Undefined 0 (0) 1
IOS Press

GeoSPARQL: Enabling a Geospatial Semantic Web

Robert Battle, Dave Kolas
Knowledge Engineering Group, Raytheon BBN Technologies
1300 N 17th Street, Suite 400, Arlington, VA 22209,
USA
E-mail: {rbattle,dkolas}@bbn.com

Abstract. As the amount of Linked Open Data on the web increases, so does the amount of data with an inherent spatial context. Without spatial reasoning, however, the value of this spatial context is limited. Over the past decade there have been several vocabularies and query languages that attempt to exploit this knowledge and enable spatial reasoning. These attempts provide varying levels of support for fundamental geospatial concepts. In this paper we look at the overall state of geospatial data in the Semantic Web, with a focus on the upcoming OGC standard GeoSPARQL. GeoSPARQL attempts to unify data access for the geospatial Semantic Web. We describe the motivation for GeoSPARQL, the current state of the art in industry and research, an example use case, and the implementation of GeoSPARQL in the Parliament triple store.

Keywords: GeoSPARQL, SPARQL, RDF, geospatial, geospatial data, query language, geospatial query language, geospatial index

1. Introduction

Geospatial data is increasingly being made available on the Web in the form of datasets described using the Resource Description Framework (RDF). The principles of Linked Open Data, detailed in [6], encourage a set of best practices for publishing and connecting structured data on the Web. Linked Open Data promotes the use of the SPARQL Protocol and RDF Query Language (SPARQL) and RDF to query and model data. While this is useful for querying for relationships that are explicitly represented in data, implicit relationships, such as geospatial relationships, cannot easily be queried. For instance, datasets may exist that describe monuments and parks, but being able to link these datasets based on their undeclared relationships is difficult. The ability to answer a meaningful query, like "What parks are within 3km of the Washington Monument?", depends on how the data is represented, whether all of the resources are related to the Washington Monument, and if that relationship is explicit.

In this paper, we discuss an emerging standard, GeoSPARQL[24] from the Open Geospatial Consortium (OGC). This standard aims to address the issues of geospatial data representation and access. It provides a common representation of geospatial data described using the RDF, and the ability to query and filter on the relationships between geospatial entities. First, we introduce some geospatial concepts that are critical to understanding some of the design choices for GeoSPARQL. We then describe topological relationships that are important to understand when designing a language for querying between spatial entities. Next, we describe the motivation for GeoSPARQL, the current state of the art in modeling and querying geospatial data in the Semantic Web, and we introduce GeoSPARQL. A discussion of the Parliament[1] triple store, its GeoSPARQL spatial index, and a use-case that illustrates a simple example of data integration with GeoSPARQL follows.

---

[1] `http://parliament.semwebcentral.org`

Dave Kolas is a co-chair for the GeoSPARQL Standards Working Group, and Robert Battle has worked on the Parliament implementation and provided feedback to the development of the standard.

2. Geospatial Concepts

Some basic understanding of geospatial concepts is required for discussion of GeoSPARQL. The following sections define some of the terms used throughout the rest of this paper.

2.1. Features and Geometries

Features and geometries are two fundamental concepts of geospatial science. A feature is simply any entity in the real world with some spatial location. This could be a park, an airport, a monument, a restaurant, etc. A feature can have a spatial location that cannot be precisely defined, such as a swamp or a mountain range. A geometry is any geometric shape, such as a point, polygon, or line, and is used as a representation of a feature's spatial location. For instance, Reagan National Airport is a geospatial feature because it is an entity that has a specific location in the world. It has a geometry that is a point with coordinates 38.852222, -77.037778 (in the WGS84[2] datum). Geometries can be measured at varying resolutions, from a simple point in the center of a feature to a complex, precise measurement of a feature's entire border. Spatial data typically separates features from geometries, although that is not always the case.

2.2. Coordinate Reference Systems

An important part of the metadata associated with a geometry is its coordinate reference system (CRS) (alternatively known as its spatial reference system). The elements of a coordinate reference system provide context for the coordinates that define a geometry in order to accurately describe their position and establish relationships between sets of coordinates. There are four parts that make up a CRS: a coordinate system, an ellipsoid, a datum, and a projection.

A coordinate system describes a location relative to some center. A geocentric coordinate system places the center at the center of the Earth and uses standard X, Y, Z ordinates. A geographic (or geodetic) coordinate system uses a spherical surface to determine locations. In such a system, a point is defined by angles measured from the center of the Earth to a point on the surface. These are also known as latitudes (horizontal) and longitudes (vertical). A Cartesian coordinate system is a flat coordinate system on the surface. It enables quick and accurate measurements over small distances and is useful for applications such as surveying.

An ellipsoid defines an approximation for the center and shape of the Earth. A datum defines the position of an ellipsoid relative to the center of the earth. This provides a frame of reference for measuring locations and, for local datums, allows for accurate locations to be defined for the valid area of the datum. WGS84 is a datum that is widely used by GPS devices that approximates the entire world. Geographic coordinate systems use an Earth based datum that transforms an ellipsoid into a representation of the Earth.

In order to create a map of the Earth, it must be projected from a curved surface onto the plane. This projection will distort the surface in some fashion which will mean that the coordinates for some locations are are more accurate than those in another. Some projections will preserve area, so the size of all objects is relative, while others preserve angles, and others try to do both. A coordinate system projected onto a plane enables faster performance, as Cartesian calculations require fewer resources than Spherical calculations. Computations across the plane, however, are inaccurate when they deal with large areas as the curvature of the Earth is not taken into account.

The combination of these elements defines a CRS. One common source of well defined coordinate reference systems is the European Petroleum Survey Group (EPSG)[3].

2.3. Topological Relationships

All spatial entities are inherently related to some other spatial entity. Whether two entities intersect somehow or are thousands of miles apart, the relationship that they share can be described and evaluated.

In [28], Randell et al. describe an interval logic for reasoning about space using a simple ontology that defines functions and relations for expressing and reasoning over spatial regions. This logic is referred to as Region Connection Calculus (RCC). A subset of RCC,

---

[2] `http://en.wikipedia.org/wiki/World_Geodetic_System`

[3] `http://www.epsg-registry.org`

Table 1
Simple Features, Egenhofer and RCC8 relations equivalence

Simple Features	**Egenhofer**	**RCC8**
equals	equal	EQ
disjoint	disjoint	DC
intersects	¬disjoint	¬DC
touches	meet	EC
within	inside + coveredBy	NTPP + TPP
contains	contains + covers	NTPPi + TPPi
overlaps	overlap	PO

RCC8, defines eight mutually exhaustive pairwise disjoint relations which can be used to imply the rest of the relations in RCC. These eight base relations are:

1. DC(x, y) (x is disconnected from y)
2. x = y (x is identical with y)
3. PO(x,y) (x partially overlaps y)
4. EC(x,y) (x is externally connected with y)
5. TPP(x,y) (x is a tangential proper part of y)
6. NTPP(x,y) (x is a non-tangential proper part of y)
7. TPPi(x,y) (y is a tangential proper part of x)
8. NTPPi(x,y) (y is a non-tangential proper part of x)

The same set of eight geospatial topological relations is described with different names by Egenhofer in [8], and includes the capacity to describe the relationship between different dimensioned geometries. This model was later generalized in the Nine Intersection Model [11]. The Open Geospatial Consortium Simple Feature Access Common Architecture specification [25] uses the Nine Intersection Model introduced by Egenhofer to describe spatial relations for use in geographic access systems. Table 1 illustrates the equivalence between all of these spatial relations.

3. Motivation for GeoSPARQL

The Open Geospatial Consortium is a non-profit standards organization focused on geospatial data. The OGC is composed of members of industry, academic institutions, and government organizations. By standardizing GeoSPARQL within the OGC, we seek to leverage the experience of its members to ensure that geospatial Semantic Web data is represented in a consistent logical way. With the input and acceptance of all of both knowledge base vendors and data producers and consumers, GeoSPARQL has the potential to unify geospatial RDF data access.

Geospatial reasoning is critical in a large number of application domains (emergency response, transportation planning, hydrology, land use, etc.). Users in these domains have long utilized relational databases with spatial extensions [9]. These spatially extended databases have given the combination of efficient, stable storage and retrieval of data with geospatial calculation and indexing. This allows questions like "Which students live within 2km of the school they attend?" to be answered efficiently.

Within the last decade, RDF storage solutions have become increasingly popular. These knowledge bases, sometimes called triple stores, are capable of better handling several types of problems at which relational databases struggle or are not intended to perform: queries with many joins across entities [32], queries with variable properties [32], and ontological inference on datasets. These features lend themselves towards problems that involve data exploration, linkage across datasets, and abstraction from low level data.

Because of RDF stores' ability to do inference and easily link data sets, they have been of growing interest to the geospatial data community. Often geospatial domains have complicated type hierarchies which cannot be fully expressed in current geospatial information systems. For instance, a river is both a waterway and a transportation route. Also, geospatial domain problems often require marrying multiple data sources together to solve a particular problem. In emergency response scenarios, population data, transportation data, and even realtime police and fire data must be combined to deliver a timely result. Combining data sources on the web is useful to consumers as well; geospatial data about points of interest combined with hotel information and travel route information could lead to significantly more sophisticated travel planning than currently exists.

As such, it was inevitable that those people interested in expressing geospatial data on the Semantic Web would want to combine the spatial indexing and calculation of spatial databases with the inferential power and data linkage of RDF triple stores [16]. This has been done by many groups in varying ways for varying purposes, as will be discussed later.

To provide geospatial reasoning and querying in a triple store, the implementors must define both an ontology for representing spatial objects and query predicates for retrieving these spatial objects. However, each organization that has attempted this task has approached it in a slightly different way. As a result, spatial RDF data that would be properly indexed and queryable in one implementation would simply be treated as plain RDF data in another implementation.

This is the primary problem which the standardization of GeoSPARQL attempts to solve. The GeoSPARQL language defines both a small ontology for representing features and geometries and a number of SPARQL query predicates and functions. All of these are derived from other OGC standards so that they are well grounded and understood. Using the new standard should ensure two things: (1) if a data provider uses the spatial ontology in combination with an ontology of their domain, that data can be properly indexed and queried in spatial RDF stores; and (2) compliant RDF triple stores should be able to properly process the majority of spatial RDF data.

Aside from providing the ability to perform spatial queries, the small ontology portion of the GeoSPARQL specification is intended to provide an interchange format for geospatial data in a wide variety of use cases. The ontology is meant to be attached to other ontologies for various domains, providing only the bare spatial aspects. It is intended to be simple enough to cover the most light-weight uses, and scale in complexity for complex use cases. GeoSPARQL goes a long way to solving one of the research issues posed by Egenhofer in [10]. Namely that "we need a plausible canonical form in which to pose geospatial data queries".

GeoSPARQL is intended to inter-operate with both quantitative and qualitative spatial reasoning systems. A quantitative spatial reasoning system involves concrete geometries for features. With these concrete geometries present, distances and topological relations can be explicitly calculated. Qualitative geospatial reasoning systems allow RCC type topological inferences for features where the geometries are either unknown or cannot be made concrete [13]. For example, if there are assertions that a monument is inside a park, and the park is inside a city, a qualitative reasoning system should be able to infer that the monument is in the city through transitivity. Hybrid versions of these systems are also possible, where some features have concrete geometries and others have abstract geometries. By sharing a set of terms for topological relations, GeoSPARQL allows conclusions from quantitative applications to be used by qualitative systems and a single query language for both types of reasoning.

4. Geospatial RDF: State of the Art and Related Work

Over the past decade, there have been many different attempts to create a geospatial RDF standard. Several different organizations, including the W3C, research groups, and triple store vendors have created their own ontologies and strategies for representing and querying geospatial data.

In 2003, a W3C Semantic Web Interest Group created the Basic Geo Vocabulary [31] which provided a way to represent WGS84 points in RDF. This work was extended from 2005 through 2007 by the W3C Geospatial Incubator Group [21] to follow the GeoRSS [23] feature model to allow for the description of points, lines, rectangles, and polygon geometries and their associated features. This group produced the GeoOWL ontology[4] which provides a detailed and flexible model for representing geospatial concepts. These ontologies were produced as products from their respective groups within the W3C as the result of collaborations across university and industry partners.

There are published datasets that use the W3C vocabularies [4] and a variety of triple stores that support data represented by these ontologies [17,22]. However, the associated working groups never moved beyond the incubator state and the respective ontologies never became official W3C recommendations. These ontologies also only work with data in the WGS84 datum. In order to be valid, data in any other CRS must be projected which can lead to inaccuracy in the data.

Support for spatial data in triple stores is mixed. Several vendors support spatial data, but not all vendors support the same representation of data or share the same support for relational queries. Some triple stores use the aforementioned W3C ontologies, while others have invented their own.

Parliament, the high performance [18] triple store from Raytheon BBN Technologies, provided the first geospatial index for semantic web data in 2007[16,17]. This index supports data in the GeoOWL ontology and introduces ontologies for querying spatial data via RDF properties that correspond to the RCC spatial relations and OGC Simple Features relations. However, data in the W3C Basic Geo vocabulary is unsupported.

---

[4] `http://www.w3.org/2005/Incubator/geo/XGR-geo-20071023/W3C_XGR_Geo_files/geo_2007.owl`

Ontotext's OWLIM-SE[5] triple store can index point data represented in the W3C Basic Geo Vocabulary [22]. The only spatial relationship that can be queried is whether a point is contained within a circle or polygon. The ability to query for relationships between higher order geometries such as lines and polygons is not supported.

OpenLink Virtuoso[6] also has support for the W3C Basic Geo Vocabulary. A SPARQL function is provided to convert a pair of latitude and longitude property values into a point geometry. A special literal datatype, `virtrdf:Geometry`, is also provided for indexing point literals. Support for testing intersection and containment relationships is provided via property functions. Through a combination of these relations and their negations, most of the Egenhofer relations can be tested. However, some relations, such as testing for overlap, cannot be supported in Virtuoso.

Other triple stores take a completely different approach. As described in [12], Franz AllegroGraph[7] defines a custom datatype to represent geospatial data and map it to a "strip" of space in the index that contains the data. A modified SPARQL syntax provides a new `GEO` operator where the geospatial aspect of the query is defined.

OpenSahara[8] provides a service for adding external indexing and geospatial querying capabilities to any triple store with a Sesame[9] Sail layer. The implentation is a wrapper for the PostgreSQL[10] database with PostGIS spatial extensions[11] and as such, OpenSahara supports all of the geometries and relations defined in the OGC Simple Features Access [25]. Literal datatypes are introduced for Well-Known Text (WKT), Well Known-Binary (WKB), and their compressed forms while the spatial relations that PostGIS supports are implemented as SPARQL filter functions.

In addition to vendor supported options, there has been significant community and research interest in-representing and querying geospatial data in the last few years. Perry proposed an extension to SPARQL, SPARQL-ST in [26]. This introduces a modified SPARQL syntax for posing spatial queries to data that is modeled in an upper ontology based on GeoRSS. A focus on describing data and metadata such as the CRS for geometries allows SPARQL-ST to operate with data of different system which is something that is lacking from many vocabularies such as GeoOWL and the Basic Geo vocabulary. This increased flexibility with data is hindered, however, by the proposed query syntax which deviates from the standard SPARQL language. Any data that is in the SPARQL-ST format can only be accessed by a SPARQL-ST system.

Taking a simpler approach, Zhai et al., in [34] and [33], discuss the need for adding topological predicates to SPARQL. The OGC Simple Features relations and a subset of geometries are used as the basis for their ontologies. Unfortunately, the relations have to be specifically encoded in RDF and the data cannot support multiple coordinate reference systems.

Another approach described by Koubarakis and Kyzirakos in [19] is based on research from the constraint database community. Constraint databases are a promising technology for integrating spatial data [5]. The authors propose to enrich the Semantic Web with spatial and temporal data by extending RDF and SPARQL with constraints. The proposed extension to RDF, stRDF, uses typed literals to describe a semi-linear point set. The query language, stSPARQL, extends SPARQL to includes additional operators for querying RCC relationships and introduces a new syntax for specifying spatial variables. Support for multiple coordinate reference systems is not discussed. Compatibility with existing data is also a problem as not all data is represented as a semi-linear point set. stSPARQL is intentionally not compatible with OGC standards as the authors believe that the semi-linear point set can be used to describe different geometries without forcing a hierarchy of datatypes on users. In [20], stSPARQL and stRDF are described with an additional context of GIS applications. The authors contrast stRDF and stSPARQL with prior work (such as GeoOWL) and note that it is hard to find information about geometries when you do not what type of geometries will be in the answer set and what properties to ask about. This is a problem with representations like GeoOWL where different geometries have different properties. The semi-linear point set avoids this by encapsulating the representation into a single datatype.

The NeoGeo Vocabulary [29] is a vocabulary that arose from the NeoGeo community[12], the Linked Data community, and several universities. They recognized

---

[5] `http://www.ontotext.com/owlim/editions`

[6] `http://www.openlinksw.com`

[7] `http://www.franz.com/agraph/allegrograph`

[8] `http://www.opensahara.com`

[9] `http://www.openrdf.org`

[10] `http://www.postgresql.org`

[11] `http://postgis.refractions.net`

[12] `http://sites.google.com/site/neogswvocs/`

the need for a well formed standard representation for geospatial data and provided one that is based on the Geography Markup Language (GML) Simple Features Profile[13] . To describe spatial relations, an ontology based on RCC8 is also provided. A limitation of the NeoGeo approach is the need to represent each coordinate as a resource. In particular, polygons and lines are represented with an RDF collection of Basic Geo points. While this does allow points to be shared across geometries, it significantly increases the verbosity of the data. Unfortunately this extra verbosity does not result in particular gains in expressive power, since each point in a polygon provides little value in isolation and RDF list contents are difficult to query for in SPARQL. Moreover this prevents geometry literals from being compared easily in SPARQL filter functions.

In [15], Hu and Du compose a three level hierarchical spatiotemporal model: a meta level for abstract space-time knowledge, a schema level for well-known models in spatial and temporal reasoning (e.g., RCC, Allen time[1]), and an instantiations level that provides mappings and formal descriptions of the various ground spatiotemporal statements in the Linked Data clouds. This meta approach provides a convenient way to abstract out spatial knowledge from its underlying representation. Mappings, however, have to be defined for each dataset at the instantiations level.

A complete implementation of integrating spatial data and queries into an RDF triple store is described by Brodt et al. in [7]. By typing WKT string representations of geometry literals with a spatial datatype, the triple store is able to efficiently store and query data. Once again, the OGC Simple Features relations are used as the basis for posing queries for the spatial relation. These relations are mapped to SPARQL filter functions which allow for direct comparison between spatial literals. This implementation is similar to the approach taken by Parliament and OpenSahara in the way that the SPARQL language itself does not need to be modified in order to query for the spatial relations between entities.

5. Introduction to GeoSPARQL

As illustrated above, many groups have created ontologies and query predicates to make indexing and query of geospatial data possible. However, since there are many of these and they all differ slightly, data that can be spatially queried in one knowledge base may not be able to be spatially queried in another. GeoSPARQL provides a standard for geospatial RDF data insertion and query, which covers the use cases of the other previous approaches.

The GeoSPARQL specification attempts to enable a wide range of geospatial query use cases, from simple points of interest knowledge bases to detailed authoritative geospatial data sources for transportation. Moreover, use of GeoSPARQL for both of these types of data should enable the data sets to be easily used together. In order to achieve this goal, different conformance classes are provided. This means that a simple knowledge base implementation intended for simple use cases need not implement all of the more advanced reasoning capabilities of GeoSPARQL, such as quantitative reasoning or query rewriting.

There are also several sets of terminology for the topological relationships between geometries. Rather than mandate that all implementations use the same set of terminology, each implementation can choose which sets of terms to support. This is discussed further in the section on GeoSPARQL relationships.

GeoSPARQL attempts to address the problems with the disparate and incompatible implementations for representing and querying spatial data. It achieves this by defining an ontology that closely follows the existing standards work from the OGC with regard to spatial indexing in relational databases.

The GeoSPARQL specification contains three main components:

1. The definition of a vocabulary to represent features, geometries, and their relationships
2. A set of domain-specific, spatial functions for use in SPARQL queries
3. A set of query transformation rules

5.1. GeoSPARQL Ontology

The ontology for representing features and geometries is fundamental to being able to build and query spatial data. The ontology is based on the OGC's Simple Features model, with some adaptations for RDF. Note that prefix definitions are omitted from examples for clarity; a definition of all of the prefixes used is at the end of the paper in listing 14. The ontology includes a class `geo:SpatialObject`, with two primary subclasses, `geo:Feature` and `geo:Geometry`. These classes are meant to be connected

---

[13] `http://www.ogcnetwork.net/gml-sf`

to an ontology representing a domain of interest. Features can connect to their geometries via the `geo:hasGeometry` property.

For example, an airport is an `geo:Feature`. It is a conceptual thing that exists in the real world in a particular place. In the real world, it has a geometry that corresponds to the coordinates of all of the points along the border of the airport area. This real-world geometry has to be measured and estimated in some way. It is possible to do this measurement at various resolutions, each of which may serve well for different purposes. A representation of the real world geometry which has been measured becomes an `geo:Geometry`. Thus the airport may have several `geo:Geometry`s, ranging from a single point in the center of the airport to an extremely detailed polygon that closely follows the airport's outside border. A geometry that will function for most purposes within a dataset can be specified as the `geo:defaultGeometry`.

GeoSPARQL includes two different ways to represent geometry literals and their associated type hierarchies: WKT and GML. An implementor of a spatial triple store may choose to support either or both of these representations. GeoSPARQL provides different OWL classes for the geometry hierarchies associated with both of these geometry representations. This provides classes for many different geometry types such as point, polygon, curve, arc, and multicurve. The `geo:asWKT` and `geo:asGML` properties link the geometry entities to the geometry literal representations. Values for these properties use the `geo-sf:WKTLiteral` and `geo-gml:GMLLiteral` data types respectively.

5.2. *GeoSPARQL Relationships*

GeoSPARQL also includes a standard way to ask for topological relationships, such as overlaps, between spatial entities. These come in the form of binary properties between the entities and geospatial filter functions.

The topological binary properties can be used in SPARQL query triple patterns like a normal property. Primarily they are used between objects of the Geometry type. However, they can also be used between Features, or between Features and Geometries, if GeoSPARQL's query rewrite rules are supported (discussed in the next section). The properties can be expressed using three distinct vocabularies: the OGC's Simple Features, Egenhofer's 9-intersection model, and RCC8. Which of these vocabularies is supported can be dependent on the triple store implementation, though it is likely that implementations will support all three. The Simple Features topological relations include equals, disjoint, intersects, touches, within, contains, overlaps, and crosses.

The filter functions provide two different types of functionality. First, there are operator functions which take multiple geometries as predicates and produce either a new geometry or another datatype as a result. An example of this is the function `ogcf:intersection`. This function takes two geometries and returns a geometry that is their spatial intersection. Other functions like `ogcf:distance` produce an `xsd:double` as a result. The second type of functionality is boolean topological tests of geometries. These come in the same three vocabulary sets as the topological binary properties: simple features topological relations, Egenhofer relations, and RCC8 relations. These functions are partially redundant with the topological binary properties; however, the topology functions take the geometry literals as parameters, while the binary properties relate Geometry and Feature entities. This means that quantitative and qualitative applications can both make use of the binary properties, but only quantitative applications can make use of the topology functions. Also, comparisons to concrete geometries provided in the query can only be made via the functions. An example of the topological functions is `ogcf:intersects`, which returns true if two geometries intersect.

5.3. *Query Transformation Rules*

The query rewrite rules allow for an additional layer of abstraction in SPARQL queries. While only concrete Geometry entities can be quantitatively compared, it nonetheless sometimes makes sense to discuss whether two features have a particular topological relationship. This is represented in the natural language question, "Is Reagan National Airport within Washington, DC?". Although Reagan National is referred to as a Washington DC airport, it is actually across the Potomac river in Virginia. In GeoSPARQL, Feature to Feature and Feature to Geometry topological relations are achieved by the combination of the use of the `geo:defaultGeometry` property and the query rewrite rules. If a feature is used as the subject or object of a topological relation, the query is automatically rewritten to compare the Geometry linked as a default, thus removing the abstraction for processing.

Listing 1 shows a query with a relationship between Feature objects before and after rewrite.

Listing 1: Query Rewrite Example

```
#Before
ASK {
  ex:DCA a geo:Feature;
     geo:within ex:WashingtonDC .
  ex:WashingtonDC a geo:Feature .
}

#After
ASK {
  ex:DCA a geo:Feature;
     geo:defaultGeometry ?g1 .
  ex:WashingtonDC a geo:Feature ;
     geo:defaultGeometry ?g2 .
  ?g1 geo:within ?g2 .
}
```

The goal of this feature is to provide a more intuitive approach to geospatial querying for use cases that do not require many different geometries, while still maintaining a concrete definition of this intuitive understanding. Compliant GeoSPARQL triple stores are not required to implement this feature.

5.4. *Using GeoSPARQL*

Consider an example using a points of interest ontology in listing 2. We seek to represent points of interest of various types (Monuments, Parks, Restaurants, Museums, etc.). These types of landmarks are represented in a class hierarchy with a `ex:PointOfInterest` class at the top. These classes of course may include many non-spatial attributes, but only a label is included here. All that is required to link this ontology with GeoSPARQL, and thus give its classes a geospatial reference, is to make `ex:PointOfInterest` a subclass of `geo:Feature`.

If compliance with WKT is chosen, and latitudes and longitudes are expressed in WGS84 datum in a longitude latitude order (CRS:84), a record for the Washington Monument would look like listing 3. If a coordinate reference system other than CRS:84 is desired, that can be included within the WKTLiteral. Listing 4 expresses the same point in WGS84 with latitude longitude order.

Listing 2: Example Ontology

```
ex:Restaurant a owl:Class;
   rdfs:subClassOf ex:Service .
ex:Park a owl:Class;
   rdfs:subClassOf ex:Attraction .
ex:Museum a owl:Class;
   rdfs:subClassOf ex:Attraction .
ex:Monument a owl:Class;
   rdfs:subClassOf ex:Attraction .
ex:Service a owl:Class;
   rdfs:subClassOf ex:
      PointOfInterest .
ex:Attraction a owl:Class;
   rdfs:subClassOf ex:
      PointOfInterest .
ex:PointOfInterest a owl:Class;
   rdfs:subClassOf geo:Feature .
```

Listing 3: Washington Monument

```
ex:WashingtonMonument a ex:Monument;
   rdfs:label "Washington Monument";
   geo:hasGeometry ex:WMPoint .
ex:WMPoint a geo:Point;
  geo:asWKT "POINT(-77.03524
     38.889468)"^^geo-sf:WKTLiteral
     .
```

Listing 4: Point in WGS84 datum

```
"<http://www.opengis.net/def/crs/
   EPSG/0/4326> POINT(38.889468
   -77.03524)^^"geo-sf:WKTLiteral
```

While this representation may seem verbose, and the literal string is no longer standard WKT, it has the advantage of encoding the CRS information directly into the literal. This is all of the data needed to define the Geometry; without the CRS, another property would need to be added onto the Geometry which would increase storage requirements and make sharing data more cumbersome.

Now we will look at a few example GeoSPARQL queries using this data. One potential query over this dataset would be, "Which monuments are contained within a park?" This query requires a topological com-

Listing 5: Example Query 1

```
SELECT ?m ?p
WHERE {
  ?m a ex:Monument ;
      geo:hasGeometry ?mgeo .
  ?p a ex:Park ;
     geo:hasGeometry ?pgeo .
  ?mgeo geo:within ?pgeo .
}
```

Listing 6: Example Query 2

```
SELECT ?m ?p
WHERE {
  ?p a ex:Park .
  ?m a ex:Monument ;
     geo:within ?p .
}
```

parison between the geometries of the monuments and the geometries of parks. We show it in listing 5 using the binary topology property `geo:within`. The two entities have type statements, `geo:hasGeometry` properties to tie them to their geometries, and then the `geo:within` function to tie them together.

If the knowledge base being used supported the query rewriting rules, and the data set included default geometries, the first query could be rewritten even more simply using a feature-to-feature topological relationship. This method is demonstrated in listing 6.

Spatial user interfaces often need to look for entities of a particular type that fall within an explicit bounding box. Consider the query, "What attractions are within the bounding box defined by (-77.089005, 38.913574) and (-77.029953, 38.886321)?" Because we need to specify an explicit geometry in the query, we need to compare to it using the topological filter functions as opposed to the binary properties. We have the attraction entity and its attached geometry, and the geometry is compared with the filter function `geof:within`. This query is shown in listing 7. Note that the bounding box is expressed as a Polygon.

Listing 7: Example Query 3

```
SELECT ?a
WHERE {
  ?a a ex:Attraction;
      geo:hasGeometry ?ageo .
  FILTER(geof:within(?ageo,
      "POLYGON((
-77.089005 38.913574,
-77.029953 38.913574,
-77.029953 38.886321,
-77.089005 38.886321,
-77.089005 38.913574
))"^^geo-sf:WKTLiteral))
}
```

Listing 8: Example Query 4

```
SELECT ?p
WHERE {
 ?p a ex:Park ;
  geo:hasGeometry ?pgeo .

 ex:WashingtonMonument
  geo:hasGeometry ?wgeo .

 FILTER(geof:distance(?pgeo, ?wgeo,
    units:m) < 3000)
}
```

Queries looking for entities within a particular distance of either other entities or a current location are extremely useful as well. "Which parks are within 3km of the Washington Monument?" can be easily expressed in GeoSPARQL. We assume the same URI for the Washington Monument in the data example above. We need to retrieve the two geometries and use the function `geof:distance` to calculate the distance between them. A standard SPARQL less than function is then applied. This query is shown in listing 8. These example queries require relatively little in terms of non-spatial constraints, but serve to illustrate some basic functionality with GeoSPARQL. With a more complex ontology, queries could include more complicated thematic elements as well.

6. Spatial Indexing of RDF Data in Parliament

Storing geospatial data just as RDF triples does not allow for the spatial exploitation of that data. In order to be able to efficiently query for the relationships between spatial entities, the data must be indexed. This allows only those resources that match the spatial component of a query to be retrieved, rather than spatially filtering all bindings that match a given query. The relative performance advantages of using a spatial index versus spatially filtering a result set is discussed in [7]. Parliament uses a spatial index to not only find the data that matches a spatial query, but to also decide whether the spatial part of a query is more selective than the other. This can be used to optimize how a query is executed and is discussed below.

6.1. Enabling GeoSPARQL

Parliament has a modular architecture that enables indexes to be built on top of the storage engine. Parliament already includes a spatial index based on a standard R-tree implementation [14]. In a similar approach to [7], the spatial index is integrated in a way that provides native support for spatial relational queries and efficient storage of the data. The general goal for this index is to split SPARQL queries with geospatial information into multiple parts, allowing for an optimized query plan between the spatial components of the query and the components with non-spatial triples to be executed.

Before the emergence of GeoSPARQL, Parliament indexed data represented in GeoOWL and allowed RCC8 and OGC Simple Features relations to be queried [16,17]. We are currently implementing GeoSPARQL based on the public candidate draft standard, and we describe this implementation in the following section.

6.2. Index Specification

The index interfaces in the Parliament API include methods for building a record from data, adding and removing records, finding a record by URI, and finding records by value. Existing indexes include the aforementioned GeoOWL spatial index, a temporal index (for indexing OWL Time[14]), and a basic numeric index (for optimizing range queries on numeric property values).

The Parliament triple store is built with support for Jena's RDF API and ARQ SPARQL query engine[15].

Listing 9: Property Function Query

```
SELECT ?x
WHERE {
  ?x apf:concat( "Hello", " ", "
      World") .
}
```

An implementation of Jena's graph interface[16] provides access to the base graph store with support for adding, removing, and finding triples. By attaching a listener[17] to the graph, the addition and removal of triples can be detected and forwarded to any associated indexes. Parliament's GeoSPARQL index listens for triples that contain the `geo:asWKT` or `geo:asGML` predicates. Any triple that is added to the graph is checked to see if it matches. At this point, the index can create a record for the geometry that is represented in the object of the triple and insert it into the index. For instance, when adding the triples in listing 3 to Parliament, the index will generate a single record that contains a reference to the resource `ex:WMPoint` and its WKT value.

In order to query for data in an index, we have extended the ARQ query engine to support property functions that can access indexes. When ARQ parses a SPARQL query, it detects the different operators in the query. A particularly useful feature of ARQ is the support for property functions. Instead of matching a triple in a graph, property functions can execute custom code in the context of the SPARQL query. Consider the query in listing 9. It will yield a result set with a single binding for `?x` by concatenating all of the arguments together to form the literal "Hello World" instead of attempting to match the triple pattern.

Parliament utilizes property functions to define the relations that can be queried via SPARQL. The GeoSPARQL spatial relations, such as `geo:intersects`, are implemented as property functions.

6.3. Optimizing Query Execution

By using an index, a query can be optimized such that the most selective part is executed first. Con-

---

[14] `http://www.w3.org/TR/owl-time/`
[15] `http://www.openjena.org`
[16] `http://openjena.org/javadoc/com/hp/hpl/jena/graph/Graph.html`
[17] `http://openjena.org/javadoc/com/hp/hpl/jena/graph/GraphListener.html`

Listing 10: Optimization Example Query

```
SELECT ?m
WHERE {
  ?m a ex:Monument ;
      geo:hasGeometry ?mgeo .
  ?mgeo geo:within ex:
     NationalMallGeometry .
}
```

sider the query in listing 10. This query is asking for all monuments within the National Mall. Parliament's query optimizer splits the query into blocks for execution based on how the variables in the query are used and what special operations occur in the query. In this instance, since the predicate, `geo:within`, is an index property function, the triple containing the predicate is considered as one query block. The rest of the query is a simple basic graph pattern containing two triples describing `?m`. The basic graph pattern is analyzed and partitioned so that no variable crosses partitions. For this query, this generates two partitions. When the query is executed, the Parliament query optimizer has two choices: (1) it can execute the spatial part first and then match to the graph pattern, or (2) it can execute the graph pattern first, then execute the spatial operation. The optimizer will decide what to do based on which path estimates it will provide the fewest result bindings.

In this example, there are two sub patterns: the index property function, and the basic graph pattern for `?m`. Each sub pattern estimates how many results that it will be able to provide. For operations within the spatial index, a bounding box query can be performed to estimate how many results will be returned. In this example, the index will look up the bounding box for `ex:NationalMallGeometry` and calculate how many items it contains. For basic graph patterns, Parliament keeps statistics on the triples it contains and can quickly estimate how many matches are in the store for a given triple pattern. Sub patterns containing basic graph patterns use these statistics to estimate how many triples will be bound by the pattern. After each sub pattern has an estimate, they are ordered in ascending order executed accordingly. In this case, if there were 500 monuments with geometries, but only 100 geometries within the bounding box, the index property function would be executed first. If, however, there were 500 geometries within the bounding box, but only 10 monuments exist in the triple store, the basic graph pattern would execute first and the spatial relationship would be tested for each of the 10 results.

Listing 11: Example Query 4 - Optimized

```
SELECT ?p
WHERE {
  ?p a ex:Park .
  ex:WashingtonMonument geo:
     hasGeometry ?wgeo .
  LET (?buff := geof:buffer(?wgeo,
     3000, units:m)) .
  ?p geo:within [
    a geo:Point ;
    geo:asWKT ?buff
  ] .
```

Consider again the query in listing 8. This query is deceptive in that it appears to be asking a simple geospatial question: "What are all the parks within 3km of the Washington Monument?". However, there is no way for Parliament's query optimizer to know a priori what the distances between geometries are. While some implementations of GeoSPARQL may be optimized to handle cases like this, in Parliament this query would not use the spatial index at all; the geometries for all parks would be found before checking their distance. Listing 11 shows a more efficient version of this query. This version buffers the geometry for the Washington Monument by 3000 meters and then uses that buffer to test the `geo:within` relationship. Parliament can take this query and run the spatial component against the spatial index in order to reduce the amount of results that need to match the rest of the query. A future version of the Parliament query optimizer will be able to optimize queries like listing 8 automatically.

7. GeoSPARQL and Linked Data

As the Semantic Web materializes on the internet in the form of Linked Data, there is an increasing amount of structured data available with some sort of geospatial context attached. However, the vast majority of this geospatial context cannot be utilized for spatial queries because the hosting SPARQL endpoints cannot perform them. In the following section, we discuss four datasets that are representative of the disparate types of data that can be integrated together via there spatial context. We then demonstrate this integration on two datasets using Parliament.

7.1. Geospatial Data Sets

GeoNames[18] provides information for over eight million geospatial features. The data is exposed via an RDF webservice[19] that exposes information on a per resource basis. The geospatial aspect is represented using the W3C Basic Geo vocabulary. There is no ability, however, to perform any type of SPARQL query to retrieve data.

Another significant source of geospatial data is DBPedia[20]. As described in [2], data from Wikipedia[21] is extracted into RDF. Many of these extracted entities are geospatial in nature (cities, counties, countries, landmarks, etc…) and many of these entities already contain some geospatial location information. In fact, DBPedia contains information from GeoNames and goes so far as to include the latitude and longitude information for many entities. It also provides `owl:sameAs` links between the DBPedia and GeoNames resources. However, without any spatial computation predicates, this geospatial information can only be retrieved "as is." A query like "Show all cities within 50 miles of Arlington, VA with a population of at least 100,000 people in which at least one famous person was born" is not possible, even though the data exists to support it.

The linked data community has released the LinkedGeoData data set[3]. This data set is a spatial knowledge base, derived from Open Street Map[22] and is linked to DBpedia and GeoNames resources. It contains over 200 million triples describing the nodes and paths from OpenStreetMap. The data set is accessible via SPARQL endpoints running on the OpenLink Virtuoso platform with all the benefits and limitations that this platform provides.

Another source of geospatial data is the United States Geological Survey (USGS). The USGS has released a SPARQL endpoint[23] for triple data derived from *The National Map*[30], a collaborative effort to deliver usable topographic information for the United States. This dataset is much more specific and specialized than the data that is provided by DBPedia and GeoNames. It includes geographic names, hydrography, boundaries, transportation, structures, and land cover. The group has attempted to follow the forthcoming GeoSPARQL specification, though some aspects of GeoSPARQL have changed slightly since the data has been published. Due to a lack of available GeoSPARQL triple stores, the published dataset includes a pre-computation of all of the topological relationships between entities. A query such as "Show all rail lines that cross rivers" is in fact possible to answer by looking at the current precomputed data. However, this means that if a new entity is added, the knowledge base needs to compute everything that the entity is related to and update those entities as well. If this data was not precomputed, the only way to answer the query would be via indexing the data and querying the relations with a relationship predicate.

7.2. Integration in Parliament via GeoSPARQL

GeoSPARQL provides the means to link geospatial datasets together and extract more meaning from the relations between the data. As it is simply not possible to pre-compute all of the relations between all of the available geospatial datasets, enriching the existing datasets with GeoSPARQL representations, and creating indexes for the data is one way that data providers can share data while providing access to geospatial semantics.

For the following examples, we have processed a subset of the GeoNames RDF dataset[24] and the USGS GeoSPARQL data for Atlanta, GA. This data is accessible via a Parliament SPARQL endpoint with a GeoSPARQL spatial index[25]. The GeoSPARQL index supports indexing data conforming to WKT and GML serialization. The supported GeoSPARQL queries include those with Simple Features, Egenhofer, and RCC8 relations, the non-topological query functions and common topological query functions. Query rewriting is not supported at this time.

As so much data is represented as individual latitude and longitude data using the W3C Basic Geo vocabulary, including that provided by GeoNames, it is desirably to be able to convert it easily into Geo-

---

[18] `http://www.geonames.org`

[19] `http://www.geonames.org/ontology/documentation.html`

[20] `http://www.dbpedia.org`

[21] `http://www.wikipedia.org`

[22] `http://www.openstreetmap.org`

[23] `http://usgs-ybother.srv.mst.edu:8890/sparql`

[24] `http://download.geonames.org/all-geonames-rdf.zip`

[25] `http://geosparql.bbn.com`

Listing 12: GeoNames Conversion Query

```
CONSTRUCT {
 ?feature a geo:Feature ;
  geo:hasGeometry [
   a geo:Point ;
   geo:asWKT ?wkt
 ] .
}
WHERE {
 ?feature a gn:Feature ;
  wgs84_pos:lat ?lat ;
  wgs84_pos:long ?long .
 LET (?wkt := spatial:toWKTPoint(?
    lat, ?long)) .
}
```

SPARQL. It is trivial to provide functions that take a latitude and longitude pair and convert them into a GeoSPARQL point Parliament provides SPARQL property functions, `spatial:toWKTPoint` and `spatial:toGMLPoint` which take a latitude, longitude, and optional spatial reference system identifier as arguments and return a `geo:WKTLiteral` or `geo:GMLLiteral` representation of a point. This makes it possible to load and index existing spatial data sets without having to regenerate existing RDF. After loading the GeoNames data into Parliament, it was aligned with GeoSPARQL using the query in listing 12. This query explicitly assigns GeoNames features to be GeoSPARQL features, creates a new `geo:Point` resource containing the point information, and links the feature to the geometry.

The USGS provides their RDF data in a format that is similar to GeoSPARQL. The geospatial data from *The National Map* contains polygon, polyline, and point data. The data conforms to an earlier revision of the GeoSPARQL standard and contains features and geometries, but lacks typed literals for the `geo:asWKT` and `geo:asGML` property values. For this data, the constructor functions for the GeoSPARQL literal datatypes can be used to create correctly typed values at query time. The existing spatial relations statements in the dataset were ignored when processing the data.

7.2.1. GeoSPARQL Data Query

Once GeoSPARQL data exists for both GeoNames and the USGS and is loaded into a GeoSPARQL capable knowledge base, such as Parliament, interesting geospatial questions can be posed. Consider the following query: "What are all the schools in Georgia that are within 50 meters of a railway". GeoNames provides point data for buildings, including schools, while the USGS data contains polyline data for rail lines as well as polygons for different regions. Listing 13 shows the GeoSPARQL formulation for this question. This query takes advantage of several features of the language. First, the `geo:within` predicate is used to determine what schools exist within the boundary of Georgia (as defined by a specific USGS resource for Georgia). Each school geometry is then buffered by 50 meters using the `geof:buffer` function. The rail features are retrieved and checked to see if they fall within this buffer. Finally, for all rail line segments that intersect the buffer, the actual distance to the school is calculated using the `geof:distance` function. Table 2 displays the school resources and the distance to the nearest rail line segment.

Table 2

Schools in Georgia within 50m of a Rail Line

school	**distance**
http://sws.geonames.org/4183400/	13.87504515671802
http://sws.geonames.org/7146774/	39.795020615173165
http://sws.geonames.org/4183400/	44.09248276546987
http://sws.geonames.org/7146435/	46.6898774805717

While the sample query assumes everything is contained in a single graph, it is not unusual for the data to exist in several different graphs or endpoints. In fact, it would be ideal to not have to replicate data and to be able to query it remotely through the dataset provider. Until the means for query federation, such as the mechanism discussed in [27], are widely supported, querying across remote linked datasets will be difficult. GeoSPARQL queries should be compatible with query federation, though there will likely be performance implications. In lieu of query federation, querying across graphs is possible. The sample data for the above examples is actually contained in two different graphs. A union graph that virtually combines both graphs, however, enables a shorter and simpler query.

8. Conclusion

GeoSPARQL is the genesis of a significant amount of previous work on combining RDF and OWL with geospatial data. Its creation means that geospatial data interchange within the Semantic Web can take place

Listing 13: Georgia School's Near Rail Lines Query

```
SELECT DISTINCT ?school ?distance
WHERE {
 GRAPH <http://example.org/data> {
  # get Georgia geometry
  gu:_2403126
   geo:hasGeometry ?ga_geo .

  # get schools within Georgia
  ?school a geo:Feature ;
   geo:hasGeometry ?school_geo ;
   gn:featureCode gn:S.SCH .

  ?school_geo gep:within ?ga_geo ;
   geo:asWKT ?school_wkt .

  # buffer schools 50m
  LET (?s_buff := geof:buffer(?
     school_wkt, 50, units:m)) .

  # find rail links within buffer
  ?rail a trans:railFeature ;
   geo:hasGeometry ?rail_geo .

  ?rail_geo geo:asWKT ?rail_wkt_s .
  LET (?rail_wkt :=geo-sf:WKTLiteral
     (?rail_wkt_s)) .

  FILTER (geof:intersects(?rail_wkt
     ,?s_buff)) .

  LET (?distance := geof:distance(?
     school_wkt,?rail_wkt,units:m))
      .
 }
}
ORDER BY ASC(?distance)
```

with an expectation of efficient geospatial queries. This, by extension, should lead to users' ability to finally utilize the significant amount of geospatial context available in RDF datasets.

Many triple stores, though they do not yet support GeoSPARQL, support similar functionality or a subset thereof. This is an indicator of how important geospatial applications are. Hopefully when the GeoSPARQL standard is released, the relevant vendors will move to unify their implementations, allowing users to consistently exchange and process geospatial data.

We have worked to update our open source triple store Parliament to support GeoSPARQL, and we hope that others will find it useful both for working on geospatial Semantic Web applications and understanding the specification.

9. Acknowledgments

We would like to thank the people at the Center of Excellence for Geospatial Information Science (CEGIS) at USGS for their efforts to support GeoSPARQL even before the specification has been finalized. This has allowed us to identify and resolve additional issues with the specification earlier.

We would also like to thank the OGC and the other members of the GeoSPARQL working group for their continued efforts to bring this marriage of the geospatial community and the Semantic Web to life.

Listing 14: RDF Prefixes

```
pf: <http://jena.hpl.hp.com/ARQ/
   property#>
ex: <http://example.org/
   PointOfInterest#>
gn: <http://www.geonames.org/
   ontology#>
gu: <http://cegis.usgs.gov/rdf/gu/
   featureID#>
geo: <http://www.opengis.net/ont/OGC
   -GeoSPARQL/1.0/>
geof: <http://www.opengis.net/def/
   queryLanguage/OGC-GeoSPARQL/1.0/
   function/>
geo-sf: <http://www.opengis.net/def/
   dataType/OGC-SF/1.0/>
geo-gml: <http://www.opengis.net/def
   /dataType/OGC-GML/3.0/GMLLiteral
   >
os: <http://rdf.opensahara.com/
   search#>
ose: <http://www.example.org/
   opensahara#>
osg: <http://rdf.opensahara.com/type
   /geo/>
owl: <http://www.w3.org/2002/07/owl
   #>
```

```
rdfs: <http://www.w3.org/2000/01/rdf
    -schema#>
spatial: <http://parliament.
    semwebcentral.org/ontology/
    spatialrelations/>
time: <http://www.w3.org/2006/time#>
trans:  <http://cegis.usgs.gov/rdf/
    trans#>
units: <http://www.opengis.net/def/
    uom/OGC/1.0/>
wgs84_pos: <www.w3.org/2003/01/geo/
    wgs84_pos#>
xsd: <http://www.w3.org/2001/
    XMLSchema#>
virtrdf: <http://www.openlinksw.com/
    schemas/virtrdf#>
```

References

[1] J. F. Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26:832–843, November 1983. ISSN 0001-0782.

[2] S. Auer and J. Lehmann. What Have Innsbruck and Leipzig in Common? Extracting Semantics from Wiki Content. In E. Franconi, M. Kifer, and W. May, editors, *Proceedings of the European Semantic Web Conference (ESWC-2007)*, volume 4519 of *Lecture Notes in Computer Science*, pages 503–517. Springer Berlin / Heidelberg, 2007. ISBN 978-3-540-72666-1.

[3] S. Auer, J. Lehmann, and S. Hellmann. LinkedGeoData: Adding a Spatial Dimension to the Web of Data. In A. Bernstein, D. Karger, T. Heath, L. Feigenbaum, D. Maynard, E. Motta, and K. Thirunarayan, editors, *Proceedings of the 8th International Semantic Web Conference (ISWC-2009)*, volume 5823 of *Lecture Notes in Computer Science*, pages 731–746. Springer Berlin / Heidelberg, 2009. ISBN 978-3-642-04929-3.

[4] C. Becker and C. Bizer. Exploring the geospatial semantic web with dbpedia mobile. *Web Semantics: Science, Services and Agents on the World Wide Web*, 7(4):278 – 286, 2009. ISSN 1570-8268. . URL `http://www.sciencedirect.com/science/article/pii/S1570826809000468`. Semantic Web challenge 2008.

[5] A. Belussi, E. Bertino, and B. Catania. Manipulating spatial data in constraint databases. In M. Scholl and A. Voisard, editors, *Advances in Spatial Databases*, volume 1262 of *Lecture Notes in Computer Science*, pages 113–141. Springer Berlin / Heidelberg, 1997. ISBN 978-3-540-63238-2.

[6] C. Bizer, T. Heath, and T. Berners-Lee. Linked Data – The Story So Far. *International Journal on Semantic Web and Information Systems, Special Issue on Linked Data*, 5:1–22, 2009. ISSN 1552-6283.

[7] A. Brodt, D. Nicklas, and B. Mitschang. Deep integration of spatial query processing into native RDF triple stores. In *Proceedings of the 18th SIGSPATIAL International Conference on Advances in Geographic Information Systems*, GIS '10, pages 33–42, New York, NY, USA, 2010. ACM. ISBN 978-1-4503-0428-3.

[8] M. J. Egenhofer. A formal definition of binary topological relationships. In *3rd International Conference, FODO 1989 on Foundations of Data Organization and Algorithms*, pages 457–472, New York, NY, USA, 1989. Springer-Verlag New York, Inc. ISBN 0-387-51295-0.

[9] M. J. Egenhofer. Spatial SQL: A query and presentation language. *IEEE Transactions on Knowledge and Data Engineering*, 6:86–95, 1994.

[10] M. J. Egenhofer. Toward the semantic geospatial web. In *Proceedings of the 10th ACM international symposium on Advances in geographic information systems*, GIS '02, pages 1–4, New York, NY, USA, 2002. ACM. ISBN 1-58113-591-2.

[11] M. J. Egenhofer and J. R. Herring. Categorizing binary topological relations between regions, lines, and points in geographic databases. Technical report, Department of Surveying Engineering, University of Maine, 1990.

[12] Franz, Inc. Geospatial support in SPARQL queries, April 2011. URL `http://www.textriples.com/agraph/support/documentation/4.2/sparql-geo.html`. [Online; accessed 25-May-2011].

[13] R. Grütter and B. Bauer-Messmer. Combining owl with rcc for spatioterminological reasoning on environmental data. In *OWLED 2007 Workshop on OWL: Experiences and Directions*, volume 258, 2007. URL `http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-258/paper17.pdf`.

[14] A. Guttman. R-trees: A dynamic index structure for spatial searching. In *International Conference on Management of Data*, pages 47–57. ACM, 1984.

[15] H. Hu and X. Du. Linking Open Spatiotemporal Data in the Data Clouds. In J. Yu, S. Greco, P. Lingras, G. Wang, and A. Skowron, editors, *Rough Set and Knowledge Technology*, volume 6401 of *Lecture Notes in Computer Science*, pages 304–309. Springer Berlin / Heidelberg, 2010.

[16] D. Kolas. Supporting spatial semantics with SPARQL. In *Proceedings of the Terra Cognita Workshop, collocated with the 7th International Semantic Web Conference (ISWC-2008)*, 2008.

[17] D. Kolas and T. Self. Spatially augmented knowledgebase. In *Proceedings of the 6th International Semantic Web Conference (ISWC-2007)*, volume 4825 of *Lecture Notes in Computer Science*. Springer, 2007.

[18] D. Kolas, I. Emmons, and M. Dean. Efficient linked-list RDF indexing in Parliament. In *5th International Workshop on Scalable Semantic Web Knowledge Base Systems (SSWS 2009)*, 2009.

[19] M. Koubarakis and K. Kyzirakos. Modeling and Querying Metadata in the Semantic Sensor Web: The Model stRDF and the Query Language stSPARQL. In L. Aroyo, G. Antoniou, E. Hyvönen, A. ten Teije, H. Stuckenschmidt, L. Cabral, and T. Tudorache, editors, *Proceedings of the Extended Semantic Web Conference 2010*, volume 6088 of *Lecture Notes in Computer Science*, pages 425–439. Springer Berlin / Heidelberg, 2010. ISBN 978-3-642-13485-2.

[20] K. Kyzirakos, Z. Kaoudi, and M. Koubarakis. Data models and languages for registries in SemsorGrid4Env, 2009. Deliverable D3.1, SemSorGrid4Env.

[21] J. Lieberman, R. Singh, and C. Goad. W3C Geospatial Ontologies, October 2007. URL `http://www.w3.org/2005/`

`Incubator/geo/XGR-geo-ont/`. [Online; accessed 25-May-2011].

[22] Ontotext AD. Geo-spatial indexing in OWLIM, 2011. URL `http://www.ontotext.com/owlim/geo-spatial`. [Online; accessed 25-May-2011].

[23] Open Geospatial Consortium. An Introduction to GeoRSS: A Standards Based Approach for Geo-enabling RSS feeds. White paper, 2006. URL `http://www.opengeospatial.org/pt/06-050r3`. Document 06-050r3.

[24] Open Geospatial Consortium. *OGC GeoSPARQL - A Geographic Query Language for RDF Data*. Open Geospatial Consortium, 2011. URL `http://www.opengeospatial.org/standards/requests/80`. Document 11-052r3.

[25] Open Geospatial Consortium. *OpenGIS®Implementation Standard for Geographic information - Simple feature access - Part 1: Common architecture*. Open Geospatial Consortium, 2011. URL `http://www.opengeospatial.org/pt/06-103r4`. Document 06-103r4.

[26] M. Perry. *A Framework to Support Spatial, Temporal and Thematic Analytics over Semantic Web Data*. PhD thesis, Wright State University, 2008.

[27] E. Prud'hommeaux. SPARQL 1.1 federation extensions. W3C working draft, W3C, June 2010. http://www.w3.org/TR/sparql11-federated-query/.

[28] D. A. Randell, Z. Cui, and A. G. Cohn. A spatial logic based on regions and connection. In *Proceedings of the 3rd International Conference on Knowledge Representation and Reasoning*, 1992.

[29] J. M. Salas, A. Harth, B. Norton, L. M. Vilches, A. D. León, J. Goodwin, C. Stadler, S. Anand, and D. Harries. NeoGeo Vocabulary: Defining a shared RDF representation for GeoData. Public draft, NeoGeo, May 2011. URL `http://geovocab.org/doc/neogeo.html`.

[30] D. Varanka. National topographic modeling, ontology-driven geographic information in the context of the national map. In *First International Workshop on Information Semantics and its Implications for Geographical Analysis (ISGA '08) at GIScience 2008, the 5th International Conference on Geographic Information Science*, September 2008.

[31] W3C. W3C Semantic Web Interest Group: Basic Geo (WGS84 lat/long) Vocabulary, 2006. URL `http://www.w3.org/2003/01/geo/`. [Online; accessed 25-May-2011].

[32] C. Weiss, P. Karras, and A. Bernstein. Hexastore: sextuple indexing for semantic web data management. *Proc. VLDB Endow.*, 1:1008–1019, August 2008. ISSN 2150-8097.

[33] Z. Xiao, L. Huang, and X. Zhai. Spatial information semantic query based on sparql. In *Proceedings of SPIE*, volume 7492, October 2009.

[34] X. Zhai, L. Huang, and Z. Xiao. Geo-spatial query based on extended sparql. In *2010 18th International Conference on Geoinformatics*, pages 1–4, June 2010.