

On Searching and Displaying RDF Data from the Web

Andreas Harth
Digital Enterprise Research Institute
Galway, Ireland
andreas.harth@deri.org

Hannes Gassert
University of Fribourg, CS Dept.
Fribourg, Switzerland
hannes.gassert@unifr.ch

Keywords

Semantic Web, Notation3, XSLT

1. MOTIVATION

Although the Semantic Web is meant for consumption by machines, humans are still in the loop. Therefore it is important to be able to view and browse RDF in a convenient user interface. At the current stage of the Semantic Web, large amounts of instance data are becoming available. For the application presented in this demo, instance data gathered from the Web is at the core of the functionality. The question we try to answer is whether it is possible to integrate, query and display Web data with minimal knowledge about the vocabularies used. We present a prototype system that works towards the goal of integrating and displaying arbitrary RDF collected from the Web, which presents challenges due to the data being scattered, uncontrolled, and heterogeneous.

2. EXPERIMENTAL SETUP

We have built a data integration system based on a Semantic Web crawler, storage and retrieval facilities, a reasoner to carry out integration tasks, and a user interface to generate HTML for displaying results of queries. The system is combined of a set of languages and tools (Bash, Java, PHP, Python) loosely connected together. We use Bash and Python scripts for scuttering, Java for storage and retrieval, Python (CWM [1]) for reasoning, and PHP for creating the user interface. Rules are used for object consolidation.

The data flow is organized in a pipeline: first, data is retrieved based on keyword search. Then, the results are integrated and instances are fused together before the rendering phase in the UI layer. The approach is a variation of the method described in [2].

We collected a dataset from the Web using an RDF crawler (dubbed “scutter”). After having obtained an initial seed set of RDF files using Google’s search API in RSS, FOAF, ICAL, DOAP, DC, and OWL formats, we perform a breadth-first crawl of `rdfs:seeAlso` properties, which are used to connect RDF files, similar to the HTML `a href` construct used by traditional Web crawlers. Please note that we ran

Table 1: Most frequently found namespaces

Namespace	Count
http://www.w3.org/1999/02/22-rdf-syntax-ns#	6973
http://purl.org/dc/elements/1.1/	4564
http://xmlns.com/foaf/0.1/	3835
http://www.w3.org/2000/01/rdf-schema#	3564
http://purl.org/rss/1.0/	3285
http://webns.net/mvcb/	2183
http://web.resource.org/cc/	2048
http://xmlns.com/wordnet/1.6/	1221
http://www.w3.org/2003/01/geo/wgs84_pos	1193

the scutter once, not capturing the regular updates made on RSS files. The resulting data set is 103 MB in size from 7791 different files, 348 of which were not valid RDF/XML. Table 1 shows the distribution of namespaces. We counted 2055 `rdfs:seeAlso` links which suggests that the files have a relatively low degree of connectedness.

3. STORAGE AND RETRIEVAL

Our application is relying on instance data; hence, we provide query and retrieval facilities tailored towards this data set. The queries we envision for our application are relatively simple, consisting of literal matching, keyword searches, and simple joins. We use YARS [3] to store the collected data, and build indices for fast triple pattern, literal and keyword retrieval. Contexts (quads instead of triples) are used for provenance tracking, which is important in our data integration scenario.

4. INTEGRATION

The main task performed by the integration layer is fusing of instances, also called object consolidation or “smushing”. “Smushing” could be performed based on an ontology, but for reasons of performance and convenience we just use a set of hand-crafted rules. For FOAF, the “smushing” is based on a couple of inverse functional properties (IFPs) such as email addresses. In our experiments, using the rules as opposed to the full ontology proved to be significantly more convenient for use in the given tool chain. Additionally, we use `foaf:name` as an IFP to match a large number of instance that don’t overlap on formal IFPs.

5. USER INTERFACE LAYER

For parts of a software application requiring straightforward adaptability, scripting languages have a proven industry track record. User interface layers in multi-tier applications fall into this category, using a server-side scripting engine as a gluing and processing layer between clients and the inner layers of an application is a widely adopted practice. The interface prototype, following these ideas, uses two scripting tools in a processing pipeline: PHP for request processing, communication with the YARS server, and response postprocessing, and Python/CWM for RDF conversion, integration and formatting.

5.1 REST-style communication of RDF/N3

In our scenario, a PHP script acts as a Page Controller, dispatching to different handlers dependent on the client's request. After the construction of an N3 query and the communication with YARS, the resulting graph goes through a series of processing steps, several of them being performed by CWM. Finally, an XSLT stylesheet produces the XHTML output. This approach to page generation has proven to be sufficiently effective for the demonstrated application. For more complex tasks a number of optimizations are being considered. One impediment is the inability of the processing pipeline to effectively "stream" content, as some processing steps require complete in-memory representations of intermediate results.

The semantics of HTTP verbs combined with the power of RDF/N3 are more than enough to create a general remote access CRUD¹-capable query interface to the YARS storage engine. N3 in this scenario is used as language for data access, manipulation and definition - and the data itself. With this combination of a lightweight protocol and a simple syntax, YARS offers an interface both powerful and easily accessible.

5.2 XSLT for RDF Transformation

Applying an ordinary XSLT stylesheet to an RDF graph is no doubt a suboptimal solution. Using tree-oriented tools on graphs is a clear mismatch of data and tools. This mismatch has resulted in many attempts to reconcile the two, such as Treehugger[4] or RDF Twig[6].

Even so, we found that very strict and predictable XML serialization of a preprocessed graph already allows for an amount of functionality that can be considered generally sufficient enough for mastering a large amount of data display tasks in actual user interfaces. In such cases, the transition from an RDF model to a "flat" presentation involves but a fairly limited set of selection and formatting operations, usually resulting in an output document of significantly reduced complexity compared to the input graph. Consuming such RDF "controlled" by intermediates like YARS and CWM becomes therefore a task easily manageable with the standard capabilities of an XSLT processor.

The demonstrated application uses one small XSLT stylesheet, which implements templates to handle the following input classes: Application states (empty result, empty query, etc.) and sequences of RDF instances (FOAF persons, DBLP articles, RSS channels and items).

Each of these main templates can, once it matched, apply any template out of the pool of the smaller templates for

individual properties such as foaf:name, dc:creator and so forth.

While a large amount of RDF's expressiveness is irretrievably lost in the transformation of the initial graph to a simple XML document conforming to XHTML 1.0 Strict, the transformation should still not be unnecessarily lossy and convey some of the original semantics where feasible. One way we do this is by reusing URIs as CSS class names. This technique, which involves the syntactic translation of URIs to valid CSS names, allows to very specifically influence the styling of RDF nodes turned XHTML elements.

Additionally, generated documents link to the underlying RDF data (N3 as well as XML) for RDF-aware user agents to consume.

6. DISCUSSION

One crucial aspect in our design is layering: we don't use a single application but a set of components that are combined using some glue code in PHP. By reusing existing applications and technologies we are able to rapidly prototype such a system.

Performance is a crucial criterion for any large-scale system dealing with content from the Web. We conducted initial performance test and profiled the time each component uses and the amount of data involved. Not surprisingly, the step that involves reasoning is the most expensive. By only feeding a small part of the stored graph into the reasoning engine, we can achieve acceptable performance since we don't need to apply the rules to the whole graph, but only to the pieces needed for displaying the results. Therefore, we gain flexibility by performing reasoning at query time; however, we might lose information that could be inferred by reasoning on the entire knowledge base [5].

What we can do is a search-engine style interface, with the extension of "smushing" and precise query formulation. Also, by inferring new links between instances we can make existing data more usable (e.g relating persons to documents they have written). This is future work.

Ongoing experiments using a display ontology are exploring output generation mechanisms more adapted to RDF than XSLT.

7. CONCLUSION

The demo shows that it is possible to search over considerable amount of RDF data in a reasonable time as well as integrate and display the results in a HTML user interface by combining existing technologies and standards.

8. REFERENCES

- [1] S. T. Berners-Lee. *CWM*. W3C. <http://www.w3.org/2000/10/swap/doc/cwm.html>.
- [2] A. Harth. SECO: Mediation Services for Semantic Web Data. *IEEE Intelligent Systems*, 19(3):66–71, May/June 2004.
- [3] A. Harth. *YARS*. DERI, 01 2005. <http://sw.deri.org/2004/06/yars/yars.html>.
- [4] D. Steer. *TreeHugger 0.1*, 2003. <http://rdfweb.org/people/damian/treehugger/>.
- [5] H. Stuckenschmidt. Query Processing on the Semantic Web. *KI*, pages 22–26, 3 2003.
- [6] N. Walsh. *RDF Twig: Accessing RDF Graphs in XSLT*, 2003. <http://rdftwig.sourceforge.net/>.

¹"Create, Read, Update, Delete"