Wikipedia and Google Map Mashup

Summary

This project builds a Web-application prototype for showing surrounding interest points of universities and colleges in Texas using Google Map, Wikipedia and possibly other resources. Users select a university in Texas from the application to view surrounding points of interest as stored in Wikipedia. The results are displayed through Google Map. A pop-up box shows information about the point of interest when the mouse is on top of it. The information includes a link to the Wikipedia article of the interest point.

Goals

This project exposes students to Web 2.0 Mashup development. It familiarizes students with many modern Web development concepts, techniques and issues, including Mashup, APIs, AJAX, XML, Resource Description Framework (RDF), ontology, semantic Web, data cleaning and preparation, human computer interface (HCI), etc. It also utilizes contents and APIs from two of the top ten Websites: Wikipedia and Google.

Technical Information

The team will have some flexibility to drive the development direction of the prototype. Parts of the job of the team will be the specification of a set of requirements with enough details. Bare-bone minimum requirements are elaborated below and they must be satisfied. However, the mentor expects the team to go beyond the minimum requirements.

1. Ontology with Wikipedia

Contents of Wikipedia (http://en.wikipedia.org/wiki/Main_Page) will be needed for this project. Wikipedia's entries are written in a relatively free format. Thus, basic information, including geo-code (the longitude and latitude of a location), must be extracted.

Dbpedia (http://dbpedia.org/About) is a community effort to extract structured information from Wikipedia. Extracted information is stored in Resource Description Framework (RDF), which is XML compliant. RDF information can be queried by a querying language called SPARQL.

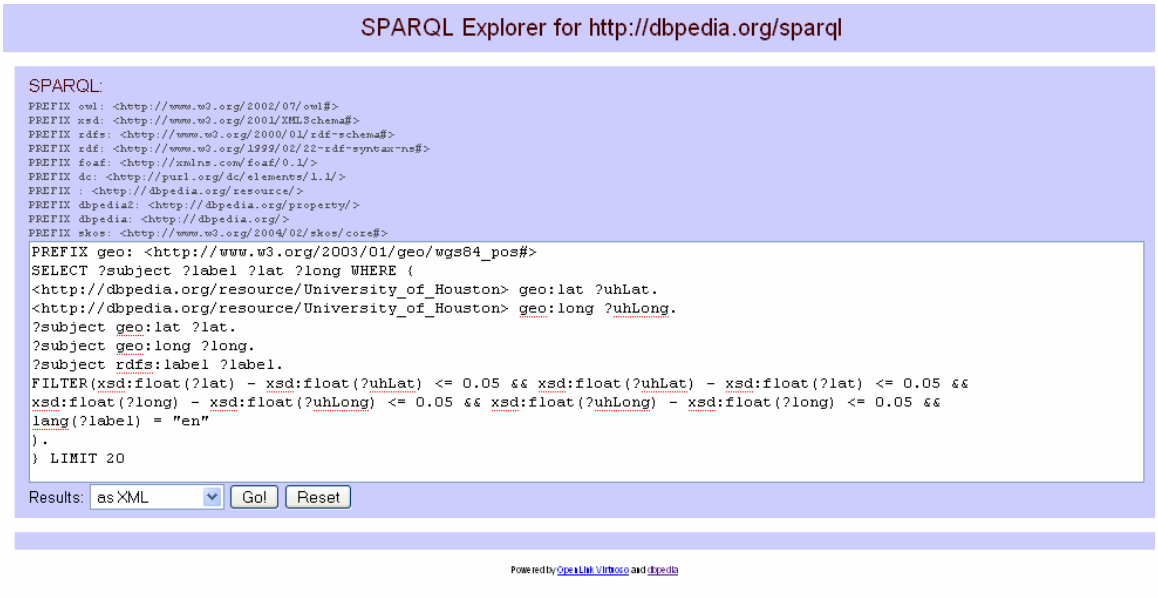
For efficiency, Dbpedia data can be downloaded and stored in a server. A SPARQL engine can then be installed to query the RDF data.

Dbpeida also have a Web front end SPARQL engine at http://dbpedia.org/snorql/.

As an example, here is the SPARQL code for finding all Wikipedia's 'things' with a geo-code that is off by 0.05 or less in both longitude and latitude with respect to the "University of Houston":

```
PREFIX geo: <http://www.w3.org/2003/01/geo/wgs84_pos#>
SELECT ?subject ?label ?lat ?long WHERE {
<http://dbpedia.org/resource/University_of_Houston> geo:lat ?uhLat.
<http://dbpedia.org/resource/University_of_Houston> geo:long ?uhLong.
?subject geo:lat ?lat.
?subject geo:long ?long.
?subject rdfs:label ?label.
FILTER(xsd:float(?lat) - xsd:float(?uhLat) <= 0.05 && xsd:float(?uhLat) - xsd:float(?lat)
<= 0.05 &&
xsd:float(?long) - xsd:float(?uhLong) <= 0.05 && xsd:float(?uhLong) - xsd:float(?long) <=
0.05 &&
lang(?label) = "en"
).
} LIMIT 20
```

Running through Dbpeida's SPARQL front end with XML output:



Dbpeida's SPARQL front end returns XML data:

```
<table class="sparql" border="1">
  <tr>
    <th>subject</th>
    <th>label</th>
    <th>lat</th>
    <th>long</th>
  </tr>
  <tr>
    <td>http://dbpedia.org/resource/The_Orange_Show</td>
    <td>The Orange Show</td>
    <td>29.71769142150879</td>
    <td>-95.32426452636719</td>
```

```
  </tr>
  <tr>
    <td>http://dbpedia.org/resource/University_of_Houston</td>
    <td>University of Houston</td>
    <td>29.71892166137695</td>
    <td>-95.33916473388672</td>
  </tr>
  <tr>
    <td>http://dbpedia.org/resource/Robertson_Stadium</td>
    <td>Robertson Stadium</td>
    <td>29.72200012207031</td>
    <td>-95.34928131103516</td>
  </tr>
  <tr>

<td>http://dbpedia.org/resource/University_of_Houston_College_of_Techno
logy</td>
    <td>University of Houston College of Technology</td>
    <td>29.72327995300293</td>
    <td>-95.3426513671875</td>
  </tr>
  <tr>
    <td>http://dbpedia.org/resource/Hofheinz_Pavilion</td>
    <td>Hofheinz Pavilion</td>
    <td>29.72468948364258</td>
    <td>-95.34700775146484</td>
  </tr>
  <tr>
    <td>http://dbpedia.org/resource/Cougar_Field</td>
    <td>Cougar Field</td>
    <td>29.72669982910156</td>
    <td>-95.34519958496094</td>
  </tr>
  <tr>
    <td>http://dbpedia.org/resource/Houston%2C_Texas</td>
    <td>Houston, Texas</td>
    <td>29.75</td>
    <td>-95.34999847412109</td>
  </tr>
  <tr>
    <td>http://dbpedia.org/resource/Toyota_Center_%28Houston%29</td>
    <td>Toyota Center (Houston)</td>
    <td>29.75072479248047</td>
    <td>-95.36211395263672</td>
  </tr>
  <tr>
    <td>http://dbpedia.org/resource/Discovery_Green</td>
    <td>Discovery Green</td>
    <td>29.75250053405762</td>
    <td>-95.35861206054688</td>
  </tr>
  <tr>
    <td>http://dbpedia.org/resource/Minute_Maid_Park</td>
    <td>Minute Maid Park</td>
    <td>29.75684356689453</td>
    <td>-95.35541534423828</td>
  </tr>
```

```
  <tr>
    <td>http://dbpedia.org/resource/Wells_Fargo_Bank_Plaza</td>
    <td>Wells Fargo Bank Plaza</td>
    <td>29.7584400177002</td>
    <td>-95.36826324462891</td>
  </tr>
  <tr>

<td>http://dbpedia.org/resource/JPMorgan_Chase_Tower_%28Houston%29</td>
    <td>JPMorgan Chase Tower (Houston)</td>
    <td>29.76055526733398</td>
    <td>-95.3638916015625</td>
  </tr>
</table>
```

For the bare-bone minimum, your application should include a component to obtain the points of interest close to a selected university from Dbpeida SPARQL front end. The team is encouraged to actually download and store the appropriate Dbpedia dataset in the server for added efficiency. However, the team will then need to address the data synchronization issues.

2. Data Preparation and Cleaning

The quality of data of Wikipedia cannot be perfect. It has both completeness and correctness issues. For example, it does not store the geo-code of the University of Houston-Clear Lake.

The list of the Texas universities and colleges may not be complete in Wikipedia and information may be missing. This is especially the case for the geo-codes.

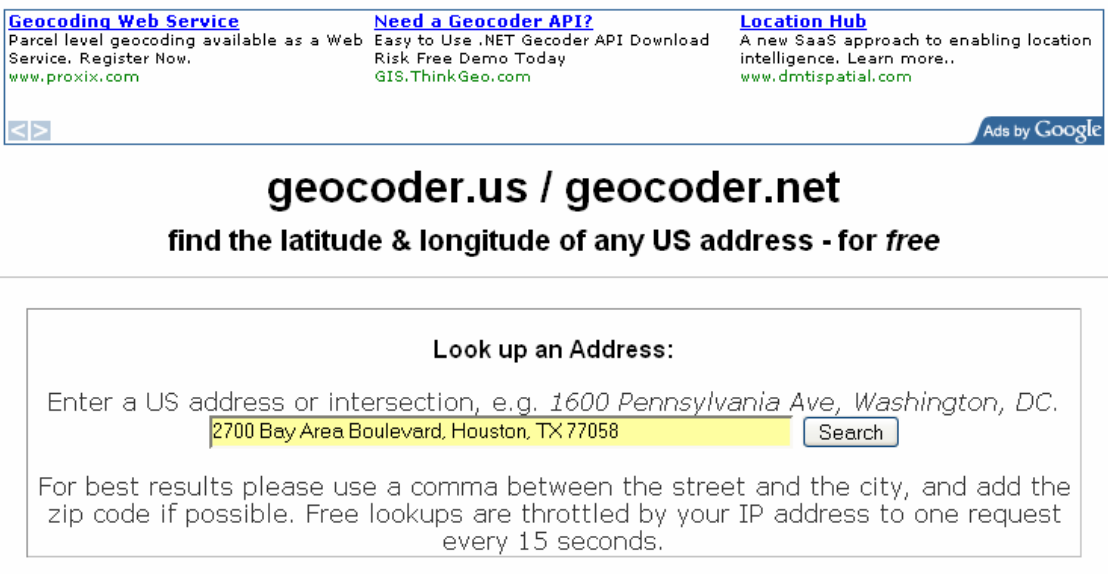
Thus, the team needs to construct a high quality list of Texas universities. It needs to research resources to be combined with Wikipedia to form such a list and mechanisms for combining the resources.

The team will also need to find a way to obtain accurate Geo-codes for all universities from Wikipedia as well as other sources. There are many ways to do so. Here is an example that can be entirely automated.

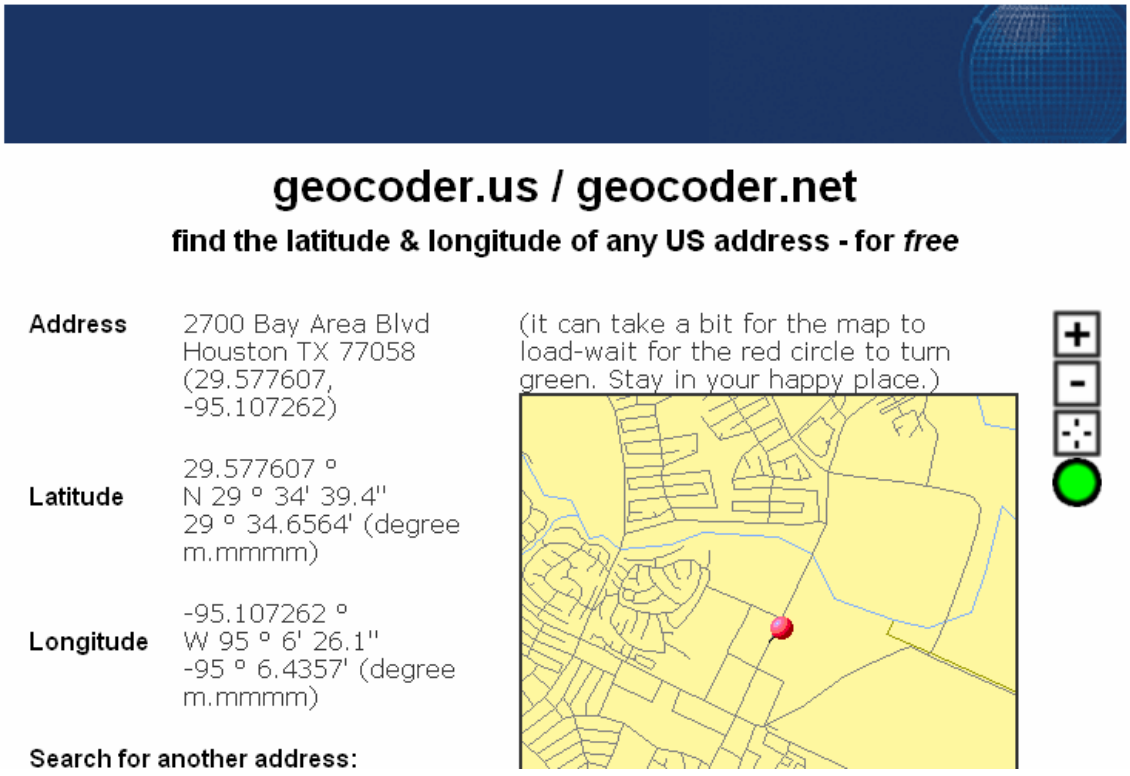
(1) Find the address of the university from Wikipedia or other sources. E.g. “2700 Bay Area Boulevard, Houston, TX 77058”.

(2) Submit this address to one of the many Web based or standalone geo-coders, such as: http://geocoder.us/.

In this case, we have:



The site returns something like:



(3) Geo-code information can then be extracted.

The provided example is not too good. For one thing, you can use the service only once per 15 seconds. Another reason is its return of HTML, not XML, making extraction inefficient and brittle. See http://geocoder.us/help/ for RDF and better output format.

Your design and implementation should be flexible and extensible. The way you work with university lists and geo-codes should be applicable to libraries and hospitals, for examples.

3. Display

The display should be via Google Map API (http://code.google.com/apis/maps/). More specifically, the AJAX API is required. AJAX is one of the backbone of Web 2.0. Note that you will need to obtain a key first.

In the bare-bone minimum, there should be an easy way for the users to select a Texas university. A Google map should then show up with the university and surrounding points of interest. When the mouse is on top of a point of interest, a pop-up box displays relevant information, including al link to the point of interest in Wikipedia.

There are hundreds of ways the team can improve on the bare-bone minimum. Use your imagination.