From Graph to GUI: Displaying RDF Data from the Web with Arago

Hannes Gassert[1] and Andreas Harth[2]

[1] Université de Fribourg, DIUF/PAI, Chemin du Musée 3, Fribourg/Switzerland.
hannes.gassert@deri.org
[2] Digital Enterprise Research Institute, NUIG, University Road, Galway/Ireland.
andreas.harth@deri.org

Abstract. In this paper we discuss approaches to display user interfaces for RDF data. Scripting languages such as PHP are used for building user interfaces on the current Web, and are a strong candidate for constructing UIs on the Semantic Web as well. We show how to use RDF for expressing and applying presentation knowledge in a way that is flexible enough to deal with arbitrary data from the Web. We present Arago, an early implementation of a presentation engine that utilizes Fresnel, an RDF display ontology.

Keywords

Semantic Web, RDF User Interfaces, PHP5, XML, N3

1 Introduction

Data on the Semantic Web is meant for consumption by machines. However, most current Semantic Web scenarios are centered around humans: as creators of data, as programmers and, last but not least, as end users. Viewing and browsing RDF in a convenient graphical user interface (UI) is crucial for each of these roles.

The question we try to answer is how to display Semantic Web data with minimal hardwired knowledge about the vocabularies used in the data set. In the scenario discussed in this paper, we look at instance data gathered from the Web, for which we cannot make any guarantees. The main goal is to explore ways to leverage *presentation knowledge* to create UIs that are able to display specific kinds of resources.

There are a couple of differences to currently used UI creation methods that are particular for RDF data from the Web:

- weak knowledge about the schema of the data; even if we know, we cannot be sure whether the instances adhere to the schema, or whether they use other vocabularies as well
- we want to specify the presentation knowledge declaratively; i.e, we want to carry out the rendering process based on a declarative specification

UIs can be identified as one of the areas where scripting languages have gained a strong foothold in industry. Particularly on the Web, server-side scripting

languages are driving a massive amount of generated pages on sites where short development cycles and fast adaptability are important. We believe that scripting languages are going to play a key role in the area of user interfaces for the Semantic Web because they enable developers to quickly prototype systems that exploit the new possibilities.

The rest of the paper is organized as follows: First, we survey existing approaches to displaying RDF which are based on CSS, XML, and XSLT. Next, we discuss Fresnel [3], an RDF-based display vocabulary currently under development. Then, we describe Arago, an early implementation of a Fresnel display engine based on the PHP5 scripting language

2 Related Work

We identified two approaches in prior work on user interface/output generation based on RDF: the model perspective, which perceives RDF data as a set of interconnected instances in a semantically coherent model, or the markup-centric view, which sees the data in a particular (XML) serialization. XML, originating in the publishing world, has proven to be a strong underlying format for many kinds of documents. A multitude of vocabularies (HTML, XSL:FO, SVG,...) have been used to express both data and style information. A great many browsers and renderers are in use today, and industrial strength publishing frameworks like Cocoon[3] provide highly sophisticated means for complex XML-based output generation tasks.

2.1 Markup-level processing

Processing RDF data on the XML level with XML tools is possible when one preprocesses the RDF and derives a canonical serialization of the RDF. Working with RDF as XML benefits from a large range of powerful tools developed for XML publishing processes. Experiences with new approaches being developed will have to be compared with these established methods. The number of existing tools and concepts in this area are evidence both of the practical importance and the maturity of the field.

RDF plus CSS Departing from an XML-centric perspective, CSS can be used directly on the RDF document, using namespace-aware selector as proposed in the CSS Namespace Enhancements document [4] and implemented in all major HTML browsers. While this approach is limited due to the inability of CSS to restructure the document, it can already be useful for very simple cases that do not require such alterations. For an example applying CSS2 to an unprepared RDF (FOAF[2]) document see `http://www.gassert.ch/foaf/hannes.rdf.xml`. However, just using CSS2 on RDF directly doesn't allow to display an URL as a hyperlink or replacing the URL of an image with the image itself.

[3] `http://cocoon.apache.org/`

RDF plus XSLT plus CSS The limitations of the approach above can be addressed by adding an additional processing phase after an XML canonicalization, applying an XSLT to the RDF/XML document. The full power of XSLT, a Turing-complete functional programming language for XML transformation can now be used to reshape the document into a form suitable for rendering, delegating styling to CSS. This combined approach, which can be considered as the current state of the art, benefits significantly from the maturity and wide deployment of XML toolkits.

While it can be argued that a process aiming at displaying RDF should therefore be built on XML technologies as early as possible, the authors don't consider it advisable to abandon the benefits and capabilities of RDF, RDFS and OWL at an early processing stage. The power of such approaches is limited by the fundamental differences between RDF and XML: Working on graphs with formal semantics using tree-oriented tools has been found to be suboptimal by a number of authors, resulting in tools like Treehugger [9] or RDF Twig [10]. In the absence of a standard canonicalization of RDF/XML both approaches offer but very limited reuse of the stylesheets and the presentation knowledge encoded within them.

For an example applying XSLT and CSS on RDF instance data that has been preprocessed (smushing, serialization) by CWM see Figure 5, a screen-shot taken from `http://sw.deri.org/search`. The PHP-based prototype uses a set of XSLT templates for instances of classes like FOAF persons, DBLP documents, or RSS items as well as a generic template for instances of unknown type.

2.2 Model-level processing

From a theoretical point of view, model-level processing is the preferred way of dealing with RDF: all its semantic richness can be accessed and exploited for processing, logical reasoning can be applied and rules be used to generate output from the model directly. The latter has been done in the work of C. Lung [5], using N3 output rules to directly generate HTML from the RDF model. This elegant rule-based approach inherently has a strong focus on markup, which makes such "logic stylesheets" not very reusable. The presentation knowledge encoded into the N3 rules will have to be entirely rephrased when targeting a different output format.

A number of other applications directly use programming languages to extract and style the contents of an RDF model, hard-coding all the presentation knowledge. This leads to similar effects as in the above approach, namely limited reusability across applications.

Striving for a means of expressing such presentation knowledge in a flexible, interoperable and integrative way, researchers have started to use RDF to model presentation knowledge. Specified and implemented by the Haystack [7] team, the Xenon [8] display ontology is an example of such practice. One of Xenon's main concerns is flexible recombination of presentation knowledge, which is necessary when dealing with combined RDF data from multiple ontologies and in different contexts. Xenon is modeled after XSLT, but replaces the XPath tree traversal language with an RDF query language. While it has been argued that Xenon, in building a powerful and highly expressive language on top of RDF, goes too far in building a full programming language on top of RDF we very

much agree with the general idea of expressing presentation knowledge in RDF. The benefits of RDF, its flexibility and formal semantics, should be also applied to meta-data describing the presentation of resources. In this way, resources can be self-descriptive also in respect to their representation displayed to humans. We think that a special-purpose declarative language allowing a suitable level of abstraction can substantially facilitate the process of expressing such knowledge. The conceptual and practical benefits compared to lower-level, graph-oriented approaches as mentioned beforehand can be seen in this improved semantic expressivity, and in the ability to use the same languages and tools for both data and its presentation.

3 Data Set

In our example scenario we use both data gathered from the Web and data converted to RDF from large databases. The data set from the Web [4] consists of around 2.5m statements in 14146 files, connected via `rdfs:seeAlso` links. The overall size of the dataset is 356 MB. A large number of the individuals are described in FOAF and RSS, additionally we have to deal with instances in a large number of numerous vocabularies. Figure 1 shows an exemplary instance of `foaf:Person` with usual properties.

```
_:boid23 rdf:type foaf:Person;
    foaf:name "Stefan Decker";
    foaf:firstName "Stefan";
    foaf:lastName "Decker";
    foaf:title "Dr";
    foaf:mbox <mailto:stefan.decker@deri.org>;
    foaf:mbox_sha1sum "1bc1f862b688a45b7e0c8d4a8467c23177c53fad";
    foaf:depiction <http://www.isi.edu/~stefan/sde.gif>;
    foaf:knows _:boid24;
    foaf:knows _:boid32.
```

Fig. 1. A sample (partial) FOAF file in RDF/N3.

4 Presentation Knowledge in the Fresnel Vocabulary

RDF-based user interfaces are built of three kinds of knowledge: that comprised in the RDF statements itself, additional information from the schema, and usually a fair amount of presentation knowledge regarding visual display. While both data and schema are being expressed in RDF, similarly common means of expressing display knowledge are currently lacking. This kind of information is being expressed in other languages, explicitly or implicitly, most of the time in procedural manner.

[4] `http://sw.deri.org/2005/04/semWebbase/`

In many cases, deriving a visual representation of an RDF graph is equivalent to transforming it to a specific XML/DOM tree. Trees have proven to be a suitable model for documents and their usually ordered and hierarchical structure; therefore, choosing a tree model when transforming a graph model for presentation is a reasonable choice.

Presentation knowledge has to encompass statements about how to perform such a transformation in a meaningful way, preserving as much of the original semantics as possible. This transformation step determines the structure and content of the visual representation. Additionally, presentation knowledge includes information on styling, covering topics like colors, fonts, and positioning, for which – on the current Web – CSS is the language of choice. This leads us to Fresnel, which has been called a "CSS for RDF".

According to its (preliminary) online manual [1] Fresnel is a "simple vocabulary for specifying which parts of an RDF graph are displayed and how they are styled using existing style languages like CSS." Fresnel is a very recent[5] approach developed in a joint effort by the teams of SIMILE[6], IsaViz[7] and RAP[8]. Fresnel builds on and interfaces with CSS and is purely declarative. It applies directly to the instance level instead of making statements on the graph-level like for example GSS [6].

In order to describe the presentational properties of such instances, Fresnel defines two main concepts for expressing presentation knowledge: *Lenses*, dealing with content selection and structuring, and *Styles*, dealing with properties such as colors determining the visual display of the selected data.

```
:foafPersonShortLens rdf:type fresnel:Lens ;
    fresnel:lensDomain foaf:Person ;
    fresnel:purpose fresnel:defaultLens;

    fresnel:showProperties ( foaf:name
                             foaf:depicts
                             foaf:mbox
                             fresnel:allProperties
                           ) ;

    fresnel:hideProperties ( foaf:title
                             foaf:mbox_sha1sum
                           ) .
```

Fig. 2. A simple fresnel:Lens for a FOAF person in RDF/N3.

[5] There are not yet any reviewed and published results on the Fresnel language itself. This early discussion is therefore based on the preliminary manual as of the end of March 2005, as well as on numerous online discussions.

[6] http://www.simile.mit.edu/

[7] http://www.w3.org/2001/11/IsaViz/

[8] http://www.wiwiss.fu-berlin.de/suhl/bizer/rdfapi/

These Lenses and Styles are to be applied to RDF instances, i.e. typed resources and their properties. An interpretation of this model consists in first finding all Lenses whose `fresnel:lensDomain` matches with an instance to be displayed, taking into account the application context a lens is said to be suitable for as stated in its `fresnel:purpose` property. The application of a lens to an instance results in a new, structured instance containing an ordered list (or tree) of the properties selected by the lens. Please see Figure 2 for a simple Lens to be applied by default to `foaf:Person` instances. The lens first selects three properties considered important for display, followed by all the properties of the instance. Additionally, the lens specifies that the properties `foaf:title` and `foaf:mbox_sha1sum` are not displayed. In this way, Fresnel lenses allow for mixed inclusive/exclusive display schemes applied to RDF instances[9]. After the selection phase carried out according to the Lens definitions, Fresnel Styles are applied in a similar manner, where styling instructions are associated with the different parts of the structured RDF instance.

In the following paragraphs we discuss a number of practical aspects of our work with Fresnel, pointing out some of its features important for our scenario. We leave a formal introduction of Fresnel's semantics and the associated processing model to the upcoming paper by the Fresnel core group.

```
:styleHomepage rdf:type fresnel:Style ;
    fresnel:styleDomain foaf:homepage ;
    fresnel:propertyStyle "color: blue"^^fresnel:cssStylingInstruction ;
    fresnel:valueStyle [ fresnel:contentNoValue
                "Homepage not indicated."^^xsd:string ] .
```

Fig. 3. Handling missing content with Fresnel style.

When working with RDF data integrated from various (uncontrollable) sources, two situations affecting the user-friendly display of information can easily arise: Often there is either too much data in an instance, or not enough. Fig. 3 presents an example of "presentation knowledge" to be applied to the `foaf:homepage` property. Besides stating that this property should be blue, the Style further states that in case of a missing property value a certain string is to be displayed instead.

[9] Note that there are issues with the exact definition of an "instance", the so-called "Decker problem" (see `http://lists.w3.org/Archives/Public/public-sws-ig/2004Feb/0037.html`). Fresnel (partially) addresses this issue by introducing the term of "recursive sublens relationships", which includes a property for defining the number of "hops" to follow when determining the content to be displayed.

5 Arago, a Fresnel Implementation

Arago[10] is our early PHP5 implementation of the Fresnel specification as of the end of March 2005.

Using Fresnel to drive a scripted UI layer combines the very practicable application of scripting languages with the strengths of Fresnel as an RDF-based encoding for presentation knowledge. By implementing Fresnel in PHP5 we hope not only to create a UI layer matching the demands of our integration and search system, but also help to advance the development of Fresnel itself.

To date, Arago implements the core Lens and Style vocabularies along with an XHTML serializer. The examples included in this paper can be successfully parsed, interpreted and applied to real-world instance data. In Figure 4 we present the class diagram of our implementation. The overall control flow is coordinated by the main `Arago` class, which delegates to "management and matchmaking" classes for Lenses and Styles, respectively. These classes manage the process of extracting, matching and applying Lenses and Styles.

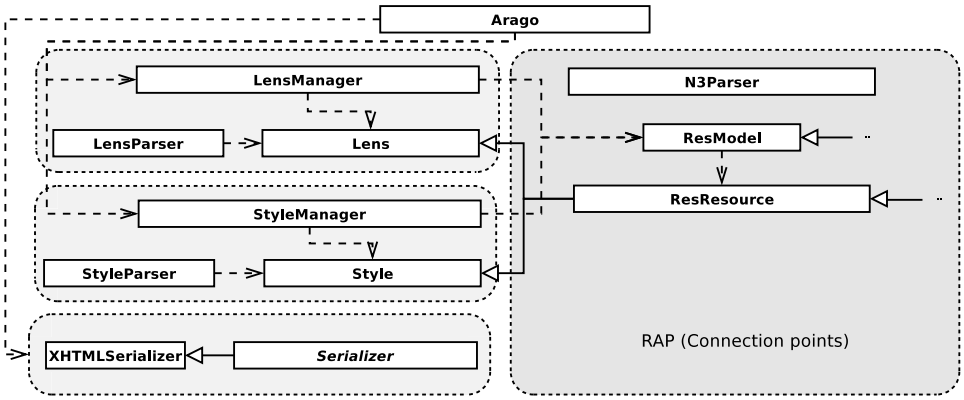

Fig. 4. Abridged UML class diagram of Arago

5.1 Selection - Structuring - Styling

As a first step, we extract a superset of the subgraph going to be displayed. Possible sources are files, any RAP model (in-memory, database-backed,...) or a YARS repository interfaced with using RDF/N3 SPARQL-like queries over HTTP. In our example scenario, the resulting RDF contains an instance of `foaf:Person` together with arbitrary RDF for which we have no presentation knowledge.

[10] Fresnel is named after a French physicist of the 19th century, an inventor of innovative lenses. Thus we named our implementation after a colleague and friend who warmly supported Augustin Fresnel: Francois Arago, known for his "adventurous conduct in the cause of science". The software package is to be found online at `http://sw.deri.org/~hannes/arago/`.

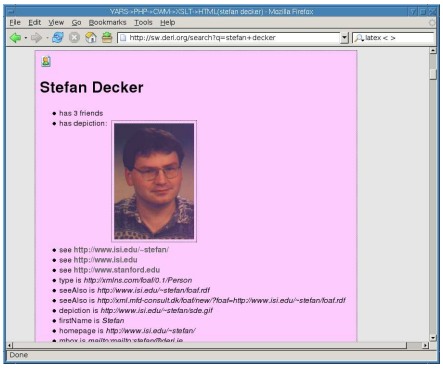

Fig. 5. Screenshot: Rendering of the example FOAF instance

We then hand over control to the `LensManager`, which matches the instance data with the available Lenses in order to find the correct Lens to apply to each instance identified. Matching is carried out according to domain indications of Lenses and Styles — if no applicable Lenses or Styles can be found, the generic display indications applying every `owl:Thing` provided in the Arago distribution. The Lenses found in this process will be then applied to the instances to be displayed, effectively determining *what* will be displayed. As a visual representation usually implies a certain order, and usual RDF does not, we need to create an ordered sequence of properties. This is done by consecutively applying the indications in `fresnel:showProperties` and `fresnel:hideProperties`, taking into account `fresnel:allProperties`. These operations are carried out (without loss of information) on an array structure, because array manipulation in PHP is both considerably easier and faster. Because mapping these ordered, hierarchical data structures to RDF is rather inconvenient, we pass them right on to the styling component. Writing the changes back into the RDF model for consumption by the next component would have been worthwhile for interoperability reasons, but doing so has turned out to be not feasible.

The styling component, taking control after the application of Lenses, is structured similar to the Lens part. Again a management class implements the process of bringing together instances and the styling instructions to be applied to them and their properties. This involved the application of "container styles", which determine how an instance as a whole is to be embedded and displayed in the output area, label lenses specifying the style in which instances (resources) are to be labeled, as well as value and property styles, which specify the visual display of single properties and their respective values. For all instances and all their properties (and possibly instances connected to them), these styling instructions are collected and linked into an intermediate structure. This application is steered by objects of the `Style` class, again like the `Lens` class implemented as "active resources", resources with behavior. The resulting structure of these processing steps is then serialized as XHTML output stream by iterating over the statements in the order determined in the selection/structuring phase. As typical with data from the Web, additional care has to be taken when external

files such as images are required for displaying a certain instance - such files may or may not be available.

5.2 Extensions to RAP

As a side effect to Arago's implementation, we try to contribute to the continuous improvement of the tools we use. This intention has manifested already in a series of patches to RAP, the RDF application programming interface in and for PHP5. As of today, these improvements have mainly concerned the areas of RDF/N3 parsing and RDF list processing: We implemented RDF/N3 list parsing (unavailable beforehand) as an enhancement for RAP's N3 parser, and consecutively improved its robustness in order to help it deal better with uncontrolled "low-quality" RDF from the Web. This included fixing and committing bugs in namespace and whitespace handling that previously lead to corrupt RDF models.

We expect the ongoing development to yield a number of additional improvements to this maturing toolkit, for which the processing of all kinds of data gathered from the Web is promising to be a good test. Further insights and improvements are expected to arise from using RAP, which has been designed and built for PHP4, with PHP5. As RAP is thoroughly object-oriented, we expect the improved scripting engine of PHP5 to have a significant impact on the performance of RAP. The effects of the new object model on a system of RAP's complexity remain to be seen, at least minor complications and inconsistencies can be expected.

6 Future Work

First experiences with this implementation advise the use of a caching subsystem, as parsing RDF and building an in-memory object model of the RDF graphs to be processed are expensive operations. Due to the PHP's nature as a webserver module reinitializing scripts on every HTTP request, these operations would have to be carried out every time a user interacts with the system. We therefore plan to integrate our prototype with the Popoon framework[11]. Popoon is an Open Source XML publishing framework for PHP, based on the idea of processing pipelines. It implements a component caching system, which allows output to be combined from cachable and non-cachable sources and processes. Besides from the efficient caching mechanisms, integration into Popoon will also allow for post-processing and integration of Arago's output, e.g. to further refine of the produced output, to combine the output with non-RDF data or to embed it within other Web applications running in the Popoon framework. We want to emphasize the component nature of Arago, which is being integrated with a "faceted browsing" component written in Java. After the full implementation of the core Lens and Style vocabularies, an immediate goal is to include rudimentary presentation logic functionality, such as linear traversal of the instance graph.

[11] `http://popoon.org/`

7 Conclusions

We have presented Arago, a presentation engine which is able to implement the functionality needed by the Fresnel vocabulary to display data from the Web. Our survey of different existing approaches to UI building for RDF applications reveals a trend towards the coalescence of domain knowledge and presentation knowledge, both modeled and expressed in RDF. Blurring the distinction between domain knowledge and presentation knowledge helps in making the user interface and output generation profit from the benefits of RDF.

Although RDF support stills is scarce in the PHP world (compared e.g to Java or Python), with RAP there is development platform of considerable maturity. We pointed out that improving its robustness in dealing with uncontrollable data from the Web. Failover issues where also discussed when RAP was ported to PEAR (see `http://pear.php.net/package/RDF/`). Advancement in reliability therefore is key in further advocating RAP — and RDF — in the world of PHP. Nevertheless, the combination of PHP and the RAP toolkit already allowed for rapid prototyping of the system presented, whose speed can be mostly credited to the powerful scripting language and its ease of use.

We think that the Fresnel display vocabulary has the potential to serve as a base for presentation-oriented RDF-based Web applications, due to its good integration with established Web standards like CSS and XHTML. By implementing Fresnel's ideas in PHP5, we attempt to validate and test the approach and study its feasibility, working towards the final aim of being able to build flexible, RDF-based data integration systems with powerful and adaptable end user interfaces.

References

1. C. Bizer and R. Lee. *Fresnel - Display Vocabulary for RDF - User Manual*, 3 2005. `http://simile.mit.edu/repository/fresnel/trunk/docs/manual/`.
2. D. Brickley and L. Miller. FOAF Vocabulary Specification. `http://xmlns.com/foaf/0.1/`.
3. R. Lee. Fresnel web page, 3 2005. `http://simile.mit.edu/fresnel/`.
4. P. Linss. CSS Namespace Enhancements (Proposal). Technical report, W3C, 6 1999. `http://www.w3.org/1999/06/25/WD-css3-namespace-19990625/`.
5. C. Y. Lung. Rules Processing and HTML Translation for the Semantic Web. Technical report, HP Labs Bristol, 2003.
6. E. Pietriga. *Graph Stylesheets (GSS) in IsaViz.* `http://www.w3.org/2001/11/IsaViz/gss/gssmanual.html`.
7. D. Quan, D. Huynh, and D. Karger. Haystack: A Platform for Authoring End User Semantic Web Applications. In *ISWC 2003*, 2003.
8. D. Quan and D. Karger. Xenon: An RDF Stylesheet Ontology. In *WWW 2005, Chiba*, 5 2005.
9. D. Steer. *TreeHugger 0.1.* `http://rdfweb.org/people/damian/treehugger/`.
10. N. Walsh. *RDF Twig: Accessing RDF Graphs in XSLT*, 2003. `http://rdftwig.sourceforge.net/`.

Acknowledgements

The authors would like to thank the Fresnel core team for the openness and accessibility of their ongoing work and discussions. We thank for the support of Soraya Kouadri Mostéfaoui and Béat Hirsbrunner at the University of Fribourg.