> Semantic Web Use Cases and Case Studies

Case Study: Improving Web Search using Metadata

Peter Mika, Yahoo! Research, Spain

November 2008

Presenting compelling search results depends critically on understanding what is there to be presented on the first place. Given that the current generation of search engines have a very limited understanding of the query entered by the user, the content returned as a result and the relationship of the two, the opportunities for customizing search results have been limited.

Opening up search

The majority of Web pages today are generated from databases, and Web site owners increasingly are providing APIs to this data or embedding information inside their HTML pages with microformats, eRDF, or RDFa. In other cases, structured data can be extracted with relative ease from Web pages that follow a template using XSLT stylesheets.

SearchMonkey reuses structured data to improve search result display with benefits to both search users, developers, and publishers of web content. The first type of applications are focusing on remaking the abstracts on the search result page: Figure 1 shows the kind of presentations that structured data enables in this space. Based on data, the image representing the object can be easily singled out. One can also easily select the most important attributes of the object to be shown in a table format. Similarly for links: the data tells which links represent important actions the user can take (e.g. play the video, buy the product) and these links can be arranged in a way that their function is clear. In essence, knowledge of the data and its semantics enables to present the page in a much more informative, attractive, and concise way.

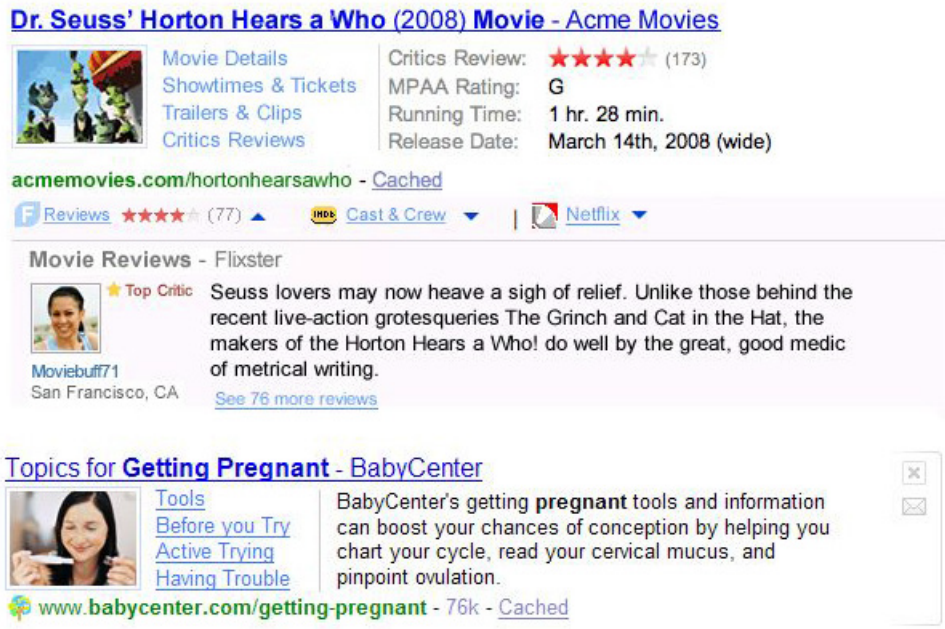

Figure 1: search results using SearchMonkey

The benefits for *publishers* are immediately clear: when presenting their page this way publishers can expect more clicks and higher quality traffic flowing to their site. In fact, several large publishers have moved to implement semantic metadata markup (microformats, eRDF, RDFa) specifically for providing data to SearchMonkey.

On the other hand, *users* also stand to benefit from a better experience. The only concerns on the users' part is the possibility of opting out and having a system free of spam. Both concerns are addressed by the Yahoo Application Gallery. As shown in Figure 2, the Gallery allows users to selectively opt in to particular SearchMonkey applications. Users also have a small number of high-quality applications that are turned on by default. The gallery is also an effective spam detection mechanism: applications that are popular are unlikely to contain spam. Also important to note that the presence of metadata does not affect the ranking of pages. Pages that are trusted by the search engine based on other metrics can be also expected to contain trustable metadata.

For *developers*, the excitement of the system is in the possibility to take part in the transformation of search and develop applications that are possibly displayed to millions of users every day. Needless to say, many publishers become developers themselves, creating applications right after they have added metadata to their own pages.

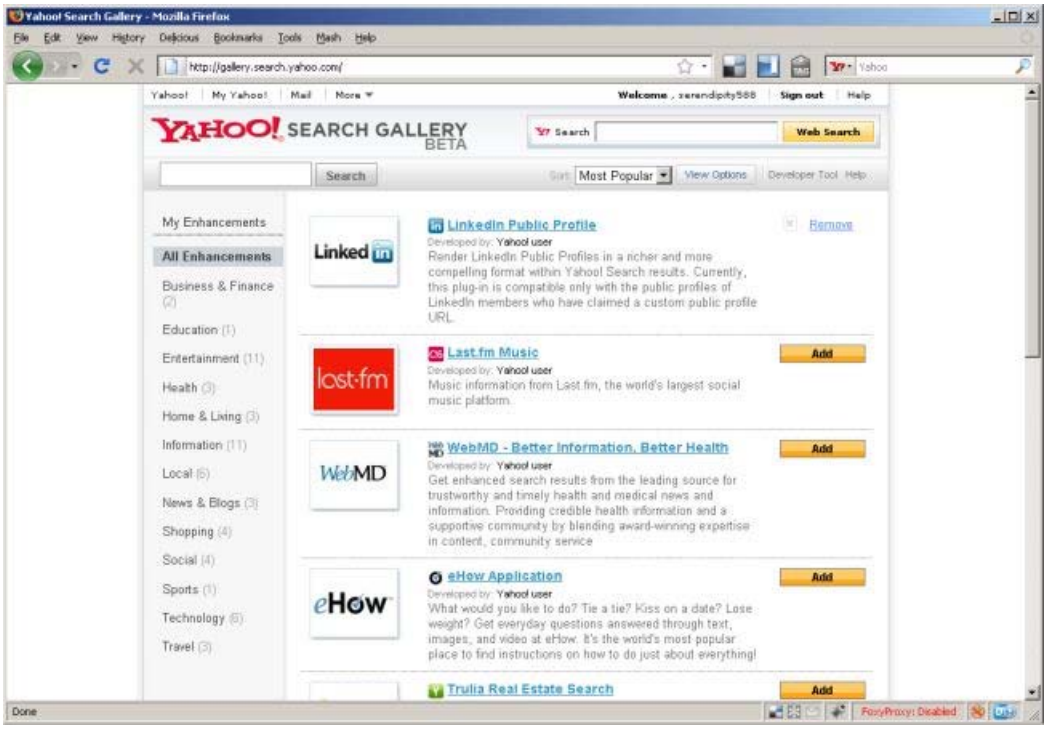

Figure 2: Yahoo Application Gallery (a larger version of the image is also available)

Architecture

The high level architecture of the system (shown in Figure 3) can be almost entirely reconstructed from the above description. The user's applications trigger on URLs in the search result page, transforming the search results. The inputs of the system are as follows:

- Metadata embedded inside HTML pages (microformats, eRDF, RDFa) and collected by Yahoo Slurp, the Yahoo crawler during the regular crawling process.
- Custom data services extract metadata from HTML pages using XSLT or they wrap APIs implemented as Web Services.
- Metadata can be submitted by publishers. Feeds are polled at regular intervals.

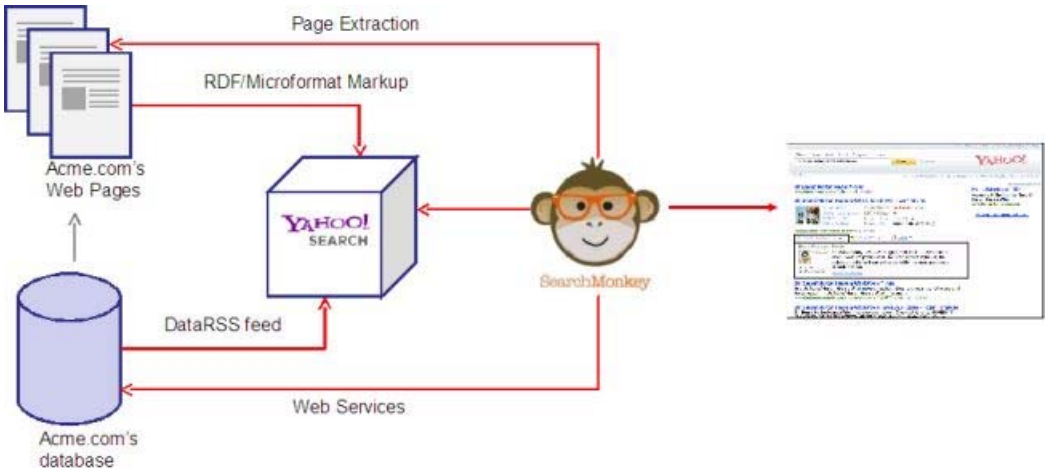

Figure 3: High level architecture of the system

Developers create custom data services and presentation applications using an online tool (see Figure 4). This tool is a central piece of the SearchMonkey experience: it gives developers access to all their services and applications. When defining new custom data services, first some basic information is provided such as name and description of the service and whether it will execute an XSLT or call a Web Service. In the next step, the developer defines the trigger

pattern and some example URLs to test the service with. Next, the developer constructs the stylesheet to extract data or specifies the Web Service endpoint to call. (Web Services receive the URL of the page as an input.) When developing XSLTs, the results of the extraction are shown immediately in a preview to help developers debug their stylesheets. After that the developer can share the service for others to build applications on and can also start building his own presentation application right away. Note that custom data services are not required if the application only uses one of the other two data sources (embedded metadata or feeds).

Creating a presentation application follows a similar wizard-like dialogue. The developer provides the application's name and some other basic information, then selects the trigger pattern and test URLs. Next, SearchMonkey shows the schema of the data available for those URLs taking into account all sources (embedded metadata, XSLT, Web Services, feeds). The developer can select the required elements which again helps to narrow down when the application should be triggered: if a required piece of the data is not available for a particular page, the application will not be executed. Then as the main step of the process, the developer builds a PHP function that returns the values to be plugged into the presentation template. As with stylesheets, the results of the application are shown in a preview window that is updated whenever the developer saves the application. In practice, most PHP applications simply select some data to display or contain only simple manipulations of the data. The last step is again the sharing of the application.

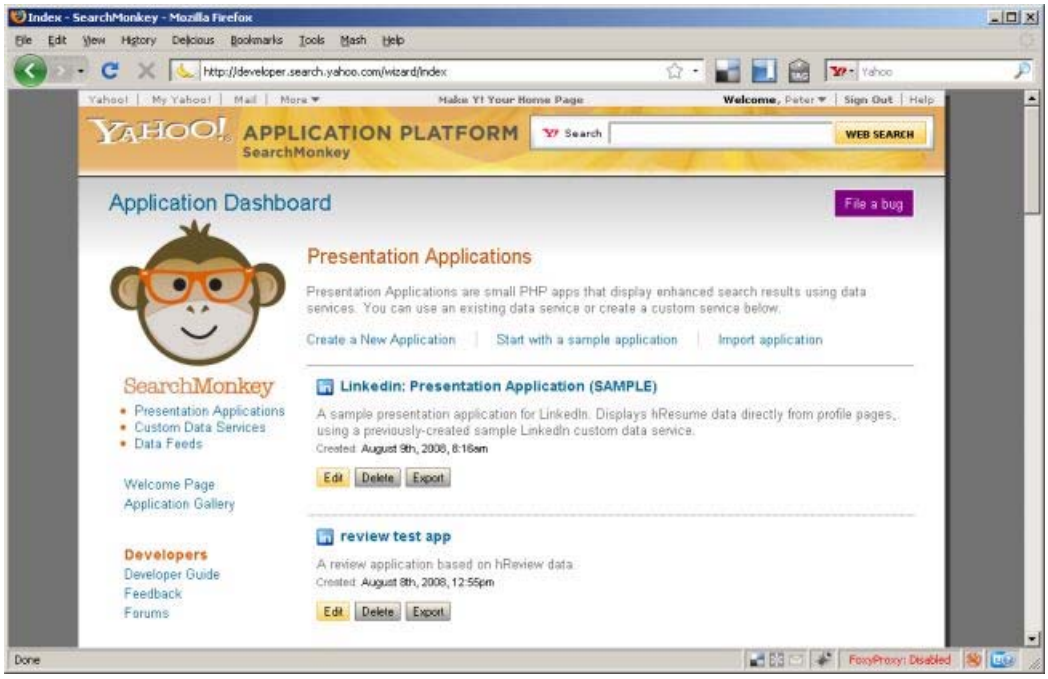

Figure 4: Online application development tool (a larger version of the image is also available)

As the description shows, the representation of data is a crucial aspect of SearchMonkey, since the output is simply a result of executing a set of transformations on Web data. Some of these transformations are performed using XSLT and some are complex enough to require a full-blown programming language such as PHP. What connects these pieces is a canonical representation of structured data.

To understand better the choices made in dealing with structured data it is useful to summarize the main requirements that any solution must fulfill. These are as follows:

- **An application platform based on structured data.** As described above, the starting point of SearchMonkey was the observation that the (vast) majority of Web pages on the Web are generated from some sort of a database, in other words driven by structured data. Publishers are increasingly realizing the benefits of opening up both data and services in order to allow others to create mash-ups, widgets or other small, non-commercial applications using their content. (And in turn, developers are demanding more and more the possibility to have access to data.) The preferred method of opening up data is either to provide a custom API or embed semantic markup in HTML pages using microformats. Thus SearchMonkey requires a data representation (syntax) that could generally capture structured data and a schema language that provides a minimally required set of primitives such as classes, attributes and a system of data types. The syntax and semantics should allow to capture the full content in typical microformat data and the languages used should be open to extensions as much as possible.
- **Web-wide application interoperability.** The queries received by a search engine and the content returned as a result cover practically all domains of human interest. While it would have been easier to develop, a solution that would limit the domains of application would not meet the need of a global search product as it would not able to capture the long tail of query and content production. The question of interoperability is complicated by the fact that the application may be developed by someone other than the publisher of the data. On the one hand, this means that data needs to be prepared for serendipitous reuse, i.e. a developer should be able to understand the meaning of the data (semantics) by consulting its description (minimally, a human readable documentation of the schema). On the other hand, the framework should support building applications that can deal with data they can only partially understand (for example, because it mixes data from different schemas). Changes in the underlying representation of the data also need to be tolerated as much as possible.
- **Ease of use.** It has been often noted that the hallmark of a successful Semantic Web application is that no user can tell that it was built using semantic technologies. That novel technology should take a backstage role was also a major requirement for SearchMonkey: the development environment is targeted at the large numbers of Web developers who are familiar with PHP and XML technologies (at least to the extent that they can understand an

example application and start extending or modifying it to fit their needs). However, developers could not have been expected to know about RDF or RDFa, technologies that still ended up playing a role in the design of the system.

Key benefits of semantic technology

Semantic technologies promise a more flexible representation than XML-based technologies. Data doesn't need to conform to a tree structure, but can follow an arbitrary graph shape. As the unit of information is triple, and not an entire document, applications can safely ignore parts of the data at a very fine-grained, triple by triple level. Merging RDF data is equally easy: data is simply merged by taking the union of the set of triples. As RDF schemas are described in RDF, this also applies to merging schema information. (Obviously true merging requires mapping equivalent classes and instances, but that is not a concern in the current system.) Semantics (vocabularies) are also completely decoupled from syntax. On the one hand, this means that RDF doesn't prescribe a particular syntax and in fact triples can be serialized in multiple formats, including the XML based RDF/XML format. (An XML-based format was required as only XML can serve as input of XML transformations.) On the other hand, it also means that the resources described may be instances of multiple classes from possibly different vocabularies simply by virtue of using the properties in combination to describe the item. The definition of the schema can be simply retrieved by entering URIs into a web browser. RDF-based representations are also a good match for some of the input data of the system: eRDF and RDFa map directly to RDF triples. Microformats can also be re-represented in RDF by using some of the pre-existing RDF/OWL vocabularies for popular microformats.

Given the requirements, the benefits and drawbacks of these options and the trends of developments in the Web space, the choice was made to adopt RDF-based technologies. However, it was also immediately clear that RDF/XML is not an appealing form of representing RDF data. In particular, it does not allow to capture metadata about sets of triples such as the time and provenance of data, both of which play an important role in SearchMonkey. Other RDF serialization formats have been excluded on the same basis or because they are not XML-based and only XML-based formats can be input to XSL transformations.

These considerations led to the development of DataRSS, an extension of Atom for carrying structure data as part of feeds. A standard based on Atom immediately opens up the option of submitting metadata as a feed. Atom is an XML-based format which can be both input and output of XML transformation. The extension provides the data itself as well as metadata such as which application generated the data and when was it last updated. The metadata is described using only three elements: item, meta, and type. Items represent resources, metas represent literal-valued properties of resources and types provide the type(s) of an item. These elements use a subset of the attributes of RDFa that is sufficient to describe arbitrary RDF graphs (resource, rel, property, typeof) . Valid DataRSS documents are thus only valid as Atom and XML, but also conform to the RDFa syntax of RDF, which means that the triples can be extracted from the payload of the feed using any RDFa parser. Figure 5 provides an example of a DataRSS feed describing personal information using FOAF (the Friend-of-a-Friend vocabulary).

```
<?profile http://search.yahoo.com/searchmonkey-profile ?>
<feed xmlns="http://www.w3.org/2005/Atom"
xmlns:foaf="http://xmlns.com/foaf/0.1/"
xmlns:y="http://search.yahoo.com/datarss/">

<id>http://www.linkedin.com/datarss/</id>
<author><name>Peter Mika (pmika@yahoo-inc.com)</name></author>
<title>Example data feed for social</title>
<updated>2007-11-14T04:05:06+07:00</updated>
<entry>
<title>Peter Mika</title>
<id>http://www.linkedin.com/ppl/webprofile?id=5054019</id>
<updated>2007-11-14T04:05:06+07:00</updated>
<content type="application/xml">
  <y:adjunct version="1.0" name="com.yahoo.social">
    <y:item rel="dc:subject">
      <y:type typeof="foaf:Person vcard:VCard">
        <y:meta property="vcard:fn foaf:name">Peter Mika</y:meta>
        <y:meta property="foaf:age">29</y:meta>
        <y:meta property="vcard:bday">1978-10-05</y:meta>
        <y:item rel="vcard:org">
          <y:type typeof="vcard:Organization">
            <y:meta property="vcard:organization-name">Yahoo!</y:meta>
            <y:meta property="vcard:organization-unit">Yahoo! Research Barcelona</y:meta>
          </y:type>
        </y:item>
      </y:type>
    </y:item>
  </y:adjunct>
</feed>
```

Queries:

foaf:Person/foaf:name = Peter Mika
foaf:Person/vcard:org/vcard:organization-unit = Yahoo! Research Barcelona

Figure 5: DataRSS feed example

A new format also brings along the question of query language. Since DataRSS is both XML and RDF, the two immediately available options were XPath and SPARQL. However, it was also immediately clear that both languages are too complex for the task at hand. Namely, presentation applications merely need to filter data by selecting a simple path through the RDF graph. Again, the choice was made to define and implement a simple new query language. (The choice is

not exclusive: applications can execute XPath expressions, and the option of introducing SPARQL is also open.) As the use case is similar, this expression language is similar to a Fresnel path expressions except that expressions always begin with a type or property, and the remaining elements can only be properties. Figure 5 also gives some examples of this query language.

Conclusions

The current applications populating this platform are relatively modest transformations of structured data into a presentation of the summary of web pages. However, the platform is open to extensions that use structured data to enrich other parts of the search interface or to be deployed in different settings such as a mobile environment. Lastly, in building on semantic technologies SearchMonkey has not only accomplished its immediate goals but is well prepared for a future with an increasingly more semantic web where the semantics of data will drive not only presentation but the very process of matching user intent with the Web's vast sources of structured knowledge.

Links:

- SearchMonkey
- Peter Mika. Anatomy of a SearchMonkey. Nodalities Magazine, September/October 2008.
- Peter Mika. Talking with Talis (podcast).