SemantGeo

SemantEco User Interaction Extension Through Geospatial Data Reasoning

SemantGeo is a powerful extension to the SemantEco framework giving the end-user the ability to drill down or expand upon their initial searches based off of Geospatial selections and subsequent reasoning. The extension acts as an additional level of reasoning empowering users to filter and parse the breadth of information residing in the SemantEco universe.

The additions introduced in the SemantGeo extension would rely on the Geospatial reasoning introduced from the GeoSparql query language. GeoSparql's "intent is to provide a standard way to express and query spatial elements in RDF, so that users can exchange data easily, and triple store implementers can have a standard format for indexing."[1] Here we cover three core use cases that were targeted early in the development of SemantGeo.

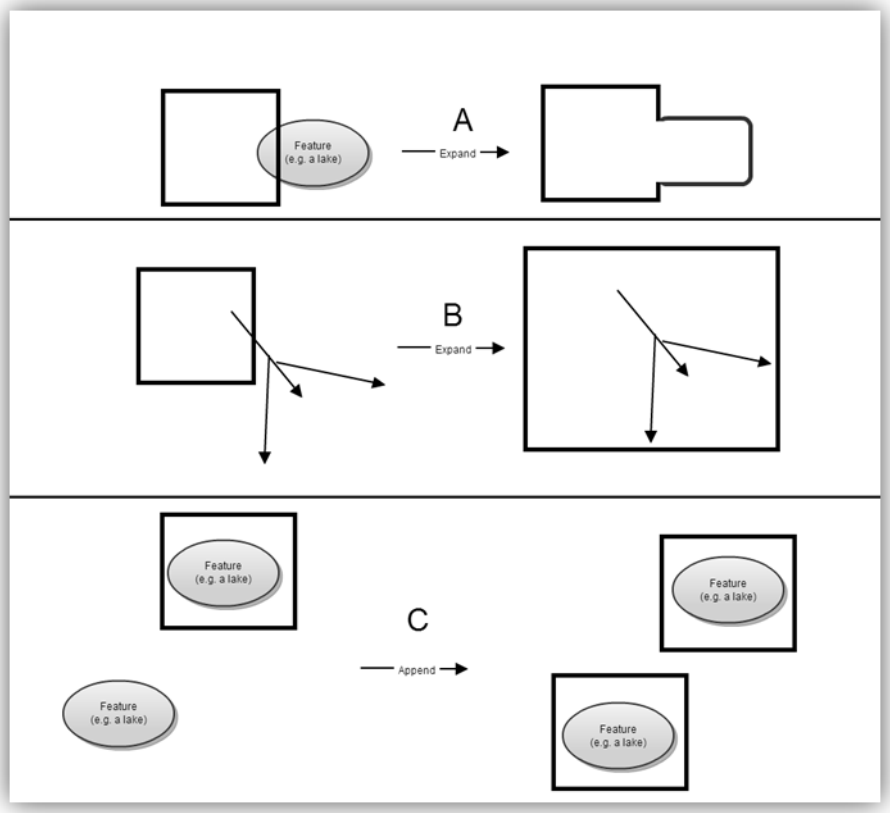

Figure 1: Early Use Cases

[1] Extracted from GeoSparql User Guide {link}

A user first forms an initial query to be made to SemantEco. Refer to Figure 1 Use case A where we abstract the idea of a query to a single square bounding box. The user is looking for information on everything contained in said bounding box. Knowing how information is related to each other empowers SemantGeo to suggest expanding the bounding box to include partially selected information. A real world example would be a geographic feature such as a lake. If the original bounding box only partially covered the lake's geographic shape, the user may want to expand their selection to contain the entire lake's shape. The SemantGeo system can see this option and suggest it to the user. The practice of helping the user refine their selections to make optimal queries is the heart of the SemantGeo extension. We can move from the abstract to a more concrete look at how this use case manifests in the GeoSparql query language.

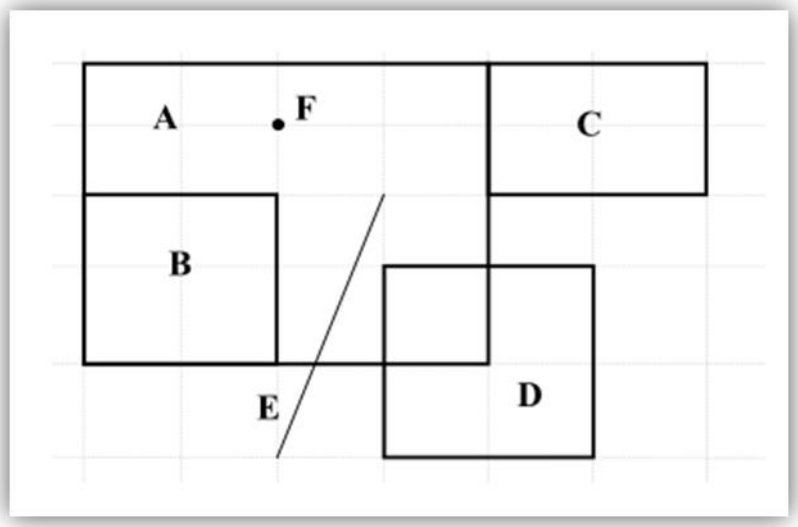

Figure 2: Spatial Data Illustration

Before we can consider interactions between the features in Figure 2, we need to understand how these features are defined in their 2D domain. They are defined by n-sided polygons built from a list of tuples[2]. We can see an example of the tuple data defining feature "A" in Figure 2.

```
<![CDATA[
        <http://www.opengis.net/def/crs/OGC/1.3/CRS84>
        Polygon((-83.6 34.1, -83.2 34.1, -83.2 34.5,
                -83.6 34.5))
    ]]>
```

Here we see the square is defined by 4 tuples which are in a list. They determine the 4 corners of the 4-sided polygon. Indeed, the pattern follows that for an n-sided polygon there are n-tuples defining the polygon. This polygon is also referred to the feature's "bounding box". The GeoSparql Query Language[3] allows for computing if one bounding box is overlapping another bounding box.

[2] Be it x,y in this example and lat, long tuples in the real world SemantGeo ontologies
[3] This language is formally defined by the Open Geospatial Consortium (OGC) {link}

Specifically, we utilize the RDF predicate "geo:sfOverlaps"[4] which "…exists if the subject SpatialObject spatially overlaps the object SpatialObject." We now apply this predicate against the layout of features in Figure 2 to reflect Use case A.

```
PREFIX my:   <http://example.org/ApplicationSchema#>
PREFIX geo:  <http://www.opengis.net/ont/geosparql#>

SELECT ?f
WHERE { ?f geo:sfOverlaps my:A }
```

We are selecting features "?f" where a triple is defined mapping the feature to the existing selection's bounding box "my:A" seen as the black square labeled "A" in Figure 2. The results of this query are Feature "E" (the line) and Feature "D" (the square). Relating back to our use case, "A" would be the selection made manually by the user, "E" could be a river or stream and "D" would be a lake defined by a square bounding box. At this point it becomes clear how we can implement the expanding action of Use case A. By seeing what features are returned from a "sfOverlaps" query we can suggest these returned features to the user to be added to the current selection and depending on what the user says act accordingly.

Consider Figure 1 Use case B. Again, the SemantGeo system intends to expand the original user selection to cover more of what the user would be interested in querying over. A common real world example of this use case would be if the current selection is a major river. There may be many smaller rivers and tributaries that branch off of the currently selected river and might be of interest. In this case, there is no overlap at all with the original selection, so we have to rely on a different logic. There are two approaches to this use case that were explored in-depth while developing SemantGeo.

The first leverages the actual spatial adjacency of the current selection to other features to determine if they are worth including, and the second relies on referenced ontologies that describe the features in a way that relates one feature to another.

For the first approach we can again use the GeoSparql language. Specifically, we are filtering features on the "geof:distance"[5] relationship. We can illustrate use case B using the features in Figure 2. We can query for the three closest features to "C" via the following:

```
PREFIX uom:  <http://www.opengis.net/def/uom/OGC/1.0/>
PREFIX my:   <http://example.org/ApplicationSchema#>
PREFIX geo:  <http://www.opengis.net/ont/geosparql#>
PREFIX geof: <http://www.opengis.net/def/geosparql/function>

SELECT ?f
WHERE { my:C my:hasExactGeometry ?cGeom .
```

[4] Defined formally in the opengis schema for GeoSparql v1.0 {link}

[5] Defined as "Returns the shortest distance in units between any two Points in the two geometric objects as calculated in the spatial reference system." {link}

```
        ?cGeom geo:asWKT ?cWKT .
        ?f my:hasExactGeometry ?fGeom .
        ?fGeom geo:asWKT ?fWKT .
        FILTER (?fGeom != ?cGeom) }
ORDER BY ASC (geof:distance(?cWKT, ?fWKT, uom:metre)) LIMIT 3
```

We are looking for features "?f" in order of ascending distance from the polygon that defines our feature "C". We do some checks to make sure we do not return feature "C" itself, and we get only the three closest features by limiting our query via "limit 3". We also want to point out that to be able to compare the geometry of the features we are referencing we get a reference to their actual geometry (list of tuples) via the "geo:asWKT[6]" predicate. The result of this query would be features "A", "D", and "E". Just like use case A, we could suggest these features to be added to the current user selection.

Depending on how the user may want adjacent features to be determined for them, we can modify our filter to choose all adjacent features within a certain distance instead of simply the closest three. This is done by changing the

```
ORDER BY ASC (geof:distance(?cWKT, ?fWKT, uom:metre))
```

line to

```
FILTER(geof:distance(?pwkt, ?wwkt, units:m) < 3000)
```

The message to take in is that the GeoSparql query language allows for a wide range of reasoning to be applied easily and quickly by making small changes to the underlying query structure.

The second approach to use case B is to rely on referenced ontologies that describe the features in a way that relates one feature to another. A real world example of this would be if a water body is known to be downstream from a different water body which is currently selected by the user. Logic would dictate that whatever may apply to the currently selected feature may also apply to something downstream. We have selected feature "C", but what if we look for all features downstream from feature "C" and we get back feature "A". We could then make a suggestion to expand the current selection to include feature "A". The critical element to this logic is the need for a "downstream" relationship defined from one feature to another.

As SemantGeo was developed, we explored various possible datasets that would have these relationship defined in them. Our primary candidate was pulling data from the USGS[7]. They maintain an extremely expansive repository of data on water bodies and water features in the United States. Users are able to request data from their backend database by selecting regions of the U.S and requesting data on all features contained in the selected region. This may sound familiar. Indeed, the USGS query interface exhibits use cases in parallel with the use cases that SemantGeo was developed to cater to. With this observation in mind, obtaining the data maintained by the USGS would be guaranteed to

[6] Defined by GeoSparql as "`serialization of a geometry`". {link}
[7] Their main site is {link} but we used {link} specifically.

further the abilities of SemantGeo. An example of querying for data from the USGS is seen in the following figure.

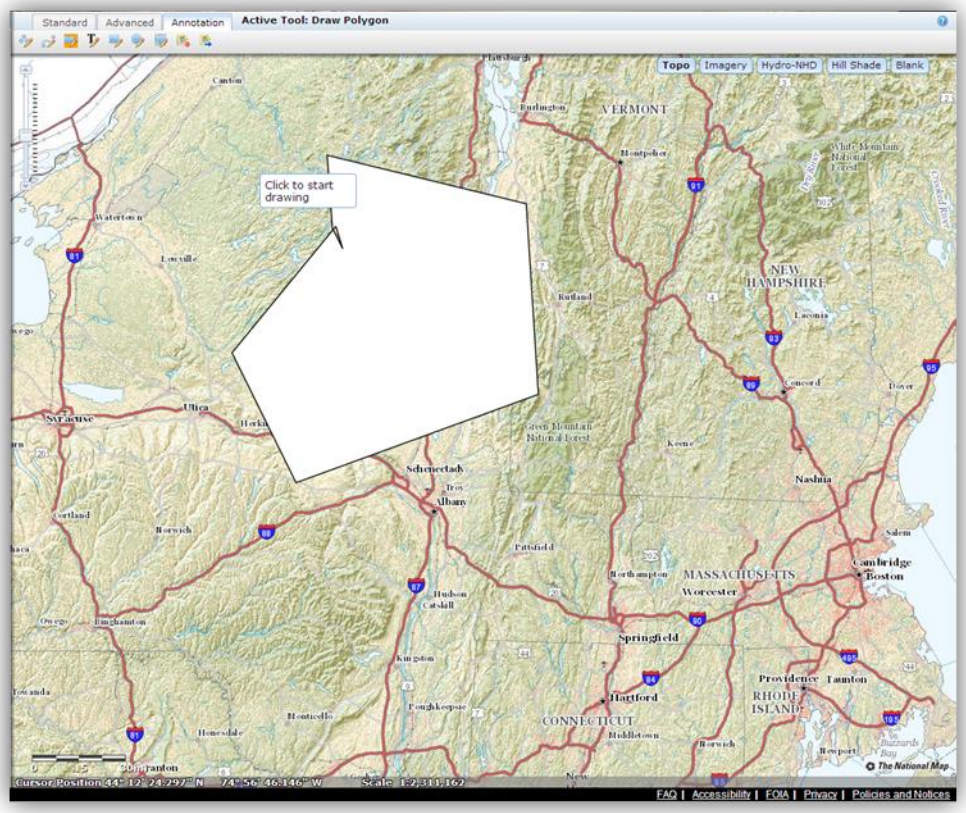

Figure 3: Using USGS Viewer

What is depicted in Figure 3 is a selection bounding box drawn by an end user over a region of the U.S. near Albany New York. A user would then request data on all features contained within this bounding box. The data would be in the format of n-triples[8] and is delivered by email within 2 weeks. During the development of SemantGeo in an effort to explore utilizing the data maintained by the USGS, we requested data on all water bodies in the entire state of New York. With this data at hand, over 500 megabytes of RDF triples uncompressed, We were able to compare the structure of the USGS data to the defined structure of GeoSparql.

[8] N-Triples is a line-based, plain text format for encoding an RDF graph. {link}

Showing statements for: **http://cegis.usgs.gov/rdf/nhd/Features/124618916**

graph	subject	predicate	object
Default Graph	-	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://cegis.usgs.gov/NHDOntology/Rapids
	-	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://cegis.usgs.gov/rdf/nhd/area
	-	http://www.opengis.net/ont/geosparql#hasGeometry	http://cegis.usgs.gov/rdf/nhd/Geometries/124618916
	-	http://cegis.usgs.gov/rdf/nhd/fType	http://cegis.usgs.gov/rdf/nhd/fType/431
	-	http://cegis.usgs.gov/rdf/nhd/shapeLength	"0.00459628191991"^^http://www.w3.org/2001/XMLSchema#float
	-	http://cegis.usgs.gov/rdf/nhd/fCode	http://cegis.usgs.gov/rdf/nhd/fCode/43100
	-	http://cegis.usgs.gov/rdf/nhd/areaSqKM	"0.005"^^http://www.w3.org/2001/XMLSchema#float
	-	http://cegis.usgs.gov/rdf/nhd/fDate	"2008-02-07T09:11:21"^^http://www.w3.org/2001/XMLSchema#dateTime
	-	http://cegis.usgs.gov/rdf/nhd/resolution	http://cegis.usgs.gov/rdf/nhd/resolution/medium
	-	http://cegis.usgs.gov/rdf/nhd/permanentIdentifier	"124618916"^^http://www.w3.org/2001/XMLSchema#string
	-	http://cegis.usgs.gov/rdf/nhd/shapeArea	"5.40621493886e-07"^^http://www.w3.org/2001/XMLSchema#float

Figure 4: USGS Data Structure

Referring to Figure 4 it becomes clear the structure of the USGS data reflects the structure exhibited in the schema definition for GeoSparql by the Open Geospatial Consortium (OGC)[9]. We are assured the ability to query against the USGS data in the same manner as when fulfilling the use cases above. Indeed, even the polygonal data is stored as a list of tuples, in this case (lat., long.):

```
"POLYGON (( -78.3859011888 43.1896324663, -78.3861483221 43.189643933, -
78.3867511888 43.189666133, -78.3872493221 43.1897713996, -78.3876340554
43.189805733, -78.3875154554 43.190011133, -78.3871243221 43.1900552663, -
78.3869177221 43.1899927996, -78.3868050554 43.1899331996, -78.3866665221
43.1899265996, -78.3864619888 43.1899168663, -78.3862879221 43.189908533, -
78.3860695221 43.1899149996, -78.3858767221 43.189960733, -78.3857585221
43.190084333, -78.3857296554 43.189999133, -78.3856778554 43.189846133, -
78.3856345888 43.189712333, -78.3856084554 43.189631333, -78.3857563221
43.1896212663, -78.3859011888 43.1896324663, -78.3859011888
43.1896324663 ))"^^http://www.opengis.net/ont/geosparql#wktLiteral
```

Where does this leave us when approaching use case B? We are clearly able to apply reasoning to any data we have from the USGS or data structured similarly. We were originally seeking for data connecting water features across relationships such as “downstream from”. It is clear there is no “downstream” logic in the USGS data, but other ontologies were explored for other data sets tied to SemantGeo that will be discussed later that are closer to what we were originally seeking from the USGS.

Lastly, Figure 1 use case C follows a similar structure to use case B. Given a feature selected by the user, select other features to be appended to the current selection based on similar attributes. A key difference is that we are not looking at adjacency or partial selection as seen in use cases A and B, but instead we are reasoning on properties of the original selection like what State or Region of the world

[9] http://www.opengeospatial.org/

the original selection was in. A simple real world example of this use case assumes the user has selected a lake in the state of New York. Given the attributes of the selection (Lake, New York), SemantGeo can suggest similar water bodies in the same State or Region. This use case requires the same class of reasoning as use case B.

The first iteration of SemantGeo needs to address the use cases that have been outlined. Given the use cases considered during the development of SemantGeo, and the issues that arose when determining how to address them, we can now step back and theorize how a functional first iteration of SemantGeo would be structured.

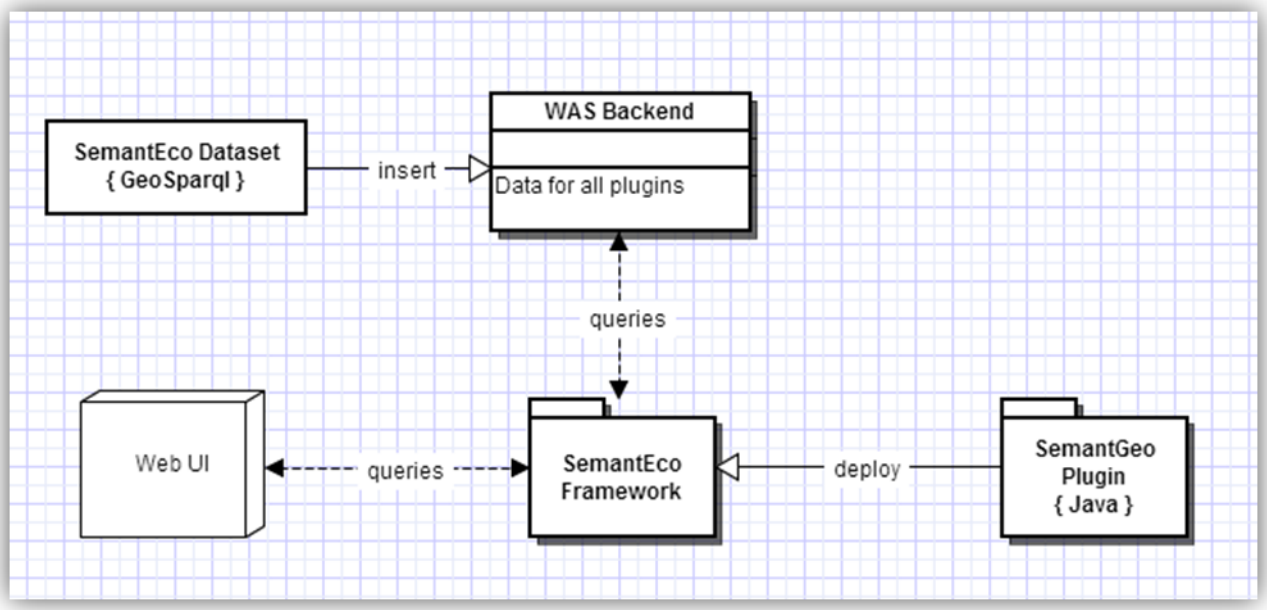

Figure 5: SemantGeo Deliverable

Figure 5 is a rough layout of SemantEco and how SemantGeo would be deployed into it. Users interact with SemantEco through a web browser interface such as Chrome or Firefox. The user interface (UI) utilizes SemantEco to modify Sparql queries, transparently query a backend database, apply reasoning on a response from the database, and update the UI according to retrieved data. Plugins running in the framework alter how the UI may look or interact with the user. They have the ability to extend how a user can query for information from the UI and modify queries before they get sent to the database. The database on the backend holds all the triples available to the user.

Two key elements make up SemantGeo, the triples it will leverage in its queries and the Java code that will extend the UI and describe the logic needed to implement the new features of SemantGeo into the existing abilities of SemantEco. From what we have discussed, the triples would in part be the data pulled from the USGS. We could insert this data into the backend triple store, and then query against it using GeoSparql in our Sparql queries. However, what about the Java code? Let's consider that in more detail.

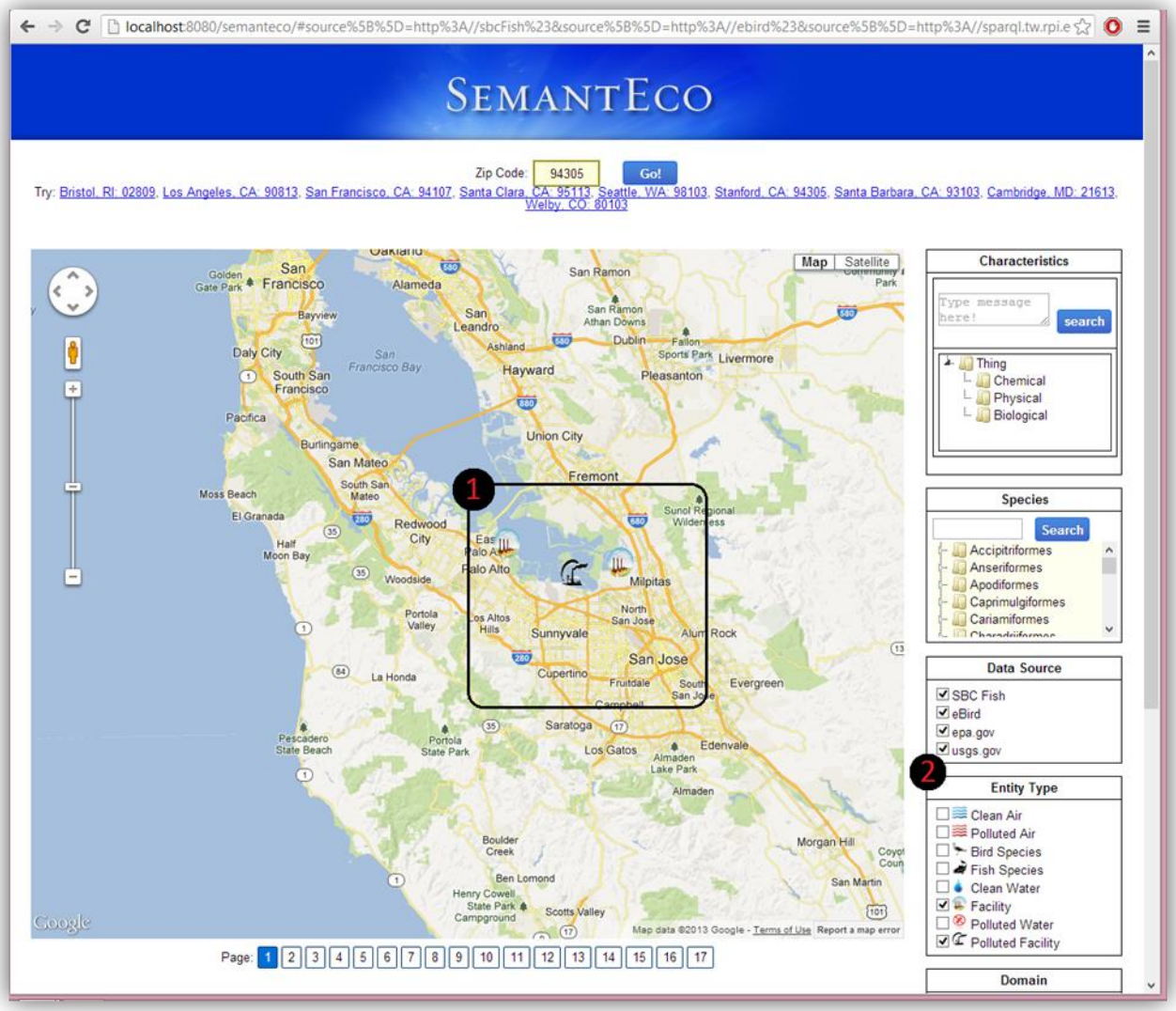

Figure 6: SemantEco UI

Figure 6 is a screenshot of the current look of SemantEco's UI. The 1 and 2 labels outline how SemantGeo would alter the UI to allow for its use cases. 1 shows the ability of a user to select a region of geography by drawing a n-sided polygon with the mouse[10]. 2 entails where additional facets would be inserted to cater to SemantGeo. Following the use cases a facet may be inserted allowing the user to determine what kind of suggestions they would like SemantGeo to make to them during normal operation. Suggestions could range from "Suggest features I have only partially selected" or "Suggest features adjacent and relevant to features I currently have selected". The Java code element of the first SemantGeo deliverable would populate this facet with these suggestions by looking at the polygonal selection data in the Google Map UI (the 1 section of figure 6), query the backend for data relevant to

[10] Of course, a user may want to paste a list of tuples to define a polygon, or any other corner case. The main point is we enable the ability to filter by a geographic bounding box.

this selection using the GeoSparql standards and the USGS triples we have mentioned before, and determine possible suggestions from the response.

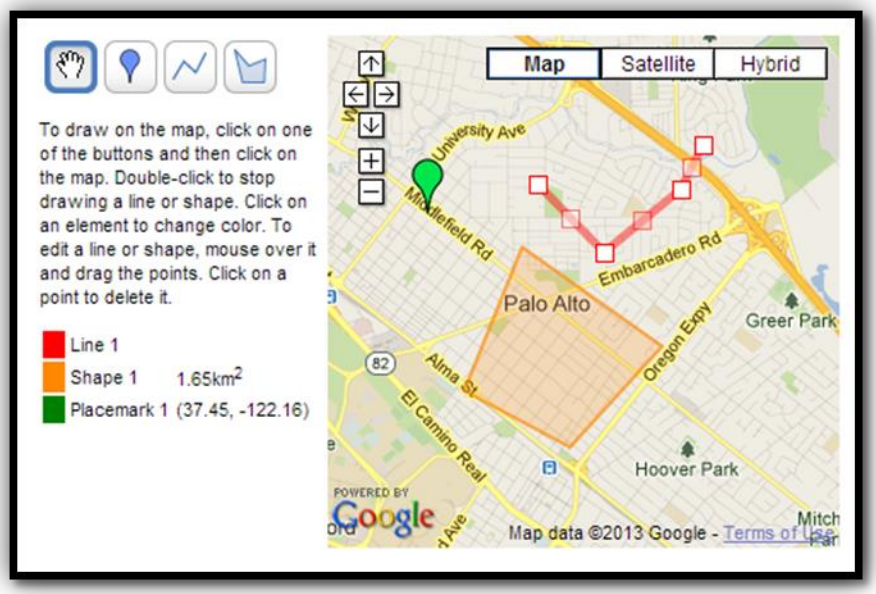

Figure 7: Google API Polygon Selection

A feature missing in the UI depiction of figure 6 was the ability to draw polygons in the Google Maps element of the SemantEco UI. Consider the screenshot in Figure 3 of the USGS interface. The ability to draw these polygons needs to be added to the SemantEco framework for SemantGeo to be able to fulfill its use cases. Adding this ability is not part of the previously mentioned two key elements of the first SemantGeo iteration. In fact, a SemantEco plugin would not even have the permissions to alter the Google Maps element of the SemantEco UI. Fortunately, adding this functionality simply requires following well documented guides by Google to allow for Polygon selections. Google refers to these as overlays. Figure 7 is an example of what the functionality would look like. In the image we can see a point selection as a green marker with a (lat., long.) tuple (37.45, -122.16), a n-sided polygon as an orange bounding box called “Shape 1”, and a series of red connected points as “Line 1”. We was able to recreate this functionality by following Google’s documentation on the subject[11].

We have defined the first iteration of a working SemantGeo plugin. Triples on the backend, and Java logic on the frontend. We can now cover the flow of activity when a user makes a query against SemantEco running SemantGeo as a plugin.

[11] See https://developers.google.com/maps/documentation/javascript/overlays

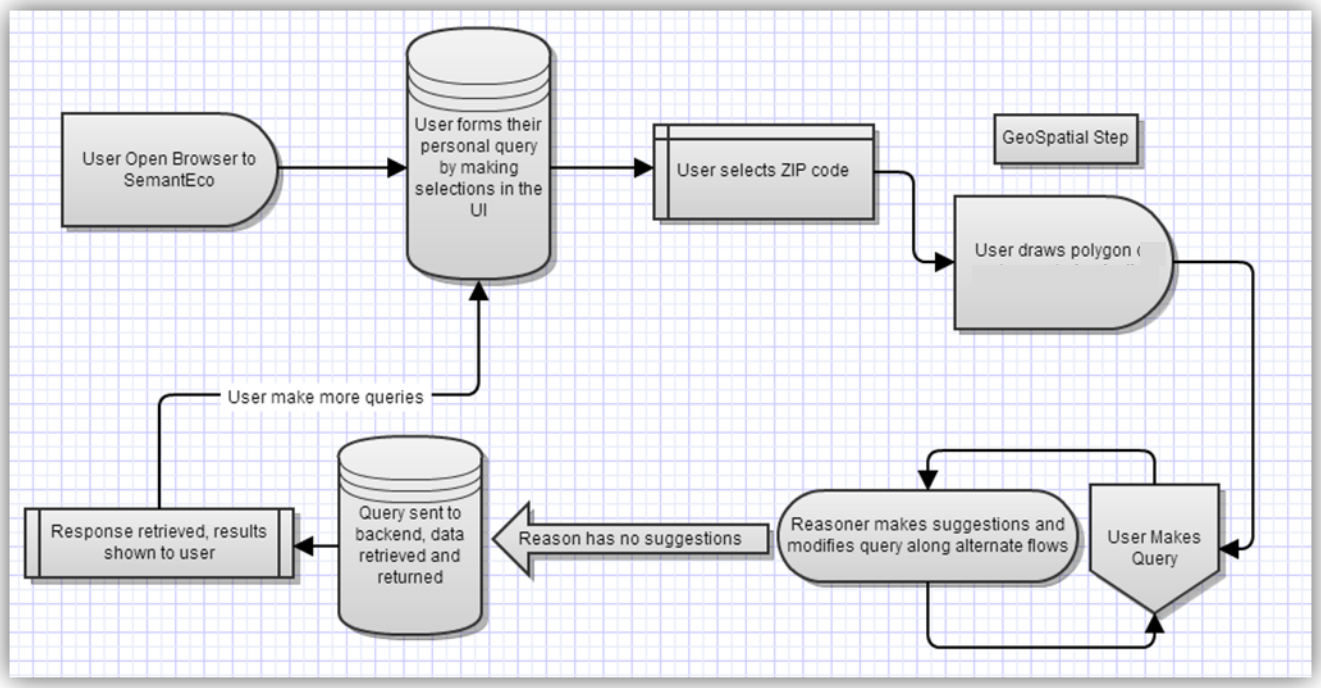

Figure 8: Activity Flow Diagram

A user begins by directing their browser to SemantEco. A user builds their query by selecting options from the facets on the right. A Zip code is selected to query against. The next step is the first deviation from normal SemantEco flow due to SemantGeo. The user draws a polygon in the Google Maps UI element. The user makes their query. The query goes to the back end server then comes back to the frontend with results. The system comes back to the user with suggestions concerning the original query after reasoning against the returned results from the backend. This is where, following the design use cases of SemantGeo, our plugin may suggest other features to include in the original polygon that was selected. From here a cycle occurs of a user querying, a user modifying their query due to suggestions, and then the user re-querying until they are happy with their results. Finally, the user may start all over again with an entirely new query. The system is cyclic by design.

Earlier we mentioned, "…other ontologies were explored for other data sets tied to SemantGeo that will be discussed later that are closer to what we were originally seeking from the USGS data." These other ontologies tie into a design goal that has yet to come to fruition within SemantGeo.

The ability to search not only against a polygon defined by latitude and longitude pairs, but also against geographical terms such as "South Side of Town" or "Hampton District" is crucial when trying to keep the usability of SemantGeo high. We realized that when a person knows a region of a city by "South Side of Town", they are simply referring to a bounding box by a common name. If a mapping of known regions of a city or state to bounding boxes could be established, then all the benefits of querying along GeoSparql could be utilized while still allowing the user to search using human readable

terms known to them on a more personal level. We understand it is not reasonable to think that humans will relate to geographical regions by bounding boxes alone, if at all.

A good example of this point is to think of how someone usually refers to a lake. They would call it "Lake X" and could easily point it out on a map. However, doing this with SemantGeo would be tedious and inefficient for them as it stands. If SemantGeo has a mapping to the known bounding box of Lake X, then the user could select "Lake X" from a facet on the right of the SemantEco UI and have the bounding box drawn for them automatically. The key factor in enabling this behavior is having data that links proper or informal names to pre-defined bounding boxes. This crucial data is still being searched for. There *is* promising data in the ontologies present at the BioPortal[12] ontology repository, specifically in relation to data obtained concerning the Darrin fresh water study on the Adirondack lakes. Another option in the short term is the hand make some triples relating common names to bounding boxes if only for proof of concept in the short term.

We consider some overarching questions concerning how SemantGeo is leveraging semantic technology. We have shown that SemantGeo leverages the GeoSparql language. We have USGS data that uses this Geosparql language in its organization meaning we can applying some reasoning to get logic out of our data that we would not normally have. This is shown in the use cases we covered. For example, seeing if one region is in another or which region is closest to another is done using Geosparql. Example Sparql queries are given that facilitate making these queries.

We are leveraging "Automated processing of Web information by software agents[13]" by utilizing the reasoning done by the various plugins running in SemantEco together. We understand that each plugin does its job to modify the Sparql queries being sent to the triple store database on the back end. The input from each of the plugins, put together, helps find new information or behaviors in the data that was not obvious when seen separately. An example relating to the use cases outlined could be showing pollution tied to a stream being a good indicator of pollution in another body of water that is known to be downstream. Tying the spatial link into the pollution data can expose how far a pollutant can travel downstream and still be sampled as at a toxic level. This data emerges as we bounce data sets off each other in the SemantEco system.

Why do we use these technologies? Why don't we just keep everything in a MySQL database locked away? It is known that relationships may exists that we cannot see until it is framed in reference to other data (that we may not even have). If that "other" data is hidden away, only accessible via a proprietary interface, or stored in a proprietary manner, than manipulating and interpreting that data has the overhead of "decrypting" the protocol into something that we can all work with first. By opening the data up from the get go, people can get access to and work with new data much easier allowing for the hidden behaviors to emerge quicker and easier than before. The move from simple row and cell based organization of data into the triples format allows for more freedom and a bigger "address space" for each term. Instead of locked into a single cell, each part of the triple has its own URI which is a unique address each able to be linked to by other terms. Much less restrictive.

[12] See: http://bioportal.bioontology.org/
[13] See: http://www.w3.org/TR/rdf-concepts/#motivation

Advanced Semantic Technologies (AST) has been a gateway course for me. I have been working on my Master's Thesis as early as my undergraduate Senior year. Working under my adviser's guidance, there was a concerted effort to intertwine courses into my curriculum that were in the same department as my Thesis work. It was always helpful to take these classes, but I never felt like I was truly getting the big picture when it came to applying Semantic Technologies to the problem of parsing, maintaining, and utilizing large amounts of data. Thanks for AST I recently had a huge 'aha' moment that has jump started my understanding and has been a big help with my overall grasp of the Semantic world. For example I can write basic to moderately advanced (filter, order, distinct, order by, sum(), average(), etc) Sparql queries now. I was even querying logd.tw.rpi.edu datasets for practice (Seriously I was!).

I still have a lot to get done. I wanted to get a lot done this past week, and despite being sick I was able to get more than I expected done. I wrote Java code for SemantGeo's plugin by modeling the code off other plugins that already existed in the project source. This needs the most focus, but my ability to make progress here was a good sign. I will be porting the Javascript code I wrote for the demos with the Darrin fresh water data to Java code. I need to Run my Cmap by Patrice (I have a Cmap for the Darrin freshwater data enhancements, but have not been in touch with Patrice this week) and get my USGS and other data into the WAS backend server so my module can query it. I still need to sit down with Jeremy and go over all the progress I have been making these past few weeks.

There is a great attention to Provenance when it comes to working with Tim's tool. I am able to capture the source and version of the data set I am working on. When this data was first seen. Which enhancements were in what version of the data, what computer system did the enhancements, and at what time UTC. The provenance helps show that the Darrin fresh water data started as XLS files on the RPI webserver, then they became CSV files via Microsoft Word, and finally which tool was used to convert/enhance them into compliant RDF. I have not used provenance to filter my Sparql queries, but I have leveraged provenance to discern between datasets sitting on the same environment. Thanks to how conversion cockpits are setup along dataset/version/* I can maintain multiple cockpits in the same environment. Their provenance helps sort them.

It is hard for me to comment on the "limitations of the tech" since I still see myself as the limiting factor in using the tech in the first place. The real limitation is still the breadth of data we have to query against. For instance, getting SemantGeo to interpret and reason against polygonal data is nothing for the user despite all the hard work if they make a query in China and we have not data on the region to query against. The real limitation is partial data sets. Another example being I was only able to get data on NYS from the USGS, and not the entire United States, or the World for that matter.

I would like to mention here that all the raw data being worked with on this project, papers used when learning about Darrin fresh water data, example conversions using Tim's tool, and more can be found on SemantGeo's Github page[14].

[14] See: https://github.com/apseyed/SemantGeo