Event Processing using an RDF Database

Jans Aasman
Franz, Inc.
555 12th Street, Suite 1450
Oakland, CA 94607
+1-510-452-2000
ja@franz.com

Abstract

This paper is about a new type of event database that enables efficient reasoning about things, people, companies, relationships between people and companies, and about places and events. The 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. On top of this event database we implemented libraries for RDFS++ logic reasoning, for geospatial and temporal capabilities, and an extensive social network analysis package. This paper focuses on a query framework that makes it easy to combine all of the aforementioned capabilities in a user friendly query language.

I. 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 [1]. 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) [2] 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, including location. 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 Intelligence community basically everything revolves around events and actors. The REWERSE program from the 6th Framework Programme of the EU Commission [3] 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" [4] currently provides the state of the art overview on how to work with people and events on a web scale and what kind of applications we might expect in the near future.

II. 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 [5], temporal reasoning [6], social network analysis [7], and event ontologies [8]. All of this is based on RDF as the 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 have

seen a direct need for query capabilities that handle geospatial/temporal/social/rdfs/owl, we have designed 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 sufficient to address most use cases.

2. Implementer friendly: We need many people to experiment with this proposed 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 the query language to work 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.

III. DISCUSSION

In the remainder of the paper we show how we can combine geospatial reasoning, temporal logic, social network analytics, and RDFS reasoning all in one query language.

One question that people ask who are familiar with triple stores is: how can this work efficiently on very large data sets containing billions 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, 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 which is actually a hybrid between a relational database and triple store, the internal representation of the triples is such that is allows for very efficient range queries.

A. Temporal Reasoning

Our temporal reasoning is based on James Allen's Interval Logic [9]. This logic looks at all the 13 ways two temporal intervals can relate to each other. We provide predicates for each of Allen's 13 interval predicates. Note that we do purely quantitative temporal reasoning. So if you provide a number of events with a start time and an end time or a duration then we can perform queries like the following. This example will return all intervals ?i2 that happened in interval ?i1.

```
(select ?x (interval-during ?i1 ?i2))
```

Temporal reasoning uses the range query capabilities 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 performs 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.

B. Geospatial Primitives

Our original intention of adding Geospatial capabilities was not so much to compete with existing spatial databases but instead make it very easy for RDF users to be able to deal with locations of objects very efficiently. In order to make this fast we implemented a variation of an R-Tree to encode two-dimensional data very efficiently directly in the triple indices [10]. A detailed description of how this geospatial representation works can be found in the geospatial tutorial included with the AllegroGraph documentation [11]. Currently we support a number of predicates that can be used in the query language. Some examples of the predicates:

```
(geo-distance ?x ?y ?dist)  -> given, x and y, return distance
```

(geo-within-radius ?x ?y 10.0) -> find y within 10 miles from x

(geo-inside-polygon ?polygon ?place ?lon ?lat)

For our benchmarking we use the open source GeoNames database that can be freely downloaded from GeoNames.org [12]. The database contains nearly 7 million points of interest on earth. From interesting points in nature, to populated areas, to schools and churches, etc. Each point has 12 features such as asciiname, the local name, elevation level, longitude, latitude, population, etc. Actually, it is not a database but a csv file that programmers can modify as necessary. For our purposes we obviously transform it into RDF triples. 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 current commercial spatial databases. Here are some typical example queries that you can do on the GeoNames database:

Find the distance between Oakland and the one and only Berkeley in California.

```
(select (?dist)
    (q ?x geo:name "Oakland")
    (q ?y geo:name "Berkeley")
    (q ?y geo:admin1_code "CA")
    (geo-distance ?x ?y ?dist))
```

Put in a Google map all the places within 10 miles from Oakland

```
(google-map (select (?name ?lat ?lon)
        (q ?x geo:asciiname "Oakland")
        (geo-within-radius ?x ?y 10)
        (q ?y geo:asciiname ?name)
        (q ?y geo:isAt5 ?pos)
        (pos->lon/lat ?pos ?lon ?lat)))
```

C. Social Network Analysis (SNA)

Many RDF resources are about people and relationships between people, or between people and companies, or between companies and other companies. We added Social Network Analysis methods to make it easier to reason about relationships and groups. The functions that we provide address the five basic questions from Social Network Analysis. (1) How far is person A from person B, (2) if there is a link between A and B then how strong is this relationship, (3) given a particular actor A, in what group does this actor 'live', (4) given an actor in a group, how important is this actor in the group and finally, (5) given a group, how dense are the relationships in the group and does this group have a leader or a set of leaders. The SNA library encompasses a set of well know SNA algorithms. We provide a set of general functions and have developed the concept of a generator. A generator is basically a function that takes as an input one node and then creates a set of output nodes. The search functions and SNA functions that we provide take these generators as first class arguments. For example: say we have a database with relationships between people, the generator 'knows' will take as an input a person and return a set of person(s) by following fr:went-to-dinner-with and fr:went-to-movies in both directions.

```
(defgenerator knows ()
    (bidirectional fr:went-to-dinner fr:went-to-movies))
```

We can use this generator to find, for example, the shortest path between two people. In this case the query will return a list of persons.

```
(select ?x
        (shortest-path knows fr:Person1 fr:Person2 ?x))
```

Or we can use the generator to first create a group of friends and friends of friends in the ego-group predicate, and then we find the importance of each member using the actor-centrality measure. This predicate will start with the most important one first.

```
(select ?x
        (ego-group fr:Person1 knows 2 ?group)
        (actor-centrality-members ?group knows ?x))
```

AllegroGraph is a native, general graph database, 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 without minimal IO.

IV. AN OVERVIEW EXAMPLE

In order to give the reader an impression of the breadth and depth of the query language, we provide a typical example that combines geospatial, temporal, SNA and RDFS reasoning.

```
(select (?x)
  (ego-group person:jans knows ?group 2)
  (actor-centrality-members ?group knows ?x ?num)
  (q ?event fr:actor ?x)
  (qs ?event!rdf:type fr:Meeting)
 (interval-during ?event "2008-12-01" "2008-12-05")
  (geo-box-around geoname:Berkeley ?event 5 miles)
  !)
```

In English this translates into:

> *Find the group of friends and friends of friends around the person "Jans". Find within this group the most important person first. Find if this person was part of an event that was of type Meeting and happened in a particular time interval within 5 miles of Berkeley.*

Note that we seamlessly mix Social Network Analysis in the first two clauses, a simple database look up in the third, an RDFS inference about the type of event, and then a temporal and a geospatial constraint. This current example and the examples shown above utilize Prolog. We expect in early 2009 to have a SPARQL engine that will perform this identical query.

The syntax of the SPARQL query will be slightly more contrived due to the fact that SPARQL normally only allows patterns that map directly on triples (see example below). Note that we introduced the non-standard '=' or assignment construct. We are planning to discuss this topic with the SPARQL committees.

```
select ?x where {
  ?group =  ego-group(person:jans knows 2) .
  ?x = actor-centrality-members(?group knows ?x) .
  ?event fr:actor ?x ;
         rdf:type fr:Meeting .
  FILTER (interval-during ?event '2007-12-01' '2007-12-31')
  FILTER (geo-box-around geoname:Berkeley ?event 5miles)
  }
```

V. SUMMARY AND FUTURE RESEARCH

In this paper we have discussed how RDF can serve as a basis for an event database where events are defined as 'things' that (1) require RDFS++ reasoning because events have types, (2) require geospatial reasoning because events happen somewhere, (3) require temporal reasoning because events nearly always have a start and duration and (4) require some form of social analysis because most interesting events have one or more actors.

We demonstrated how all of these capabilities can be used in one query language, in this case Prolog. And we expect that in the near future these capabilities will be available in SPARQL as well.

The primary research effort for the current version of the query framework is to enhance query-optimization. Notice that in the example shown above, most clauses are not direct matches against the database but functors that do computations. Some of these functors can act both as generators and as filters (as is common in Prolog). In case a functor acts as generator we need to research better statistical predictions for how many solutions can be expected so that we can do better re-ordering of clauses.

VI. REFERENCES

[1] Aasman, J., Unification of geospatial reasoning, temporal logic, & social network analysis in event-based systems, Distributed Event Based Systems (DEBS 2008) http://portal.acm.org/citation.cfm?id=1386007

[2] 1st Scientific American article on the Semantic Web, http://www.sciam.com/article.cfm?articleID=00048144-10D2-1C70-84A9809EC588EF21&ref=sciam

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

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

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

[6] 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

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

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

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

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

[11] Geospatial tutorial section of Franz Inc.'s
AllegroGraph 3.0 documentation,
http://agraph.franz.com/support/documentation/curren
t/geospatial-tutorial.html

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