Franz Semantic Technologies

Unification of Geospatial Reasoning, Temporal Logic, & Social Network Analysis in a Semantic Web 3.0 Database

Jans Aasman
Franz, Inc.
2201 Broadway, Suite 715
Oakland, CA 94612
+1-510-452-2000
ja@franz.com

ABSTRACT

This paper is about a new type of event database that makes it efficient to reason about things, people, companies, relationships between people and companies, and about places and events. This event database is built on top of a scalable distributed RDF triple store that can handle literally billions of events. Like objects, events have at least one actor, but usually more, a start-time and possibly an end-time, a place where the event happened, and the type of the event. An event can have many additional properties and annotations.

For example, telephone call detail records, email records, financial transactions, purchases, hospital visits, insurance claims, library records, etc. can all be viewed as events. On top of this event database we implemented very efficient geospatial and temporal queries, an extensive social network analysis library and simplified description logic. This paper focuses on a query framework that makes it easy to combine all of the aforementioned capabilities in a user friendly query language.

Keywords

Geotemporal logic, geospatial reasoning, RDF database, graph database, RDFS, OWL, SPARQL, social network analytics, business intelligence, event-based systems, event-driven architectures, metadata, semantic technologies.

1. INTRODUCTION

This paper describes the design and use of a unifying query framework for geospatial reasoning, temporal logic, social network analytics, RDFS and OWL in Event-based systems. In this introduction we will first go into why we need such a framework and the requirements for such a framework.

The reason for such a framework can be answered by looking at the vision of the semantic web and understanding how companies use semantic technologies. Tim Berners-Lee, James Hendler and Ora Lassila's Scientific American article (May, 2000) [1] provides a compelling vision of the Semantic Web. It contains some interesting use cases for what the Semantic Web will bring. These use cases assume that software agents know how to roam the web and reason over things, people, companies, relationships between people and companies and about places and events. Clearly these agents need a query capability that supports a combination of description logic, geospatial reasoning, temporal reasoning, and knowledge about the social relationships between people.

The commercial vendors of Semantic Technologies also see a number of use cases that all center around events and require the aforementioned query capabilities. We currently see companies using large data warehouses with very disparate RDF based triple stores describing various types of events where each event has at least two actors, usually a begin and end time, and very often a geospatial component. These events are literally everywhere: in Health Care applications we see hospital visits, drugstore visits, and medical procedures. In the Communications Industry we see telephone call detail records, now with location too. An email and calendar database of a large company is nothing more than a social network database filled with events in time and, in many cases, space. In the Financial Industry every transaction is essentially an event. In the Insurance Industry claims are important events and they desperately need more activity recognition. In the Homeland Security Industry basically everything revolves around events and actors. The REWERSE program from the 6th Framework Programme of the EU

Commission [2] is one of the few systematic efforts to combine RDFS/OWL with geotemporal reasoning, although the social aspect hasn't been addressed yet. The recent book “The Geospatial Web” [3] provides currently the best state of the art overview of how to work with people and events on a web scale and what kind of applications we might expect in the near future.

1.1 Framework Requirements

The Semantic Web community has made great strides in the area of ontologies and description logic, and some initial work in the areas of geospatial reasoning [4], temporal reasoning [5], social network analysis [15], and event ontologies [7]. All of this is based on RDF as the basic data representation. Based on this W3C standard the combination of all these different reasoning capabilities in one unified framework will propel further industry adoption of Semantic Technology. Given that we’ve seen a direct need for query capabilities that handle geospatial/temporal/social/rdfs/owl, we began designing a framework. The main requirements we identified were:

1. User and programmer friendly: We wanted the framework to be an extension of SPARQL, with SPARQL as the foundation. Certainly the framework should not be anymore complex than SPARQL. SPARQL is relatively user friendly, and as languages go, the adoption rate is such that one could make the argument that it is good enough.

2. Implementer friendly: The design that we propose in this paper is a work in progress. We need many people to experiment with this framework such that the Semantic Web community can converge on a standard.

3. Efficient: Given that we work with very large databases with millions of events where the response time has to be on the sub second level, the implementation of the query language and query engine needs to be very fast

4. We want to work the query language on distributed databases. Currently we’ve designed the query engine to work on federations of triple stores. Once we develop efficient caching techniques for distributed RDF knowledge stores residing all over the web, it will also be efficient for agents that need to roam the web.

5. Practical & Easily Extendible: We want the API to be such that it can be easily modified to allow for ongoing experimentation.

6. Works well with RDFS and OWL reasoning.

In the rest of this paper, we discuss our Event Ontology, the use of RDFS and OWL reasoning for events, our query API for temporal logic, our query API for geospatial reasoning, our API to a set of Social Network Algorithms, and how we combine it all in our query language.

2. EVENT ONTOLOGY

We examined a several event ontologies that have been published and are in use. Cyc [6] has a very elaborate event system. An example of a simple one that comes close to what we use was created by the Centre for Digital Music in Queen Mary, University of London [7]. We also looked at the type of industry applications that we envision and came up with a fairly simple set of common predicates that typically constitute an event.

Here is an example of an event:

```
@prefix rdf:     <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix event:       <http://www.acme321.com/ontology/0.1/> .
@prefix geo:    <http://www.geonames.org/#>.

event:event100100023
  rdf:type event:MobileTelephoneCall
  event:actor event:Person1
  event:actor event:Person2
  geo:longitude 37.223231
  geo:latitude -122.0177743
  event:start "2007-10-10T12:00:01"
  event:end ""2007-10-10T12:20:01"
```

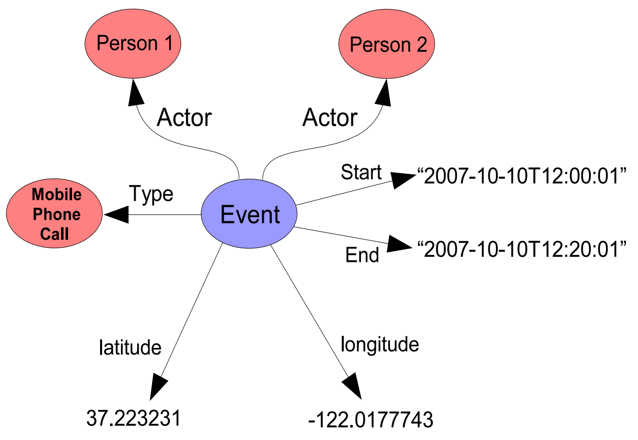


Figure 1. An Example of Common Event Properties

As is clear from figure 1, we use:

- **Type.** In most applications it is advisable to add an rdf:type to the event. The type itself can of course be a subclass of more general types. We found that even in practical situations we do need RDFS entailment reasoning to work with typed events. Example: in an RDF application dealing with meeting and calendar information, we used a class hierarchy for various types of meetings (general-meeting, work-meeting, personal-meeting, informal-meeting, interview, etc.)

- **Actors.** Although it is conceivable to have events without actors, say a sensor reading, we are interested in those types of events where we have multiple actors. Some examples include a telephone call, an email, a purchase order, a meeting, a hospital visit, etc. We are considering the FOAF [8] ontology and namespaces for actors.

- **Longitude, Latitude, and Place References.** You usually want to reason about where something happened. Sometimes you have a place name, sometimes you have the longitude and latitude. We use the geonames namespace of the wonderful GeoNames database [9] with over 6 million places on earth encoded in RDF.

- **Start Time, End Time, and Duration.** You usually want to reason about when something happened. We choose to use start time and end time, and we can deal with durations too. The dates that we use are defined by the ISO 8061 specification [10].

3. PROPOSED QUERY LANGUAGE

The standard syntax of our queries closely resembles SPARQL except that namespaces are globally defined before performing the queries. Actually, the query framework is currently implemented in Prolog; however, the Prolog layer is very thin. We implement all the underlying mechanisms directly in the native language of the triple store. We only use the Prolog for its backtracking and unification, and for having a unified way to pass around bound and unbound variables. Here is an example:

```
(register-namespace "event"
"http://www.acme321.com/ontology/0.1")
(select (?x ?y)
  (?x event:knows ?z)
  (?z event:loves ?y))
```

3.1 RDFS and OWL Reasoning

Our query framework uses the standard RDFS and OWL-lite reasoning. So in that sense there is not much to say about this particular topic. Here is an example:

acme:project-meeting rdfs:subClassOf acme:meeting

acme:informal-meeting rdfs:subClassOf acme:meeting

acme:informal-project-meeting rdfs:subClass acme:informal-meeting

acme:informal-project-meeting rdfs:subClass acme:project-meeting

acme:meeting1000 rdf:type acme:informal-project-meeting

acme:meeting1001 rdf:type acme:project-meeting

```
(select (?x)
    (?x rdf:type acme:project-meeting))
```

=>

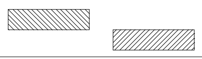
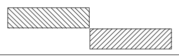
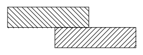
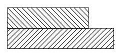
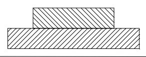
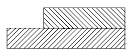
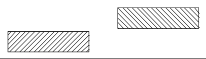
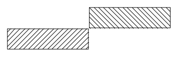
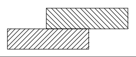
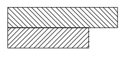
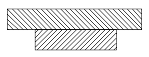
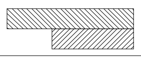
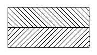
acme:meeting1000

acme:meeting1001

3.2 Temporal Logic

The kinds of temporal reasoning that you can do with events as defined above in the event ontology are fairly standard. We provide two types of temporal reasoning. The first type is reasoning over events as if they were points in time, so you can ask for events that started before or after another event started, or we can ask if a number of events are in order. The second type is about reasoning with intervals. You might want to find all of the events where the entire event happened before another event, or you might want to find all of the events that happened during another event.

Table 1. Allen's Interval Primitives

	**(interval-before ?e1 ?e2)**
	**(interval-meets ?e1 ?e2)**
	**(interval-overlaps ?e1 ?e2)**
	**(interval-starts ?e1 ?e2)**
	**(interval-during ?e1 ?e2)**
	**(interval-finishes ?e1 ?e2)**
	**(interval-after ?e1 ?e2)**
	**(interval-met-by ?e1 ?e2)**
	**(interval-overlapped-by ?e1 ?e2)**
	**(interval-started-by ?e1 ?e2)**
	**(interval-contains ?e1 ?e2)**
	**(interval-finished-by ?e1 ?e2)**
	**(interval-cotemporal ?e1 ?e2)**

The temporal logic our framework supports is entirely based on Allen's temporal logic [11] and was inspired by SNARK (SRI's New Automated Reasoning KIT) [12]. Like SNARK, our temporal representation supports two kinds of temporal entities, time points and time intervals, and relationships between them. Staying with the example we described above, say we want to find an informal project meeting that happened before the official project meeting during workday 200.

acme:workday200 event:start 2007-10-10T07:00:00

acme:workday200 event:end 2007-10-10T21:00:00

acme:meeting1000 event:start 2007-10-10T12:10:00

acme:meeting1000 event:end 2007-10-10T12:50:00

acme:meeting1001 event:start 2007-10-10T12:50:00

acme:meeting1001 event:end 2007-10-10T13:10:00

```
(select (?x ?y)
  (interval-during acme:workday200 ?x)
  (?x rdf:type acme:informal-meeting)
  (interval-during acme:workday200 ?y)
  (?y rdf:type acme:project-meeting)
  (interval-before ?x ?y))
```

As we mentioned before, we use Prolog as the carrier language for the query framework and that provides us many advantages. As an example, the following shows how to use the point-before primitives in various ways.

Find everything before meeting1001.

```
(select (?x) (point-before ?x
acme:meeting1001))
```

Find everything after meeting1000.

```
(select (?x) (point-before
acme:meeting1000 ?x))
```

Find all before/after pairs, (don't ever do this in a large database).

```
(select (?x ?y) (point-before ?x ?y))
```

Is meeting1000 before meeting1001?

```
(select () (point-before
acme:meeting1000 acme:meeting1001))
```

3.3 Geospatial Reasoning on Events and Shapes

In some cases you want to find the distance between two events, or whether an event happened within a given distance of another event. Or you defined a particular closed polygon and you want to find out whether polygon1 is in polygon2, or whether polygon1 and polygon2 overlap. We support all these in the query language. The following are a few example API calls:

This finds all the ?x within the bounding box specified by minlat, maxlat, minlon and maxlon. If ?x is bound it will return true/false.

```
–  (geo-bounding-box ?x minlat maxlat
minlon maxlon)
```

This will find all the events ?y that are in the bounding box around ?x where the bounding box is specified by going 'miles' miles left and right and up and down. If ?y is bound it will return true/false.

```
–  (geo-box-around ?x ?y miles)
```

Like the previous but now using the true radius.

```
– (geo-radius-around ?x ?y miles)
```

Find the distance between ?x and ?y and unify with ?dist.

```
– (geo-distance ?x ?y ?dist)
```

Find if polygon1 is in polygon2.

```
– (geo-polygon-contains p1 p2)
```

Now we add some longitudes and latitudes to the events, and ask the same question we did before but we want to make sure the distance between the two meetings is less than 9 miles.

[My home town Moraga, CA]

acme:meeting1000 event:latitude 37.83492660522461

acme:meeting1000 event:longitude -122.12968444824219

[Berkeley, CA]

acme:meeting1001 event:latitude 37.8715934753418

acme:meeting1001 event:longitude -122.27274322509766

```
(select (?x ?y)
  (interval-during acme:workday200 ?x)
  (?x rdf:type acme:informal-project-
meeting)
  (interval-during acme:workday200 ?y)
  (?y rdf:type acme:project-meeting)
  (interval-before ?x ?y)
  (geo-box-around ?x ?y 9))
```

Note: the triple store we use includes the well-known great-circle distance to compute distances between points accurately, see wikipedia for law of haversines [13].

3.4 Social Network Analysis

Social Network Analysis (SNA) is concerned with understanding the structure of groups, the relationships and information flow in and between groups, and the role of an individual actor in their groups. SNA has been around since the 1930's. It picked up steam in the 1950's when it started to bloom within Sociology and Social Psychology. Since the 60's it is closely related with the mathematical field of graph analysis. For readers new to this area, the University of Essex offers a wonderful course on the basics of SNA [14].

The importance of SNA for the Semantic Web was recently brought into the open by Peter Mika's book Social Networks and the Semantic Web [15]. A quote from his book that summarizes this:

> "We are not just building the Web anymore: we are on it. The latest set of applications have transformed the Web from a mere document collection into a social space: the new services developed under the banner of Web 2.0 cater to our needs of connecting through the medium and allow us to explicitly describe, maintain and develop our online self. At the same time, documents and other forms of content are not only up- and downloaded anymore, but actively exchanged, filtered, organized and discussed in groups of all sizes …"

Social Network graphs are the most challenging graphs to work with: they are mostly a mix of directed and undirected, mostly not hierarchical, and are usually cyclic. As part of this unification effort, we implemented a complete set of SNA algorithms that help us answer some important questions with respect to groups.

Q1: What is the degree of separation from actor1 (A1) to actor2 (A2) and what is the connection strength from A1 to A2?

The degree of separation from A1 to A2 is computed by finding the shortest path from A1 to A2 in the graph. Each actor might have a number of attributes, some directly encoded in the triples, some can be found by the RDFS/OWL reasoner and some can be computed on the fly by applying rules or prolog functors. Before we compute a shortest path we first have to define a generator that determines for every childnode how it should expand into its childnodes. The following says that if you want to go through the graph from A1 to A2, only follow the links where this particular node had a phone call or email with the next actor.

```
(defgenerator gen1
(undirected(telephone-call),
undirected(email)))
```

So now we find the degrees-of-separation between two people through the following call where ?a1 and ?a2 need to be bound. If ?generator is bound it will use the generator to expand nodes, otherwise it will follow all links.

```
(sna-shortest-path ?a1 ?a2 ?generator
?path)
```

The connection strength between two actors is computed by counting all the paths between actors ?a1 and ?a2, where the generator determines which paths are followed, and 'depth' determines the maximum length of a path.

```
(sna-connection-strength ?a1 ?a2
?generator ?depth ?result)
```

Q2: In what groups is actor A?

Find the group around a person to the Nth level. This is useful in some cases to avoid catastrophic complexity.

```
(sna-ego-group ?a ?generator ?depth
?group)
```

It is more interesting to find all the fully connect graphs around a person. Such a connected graph is also called a 'clique'. As an example: I am in a family-clique that consists of my family members and some friends of the family members that know all the others in the family. In addition, I am part of a work-clique, a tennis-clique, a Starbucks-clique, etc. In figure 2 we can see that example person 3 is in three different cliques.

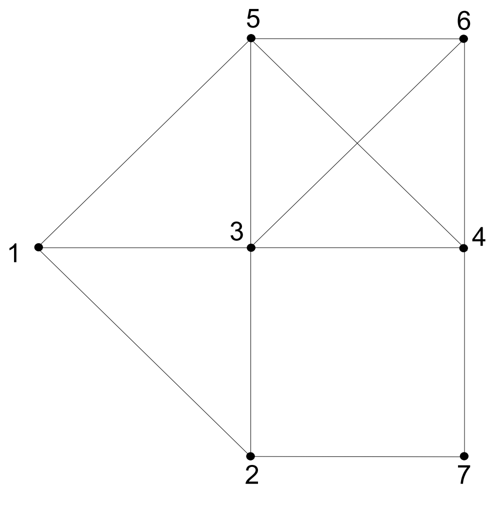


Figure 2. Cliques: {1,2,3}, {1,2,5}, and {3,4,5,6}

```
(sna-clique ?p1 ?generator ?minimum-
depth ?clique)
```

Q3: Who are the key players in a network?

There are several ways to compute the importance of an actor in a particular group. The most obvious ones are 'in-degree' (how many people point at me), 'out-degree' (how many people do I point to) and nodal degree (add the previous two). A more sophisticated method is to compute the centrality of an actor. The three most important actor-centrality measures that we've implemented are:

1. Actor degree centrality: I have the most connections in a group so I am more important.
2. Actor closeness centrality: I have more shortest paths to anyone else in the group so I am more important.
3. Actor betweenness centrality: I am more often on the shortest path between other people in the group so I am more important. I can control flow of information better than other people.

Q4: How strong and well connected is this group or how centralized is this group?

If you want to know how well connected a group is you can compute its density. Density is defined as the ratio between actual number of connections of all the actors in the group and the theoretical possible number of connections in a group. More complicated is to compute to what a group is centralized around. We compute group-degree-centrality, group-betweenness-centrality and group-closeness-centrality.

3.5 Putting It Together

So, we now have discussed how to use RDFS/OWL in this framework, we showed how to use Geospatial and Temporal reasoning; now we finally can do a query where we also add Social Network Analysis to the mix.

If we go back to our previous example: do we have a shared friend that was in a meeting today close to us that we were not in? There are undoubtedly other ways to do this but here is one example:

```
(select (?x ?y ?a2)
  (interval-during acme:workday200 ?x)
  (?x rdf:type acme:meeting)
  (?y rdf:type acme:meeting)
  (?z rdf:type acme:meeting)
  (not (= ?x ?y ?z))
  (geo-box-around ?x ?y 0.5)
  (geo-box-around ?x ?z 0.5)
  (?x event:actor ?a)
  (sna-clique ?a gen1 3 ?ac)
  (?y event:actor ?b)
  (sna-clique ?b gen1 3 ?bc)
  (?z event:actor ?c)
  (not (= ?a ?b ?c))
  (member ?c ?ac)
  (member ?c ?bc))
```

3.6 SPARQL Research Questions

One question that we get repeatedly is whether we can make this work in SPARQL. SPARQL very quickly became the query language of choice so from that perspective it would be great if all the technology discussed in this paper would actually work from SPARQL.

The biggest challenges with the current SPARQL specification are:

1. SPARQL only works on raw and inferenced triples. If we look at the examples that have been described in this paper above we see that we've added many query types that are actually rules. One solution the SPARQL community has come up with is to define so-called 'magical' predicates. The SPARQL engine recognizes the special 'magical' predicates and instead of looking in the triple database, it will invoke a rule or function with the arguments supplied in the specific SPARQL clause.

2. SPARQL only supports 2-ary predicates. In the examples we see 3 or 4-ary predicates. Take as an example geo-box-around. It takes three arguments. One solution would be to change SPARQL to take magic predicates that take more than 2 arguments. Another option would be to break up the 3-ary predicate in two clauses with 2 arguments and then have the engine turn it into a 3-ary magical predicate.

4. DISCUSSION

In this article we have shown how we can combine geospatial reasoning, temporal logic, social network analytics, and RDFS reasoning all in one query language. It is relatively unimportant that this is prototyped in Prolog, as we will in the near future rewrite this into something more amenable to query optimization.

One question that people ask who are familiar with triple stores is: how can this work efficiently on very large data sets containing hundreds of millions of triples? Most first generation triple stores store the URIs and literals that constitute the parts of a triple as strings in a dictionary. So then, when doing range queries over numeric values, for example, "select * from person where age > 50", the triple store engine has to go through each value for the predicate age. One way around this is to add btrees for every numeric type but that in general is a very inefficient solution in triple stores. The triple store that we use is AllegroGraph [16] which is actually a hybrid between a relational database and triple store, the internal representation of the triples is such that it allows for very efficient range queries.

So let us conclude with some notes on the efficiency of our query framework.

4.1 Temporal Reasoning

Temporal reasoning obviously uses the range queries to the fullest extent. If you want to find all the events that happened between Jan. 1, 2008 and Jan. 2, 2008, the triple store does a straight triple query with only one cursor scan. It is still possible to blow up the query time spectacularly by doing things like

```
(select (?x ?y) (point-before ?x ?y))
```

as that will generate every before/after pair. However, we do consider that to be the responsibility of the user. In many cases a query optimizer can warn for that or rearrange the clauses to bind ?x or ?y.

4.2 Geospatial Primitives

In order to make this fast we implemented a variation of an R-Tree [17] to encode two-dimensional data very efficiently directly in the triple indices. A detailed description of how this geospatial representation works can be found in the Geospatial Tutorial included with the AllegroGraph documentation [18].

Using the GeoNames database [9], we can retrieve all 459 geo-points around Berkeley less than 4 miles away in less than 5 milliseconds. We would argue that the basic retrieval speed is comparable to or better than full-scale spatial databases.

4.3 Social Network Analysis

A native, general graph database is written specifically to make graph search faster. However, the bottleneck is still getting triples from disk as fast as possible and having the smartest algorithms and best caching available. For example, many of the centrality measures that are used to compute the importance of an actor in a known group need to compute the shortest path between every actor in the group. We have created special constructors to cache these groups in a transparent way so that most computations can be done in memory. We are satisfied with the current performance but we are also happy there are still many places where we can improve performance.

4.4 Query Optimization

The primary research effort for the current version of the query framework is to make query-optimizing work. In a framework where it is hard to predict how many new bindings a clause will generate, query optimizing is hard. We still need to rely on smart users to order clauses appropriately.

5. REFERENCES

[1] 1st Scientific American article on the Semantic Web, http://www.sciam.com/article.cfm?articleID=00048144-10D2-1C70-84A9809EC588EF21&ref=sciam

[2] REWERSE, http://rewerse.net

[3] Scharl, A., Tochtermann, K.: The Geospatial Web: How Geobrowsers, Social Software and the Web 2.0 are Shaping the Network Society. Springer (2007)

[4] W3C Geospatial Incubator Group, http://www.w3.org/2005/Incubator/geo/

[5] Gutierrez, C., Hurtado, C., and Vaisman, A. Temporal RDF. In European Conference on the Semantic Web (ECSW'05) (Best paper award), pages 93–107, 2005, http://www.dcc.uchile.cl/~cgutierr/papers/temporalRDF.pdf

[6] Cycorp, http://www.cyc.com/

[7] Raimond, Y. Abdallay, S., Event Ontology, 2007, http://motools.sourceforge.net/event/event.html

[8] The Friend of a Friend Project, http://www.foaf-project.org/

[9] GeoNames Data Access, http://www.geonames.org/export/

[10] The international standard date and time format, http://www.cl.cam.ac.uk/~mgk25/iso-time.html

[11] Allen, J.F.: Time and Time Again: The Many Ways to Represent Time. International Journal of Intelligent Systems, Vol. 6, No. 4 (1991)

[12] SNARK, SRI's New Automated Reasoning Kit, http://www.ai.sri.com/~stickel/snark.html

[13] Wikipedia, the Haversine formula, http://en.wikipedia.org/wiki/Haversine_formula

[14] University of Essex, Social Network Analysis course, http://www.analytictech.com/essex/schedule.htm

[15] Mika, P.: Social Networks and the Semantic Web. Springer (2007)

[16] Franz Inc.'s AllegroGraph 3.0, http://agraph.franz.com/allegrograph/

[17] Wikipedia R-tree data structure, http://en.wikipedia.org/wiki/Rtree

[18] Geospatial tutorial section of Franz Inc.'s AllegroGraph 3.0 documentation, http://agraph.franz.com/support/documentation/3.0/geospatia l-tutorial.html