Spatially-Augmented Knowledgebase

Dave Kolas, Troy Self

BBN Technologies
1300 N. 17th St., Suite 400, Arlington, VA 22209
{dkolas, tself}@bbn.com

Abstract. As an increasing number of applications on the web contain some elements of spatial data, there is a need to efficiently integrate Semantic Web technologies and spatial data processing. This paper describes a prototype system for storing spatial data and Semantic Web data together in a SPatially-AUgmented Knowledgebase (SPAUK) without sacrificing query efficiency. The goals are motivated through use several use cases. The prototype's design and architecture are described, and resulting performance improvements are discussed.

1 Introduction

With the advent of social networking sites, wikis, and other web environments that fall under the umbrella of web collaboration technologies, exposing the data behind web sites in machine-readable formats is becoming ever more popular. Much of the information linked and shared across the Web becomes more useful when combined with its spatial context. Crime statistics, real-estate information, and restaurant reviews are examples of information that is more useful when consumed from a spatial perspective. Using Web 2.0 techniques, web sites commonly referred to as "mash-up" sites are able to display information spatially. For example, one site may overlay crime statistics on a map using Google Maps[1] while another site displays houses for sale on Google Maps. In both cases, the combination of data and capabilities is predefined by the mash-up site and is only used for display purposes.

Semantic Web technologies, such as the Resource Description Framework (RDF) and the SPARQL Protocol and RDF Query Language (SPARQL) are beginning to eliminate this limitation. The graph structure of RDF along with the graph query capabilities of SPARQL make them ideal candidates for representing and searching the ever-changing, interlinked, flexible data of the Web, which is not easily done using a traditional relational database [1].

RDF databases, sometimes called triplestores, offer significant advantages over traditional structured databases for Semantic Web data [2], but are not optimized for spatial information such as geographic coordinates. In this paper, we describe a Spatially AUgmented Knowledgebase (SPAUK) that provides the high-performance graph query capabilities needed for searching webs of data, without sacrificing the spatial indexing and processing capabilities necessary for performing searches involv-

[1] http://maps.google.com

ing spatial extents and operators. Here we describe our motivations and example use cases for the augmented knowledge base as well as the design and implementation results. Finally, we discuss the status of the prototype and future direction.

2 Motivation

While RDF and modern triple stores are efficient at storing and querying data linked across multiple sources of information, they are poor performers when it comes to spatial processing. The current standard for storing spatial data generally involves using an object-relational database augmented with spatial capabilities, such as Oracle Spatial. While this approach has proven effective within a predominantly spatial environment, the object-relational model lacks the flexibility of RDF and triple-stores that make them attractive for searching linked data across multiple sources. The goal of SPAUK here is to provide efficient storage and query of spatial data without sacrificing the flexibility and graph search ability of RDF and triplestores.

2.1 Use Cases

Query Mash-ups
Online communities, specifically social networking sites, have led to a surge in available data about relationships between people. In many cases, a person will own an identity on several sites and provide location information about where they live or work. The graph structure of RDF makes it natural for representing the information distributed across these sites. Combining the graph query capabilities enabled by RDF with efficient spatial processing allows us to search for people based on profile data from multiple online identities, filtered within a particular spatial boundary. As a developer, I may be organizing a working group and wish to find other developers near me with similar interests. A graph search supplemented with spatial information allows me to search for all employees of companies located within 2 miles of my company who are developers on SemWebCentral[2] and have listed their employers on their Facebook[3] account. Given location information about local coffee shops, I can also search for a coffee shop centrally located between us where we can meet. Searches like these require the ability to link and search information from multiple online sources while bounding the query and results within spatial constraints. Without spatial query techniques, graph queries like these may waste time processing all coffee shops, all SemWebCentral users, or all Facebook users before testing the location information to determine if they match the query.

Spatial Annotation
Web sites exist that allow users to submit reviews about all kinds of topics, including movies, books, and restaurants. In the latter case, the geospatial information is impor-

[2] http://www.semwebcentral.org
[3] http://www.facebook.com

tant when it is time to search reviews. A user will most likely only be interested in reviews of restaurants within a particular boundary or near a particular event. A knowledge base that allows efficient storage and search of interlinked data along with geospatial data enables an application that allows users to annotate restaurants on a map, review the restaurant, and provide details about the restaurant by linking to other data or reviewers on the Web. Such an application would allow a search for good restaurants near a particular conference.

2.2 Query Types

In order to support the combination of semantic and spatial data, one must consider several different types of queries. The work [3] of Egenhofer on spatial query languages based on SQL defined three types of queries:

- Queries about spatial properties
- Queries about non-spatial properties
- Queries about both spatial and non-spatial properties

Applying these straightforward concepts to a Semantic Web system yields three analogous query classes:

- Queries about spatial properties
- Queries about ontological properties
- Queries about both spatial and ontological properties

Among the spatial properties that can be queried over, several types of spatial queries have been identified:

- Queries about the spatial properties of an individual
- Queries that relate individuals to a known location (point and range queries)
- Queries that relate individuals to one another (spatial join, nearest neighbor queries)
- Queries that spatially aggregate individuals

We will now explore each of these types of spatial queries individually, as applied to a storage mechanism that also supports ontological data.

The simplest type of spatial queries is queries for the location of a known object: *"Where is the location of Jimmy's Pizza Parlor?"* This type of query is essentially straightforward data retrieval, and does not necessarily require any specialized spatial processing. As such, a semantic system could support these queries without modification.

The second type of spatial queries relates individuals to a known location. This location could be another specific object in the knowledge base, i.e.: *"Which gas stations are within 1 mile of Jimmy's Pizza Parlor?"* or an absolute location, i.e.: *"Which gas stations are within 1 mile of 38°N, 77°W?"*. Naturally, these queries must be crossed with ontological inference as well: *"Which restaurants are within 1 mile of Gus's Gas?"* where 'restaurants' must include entities defined not specifically as restaurants, but those defined as Pizza Parlors, Sub Shops, etc., also.

The third type of spatial queries relates individuals to one another. This class includes both spatial joins and nearest neighbor queries. For example, *"Where can I go to buy bananas, milk, and a drill within a 2 mile radius?"* involves not only the spa-

tial join between the individual places, but also the ontological inference of the types of stores that sell the items in question.

Our spatial semantic knowledge base must be able to support all of these types of queries, and combinations thereof, efficiently.

3 Related Work

Significant research has gone into creating various types of efficient spatial index structures. These index structures are generally used as a supplemental index to an object-relational database. Adding the supplemental indices allows the object-relational databases to significantly increase their performance with respect to spatial queries. The indices are attached to a column or columns defined as a spatial datatype.

A wide variety of useful spatial index structures exist, each with its own positive and negative characteristics. Most fall within a small number of major families, however. These are R-trees [4], quadtrees [5], and grid files [6]. Since we will not be attempting to enhance these index structures in any way, our discussion in this area will focus on which is appropriate to attach to a semantic knowledgebase.

4 Design

The primary goal of SPAUK is to provide efficient spatial processing for spatial semantic systems. We can leverage the significant work that has gone into optimizing database systems for spatial data processing. These systems typically employ a supplementary spatial index to provide efficient spatial queries. As such, we chose to design SPAUK as a semantic knowledgebase capable of supporting supplementary spatial (and other) indices.

A secondary goal was to design a system such that the addition of spatial processing to the system is as transparent as possible to the user. This means that from a client's point of view, all of the data, both the semantic data and the spatial data, is still presented as a graph. To do this, the knowledge base presents itself as a standard SPARQL endpoint. This allows any clients capable of interfacing via the SPARQL protocol to utilize SPAUK.

Thus, the design must present one conceptual graph to its clients, and queries over this graph must be divided appropriately into sub-queries which can be answered by the various parts of the knowledgebase. Spatial parts of the query, including locations and spatial relationships, must be sent to the spatial index and query processor. Non-spatial components of the query must be sent to the underlying triplestore. Results must be combined from the two parts to form a coherent answer. Moreover, data which is inserted must find its way into the appropriate parts of the knowledge base.

4.1 Interface

As noted before, SPAUK's external interface utilizes the SPARQL protocol for query access. However, mapping queries that include spatial instances and relationships to SPARQL is not necessarily straightforward. There are many possible ways that one could use SPARQL for spatial data, and the ideal way has yet been attained [7]. For our prototype, we stayed within the bounds of SPARQL as it is currently defined. While this did not necessarily provide the cleanest possible spatial-semantic query interface, it did allow us to utilize other semantic web software without modification.

In order to do this, we needed to define a set of classes and properties to represent objects, attributes, and relationships that the knowledgebase could understand. Numerous candidate representations already exist. GeoRSS is a good choice for representing spatial extents because it is simple, it already has an RDF syntax, and it is based on the Open Geospatial Consortium's standard for representing spatial extents, Geography Markup Language (GML) [8]. Using another representation, such as the spatial portions of a commonly used upper ontology, could have worked just as well. For the spatial relationships, we decided to start with a set of qualitative topological relationships based on the Region Connection Calculus [9]. First, we look at an example of a Gas Station expressed using these concepts:

```
[] a gas:GasStation;
          gas:name "Gus's Gas";
          gas:brand gas:Exxon;
          gas:numberOfPumps "8";
          georss:where [
            a gml:Point;
            gml:pos "38 -77"
          ].
].
```

The following is an example of the query, "*Which gas stations are within 1 mile of 38°N, 77°W?"* encoded as described.

```
SELECT ?x
WHERE {
          ?x a gas:GasStation;
            georss:where ?y.
          ?y rcc:part [
            a gml:Buffer;
            gml:radius "1";
            gml:bufferGeometry [
              a gml:Point;
              gml:pos "38 -77"
            ].
          ].
}
```

This provides an interface for querying, but does not allow for insertion or deletion of triples. Since this is a necessary for our system and is not yet part of the SPARQL specification, we added HTTP interface methods for both insertion and deletion. These methods merely require a set of RDF triples being posted to the appropriate

URLs. Together with the SPARQL methods, these methods define the entirety of the external interface of SPAUK.

4.2 Architecture

In order to facilitate interoperability and leverage existing semantic web software, the architecture of SPAUK is based on the Jena Semantic Web Framework[4] and Joseki[5]. Utilizing these tools allowed us to focus on the core query-splitting and spatial components of SPAUK.

The basic idea of the architecture is to have a specialized SPAUKGraph implementation of the com.hp.hpl.jena.graph.Graph Java interface that deals with the splitting and combining of the information that goes in and out of the knowledgebase. SPAUKGraph deals directly with some set of IndexProcessors, which represent the interface to the data stored in 1 or more supplemental indices. We address how the Graph handles queries and insertion below.

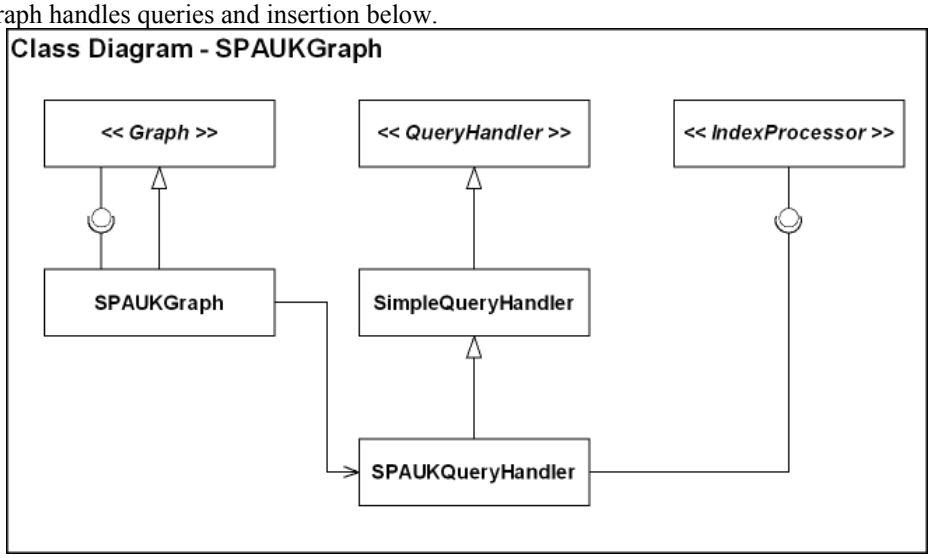

Fig 1. Class diagram for the SPAUKGraph and its relation to the IndexProcessors.

4.1.2 Data Insertion

The underlying triplestore continues to be the master copy of all information in SPAUK. All data inserted is in the form of statements, which are inserted directly into the underlying triplestore.

There is an important dichotomy between the statements in the triplestore and the contents of the supplemental indices. While the data in the tirplestore is a graph, the data that goes in the supplemental indices are sets of discrete objects, i.e. spatial extents. This makes knowing when to insert an object into the supplemental index

[4] http://jena.sourceforge.net/
[5] http://www.joseki.org/

somewhat tricky. The insertion interface sees the statements being added one at a time, and must combine sets of them to form objects to be inserted. We accomplished this through the use of Jena's InfGraph. For each type of object that the system must watch for, a rule is added to an InfGraph layer above the underlying triplestore. The head of the rule is a function that connects to the appropriate IndexProcessor to add an object to the index. The rule will not fire until all required components are available. This rule, for example, adds points to the spatial index processor when they are inserted:

```
[point:
   (?x rdf:type gml:Point)(?x gml:pos ?pos) ->
         point(?x, ?pos)
]
```

This scheme allows for the insertion of objects into the indices without concerning us with the transactionality of the data store. In fact, if part of a geometry definition is inserted at some point, and then much later the rest of the definition is inserted, the geometry will be indexed successfully at the later point. However, it does not account for updates or deletions of statements corresponding to indexed objects. As such, indexed objects are treated as immutable within the system. If the location of a restaurant changes, rather than changing the properties of the location object to which the restaurant is attached, the restaurant must be severed from the location and a new location created.

4.1.1 Querying

Querying the combined data storage is the most complicated part of the system. The appropriate parts of the query must be partitioned among the underlying triplestore and the supplemental indices, depending upon which parts are capable of most efficiently answering each piece of the query.

When the SPAUKGraph receives the query, it first splits the query based on the namespaces of the associated with the attached IndexProcessors. For each triple, the namespace of the predicate or the Class (for rdf:type statements) is matched against the namespaces associated with the IndexProcessors. If a namespace is not associateed with an IndexProcessor, it defaults to association with the underlying triplestore. For instance, in the following query, the portion that must be processed by the spatial index processor has been italicized:

```
SELECT ?x
WHERE{
          ?x a gas:GasStation;
            georss:where ?y.
          ?y rcc:part [
            a gml:Buffer;
            gml:radius "1";
            gml:bufferGeometry [
              a gml:Point;
              gml:pos "38 -77"
            ].
          ].
```

}

Unfortunately, this assigns artificial meaning to the namespaces of which the query processor is aware. This could lead to errors in processing if the users attempted to extend the spatial ontologies in a way that the system did not understand. Since no better way to divide the statements at the query level has yet come to light, this is the method that the SPAUK prototype uses.

Once the query has been divided into appropriate parts, the SPAUKGraph must make a best-effort attempt to determine which part of the query is the most selective. Since it does not have any information about the selectivity of the query parts directly, it must ask the underlying triplestore and the IndexProcessors to approximate the selectivity of their parts of the query as an estimated number of results. Developing an appropriate cost model for this is an area of future work. In the current implementation, if the spatial query processor receives a query for objects within a specified area, it returns the highest possible selectivity and thus is chosen first. In all other cases, the system defaults to allowing the triplestore to bind first. Other possibilities include attempting to execute the different parts in parallel, however this was beyond the scope of our prototype.

An initial subquery is chosen and then executed by either the underlying triplestore or an IndexProcessor as appropriate. The bindings from this subquery are then applied to the other subqueries as they are executed. Since the linkage objects between the spatial and non-spatial portions are bound at this point, it is expected that the selectivity of the remaining bound subqueries should be extremely high.

Though SPAUK supports SPARQL, the SPARQL support is provided exclusively by the ARQ component of Jena. Thus our system deals only with queries in the form of simple graph patterns. This drastically reduces the amount of query processing work that needs to be done; however, there are cases where this design creates SPARQL queries that could not be properly optimized using the index. This could happen if the definition of the range in a range query were split over an OPTIONAL clause. Since these cases are primarily connected to poor query construction, we currently ignore them in the design.

4.3 Indices

The spatial index used in the prototype was a simple in-memory gridfile. This is not a particularly sophisticated spatial index; however, the software was designed such that substituting another indexing mechanism should be straightforward.

Ideally, we would like to add either a quadtree or R-tree indexing mechanism to SPAUK. Having both available to spatial applications is ideal, since both have strengths and weaknesses depending on the distribution of the data being stored. Particularly, quadtrees function better than R-trees when data is more evenly spatially distributed, and R-trees function better when data is more spatially clustered. Since there are applications which could potentially make use of both types of indices, the option of both should exist. This is analogous to the spatial index support provided in common spatial relational databases such as Oracle Spatial 10*g*.

5 Results

The SPAUK system was successfully implemented for a subset of the desired problem. Processing was implemented for two major types of spatial geometries: Points and Polygons defined by an exterior linear ring. Rules were created to detect these geometries and insert them into the index. Two spatial relationships were implemented over these polygons, *connected* and *part*. These allowed us to sufficiently test the query splitting mechanism.

Unfortunately, without the creation of spatial semantic benchmarks, we do not yet have a way to empirically test the performance of the SPAUK system. However, consideration of the prior art in the object-relational database realm and a careful look at the index structures demonstrates that the technique is superior.

Consider attempting to build a system for spatial semantic data without any spatial indexing. A query for all restaurants that are in a 2 mile radius from a given point would clearly be *O(n)* in the number of restaurants, since the system would have to compare each and every restaurant's location to the spatial buffer area. However, if a quadtree was used for spatial indexing, we would expect the time to find objects in the radius to be logarithmic.

6 Conclusion

While we have not yet done formal analysis of the performance improvement caused by using a supplemental spatial index, examples of the technique in the object-relational database world, simple analysis of the algorithms involved, and preliminary usage of the SPAUK system have shown that the approach is indeed valid. Attaching a semantic GIS client to the SPAUK system provides responsive spatial semantic query capability. We believe that this type of system enables a new class of semantic applications whose full potential cannot yet be conceived. Waldo Tobler's "first law of geography" states, "Everything is related to everything else, but near things are more related than distant things." [10] Since a goal of the Semantic Web is to maximize the meaning of relationships, spatial information processing cannot be ignored.

7 Future Work

The first major piece of future work for the SPAUK system will be to fully implement the GeoRSS geometry types and the RCC8 spatial relations. This will provide a fully usable system for experimentation with spatial semantic data storage, and hopefully provide others with a method of building spatial semantic applications when it soon becomes open source.

The second piece of future work involves significantly more formal performance testing. However, this will require several other advancements. First, a benchmark for spatial semantic data must be created. This could very well be an enhancement of the Lehigh University Benchmark (LUBM) [11]. Secondly, SPAUK would need to

be attached to a more robust spatial index, such as a persistent R-tree. With these modifications in place, SPAUK will be formally compared to a semantic spatial system in which spatial calculations are performed only as function calls in rules.

Finally, extending the SPAUK implementation with a temporal index or other indices is very desirable. The architecture is built not just for one supplemental index, but for many; hopefully it can provide benefit for a wide variety of application areas.

Acknowledgements We wish to thank Mike Dean and BBN Technologies for their support of this effort and helpful comments on the paper.

References

1. Connected Services Framework 3.0 Developers Guide. Microsoft (2006)
2. Lassila, O., Hendler, J.: Embracing "Web 3.0". Internet Computing, IEEE **11** (2007) 90-93
3. Egenhofer, M.: Spatial SQL: A Query and Presentation Language. IEEE Transactions on Knowledge and Data Engineering **6** (1994) 86-95
4. Guttman, A.: R-trees: a dynamic index structure for spatial searching. ACM Press New York, NY, USA (1984)
5. Finkel, R.A., Bentley, J.L.: Quad trees a data structure for retrieval on composite keys. Vol. 4. Springer (1974) 1-9
6. Nievergelt, J., Hinterberger, H., Sevcik, K.C.: The Grid File: An Adaptable, Symmetric Multikey File Structure. Vol. 9. ACM Press New York, NY, USA (1984) 38-71
7. Kolas, D.: Supporting Spatial Semantics with SPARQL. (2007)
8. ISO/TC 211/WG 4/PT 19136 Geographic information - Geography Markup Language (GML). (2004)
9. Cohn, A.G., Bennett, B., Gooday, J., Goss, N.M.: Qualitative Spatial Representation and Reasoning with the Region Connection Calculus. GeoInformatica **1** (1997) 275-316
10. Tobler, W.R.: A Computer Movie Simulating Urban Growth in the Detroit Region. Vol. 46. JSTOR (1970) 234-240
11. Guo, Y., Pan, Z., Heflin, J.: LUBM: A Benchmark for OWL Knowledge Base Systems. Journal of Web Semantics **3** (2005) 158-182