depicts a portion of the bigger picture of a data model. If we as a community expect the average user to choose one particular model over another, then factors such as "clear information" and being "easily readable" are going to come into play just as much as how well the data is actually represented. Thus, we created a fourth survey comparing all three models on a single page, asking subjective evaluation questions that may be left unanswered by a survey that looks only to evaluate a user's ability to draw out information.

A total of 169 people participated in our surveys (Callimachus - 45, RDF Gravity - 48, Turtle - 46, Comparison - 30). To ensure no differences in question difficulty between models, the questions on all three model surveys were identical, with the representations being drawn from the exact same data sets. Questions were asked about the information being presented, with increasing levels of complexity as the user moved through the survey. Users were always given the option of answering "I don't know" if they were unable to locate the answer to a question. The comparison survey was a bit different, displaying all three models at once and asking the participant questions such as which model they would be more likely to use and why.

Considering the structure of our surveys, we thought it would be most interesting to compare the number of times a participant answered "I don't know" between the three different models. To do this we conducted a between-subjects ANOVA where the number of times someone responded "I don't know" was the dependent variable and the model they saw (Callimachus, RDF Gravity, or Turtle) was the independent variable. There was a significant effect of model evaluated, $F(2, 136) = 35.25$, $p < .001$. Following the significant effect, Tukey's post-hoc analyses showed that there were significant differences between all groups. RDF Gravity ($m = 21.44$) had a significantly higher number of "I don't know" responses than either Turtle or Callimachus. Turtle ($m = 10.33$) had a significantly higher number of "I don't know" responses than Callimachus and a significantly lower number of "I don't know" responses than RDF Gravity. Callimachus ($m = 2.55$) had a significantly lower

number of “I don’t know” responses than either RDF Gravity or Turtle. Responses are summarized in Figure 1.

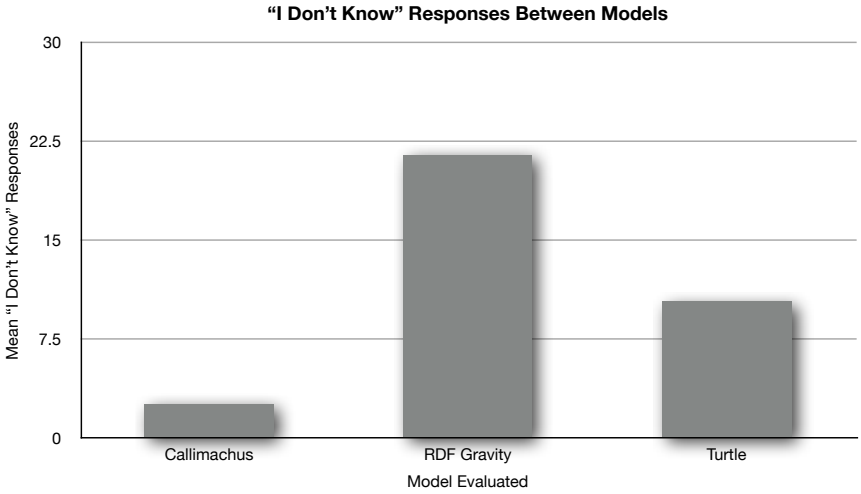

Figure 1. Mean Comparison Between Models of “I Don’t Know” Responses

Lastly, we did a basic analysis of the survey comparing all three models. In this survey we asked participants to rank the models from 1-3 (1 being the best, 3 being the worst) with regards to “visual appeal” and “likelihood of use.” Participants also recorded why they chose their highest and lowest rank models from a list of possible factors with the option to add their own justification if the reason was not present.

The most telling statistic was simply the mean ranking of the three different models. In response to which they found most visually appealing, the average rating for Callimachus was 1.17, RDF Gravity was 2.07, and Turtle was 2.77. As you can see there was a small amount of variation in ranking, but overwhelmingly Callimachus was the most visually appealing model, followed by RDF Gravity and Turtle, as shown in Figure 2.

In response to the question concerning which model they would be most likely to use when exploring a new set of information, the results were not as pronounced but came to the same conclusion. Callimachus had the highest average ranking at 1.57, followed by RDF Gravity which had an average ranking of 1.80, followed up by Turtle with an average ranking of 2.63, as shown in Figure 3.

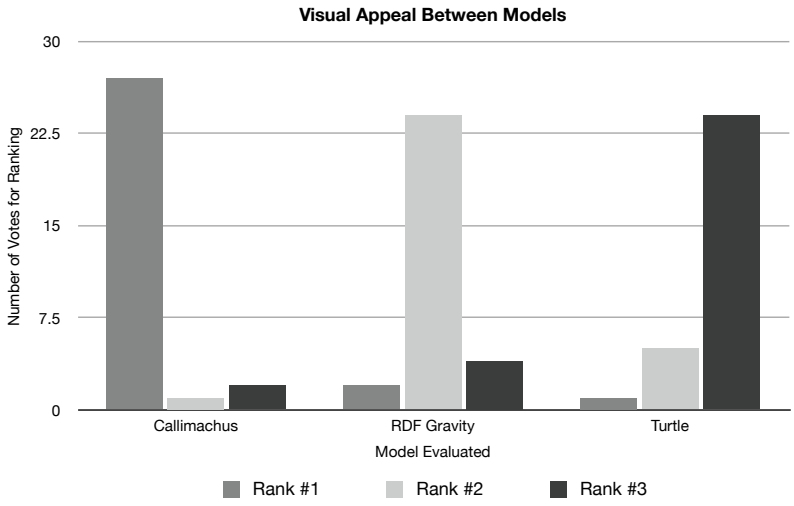

Figure 2. Visual Appeal Between Models

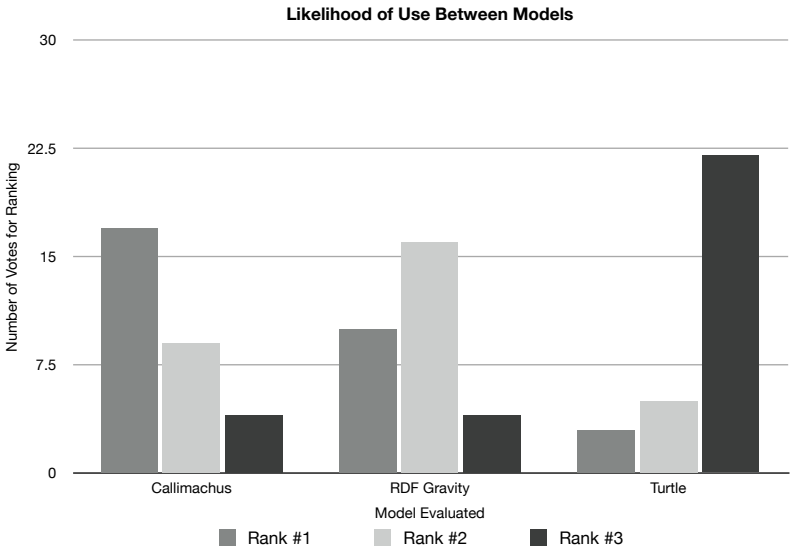

Figure 3. Likelihood of Use Between Models

A brief visual examination of the two interfaces shown below reveal the organization of similar properties in different ways. In Figures 4 and 5 below the green box surrounds description information such as the name of the facility and the ID number; the red box contains latitude and longitude information; and the blue box surrounds address information. Comparing these boxes between the two interfaces reveals not only a more easily readable version of the data in Callimachus, but also

the ability to create interesting visualizations using the data – a map with the latitude and longitude plotted.

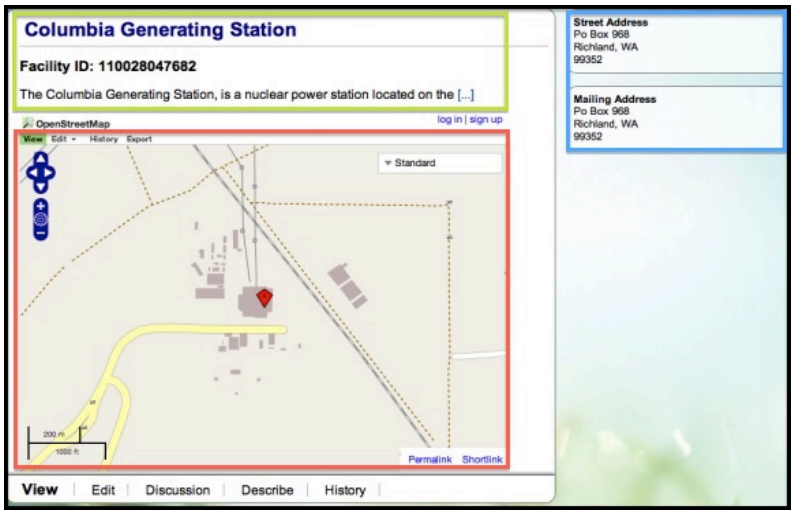

Figure 4. Callimachus Interface

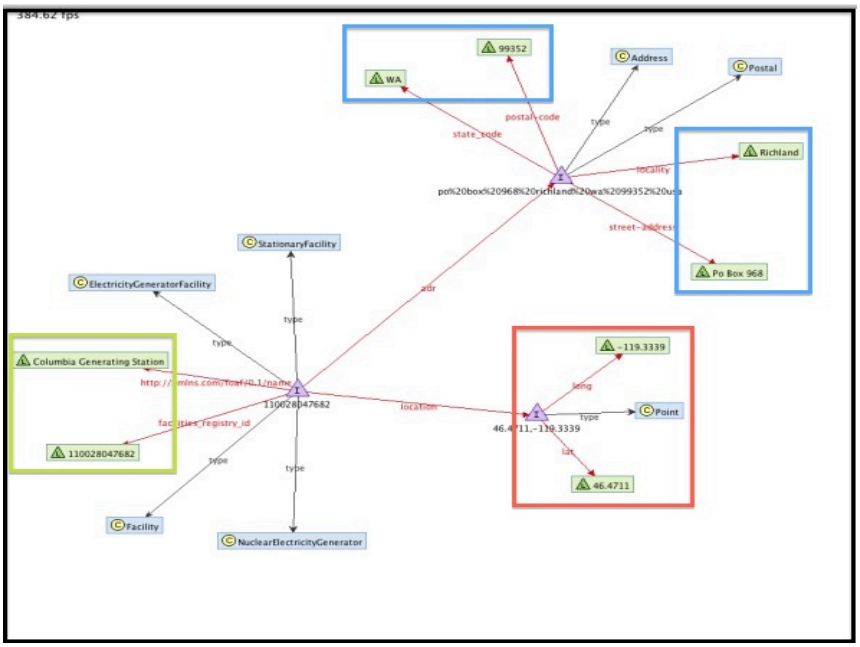

Figure 5. RDF Gravity Interface

These results and findings closely echo the results of the objective measures, indicating that not only do people find the user experience of Callimachus superior to

its peers, but they are also able to extract the highest amount of, and most accurate, data from its interface.

6 Conclusion

We have shown that, despite not having been designed as a template language, RDFa performs well in that role. Moreover, it provides the missing link between the "Web of Data" and the "Web of Documents", making it far easier to create views on Linked Data. Web pages generated in bulk from the combination of Linked Data and XHTML+RDFa templates allow for better human readability of complex data, as shown in our survey results. The utilization of widely-accepted and reviewed Web standards like HTML, Javascript, and CSS in conjunction with emerging standards such as RDFa allow people publishing to the Web to easily create large amounts of HTML pages from RDF data. These features establish Callimachus as a powerful platform for creating fully rounded Linked Data applications with both human- and machine-readable content.

References

[1] 3 Round Stones (2009) The Callimachus Project. http://www.callimachusproject.org. Accessed 1 August 2012.
[2] Lindström, N. (2006) Oort – a Python toolkit for accessing RDF graphs as plain objects. http://oort.to/. Accessed 1 August 2012.
[3] Kjernsmo, K., RDFa Templating. http://www.kjetil.kjernsmo.net/software/rat/. Accessed 1 August 2012.
[4] Adida, B., Birbeck, M., McCarron, S., Herman, I. (2012) RDFa Core 1.1. http://www.w3.org/TR/rdfa-core/. Accessed 1 August 2012.
[5] Berriman, F. et al., Microformats. http://microformats.org/. Accessed August 1 2012.
[6] Hickson, I. (2012) HTML Microdata. http://www.w3.org/TR/microdata/. Accessed 1 August 2012.
[7] Adida, B., Birbeck, M., Pemberton, S. (2012) HTML + RDFa 1.1. http://www.w3.org/TR/rdfa-in-html/. Accessed 1 August 2012.
[8] Hepp, M. (2008) GoodRelations: The Web Vocabulary for E-Commerce. http://purl.org/goodrelations/. Accessed 1 August 2012.
[9] eCl@ss, eCl@ss classification and product description 7.0. http://www.eclass.de. Accessed 1 August 2012.
[10] Heath, T. & Bizer, C. (2011) Linked Data: Evolving the Web into a Global Data Space (1st edition). Synthesis Lectures on the Semantic Web: Theory and Technology, 1:1, 1-136. Morgan & Claypool.
[11] Pilgrim, M. (2010) "HTML5: Up and Running". O'Reilly Google Press.
[12] Hepp, M., Leukel, J., Schmitz, V. (2006) A Quantitative Analysis of Product Categorization Standards. Springer. doi: 10.1007/s10115-006-0054-2
[13] Wood, D. (Ed.) (2010) Linking Enterprise Data. Springer.
[14] Goyal, S. & Westenthaler, R. RDF Gravity. http://semweb.salzburgresearch.at/apps/rdf-gravity/. Accessed 1 August 2012.