RDF Social Tagging Data Queries
Project Report

Daniel Whitehead
China Williams
December 11, 2009
S636 Semantic Web
I. Introduction

As a group we knew we wanted to learn more about the social web. We were interested in analyzing how individuals behave, interact, and socialize on the internet. We considered looking at social identifiers, the social and semantic web, or social tagging. The topic that we decided on was social tagging.

The analysis of social tagging is an interesting subject area for many different reasons. One notable reason is that tagging provides meaningful information about the tagged resources. Tapping into the collective understanding of many individuals helps to alleviate the heavy costs of metadata creation. In addition to information about the objects themselves, social tagging provides for the analysis of the habits of individual taggers. When the tags of a given resource are analyzed, they can reveal a great deal of information about the underlying content and the collective perception of that content. When the tags of a given tagger are examined, the resulting data reveals illuminating information about the tagger’s interests, personality, and values.
II. Ontology Design

Our work in understanding social tagging data came from two sources: first, the hypothetical structure of the data which we created with references to UTO (Upper Tag Ontology, a framework for describing tagging events) and implemented using Protégé; second, the representation and SPARQL queries for use with the RDF data supplied by Dr. Ding.

a. Hypothetical Social Tagging Representation

For the construction of the first social tagging ontology we followed the Methontology engineering method. Following the first task of the Methontology method, we checked to see if another ontology had already been created by a third party that would be available. Since we found no available ontology (other than UTO itself), we began designing an ontology to represent social tagging data.

First, we brainstormed the classes of data that would be necessary to keep track of individual tagging events. We then determined what entities and attributes would be needed. From these relationships, we constructed a hypothetical data structure for social tagging. After we had developed our ontology, we found a more complete model of UTO that would have proved very useful, had we not already built our ontology. This did allow for the differences in modeling to come to light, however. For a schematic of the UTO data structure, see Figure A.

Once these relationships were sketched out, we began to implement the ontology in the OWL format, using the Protégé software. We made each class a child of the Thing class, and then linked them to the object properties. We decided to implement inverse object relations for most of the entities: is_Source_of, is_Object_of, is_Tagger_of, etc. In addition to these inverse relations, we recognized the
need for the **has_related_tag** property to be symmetric, providing a sibling relationship among tags. See Figures C, D, and E for screenshots of our Protégé implementation.

After we had completed the ontology, we discovered a more complete table of UTO (“Mediating and Analyzing Social Data,” p. 5) that we had failed to take into account. While the majority of our recreation was accurate, there were several areas in which our ontology differs from UTO. For instance, in UTO, the date and vote are constructed as classes unto themselves, whereas in our model, we chose to represent the date and vote as data properties of the tagging event. Additionally, we had not implemented some of the comments for the classes, which, on reflection, are highly important for the creation of an ontology, as it will help other potential users understand what the classes represent. Our ontology was also very basic in that it lacked the cardinality rules of UTO.

### b. RDF Data Structure

For querying our RDF data set (explained below), we decided to make sure that our hypothetical data structure would be interoperable with the data. To do this, we constructed another data model for the RDF data. We felt that it would be most efficient to view the data itself to determine how the ontology should be structured, effectively reverse-engineering the RDF file and re-creating the UTO used to produce the data. See Figure B for the structure of the RDF data set.

The shallow hierarchy of the social tagging data meant that the classes could be extracted relatively easily by scanning the data file, writing down all possible relations for each description. Each RDF description had some mixture of the following predicates: **has_source**, **has_tag**, **has_object**, **has_date**, **has_tagger**, **has_vote**, and **has_comment**; the first three being URIs linking to other objects and the last four linking to literal values. For the most part, we kept the existing object property labels. From these relations, we inferred that the description URI itself described not an object with a list of tags, but rather a single tagging event. We chose to represent all but two as separate classes, leaving
the list as the following: **Tagging** (the event), **Source**, **Object**, **Tag**, **Tagger**, and **Comment** with **Date** and **Vote** stored as data attributes.

Initially, each of these classes was tied directly to the **Tagging** event with the appropriate object properties. However, after studying the structure of the description elements in the RDF document more closely, and examining some of the literature provided for the assignment, we determined that the **Source** and **Object** entities were related, and that there was no one-step relationship between **Tagging** and **Source**.

Thus, as far as we can tell, our first and second representations of UTO are interoperable. The differences in the data structure can be explained by the data set that we were provided. For example, our RDF file contains no descriptions about tags, and so there was no way to tell if the **has_related_tag** predicate was used when the RDF was generated. Likewise, the inverse relations (**is_tagger_of**, etc.) are left from the second data model, as those predicates were never used, but there is nothing to prevent them from existing elsewhere.
III. RDF Dataset

The dataset we used was a file of RDF descriptions representing approximately sixty del.icio.us tagging events. While a larger dataset would have provided a fuller picture of how users are tagging resources online, our dataset allowed us to attain proof of concept for our Sparql queries.

Each RDF description of a tagging event had some or all of the following elements: has_object, has_date, has_tagger, has_vote, has_comment, and has_tag. Some tagging events had one tag associated with it, while other events had as many as ten. About forty five of the events had a numeric vote associated with them while there were about a dozen events with non-empty comment strings.

Several of the RDF descriptions were for actual website URLs (as objects of has_object are unique identifiers for tagging event), rather than tagging events themselves. These descriptions had only the has_source predicate, which indicated what social tagging site the data had been taken from. As our dataset came exclusively from data crawled a single source, every source was marked as http://del.icio.us.
IV. Query Creation

In the query design process we had a brainstorming session where we thought about the most practical and interesting questions to ask about the dataset. During this process we considered all questions regardless of whether we thought we could write the corresponding queries in Sparql.

We wrote the following queries in Sparql and ran them in the Eclipse Java development environment. Some of the queries had to be simplified, but for the most part we succeeded in creating them. Here is a list of the queries that we determined would be most common and generally useful for social tagging data:

1. **Given a tag**, return all resources associated with that tag ordered by vote.
2. **Given a resource**, return all tags associated with it ordered by tagger.
3. **Given any tagger**, return all resources that they have tagged, with the tags and date.
4. **Given any vote** as an integer, return all resources that have greater than that vote.
5. **Given a resource**, return the resource and all comments ordered by date.
6. Return all resources that have comments to see what people are talking about.
To provide an example of the corresponding Sparql query, here is the code for query number 6 above:

```sparql
PREFIX uto: <http://uto.deri.at/>
PREFIX del: <http://del.icio.us/tag/>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

SELECT ?resource ?has_comment ?date WHERE {
  ?x uto:has_object ?resource .
  ?x uto:has_comment ?has_comment .
  ?x uto:has_date ?date .
  FILTER (?has_comment != "")
}
```

A screen shot of the results for this query run against our dataset is:

<table>
<thead>
<tr>
<th>resource</th>
<th>has_comment</th>
<th>date</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://petrz.pl/%3E">http://petrz.pl/&gt;</a></td>
<td>&quot;Test - zobacz&quot;</td>
<td>&quot;Nov 05&quot;</td>
</tr>
<tr>
<td><a href="http://www.pixelio.de/search.php">http://www.pixelio.de/search.php</a></td>
<td>&quot;pixelio.de - Deine kostenlose Bilddatenbank lizenzfreie Fotos&quot;</td>
<td>&quot;May 07&quot;</td>
</tr>
<tr>
<td><a href="http://refit.sourceforge.net/doc/c2e2_startos.html">http://refit.sourceforge.net/doc/c2e2_startos.html</a></td>
<td>&quot;refit - Documentation - Starting an Operating System&quot;</td>
<td>&quot;Mar 07&quot;</td>
</tr>
<tr>
<td><a href="http://petrz.pl/">http://petrz.pl/</a></td>
<td>&quot;Petrz.pl - zobacz&quot;</td>
<td>&quot;Nov 06&quot;</td>
</tr>
<tr>
<td><a href="http://www.businessweeek.com/bios/Don_Tapscott.htm">http://www.businessweeek.com/bios/Don_Tapscott.htm</a></td>
<td>&quot;Don Tapscott&quot;</td>
<td>&quot;Jun 07&quot;</td>
</tr>
<tr>
<td><a href="http://www.dekanat-krainchau.de/55_317_.html">http://www.dekanat-krainchau.de/55_317_.html</a></td>
<td>&quot;Katholische Kirche - Dekanat Kраichgau&quot;</td>
<td>&quot;Jun 07&quot;</td>
</tr>
<tr>
<td><a href="http://sname.shirinpomori.net/casino-games/">http://sname.shirinpomori.net/casino-games/</a></td>
<td>&quot;casino games&quot;</td>
<td>&quot;Jun 07&quot;</td>
</tr>
<tr>
<td><a href="http://www.slate.com/id/2144121/">http://www.slate.com/id/2144121/</a></td>
<td>&quot;Well, it could just mean you don’t like rap music.&quot;</td>
<td>&quot;May 06&quot;</td>
</tr>
<tr>
<td>&lt;<a href="https://www.pkiotelligence/p">https://www.pkiotelligence/p</a>&lt;kixd&gt;</td>
<td>&quot;Servis internetowy PKI Intelig&quot;</td>
<td>&quot;Feb 07&quot;</td>
</tr>
<tr>
<td><a href="http://choutimes.com/2006/11/korean_tofu_pancake.html#more">http://choutimes.com/2006/11/korean_tofu_pancake.html#more</a></td>
<td>&quot;Chow Times: Korean Tofu Pancake&quot;</td>
<td>&quot;Jun 07&quot;</td>
</tr>
<tr>
<td><a href="http://themathworksheetsite.com/real_tape.html">http://themathworksheetsite.com/real_tape.html</a></td>
<td>&quot;Generates tape measures to read and practice&quot;</td>
<td>&quot;Mar 06&quot;</td>
</tr>
<tr>
<td><a href="http://www.ghouse.com/google-rechercher-un-visage_274/">http://www.ghouse.com/google-rechercher-un-visage_274/</a></td>
<td>&quot;Ghouse Google : Rechercher un Visage&quot;</td>
<td>&quot;Sep 09&quot;</td>
</tr>
<tr>
<td><a href="http://www.lifelife.com/lifelife/default.aspx">http://www.lifelife.com/lifelife/default.aspx</a></td>
<td>&quot;Senior 80 Plus People Articles - LifelifeIn.com&quot;</td>
<td>&quot;Jun 07&quot;</td>
</tr>
<tr>
<td><a href="http://enseek.com/~jschomb/articles/linux_logback.html">http://enseek.com/~jschomb/articles/linux_logback.html</a></td>
<td>&quot;Discussion of mounting partitions within disk images under Linux.&quot;</td>
<td>&quot;May 07&quot;</td>
</tr>
<tr>
<td><a href="http://petrz.pl/">http://petrz.pl/</a></td>
<td>&quot;zobacz&quot;</td>
<td>&quot;Nov 09&quot;</td>
</tr>
</tbody>
</table>

The remaining Sparql queries can be found in the appendix.

Most queries were fairly straightforward to construct, once we had figured out two things: the data structure of our RDF dataset and the Sparql syntax. One limitation of the queries that we wrote was that they were entirely static; the user would have to manually go into the query file and insert a different string to search. For instance, in our tagger query, we queried the database for a user named
“dielsy,” and the query worked fine. However, we did not build an automated system to allow the user to insert a different name into the query.

Another obstacle we had to overcome was that sometimes we had to manipulate the data set in order to test the queries. If there was only one tagging event by “dielsy”, we could not check to see if our ordering was working correctly, or if the same entries might occur multiple times, and so forth.

There were several queries we would have liked to write and run but we were limited by our knowledge about how to compose such queries. For example, we wanted to write a query that would take in two or more taggers and return information about their tagging similarities and differences. We also thought it would be insightful if we could normalize the voting data for every tagger in order to compare votes on the same resource made by unique taggers. As well we wanted to write a query to determine how often a tag is tagged over time to show its rise or fall in popularity, or how often a particular resource is tagged, to show its popularity. Finally, we wanted to write a query that would return a list of all tags and a numeric value representing how many times that tag was found.
V. Future Work & Conclusion

There are many areas of further work and research to pursue from here. It is noteworthy to point out that as behavior in social tagging changes new queries will need to be composed. In other words, an ever present limitation and challenge to this area of research is in not knowing what the next social tagging trend will be and how to best integrate new technologies into the UTO framework.

A useful tool and enhancement to this project is an online interface which would allow individuals to interact with the dataset and run queries such as the ones listed above. Additionally, if the data was stored in an online database that did periodic crawls of social tagging tools, then the interface would always present fresh data. This type of configuration would be dynamic and allow users of the interface to quickly find out what is being tagged online, what taggers think about resources, and to determine popular topics all in real time. Also, additional data sources, such as Flickr and YouTube, could be included in the dataset which would result in even more meaningful information being generated. As long as the data crawled from other social networking sites can be digested into the UTO RDF format that our data sample was in, the current queries should function. However, if the new data sources have different kinds of data (for instance, linking users together), then new queries would need to be created to compensate.

Finally, the area of social tagging and information visualization as an analysis tool is fascinating. With the data provided by crawling social tagging sites, tag clouds could be created that would synthesis the vast amount of information provided by social tags. Another example of visualization is a graph that would convey a taggers tagging behavior over time to show the evolution of their interests. Diagrams could show how a group of taggers were related based on the similarities and differences of their tags.
Since the internet is growing and changing every day, users are also changing how they interact with it. Clearly there are many questions answered regarding social tagging as well as many more questions yet to be asked. While working on this project we learned that even with a small dataset and a handful of queries, there are interesting questions that can be answered. The two representations of social tagging data helped us to better understand how social tagging data can be created, analyzed, and manipulated.
Figure A: Social Tagging Hierarchy (hypothetical), derived from UTO, and implemented using Protégé.
Figure B: Social Tagging Hierarchy (observed), derived from del.icio.us RDF data set. Note that inverse properties were not present in our data set.
Figure A: Class (Concept) Hierarchy in Protégé.
Figure B: Object Properties (Relations) in Protégé.
Figure C: Data Properties in Protégé.
Sparql Queries:

1. Given a tag, return all resources associated with that tag ordered by vote.

   
   ```sparql
   PREFIX uto: <http://uto.deri.at/>
   PREFIX del: <http://del.icio.us/tag/>
   PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
   SELECT DISTINCT ?resource ?vote
   WHERE
   {
   ?x uto:has_tag del:cute .
   ?x uto:has_object ?resource .
   ?x uto:has_vote ?vote
   }
   order by desc(xsd:integer(?vote))
   ```

2. Given a resource, return all tags associated with it ordered by tagger.

   ```sparql
   PREFIX uto: <http://uto.deri.at/>
   PREFIX del: <http://del.icio.us/tag/>
   PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
   SELECT DISTINCT ?tagger ?tag
   WHERE
   {
   ?x uto:has_tagger ?tagger .
   ?x uto:has_tag ?tag
   }
   order by ?tagger
   ```
3. Given any tagger, return all resources that they have tagged, with the tags and date.

```sql
SELECT DISTINCT ?has_tagger ?resource ?tag ?date
{
?x <http://uto.deri.at/has_tagger> "dielsy" .

?x <http://uto.deri.at/has_tagger> ?has_tagger .
?x <http://uto.deri.at/has_date> ?date
}
order by?resource
```

4. Given any vote as an integer, return all resources that are greater than that vote.

```sql
PREFIX uto: <http://uto.deri.at/>
PREFIX del: <http://del.icio.us/tag/>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

SELECT DISTINCT ?resource ?vote
WHERE
{
?x uto:has_object ?resource .
?x uto:has_vote ?has_vote .
?x uto:has_vote ?vote .
FILTER (?vote > "15")
}
```

5. Given a resource, return the resource and all comments ordered by date.

```sql
PREFIX uto: <http://uto.deri.at/>
PREFIX del: <http://del.icio.us/tag/>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

SELECT DISTINCT ?resource ?has_comment ?date WHERE
{
?x uto:has_object ?resource .
?x uto:has_object <http://patrz.pl/> .

?x uto:has_comment ?has_comment .
?x uto:has_date ?date
}
order by?date
```
6. **Return** all *resources* that have *comments* to see what people are talking about.

```sparql
PREFIX uto: http://uto.deri.at/
PREFIX del: <http://del.icio.us/tag/>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

SELECT ?resource ?has_comment ?date WHERE
{
  ?x uto:has_object ?resource .
  ?x uto:has_comment ?has_comment .
  ?x uto:has_date ?date .
  FILTER (?has_comment != "")
}
```
References

Ding, Ying; Toma, Ioan; Kang, Sin-Jae; Zhang, Zhixiong; Fried, Michael (2008). Mediating and Analyzing Social Data. *OTM Conferences.*

Ding, Ying; Toma, Ioan; Kang, Sin-Jae; Fried, Michael; Yan, Zhixian (2008). Data Mediation and Interoperation in Social Web: Modeling, Crawling and Integrating Social Tagging Data. *Workshop on Social Web Search and Mining (WWW2008).*