Introduction in stRDF and stSPARQL

M. Koubarakis, K. Kyzirakos, M. Karpathiotakis,
Ch. Nikolaou, M. Sioutis, G. Garbis, K. Bereta

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This document presents the data model stRDF and the query language stSPARQL, the new version of the data model stRDF and the query language stSPARQL respectively. NKUA, in the context of the FP7 ICT project SemsorGrid4Env [1], proposed stRDF and stSPARQL extensions of the W3C standards RDF and SPARQL for representing and querying spatial data in the Semantic Web . stRDF and stSPARQL follow the tradition of constraint databases and use linear constraints for the representation of spatial data. In the new version of stRDF and stSPARQL, a more practical solution to the problem of representing geospatial data is proposed, using the OGC standard Well-known Text (WKT) instead of constraints. These new versions were proposed by NKUA in the context of the FP7 ICT project TELEIOS [2] . In chapter 2, we provide background information about OGC standards for representing spatial information and coordinate reference systems. Then, we present the data model stRDF (chapter3), and a part of its integration in the NOA Use Case. Chapter 5 describes the query language stSPARQL. Chapter 6 introduces the system Strabon, used for storing stRDF graphs and evaluating stSPARQL queries. Finally, chapter 7 presents a the use of Strabon in a real world scenario, which involves data retrieval and rapid mapping based on heterogeneous,linked data comprised datasets.

[1] `http://www.semsorgrid4env.eu/`
[2] `http://www.earthobservatory.eu/`

Chapter 2

Background on OGC Standards

2.1 Background on OGC Standards

In this section we present some well-known standards developed by the Open Geospatial Consortium (OGC)[1]. OGC is an international consortium of companies, government agencies and universities participating in a consensus process to develop publicly available interoperability standards for geospatial data. OGC standards focus on solutions for geospatial data and services, GIS data processing and geospatial data sharing.

The OGC Abstract Specification[2] is the conceptual foundation for the OGC interoperability specifications. The purpose of this specification is to define a conceptual model which includes the core concepts related to geospatial data, for which OGC standards will be developed in other technical documents. The OGC Abstract Specification also defines the basic terminology to be used in the rest of the OGC specifications. A small subset of this terminology (which might seem unfamiliar to a newcomer) will be used many times in the rest of this tutorial, thus we introduce it here.

In OGC terminology, a *geographic feature* (or simply *feature*) is an abstraction of a real world phenomenon and can have various attributes that describe its *thematic* and *spatial* characteristics. For example, a feature can represent an airport. Thematic information about an airport can include its name, the company that manages it, etc., while a spatial characteristic is its location on Earth. The spatial characteristics of a feature are represented using *geometries* such as points, lines, and polygons. Each geometry is associated with a *coordinate reference system* which describes the coordinate space in which the geometry is defined.

The organization of the rest of this section is as follows. In Section 2.1.1 we will define the main concepts underlying coordinate reference systems, and give some background information on the coordinate reference systems we most often find in applications. In many research papers coordinate systems are given only a passing mention, and the well-known Cartesian plane (or some subset of

[1] `http://www.opengeospatial.org/`
[2] `http://www.opengeospatial.org/standards/as`

it) is assumed to be the geographic space occupied by features. However, this assumption is not always true in applications, so the following section surveys the most popular coordinate systems that are in use today. In Section 2.1.2 we will present the Well-Known Text OGC standard that can be used for representing geometries and their coordinate reference systems. In Section 2.1.3 we will present the OpenGIS Simple Feature Access standard that defines a standard SQL schema that supports storage, retrieval, query and update of collections of features using SQL. This standard is today used in all relational DBMSs that offer support for geospatial data. It is also the basis for the stSPARQL and GeoSPARQL query languages discussed later, thus it is covered in detail. In Section 2.1.4 we will present the Geography Markup Language which is a more recent standard defined by OGC for representing geographical features in XML. Both of these standards have been recently used in data models and query languages for linked geospatial data, thus we present them in detail here.

2.1.1 Coordinate Reference Systems

A *coordinate* is one of n scalar values that determines the position of a point in an n-dimensional space. A *coordinate system* is a set of mathematical rules for specifying how coordinates are to be assigned to points. For example, the well-known Cartesian coordinate system assigns positions to points relative to n mutually perpendicular axes. A *coordinate reference system* is a coordinate system that is related to an object (e.g., the Earth, a planar projection of the Earth, a three dimensional mathematical space such as $\mathbb{Z}^3$) through a so called *datum* which specifies its origin, scale, and orientation. In the relevant literature, a coordinate reference system is also referred to as a *spatial reference system*. Various kinds of coordinate reference systems exist. In what follows we discuss geographic and projected coordinate systems because they are the ones most often found in GIS applications.

A *geographic coordinate reference system* is a three-dimensional coordinate system that utilizes latitude, longitude, and optionally elevation, to capture geographic locations on Earth. Detailed definitions for latitude, longitude and elevation (technically, geodetic height) can be found in [LGMR05]. The World Geodetic System (WGS) is the most well-known geographic coordinate reference system and its latest revision is WGS84[3]. WGS84 is the reference coordinate system used by the Global Positioning System.

Although a geographic coordinate system such as WGS84 is a comprehensive way to describe locations on Earth, some applications work on a *projection* of the Earth even though there is a price to pay in terms of distortions that such a projection causes. In these cases a *projected coordinate reference system* is used that transforms the 3-dimensional ellipsoid approximation of the Earth into a 2-dimensional surface. In general, projected coordinate reference systems are always associated with a geographic coordinate system and it is important to understand the compromises made by it when computing the projection of the Earth.

An example of a projected coordinate reference system with world coverage is the Universal Transverse Mercator (UTM) system. UTM uses the WGS84 ellipsoid approximation of the Earth as the underlying geographic coordinate

[3] `http://en.wikipedia.org/wiki/WGS84/`

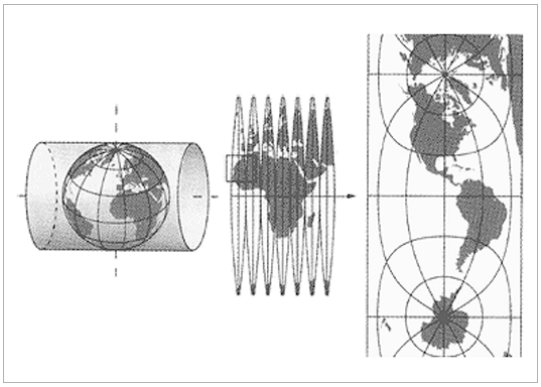


Figure 2.1: The transverse Mercator projection of UTM

system. It is based on the transverse Mercator projection of the Earth on a 2-dimensional plane. Intuitively, this projection is obtained if we form a cylinder by wrapping a piece of paper around the Earth's poles, projecting on it the ellipsoid of Earth, and then unwrapping it (see Figure 2.1). As shown on the same figure, UTM is not a single projection system. It is based on a grid which divides the Earth into sixty zones of equal width, so that each zone can use a fine-tuned transverse Mercator projection that is capable of projecting the corresponding region of the Earth with a low amount of distortion.

Individual countries or states (e.g., in the USA) have their own projected coordinate reference systems that are more precise for their geographic area. For example, the Greek Geodetic Reference System 1987 (GGRS87)[4] is a projection system commonly used in Greece which is based on the local coordinate reference system Geodetic Reference System 1980 (GRS80)[5].

Various authorities provide information about popular coordinate reference systems. For example, OGC maintains a list with the URIs of various systems[6]. The European Petroleum Survey Group (EPSG) also provides a big collection of coordinate reference systems[7]. The identifiers of coordinate reference systems assigned by these authorities are used in geospatial data standards, e.g., Well-Known Text, to be discussed below.

2.1.2 Well-Known Text

Well-Known Text (WKT) is a widely used OGC standard for representing geometries. WKT can be used for representing geometries, coordinate reference systems, and transformations between coordinate reference systems. WKT is described in the "OpenGIS Simple Feature Access - Part 1: Common Architecture" specification [OGC10d] that is the same as the ISO 19125-1 standard. This standard concentrates on ways to represent and manipulate simple features. A *simple feature* is a feature with all spatial attributes described piecewise by a straight line or a planar interpolation between sets of points.

[4] `http://spatialreference.org/ref/epsg/4121/`
[5] `http://spatialreference.org/ref/epsg/6121/`
[6] `http://www.opengis.net/def/crs/`
[7] `http://www.epsg-registry.org/`

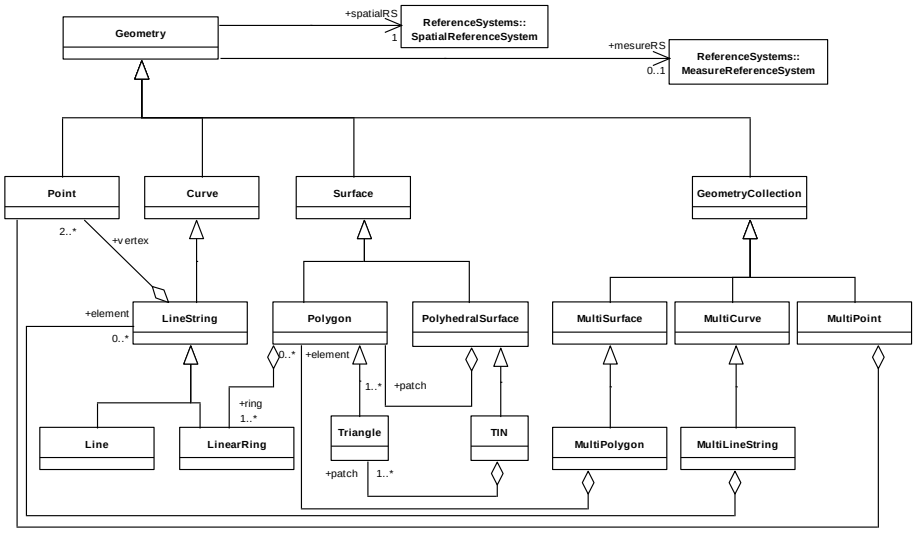


Figure 2.2: The classes of geometries in WKT (figure from [OGC10d])

Geometries in WKT are restricted to 0-, 1- and 2- dimensional geometries that exist in $\mathbb{R}^2$, $\mathbb{R}^3$, or $\mathbb{R}^4$. Geometries that exist in $\mathbb{R}^2$ consist of points with coordinates x and y, e.g., `POINT(1 2)` in WKT syntax. Geometries that exist in $\mathbb{R}^3$ consist of points with coordinates x, y, and z, or x, y, and m where m is a measurement. For example, the point `POINT(37.96 23.71 27)` might be used to represent the temperature of the city of Athens measured in Celcius degrees; where `37.96` is the latitude of Athens, `23.71` its longitude, and `27` its temperature. Geometries that exist in $\mathbb{R}^4$ have points with coordinates x, y, z, and m with similar semantics.

Geometries represented using WKT have the following properties:

- All geometries are topologically closed which means that all the points that comprise the boundary of the geometry are assumed to belong to the geometry, even though they may not be explicitly represented in the geometry.
- All coordinates within a geometry are in the same coordinate reference system.
- For geometric objects that exist in $\mathbb{R}^3$ and $\mathbb{R}^4$, spatial operations work on their "map geometries", that is, their projections on $\mathbb{R}^2$. Therefore, the z and m values are not reflected in calculations (e.g., when invoking functions equals, intersects, etc.) or in the generation of new geometry values (e.g., when invoking functions buffer, minimum bounding box, etc.). Thus, the WKT specification is inherently about 2-dimensional geometries, but it also allows z and m values associated with these geometries and functions to access them.

Let us now present the part of the standard that defines how to represent vector geometries. In Figure 2.2 we present the class hierarchy for simple feature geometries represented in WKT as defined in [OGC10d]. The top Geometry

class has subclasses Point, Curve, Surface, and Geometry Collection. The Geometry Collection is further specialized to classes of 0-, 1- and 2- dimensional geometries named MultiPoint, MultiLineString and MultiPolygon respectively. Each geometry is linked to a specific coordinate reference system and optionally to a measure reference system. A *measure reference system* may be used to interpret the third or fourth dimension of geometries that exist in $\mathbb{R}^3$ or $\mathbb{R}^4$. For example, a fire hotspot can be modeled as a point that has x, y, and m coordinates, where the coordinates x and y are used to represent the location of the hotspot, while the m coordinate represents the measured temperature.

Let us provide some more information for each class depicted in Figure 2.2.

- **Point**. A point represents a single location in coordinate space. A point has x and y coordinate values and may have z and m depending on the associated coordinate reference system.
- **Curve**. A curve is a 1-dimensional geometry. The subtypes of class curve define the type of interpolation that is used between points.
- **LineString**. A line string is a subtype of class curve that uses linear interpolation between points. A line string is closed if its start point is equal to its end point. A line string is simple if it has no self-intersections.
- **Line**. A line is a line string with exactly two points.
- **LinearRing**. A linear ring is a line string that is both closed and simple.
- **Surface**. A surface is a 2-dimensional geometry. This class is abstract (i.e., it may not be instantiated). A simple surface may consist of a single "patch" that has one "exterior" boundary and 0 or more "interior" boundaries (e.g. a polygon with holes).
- **Polygon**. A polygon is a simple surface that is planar. It has exactly one exterior boundary and may have several non-intersecting interior boundaries. Each polygon is topologically closed and no two boundaries (interior or exterior) cross. However, two boundaries may intersect at a point, but only as a tangent. The interior of a polygon is a connected point-set while the exterior of a polygon with holes is not connected.
- **Triangle**. A triangle is a polygon with 3 distinct, non-collinear vertices and no interior boundary.
- **Polyhedral Surface**. A polyhedral surface is a contiguous collection of polygons which share common boundary segments. Each pair of polygons that touch has a common boundary that is expressed as a finite collection of line strings. Each such line string is a part of the boundary of at most 2 polygon patches.
- **Triangulated Irregular Network**. A triangulated irregular network is a polyhedral surface consisting only of triangle patches.
- **Geometry Collection**. A geometry collection is a set of distinct geometries.

- **MultiPoint**. This is a geometry collection whose elements are points that are not connected.
- **MultiCurve**. A multi-curve is a geometry collection whose elements are curves.
- **MultiLineString**. A multi-line string is a geometry collection whose elements are line strings.
- **MultiSurface**. A multi-surface is a 2-dimensional geometry collection whose elements are surfaces. The geometric interiors of any two surfaces may not intersect. The boundaries of any two surfaces may not cross but may touch at a finite number of points.
- **MultiPolygon**. A multi-polygon is a multi-surface collection whose elements are polygons. The boundaries of each polygon may not intersect.

The syntax of the WKT representation of a geometry is presented in detail in [OGC10d]. Some examples of geometries represented in WKT are shown in Table 2.1.

The interpretation of the coordinates of a geometry depends on the coordinate reference system that is associated with the geometry. Note that according to the WKT standard, the coordinate reference system that is associated to a geometry is never embedded in the object's representation, but is given separately using appropriate notation.

2.1.3 OpenGIS Simple Feature Access

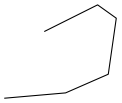
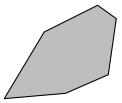
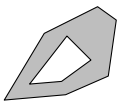
The "OpenGIS Simple Feature Access - Part 2: SQL Option" standard [OGC10c] defines a standard SQL schema that supports storage, retrieval, query and update of sets of simple features using SQL. This OGC standard is the same as the ISO 19125-2 standard. The simple features supported by this standard have both spatial and non-spatial (thematic) attributes. The spatial attributes are geometries of the types described in Section 2.1.2. This standard assumes that sets of simple features are stored as relational tables and each feature is a row in a table. The spatial attributes of the features are represented as geometry-valued columns, while non-spatial attributes are represented as columns whose types are the standard SQL data types. There are also additional tables that are used to store information about features and coordinate reference systems. The standard describes schemas for two types of feature table implementations: implementations using only the SQL predefined data types and SQL with geometry types. Only the latter approach is covered here.

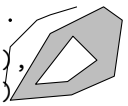
The "SQL with geometry types" approach uses WKT geometry classes presented in Section 2.1.2 to define new geometric data types for SQL together with SQL functions on those types.

The following SQL functions have been defined for requesting the desired representation of a geometry, checking whether some condition holds for a geometry and returning some properties of the geometry:

1. `ST_Dimension(A:Geometry):Integer`, returns the inherent dimension of

Table 2.1: Examples of geometries represented in WKT

Geometry type	**WKT representation**	**Geometry**
Point	`Point(5 5)`	
LineString	`LineString(5 5,28 7,44 14,47 35,40 40,20 30)`	
Polygon	`Polygon((5 5,28 7,44 14,47 35,40 40,20 30,5 5))`	
Polygon	`Polygon((5 5,28 7,44 14,47 35,40 40,20 30,5 5),` `(28 29,14.5 11,26.5 12,37.5 20,28 29))`	
MultiPoint	`MultiPoint((5 5),(28 7),(44 14),` `(47 35),(40 40),(20 30))`	

Geometry Collection	`GeometryCollection(`
	`Point(5 35),`
	`LineString(3 10,5 25,15 35,20 37,30 40),`
	`Polygon((5 5,28 7,44 14,47 35,40 40,20 30,5 5),`
	`(28 29,14.5 11,26.5 12,37.5 20,28 29))`
	`)`
		

the geometry `A`, which must be less than or equal to the coordinate dimension[8].

2. `ST_GeometryType(A:Geometry):String`, returns the name of the instantiable subtype of Geometry as defined in [OGC10d], of which the geometry `A` is an instantiable member.

3. `ST_AsText(A:Geometry):String`, exports the geometry `A` as its WKT representation.

4. `ST_AsBinary(A:Geometry):Binary`, exports the geometry `A` as its Well-Known Binary[9] representation.

[8]The prefix ST is a historical one and comes from the spatial part of the SQL/MM standard [Sto03]. It was meant to denote "spatial and temporal", but temporal issues were not included finally in that standard.

[9]Well-Known Binary is an equivalent to the WKT format for the representation of a ge-

5. `ST_SRID(A:Geometry):Integer`, returns the coordinate reference system identifier for the geometry `A`.

6. `ST_IsEmpty(A:Geometry):Boolean`, returns true if the geometry `A` is the empty geometry. Otherwise, it returns false.

7. `ST_IsSimple(A:Geometry):Boolean`, returns true if the geometry `A` has no anomalous geometric points, such as self intersection or self tangency. Otherwise, it returns false.

The following SQL functions have been defined for testing topological spatial relationships between two geometries. These functions correspond to the relations from the dimensionally extended 9-intersection model of Egenhofer and Clementini defined in [CSE94].

1. `ST_Equals(A:Geometry, B:Geometry):Boolean`, returns true if the geometry `A` is "spatially equal" to the geometry `B`. Otherwise it returns false.

2. `ST_Disjoint(A:Geometry, B:Geometry):Boolean`, returns true if the geometry `A` is "spatially disjoint" to the geometry `B`. Otherwise it returns false.

3. `ST_Intersects(A:Geometry, B:Geometry):Boolean`, returns true if the geometry `A` "spatially intersects" the geometry `B`. Otherwise it returns false.

4. `ST_Touches(A:Geometry, B:Geometry):Boolean`, returns true if the geometry `A` "spatially touches" the geometry `B`. Otherwise it returns false.

5. `ST_Crosses(A:Geometry, B:Geometry):Boolean`, returns true if the geometry `A` "spatially crosses" the geometry `B`. Otherwise it returns false.

6. `ST_Within(A:Geometry, B:Geometry):Boolean`, returns true if the geometry `A` is "spatially within" to the geometry `B`. Otherwise it returns false.

7. `ST_Contains(A:Geometry, B:Geometry):Boolean`, returns true if the geometry `A` "spatially contains" the geometry `B`. Otherwise it returns false.

8. `ST_Overlaps(A:Geometry, B:Geometry):Boolean`, returns true if the geometry `A` "spatially overlaps" the geometry `B`. Otherwise it returns false.

9. `ST_Relate(A:Geometry, B:Geometry, intersectionPatternMatrix: String):Boolean`, returns true if the geometry `A` is "spatially related" to the geometry `B` by testing for intersections between the interior, boundary and exterior of the two geometries as specified by the values in the `intersectionPatternMatrix` according to the Egenhofer and Clementini intersection pattern matrix (DE-9IM) of [CFO93, CSE94].

The following SQL functions have been defined for constructing new geometric objects from existing geometries.

---

ometry and is also defined in [OGC10d]. The object is represented as a contiguous stream of bytes, thus facilitating its exchange between a database client and server.

1. `ST_Boundary(A:Geometry):Geometry`, returns a geometry that is the boundary of the geometry `A`.

2. `ST_Envelope(A:Geometry):Geometry`, returns a geometry that is the minimum bounding box for the input geometry `A`.

3. `ST_Intersection(A:Geometry, B:Geometry):Geometry`, returns a geometry that represents the point set intersection of the geometries `A` and `B`.

4. `ST_Union(A:Geometry, B:Geometry):Geometry`, returns a geometry that represents the point set union of the geometries `A` and `B`.

5. `ST_Difference(A:Geometry, B:Geometry):Geometry`, returns a geometry that represents the point set difference of the geometries `A` and `B`.

6. `ST_SymDifference(A:Geometry, B:Geometry):Geometry`, returns a geometry that represents the point set symmetric difference of the geometries `A` and `B`.

7. `ST_ConvexHull(A:Geometry):Geometry`, returns a geometry that represents the convex hull of the geometry `A`.

8. `ST_Buffer(A:Geometry, distance:Double):Geometry`, returns a geometry that represents all points whose distance from the geometry `A` is less than or equal to `distance`. The relevant calculation is done in the coordinate reference system of this geometry.

The `ST_Distance(A:Geometry, B:Geometry):Double` SQL function is defined for calculating the shortest distance between two geometries. The calculated scalar value corresponds to the shortest distance of these two geometries measured in the unit system of their coordinate reference system.

The standard ISO 13249 SQL/MM is an international standard for multimedia and other application extensions of SQL. Part 3 of this standard defines a set of types and methods for representing, processing, storing and querying spatial data in relational databases [Sto03, ME01]. The SQL/MM standard for spatial data is very close to the OpenGIS Simple Feature Access standard presented here and, in fact, the two efforts influenced each other. Therefore, we do not give any details about it in this document.

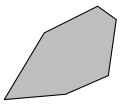
2.1.4 Geography Markup Language

The Geography Markup Language (GML) [OGC07] is the most common XML-based encoding standard for the representation of geospatial data. GML was developed by the OGC and it is based on the OGC Abstract Specification. GML provides XML schemas for defining a variety of concepts that are of use in Geography: geographic features, geometry, coordinate reference systems, topology, time and units of measurement. Initially, the GML abstract model was based on RDF and RDFS, but later the consortium decided to use XML and XML Schema. The GML *profiles* are logical restrictions of GML that might be of use to applications that do not want to use the whole of GML. GML profiles can be specified through an XML document, an XML schema, or both. Some of the

Table 2.2: Examples of geometries represented in GML

Geometry type	**GML representation**	**Geometry**

Point	`<gml:Point gml:id="p1"`
	`srsName="urn:ogc:def:crs:EPSG:6.6:4326">`
	`<gml:coordinates>5,5</gml:coordinates>`
	`</gml:Point>`
		.

Polygon	`<gml:Polygon gml:id="p3"`
	`srsName="urn:ogc:def:crs:EPSG:6.6:4326">`
	`<gml:exterior>`
	`<gml:LinearRing>`
	`<gml:coordinates>`
	`5,5 28,7 44,14 47,35`
	`40,40 20,30 5,5`
	`</gml:coordinates>`
	`</gml:LinearRing>`
	`</gml:exterior>`
	`</gml:Polygon>`
		

profiles that have been proposed for public use are: (i) Point Profile, which defines a simple point geometry in GML, (ii) GML Simple Features Profile, which is the GML encoding of Simple Features for SQL discussed in Section 2.1.3, (iii) a GML profile for JPG, and (iv) a GML profile for RSS. It should be noted that GML profiles are different from application schemas. The profiles are part of the GML namespaces (OpenGIS GML) and define restricted subsets of GML. Applications schemas are XML vocabularies that are application-specific and are valid inside the application-specific namespaces. Application schemas can be built on specific GML profiles or use the full GML specification.

The GML Simple Features Profile [OGC10a] and the Simple Features for SQL presented in Section 2.1.3 have similar structure and describe similar geometries. However, the GML Simple Features Profile can also have geometries in three dimensions while Simple Features for SQL can have geometries of up to only two dimensions. In the GML Simple Features Profile, a feature can have any number of geometric properties, and every geometry should be referenced to a coordinate reference system that has 1, 2 or 3 dimensions.

Since the GML Simple Features Profile can represent similar geometries with WKT, we present in Table 2.2 two examples only showing the GML representation of a point and a polygon. The complete syntax of the GML representation of a geometry is presented in [OGC07].

GML completes our discussion of OGC standards. We now move to discuss a category of successful systems that use these standards today: relational DBMSs with geospatial data support.

Chapter 3

The New Versions of stRDF and stSPARQL

In the most recent version[1] of stRDF [KKN+11] we use OGC standards for the representation of geospatial data. The datatypes `strdf:WKT` and `strdf:GML` are introduced to represent geometries serialized using the OGC standards WKT and GML which were presented in Section 2.1.4.

Given the OGC specification for WKT, the datatype `strdf:WKT`[2] is defined as follows. The lexical space of this datatype includes finite-length sequences of characters that can be produced from the WKT grammar defined in the WKT specification, optionally followed by a semicolon and a URI that identifies the corresponding CRS. The default case is considered to be the WGS84 coordinate reference system. The value space is the set of geometry values defined in the WKT specification. These values are a subset of the union of the powersets of $\mathbb{R}^2$ and $\mathbb{R}^3$.

The lexical and value space for `strdf:GML` are defined similarly. In this case, since the GML grammar allows us to state coordinate reference systems for the geometries we define, we do not have a separate component for them in `strdf:GML` literals as we do for `strdf:WKT` literals.

The datatype `strdf:geometry` is also introduced to represent the serialization of a geometry independently of the serialization standard used. The datatype `strdf:geometry` is the union of the datatypes `strdf:WKT` and `strdf:GML`, and appropriate relationships hold for their lexical and value spaces.

Both the original [KK10] and the new version of stRDF impose minimal new requirements to Semantic Web developers that want to represent spatial objects with stRDF; all they have to do is utilize a new literal datatype. These datatypes (`strdf:WKT`, `strdf:GML` and `strdf:geometry`) can be used in the definition of geospatial ontologies needed in applications, e.g., ontologies similar to the ones defined in [Per08]. The same approach based on spatial literals has also recently been used independently in the paper [BNM10] and also in the GeoSPARQL standard [OGC10b]. The datatype `strdf:geometry` can be used in the definition of geospatial ontologies that would like to use stRDF, e.g.,

[1] The initial versions of stRDF and stSPARQL, based on the use of constraints, can be found here: [KK10]

[2] `http://strdf.di.uoa.gr/ontology`

ontologies similar to the ones defined in [Per08, OGC10b]. Let us now give some examples of modeling thematic and spatial data in stRDF (Turtle notation is used).

stRDF triples derived from GeoNames[3] that represent information about the Greek towns Olympia and Zacharo including an approximation of their geometries.

```
geonames:264637 geonames:name "Olympia";
                owl:sameAs dbpedia:Olympia_Greece;
                rdf:type noa:Town;
                strdf:hasGeometry "POLYGON((21.5 18.5, 23.5 18.5,
                                  23.5 21, 21.5 21, 21.5 18.5));
                    <http://www.opengis.net/def/crs/EPSG/0/4326>"
                    ^^strdf:WKT.

geonames:251283 geonames:name "Zacharo";
                rdf:type noa:Town;
                strdf:hasGeometry "POLYGON((19 19, 21 19, 21 21,
                                  19 21, 19 19));
                    <http://www.opengis.net/def/crs/EPSG/0/4326>"
                    ^^strdf:WKT.
```

The above triples represent some information about the Greek towns Olympia and Zacharo including an approximation of their geometry modified for the purposes of our example. GeoNames normally utilizes the W3C Basic Geo vocabulary[4] which is a basic ontology and OWL vocabulary for representing geospatial properties for Web resources. Instead of this, we use a typed literal of the data type `strdf:WKT` to define a polygon approximation of the geometry of the town encoded in WKT. The data type `strdf:WKT` (resp. `strdf:GML`) is similar to the data type `strdf:geometry` but is used to represent spatial objects using the WKT (resp. GML) serialization. Notice that a URI is used to denote that the coordinates of these geometries are given in the WGS84 using the syntax that we discussed above. This is also considered to be the default case in our approach, thus in this example the presence of this URI is optional.

stRDF triples produced from the processing chain of the National Observatory of Athens that represent burnt areas.

```
noa:BA1 a noa:BurntArea;
        noa:hasConfirmation noa:verified;
        strdf:hasGeometry "POLYGON((20 20, 20 22,
                          22 22, 22 20, 20 20))"^^strdf:WKT.
noa:BA2 a noa:BurntArea;
        noa:hasConfirmation noa:verified;
        strdf:hasGeometry "POLYGON((23 18, 24 19,
                          23 19, 23 18))"^^strdf:WKT.
noa:BA3 a noa:BurntArea.
        noa:hasConfirmation noa:verified;
        strdf:hasGeometry "POLYGON((20 15, 21 15,
                          21 16, 20 15))"^^strdf:WKT.
```

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

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

The above triples describe burnt areas that have been generated by the National Observatory of Athens and will be utilized in the following examples of this chapter.

Let us now present the main features of the new version of stSPARQL. stSPARQL is an extension of SPARQL 1.1 with functions that take as arguments spatial terms and can be used in the SELECT, FILTER, and HAVING clause of a SPARQL 1.1 query. A *spatial term* is a spatial literal (i.e., a typed literal with datatype `strdf:geometry`), a query variable that can be bound to a spatial literal, the result of a set operation on spatial literals (e.g., union), or the result of a geometric operation on spatial terms (e.g., buffer).

The new version of stSPARQL extends SPARQL 1.1 with the machinery of the OpenGIS Simple Feature Access standard which was presented in Chapter 2.1. We achieve this by defining a URI for each of the SQL functions defined in the standard and use them in SPARQL queries. For example, for the function `ST_IsEmpty` defined in the OGC-SFA standard, we introduce the SPARQL extension function

```
xsd:boolean strdf:isEmpty(strdf:geometry g)
```

which takes as argument a spatial term `g`, and returns `true` if `g` is the empty geometry. Similarly, we have defined a Boolean SPARQL extension function for each topological relation defined in OGC-SFA (topological relations for simple features), [Ege89] (Egenhofer relations) and [CBGG97] (RCC-8 relations). In this way stSPARQL supports multiple families of topological relations our users might be familiar with. Using these functions stSPARQL can express *spatial selections*, i.e., queries with a `FILTER` function with arguments a variable and a constant (e.g., `strdf:contains(?geo, "POINT(1 2)"^^strdf:WKT)`), and *spatial joins*, i.e., queries with a `FILTER` function with arguments two variables (e.g., `strdf:contains(?geoA, ?geoB)`).

The SPARQL extension functions corresponding to the SQL functions of the OpenGIS Simple Feature Access standard can be used in the SELECT clause of a SPARQL query. As a result, new spatial literals can be generated on the fly during query time based on pre-existing spatial literals. For example, to obtain the buffer of a spatial literal that is bound to the variable `?GEO`, we would use the following select expression:

```
SELECT (strdf:buffer(?geo,0.01) as ?geobuffer)
```

One of the new features in SPARQL 1.1 is support for aggregate functions. In stSPARQL we have introduced the following three aggregate functions that deal with geospatial data:

- `strdf:geometry strdf:union(set of strdf:geometry a)`, returns a geometric object that is the union of the set of input geometries.
- `strdf:geometry strdf:intersection(set of strdf:geometry a)`, returns a geometric object that is the intersection of the set of input geometries.
- `strdf:geometry strdf:extent(set of strdf:geometry a)`, returns a geometric object that is the minimum bounding box of the set of input geometries.

More aggregate functions may be added in the future if we identify a need for them in applications. stSPARQL also supports update operations (insertion,

deletion, and update of stRDF triples) conforming to the declarative update language for SPARQL, SPARQL Update 1.1, which is a current proposal of W3C.

The following examples demonstrate the functionality of stSPARQL.

Return the names of towns that have been affected by fires.

```
SELECT ?name
WHERE { ?town a noa:Town;
              geonames:name ?name;
              strdf:hasGeometry ?townGeom.
        ?ba a noa:BurntArea;
              strdf:hasGeometry ?baGeom.
        FILTER(strdf:overlap(?townGeom,?baGeom))}
```

The result of the query evaluated on the data sets of Example 3 and 3 is displayed below:

?name
"Olympia"
"Zacharo"

The query above demonstrates how to use a topological function in a query. The results of this query are the names of the towns whose geometries "spatially overlap" the geometries corresponding to areas that have been burnt.

Isolate the parts of the burnt areas that lie in coniferous forests.

```
SELECT ?ba (strdf:intersection(?baGeom,strdf:union(?fGeom)) AS ?burnt)
WHERE { ?ba a noa:BurntArea. ?ba strdf:hasGeometry ?baGeom.
        ?f a noa:Area. ?f noa:hasLandCover noa:coniferousForest.
        ?f strdf:hasGeometry ?fGeom.
        FILTER(strdf:intersects(?baGeom,?fGeom)) }
GROUP BY ?ba ?baGeom
```

The query above tests whether a burnt area intersects with a coniferous forest. If this is the case, groupings are made depending on the burnt area. The geometries of the forests corresponding to each burnt area are unioned, and their intersection with the burnt area is calculated and returned to the user. Note that only `strdf:union` is an aggregate function in the `SELECT` clause; `strdf:intersection` performs a computation involving the result of the aggregation and the value of `?baGeom` which is one of the variables determining the grouping according to which the aggregate computation is performed.

The result of the query evaluated on the data sets of Example 3 and 3 is displayed below:

?ba	?burnt
geonames:264637	"POLYGON ((20 21, 20 22, 22 22, 22 21, 21.5 21, 21.5 20, 21 20, 21 21, 20 21))"^^strdf:WKT
geonames:251283	"MULTIPOLYGON (((23.5 18.5, 23 18, 23 18.5, 23.5 18.5)), ((23.5 19, 24 19, 23.5 18.5, 23.5 19)))"^^strdf:WKT

Chapter 4

The system Strabon

In this chapter we present Strabon, a storage and query evaluation module for stRDF/stSPARQL which is currently under development by our group. In the following sections, we present Strabon's architecture and provide instructions to install and use Strabon for storing stRDF data and evaluating stSPARQL queries respectively.

4.1 Strabon architecture

We decided to base our implementation on the widely-known RDF store Sesame[1]. Even though Sesame is not the most efficient RDF store available, we decided to base our implementation on it because of its open-source nature, its layered architecture, its wide range of functionalities and the ability to integrate a 'spatially enabled' DBMS in order to exploit its variety of spatial functions and operators. Sesame was extended in order to manage both thematic and spatial RDF data that are subsequently stored in PostGIS[2]. PostGIS is an add-on module for PostgreSQL[3] that 'spatially enables' PostgreSQL by adding support for geospatial objects.

Our aim when we started the implementation was creating a layer that could be included in Sesame's software stack in a transparent way so that it does not affect Sesame's range of functionalities, while at the same time benefits by new versions of Sesame. Strabon can now be considered a Sesame SAIL that can be placed on top of an existing Sesame installation without affecting its current functionality. We were vindicated by this approach, since our second round of system development coincided with versions of Sesame that introduced support for SPARQL 1.1.

Strabon follows the modular architecture of Sesame and comprises three modules:

Query Engine. Strabon's query engine evaluates the stSPARQL query posed to the system. After a query is received, it is parsed and optimized. Then an execution plan is created and executed, and the results are returned to the client. Sesame's evaluator and optimizer have been extended in order to handle

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

[2] PostGIS, `http://postgis.refractions.net/`

[3] PostgreSQL, `http://www.postgresql.org/`

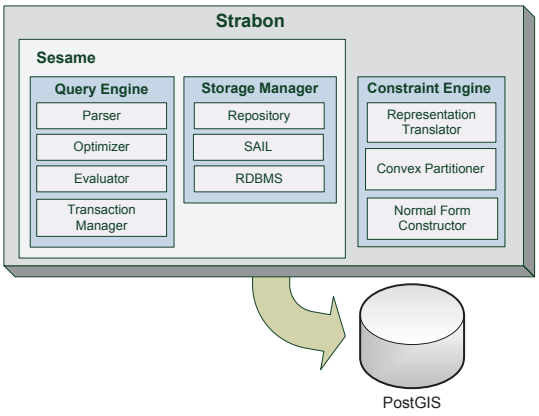


Figure 4.1: The Strabon system architecture

stSPARQL queries. The parser and transaction manager were not modified.
Storage Manager. This module handles data storage in the PostGIS RDBMS layer. We have extended Sesame's components in order to be able to store the spatial literals we described in 3.
PostGIS. PostGIS is used both for the storage of stRDF data and the evaluation of stSPARQL queries.

4.2 Installing Strabon

Using Strabon on top of PostGIS:

- Install PostgreSQL 9.0 or higher. More information can be found at `http://www.postgresql.org/download/`.

```
$> sudo apt-get install postgresql-9.1
```

- Install PostGIS 1.5 or higher. More information can be found at `http://postgis.refractions.net/download/`.

```
$> sudo apt-get install postgresql-9.1-postgis
```

- Provide a password for default user (postgres)

```
$> sudo -u postgres psql -c "ALTER USER postgres WITH PASSWORD 'postgres';"
```

4.2.1 Creating a spatially enabled database

Spatially-enabled databases permit the use of spatial function calls. MonetDB creates spatially-enabled databases by default if you have enabled the `geom` module. More information on how to create a spatially-enabled database in PostGIS can be found at `http://postgis.refractions.net/docs/`.

- Set postgis-1.5 path.

```
$> POSTGIS_SQL_PATH=‘pg_config --sharedir‘/contrib/postgis-1.5
```

- Create the spatial database that will be used as a template.

```
$> createdb -E UTF8 -T template0 template_postgis
```

- Add PLPGSQL language support.

```
$> createlang -d  template_postgis plpgsql
```

- Load the PostGIS SQL routines.

```
$> psql -d template_postgis -f $POSTGIS_SQL_PATH/postgis.sql
$> psql -d template_postgis -f $POSTGIS_SQL_PATH/spatial_ref_sys.sql
```

- Allow users to alter spatial tables.

```
$> psql -d template_postgis -c "GRANT ALL ON geometry_columns TO PUBLIC;"
$> psql -d template_postgis -c "GRANT ALL ON geography_columns TO PUBLIC;"
$> psql -d template_postgis -c "GRANT ALL ON spatial_ref_sys TO PUBLIC;"
```

- Perform garbage collection.

```
$> psql -d template_postgis -c "VACUUM FULL;"
$> psql -d template_postgis -c "VACUUM FREEZE;"
```

- Allows non-superusers the ability to create from this template.

```
$> psql -d postgres -c "UPDATE pg_database SET datistemplate='true'
   WHERE datname='template_postgis';"
$> psql -d postgres -c "UPDATE pg_database SET datallowconn='false'
   WHERE datname='template_postgis';"
```

- Create a spatially-enabled database named `endpoint`.

```
$> createdb endpoint -T template_postgis
```

4.3 Compiling and Running Strabon

The following sections describe how we can download, install and use Strabon.

4.3.1 Download and Compile Strabon

In order to download and compile Strabon, the steps below should be followed:

- Clone the source code of the TELEIOS system from the TELEIOS mercurial repository hosted by ACS.

```
$> hg clone ssh://opteleios@teleios-repo.acsys.it//home/opteleios/TELEIOS/system
```

- Change to the directory that Strabon source code resides in.

```
$> cd system/components/Strabon
```

- Build Strabon.

```
$> mvn clean package
```

4.3.2 Store

If the installation of Strabon has been completed successfully, the user can proceed to store an stRDF dataset. If PostGIS is used as the relational backend, the user has to execute the `StoreOp` class of the package `eu.earthobservatory.runtime.postgis` of module `strabon-runtime`.

This class requires the following arguments:

- `HOST`: The name of the database host (e.g., localhost)
- `PORT`: The port which your connection listens to.
- `DATABASE`: The name of the spatially-enabled database you created previously.
- `USERNAME`: The username used to connect to the database.
- `PASSWORD`: The password used to connect to the database.
- `FILE`: The stRDF document to be stored.
- `FORMAT`: The format of the triples.

We present an example on how to run Strabon from a console, connect to PostgreSQL and store an RDF file. Let us assume that the file shown in Figure 4.2 resides in `/tmp/triples.nt` (encoded according to the N3 format) and a spatially-enabled database named `endpoint` has been created:

After executing the following command the file will be stored in the `endpoint` database.

```
$> cd jars/target && java -cp $(for file in ‘ls -1 *.jar‘; do myVar=$myVar./$file":";
done;echo $myVar;) eu.earthobservatory.runtime.postgis.StoreOp localhost 5432 endpoint
postgres postgres /tmp/triples.nt N3
```

```
@prefix dc:   <http://purl.org/dc/elements/1.1/> .
@prefix :     <http://example.org/book/> .
@prefix ns:   <http://example.org/ns#> .

:book1  dc:title  "SPARQL Tutorial" .
:book1  ns:price  42 .
:book2  dc:title  "The Semantic Web" .
:book2  ns:price  23 .
```

Figure 4.2: Example triples in N3 format

4.3.3 Query

If storing has completed successfully, evaluation of stSPARQL queries can be performed using Strabon. The user has to use the corresponding `QueryOp` class of the package `eu.earthobservatory.runtime.postgis` or `eu.earthobservatory.runtime.monetdb` of module `strabon-runtime`. This class requires the following arguments:

- `HOST`: The name of the database host (e.g., localhost)
- `PORT`: The port which your connection listens to.
- `DATABASE`: The name of the spatially-enabled database you created previously.
- `USERNAME`: The username used to connect to the database.
- `PASSWORD`: The password used to connect to the database.
- `QUERY`: The stSPARQL query to evaluate.
- `FORMAT`: The format of the results.

Further to our previous example we present how one can run Strabon from a console, connect to PostgreSQL and evaluate an stSPARQL query. Suppose that the user wants to retrieve titles of books above a specific value (e.g., 30 dollars). Then, she should run the following command:

```
$> cd jars/target && java -cp $(for file in `ls -1 *.jar`; do myVar=$myVar./$file":";
done;echo $myVar;) eu.earthobservatory.runtime.postgis.QueryOp  localhost 5432 endpoint
postgres postgres "PREFIX dc: <http://purl.org/dc/elements/1.1/> PREFIX ns:
<http://example.org/ns#> SELECT ?title WHERE {?book dc:title ?title. ?book ns:price
?price. FILTER(?price > 30)}";
```

The results that will be returned are:

?title
''SPARQL Tutorial''

4.3.4 Update

In addition, one can perform stSPARQL updates in Strabon by using the `UpdateOp` class of the package `eu.earthobservatory.runtime.postgis` of module `strabon-runtime`. This class requires the following arguments:

- `HOST`: The name of the database host (e.g., localhost)
- `PORT`: The port which your connection listens to.
- `DATABASE`: The name of the spatially-enabled database you created previously.
- `USERNAME`: The username used to connect to the database.
- `PASSWORD`: The password used to connect to the database.
- `UPDATE`: The stSPARQL update query to evaluate.

Update functionality using MonetDB as a relational backend is currently work in progress, and will be fully implemented in subsequent releases of Strabon.

In the following example, the command given replaces the price of book "The Semantic Web" with a higher one.

```
$> cd jars/target && java -cp $(for file in ‘ls -1 *.jar‘; do myVar=$myVar./$file":";
done; echo $myVar;) eu.earthobservatory.runtime.postgis.UpdateOp  localhost 5432 endpoint
postgres postgres "PREFIX dc: <http://purl.org/dc/elements/1.1/> PREFIX ns:
<http://example.org/ns#> DELETE {?book ns:price ?price} INSERT {?book ns:price 32} WHERE
{?book dc:title \"The Semantic Web\". ?book ns:price ?price.}";
```

After executing this command the results of the previous query will be:

?title
‘‘SPARQL Tutorial’’
‘‘The Semantic Web’’

Chapter 5

Using Strabon in a real world scenario

In this chapter we present how Strabon can be used in a real world scenario, using linked data to compose more sophisticated queries than the ones already presented. So, we first describe briefly the corresponding ontologies, and then, we present stSPARQL queries based on them. The scenario is based on the NOA Use Case of TELEIOS, which involves the use of linked data in fire monitoring applications. Finally, some rapid mapping examples are presented.

5.1 Ontologies

The following ontologies will be presented in this section:

- the Noa Ontology
- The Corine Land Cover Ontology
- Linked Geodata
- Geonames
- Greek administrative geography

5.2 NOA Ontology

The NOA Ontology covers useful properties of the data collections that are being used or produced by the NOA processing chain, in the context of the NOA Fire Monitoring application. In the NOA Ontology, the data is distinguished into three major categories:

RawData. Instances of this class describe files with raw data.

Hotspot. Instances of this class describe hotspots that were detected by processing in raw data.

ShapeFile. An ESRI shapefile is created from all hotspots detected by a specific acquisition. Instances of this class describe these ESRI shapefiles.

Other classes that are defined in NOA Ontology are:

Coastline. Instances of this class describe areas which define coastline of Greece.

ConfirmationValue. This class contains only three instances confirmed. `noa:false\_alarm` and `noa:unknown` and `noa:confirmed`. These instances are assigned to hotspots and indicate if the hotspot has been confirmed to be a fire or a false alarm. The instance unknown represents that no confirmation has yet taken place.

Organization. This class contains instances that represent EO Organizations (e.g.: NOA, FIRMS, EFFIS)

- `noa:BurntArea`. This class corresponds to the burnt area mapping products available from FMM-2 service.
- `noa:Hotspot`. This class in its turn has a direct relation with the fire monitoring products from FMM-1 service.
- `noa:Coastline`. This class is used to describe a single geometry, the coastline of Greece.
- `noa:Region`. This class depicts all areas in Greece along with their land use.
- `clc:CorineArea`. This class imitates the Corine land cover nomenclature, since it contains subclasses that map to all the items of the three levels of the nomenclature. Instances of class `noa:Region` become instances of class `clc:CorineArea` through the property `noa:hasCorineLandCoverUse`. CorineArea class belongs to the Europe CORINE ontology, that is available online through the HarmonISA[1] project.

A description of some important properties follows:

- `noa:hasGeometry`. This property describes the geometry of an area in Well-known text (WKT) format. Well-known text is a text markup language for representing vector geometry objects on a map, spatial reference systems of spatial objects and transformations between spatial reference systems.
- `noa:hasDateTime`. This property provides information about both the time and date of an acquired hotspot. It uses the `xsd:dateTime` data type to represent this information.
- `noa:hasCorineLandCoverUse`. This property connects areas that are instances of class `noa:Region`, to instaces of class `clc:CorineArea`. This allows us to obtain all benefits offered by the CORINE ontology for the area of reference.

[1] `https://harmonisa.uni-klu.ac.at/`

5.3 Corine Land Cover Ontology

This ontology aims to modeling the CORINE Land Cover (CLC) nomenclature[2]. It has two main classes `clc:Area` and `clc:LandUse`. `clc:LandUse` is the top class of CLC taxonomy and contains classes that model the full hierarchy of land uses. `clc:Area` represents every area with a specific land use. The main properties of an instance of class `clc:Area` is

- `strdf:hasGeometry`
- `clc:hasLandUse`

The property `noa:hasGeometry` indicates where an area lies while `clc:hasLandUse` matches an area with an instance of a `clc:LandUse` subclass. The stRDF description of an area that lies in a coniferous forest

5.3.1 Linked Oped Data for Rapid Mapping

Data described above constitute the main data product for fire mapping application. However, technological solutions to such cases require integration of multiple, heterogeneous data sources in order to produce complete thematic maps. In order to increase the value of the final map product we interconnect EO data with open linked data which is a new, rapidly developing research area of Semantic Web that studies how one can make RDF data available on the Web and interconnect it with other data with the aim of increasing its value for everybody. The resulting "Web of data" has recently started being populated with geospatial data. Two representative examples of such efforts are LinkedGeoData[3] (LGD) and GeoNames[4] that are used in TELEIOS for the fire monitoring and rapid mapping applications. In the following, the datasets utilized in TELEIOS for the delivery of the fire monitoring application are described.

LinkedGeoData

LinkedGeoData is an effort aiming at enriching available geographic information published as linked data. LinkedGeoData is primarily focused on publishing OpenStreetMap (OSM)[5] data as linked data.

OpenStreetMap data model consists of three main categories:

Nodes points on earth with lat/long values

Ways ordered sequences of nodes

Relations groupings of nodes and ways

The respective ontology, LinkedGeoData, is derived mainly from OSM tags, i.e., attribute-value annotations of nodes, ways, and relations, counting up to 500 classes, 50 object properties and about 15,000 datatype properties. Points are

[2] `http://www.eea.europa.eu/publications/CORO-landcover`
[3] `http://linkedgeodata.org/`
[4] `http://www.geonames.org/`
[5] `http://www.openstreetmap.org/`

namespace	prefix
http://strdf.di.uoa.gr/ontology#	strdf
http://www.w3.org/1999/02/22-rdf-syntax-ns#	rdf
http://linkedgeodata.org/triplify/	lgd
http://linkedgeodata.org/ontology/	lgdo
http://www.w3.org/2000/01/rdf-schema#	rdfs
http://linkedgeodata.org/property/	lgdp
http://linkedgeodata.org/triplify/way10973689/	lgdw_way10973689
http://www.georss.org/georss/	georss
http://www.openlinksw.com/schemas/virtrdf#	virtrdf
http://www.w3.org/2003/01/geo/wgs84_pos#	geo
http://www.geonames.org/ontology#	gn
http://teleios.di.uoa.gr/ontologies/noaOntology.owl#	noa
http://www.w3.org/2002/07/owl#	owl
http://dbpedia.org/resource/	dbpedia

Table 5.1: Namespaces used and corresponding prefixes

the type of geometry used by OSM to represent places, cities, etc., defined by their longitude and latitude in the WGS84 coordinate reference system. Furthermore, an effort was made to match DBpedia resources with LinkedGeoData, taking into account the common classes between the two ontologies, for which the owl:sameAs link was used to connect them. Currently, the knowledge base of LGD comprises approximately 2 billion triples. For the fire mapping application we isolated only data concerning Greece which led to approximately 840,000 triples.

N3 representation of Parthenon according to LGD

```
lgd:way10973689 rdf:type         lgdo:Amenity, lgdo:Tourism, lgdo:Building,
                                 lgdo:Historic, lgdo:Attraction, lgdo:Way,
                                 lgdo:HistoricRuins ;
                lgdo:directType  lgdo:Attraction, lgdo:Building, lgdo:HistoricRuins ;
                lgdp:int_name    "Parthenon" .
lgdo:hasNodes  lgdw_way10973689:nodes .
georss:polygon  "37.9715909 23.7262015 37.9712993 23.7262856
  37.9714414 23.7270791 37.971733
  23.7269951 37.9715909 23.7262015" .

lgdw_way10973689:nodes rdf:type rdf:Seq .
lgdw_way10973689:nodes rdf:_1 lgd:node97810425 ;
                 rdf:_2 lgd:node97810428 ;
                 rdf:_3 lgd:node97810431 ;
                 rdf:_4 lgd:node97810434 ;
                 rdf:_5 lgd:node97810425 .

lgd:node97810434 geo:geometry "POINT(23.727 37.9717)"^^virtrdf:Geometry .
lgd:node97810425 geo:geometry "POINT(23.7262 37.9716)"^^virtrdf:Geometry .
lgd:node97810428 geo:geometry "POINT(23.7263 37.9713)"^^virtrdf:Geometry .
lgd:node97810431 geo:geometry "POINT(23.7271 37.9714)"^^virtrdf:Geometry .
```

```
ns2:node97810425     geo:geometry    "POINT(23.7262 37.9716)"^^virtrdf:Geometry .
```

The triples above describe Parthenon according to LGD transformation from OSM data. Please note that geometry of Parthenon is defined both with sequence of constituent nodes and their geometry and with object value of property georss:polygon that is a string with all lat/long pairs of constituent nodes.

N3 representation of Parthenon for storing in Strabon

```
lgd:way10973689 rdf:type  lgdo:Amenity, lgdo:Tourism, lgdo:Building,
lgdo:Historic, lgdo:Attraction, lgdo:Way,
lgdo:HistoricRuins;
lgdo:directType lgdo:Attraction, lgdo:Building, lgdo:HistoricRuins ;
lgdo:hasNodes  lgdw_way10973689:nodes ;
lgdp:int_name  "Parthenon" ;
georss:polygon  "POLYGON(( 23.7262015 37.9715909, 23.7262856 37.9712993,
 23.7270791 37.9714414, 23.7269951 37.971733,
23.7262015 37.9715909))"^^<http://strdf.di.uoa.gr/ontology\#WKT> ;
```

For using LGD data in a rapid mapping application in Strabon we chose to omit constituent nodes for each way and define its geometry according to data model described in [KKN+11].

GeoNames

GeoNames [6] is a gazetteer which collects both spatial and thematic information for various placenames around world. Data of GeoNames is available through various Web services but they are also published as linked data. GeoNames database contains over 10 million geographical names corresponding to over 7.5 million unique features. As in LGD we used only data about Greece that are comprised by approximately 575000 triples describing roud 41000 features.

Every feature in GeoNames ontology belongs in class Feature that has no subclasses. Features are characterized as country, town, road, etc. by codes that are assigned as object of properties gn:featureClass and gn:featureCode. Values of property gn:featureClass categorizes each feature in one of the following categories while values of property gn:featureCode narrows down these categories.

A country, state, region...

H stream, lake ....

L parks,area, ...

P city, village,...

R road, railroad

S spot, building, farm

T mountain,hill,rock,...

U undersea

[6] http://www.geonames.org/

V forest,heath,...

Furthermore, the features in GeoNames are interlinked with each other defining

childern regions that are inside the underlined feature

neighbours neigbouring countreis

nearby features features that have certain distance with the underlined feature

Finally, geonames has information only for position of a feature (described using wgs_84:lat and wgs_84:long) and not its full geometry. The following example shows how Athens is defined in GeoNames.

N3 description of Athens according to GeoNames

```
<http://sws.geonames.org/264371/> a gn:Feature ;
    wgs84_pos:lat "37.97945" ;
    wgs84_pos:long "23.71622" ;
    gn:alternateName "Athens"@en ;
    gn:countryCode "GR" ;
    gn:featureClass gn:P ;
    gn:featureCode gn:P.PPLC ;
    gn:locationMap <http://www.geonames.org/264371/athens.html> ;
    gn:name "Athens" ;
    gn:nearbyFeatures <http://sws.geonames.org/264371/nearby.rdf> ;
    gn:officialName "Athens"@gr ;
    gn:parentADM1 <http://sws.geonames.org/6692632/> ;
    gn:parentADM2 <http://sws.geonames.org/264353/> ;
    gn:parentCountry <http://sws.geonames.org/390903/> ;
    gn:parentFeature <http://sws.geonames.org/264353/> ;
    gn:population "729137" ;
    gn:wikipediaArticle <http://af.wikipedia.org/wiki/Athene> ;
    rdfs:isDefinedBy "http://sws.geonames.org/264371/about.rdf" ;
    owl:sameAs dbpedia:Athens ;
    wgs84_pos:alt "70" .
```

The triples above describe Athens according to GeoNames ontology. Please note that in order to use it in Strabon we have to replace the triples for latitude and longitute with the triple
`noa:hasGeography "POINT(23.71622 37.97945)"\^\^strdf:WKT.`

Greek administrative geography

While linked open geospatial data becomes more and more popular, many governments publish open data about administrative divisions, statistical information, etc. in their portals [GDH08]. We have taken advantage of this data openness for Greece[7] and since these haven't been published as linked data, we publish it as such focusing on the Greek administrative division and geometry of lowest divisions (municipalities).

[7] `http://geodata.gov.gr/geodata/`

5.4 Examples

In this section we demonstrate some examples about using data that are described above. The core concept of this examples is fire mapping application but our system is not limited only to such applications. First we demonstrate examples that utilize ontogies described to discover data of interest. Afterwards we demonstrate a series of examples that exhibit how Semantic Web technologies and linked open data can enhance Earth Observation applications like rapid mapping.

5.4.1 Data discovery

In this section examples about discovering data of interest are demonstrated

Retrieve shapefiles that contains acquisitions between 20:00 and 20:30 of August 21, 2010 and done by sensor MSG1_RSS

```
SELECT     ?filename
WHERE {    ?file rdf:type noa:ShpFile .
           ?file noa:hasFilename ?filename .
           ?file noa:hasAcquisitionTime ?sensingTime .
           FILTER( str(?sensingTime) > "2010-08-21T20:00:00" ) .
           FILTER( str(?sensingTime) < "2010-08-21T20:30:00" ) .
           ?file noa:isDerivedFromSensor ?sensor .
           FILTER( str(?sensor) = "MSG1_RSS" ) . }
```

The result of the query is displayed below:

?filename
MSG1_RSS_10-08-21_20:05_cloud-masked.shp
MSG1_RSS_10-08-21_20:05_plain.shp
MSG1_RSS_10-08-21_20:10_cloud-masked.shp
MSG1_RSS_10-08-21_20:10_plain.shp
MSG1_RSS_10-08-21_20:15_cloud-masked.shp
MSG1_RSS_10-08-21_20:15_plain.shp
MSG1_RSS_10-08-21_20:20_cloud-masked.shp
MSG1_RSS_10-08-21_20:20_plain.shp
MSG1_RSS_10-08-21_20:25_cloud-masked.shp
MSG1_RSS_10-08-21_20:25_plain.shp

This is a quite simple query that utilizes only data covered by NOA ontology. It searches for shapefiles (instances of class `noa:ShpFile`). Each shapefile contains information from a specific acquisition. This query searches for acquisitions by sensor MSG1_RSS taken betwenn 20:00 and 20:30 of August 21, 2010 and for every instance found the filename is projected. So, user can download the respective file.

Discover all files concerning NOA use case

```
SELECT ?filename {
  ?collection rdf:type iman:NOA-UC .
  ?file noa:belongToCollection ?collection .
  ?file noa:hasFilename ?filename . }
```

Some of the results of the query are displayed below:

?filename
H-000-MSG1__-MSG1_RSS____-IR_039___-000007___-201008211900-C_
H-000-MSG1__-MSG1_RSS____-IR_039___-000007___-201008211905-C_
H-000-MSG1__-MSG1_RSS____-IR_039___-000007___-201008211910-C_
....
MSG1_RSS_10-08-21_19:00_cloud-masked.shp
MSG1_RSS_10-08-21_19:00_plain.shp
MSG1_RSS_10-08-21_19:05_cloud-masked.shp
MSG1_RSS_10-08-21_19:05_plain.shp
MSG1_RSS_10-08-21_19:10_cloud-masked.shp
MSG1_RSS_10-08-21_19:10_plain.shp
MSG1_RSS_10-08-20_08:10_plain.shp
....

This query makes use of ImageAnnotation ontology so user who is not aware of the available data and is interested in the NOA Use Case can pose the preceding query to discover relative filenames.

Discover only shapefiles concerning NOA use case

```
SELECT ?filename {
  ?collection rdf:type iman:NOA-UC .
  ?file noa:belongToCollection ?collection .
  ?file noa:hasFilename ?filename .
  ?collection eolo:hasProcessingLevel eolo:L1 . }
```

A part of the result of the query is displayed below:

?filename
MSG1_RSS_10-08-21_19:00_cloud-masked.shp
MSG1_RSS_10-08-21_19:00_plain.shp
MSG1_RSS_10-08-21_19:05_cloud-masked.shp
MSG1_RSS_10-08-21_19:05_plain.shp
MSG1_RSS_10-08-21_19:10_cloud-masked.shp
MSG1_RSS_10-08-21_19:10_plain.shp
MSG1_RSS_10-08-20_08:10_plain.shp
....

This query is quite similar to the previous one. The difference is that it defines more restrictions. It asks only for files belonging to collections with processing level one. This means results of processing on raw data. Thus, it will retrieve filenames only of shapefiles.

5.4.2 Rapid mapping

The following examples give an overview about how Semantic Web technologies and open linked data can be used in rapid mapping applications.

As long as automatic map generation is concerned Semantic Web technologies provide tools for handling heterogeneous data in a homogeneous way, while Linking Open Data cloud supplies abundance of data in addition to internal EO data. Thus, a user has in her hands datasets covering a large variety of geospatial information from minor, but interesting, entities like fire stations or hospitals to large entities like countries and their administrative divisions. So, instead of manually combining heterogeneous data a user should only pose a stSPARQL query for each layer that she wants to depict in a map and overlay the retrieved data. In order the query results to be depicted in a map there should be exactly one geospatial variable in select clause of query. Results of such queries can be encoded in SHP or KML formats.

For example, posing the queries in following examples and overlaying their results someone can create a map that depicts fires of august of 2007 along with auxiliary information (Figure 5.2).

Get all hotspots in Peloponnese that was detected from 23$^{\text{rd}}$ to 26$^{\text{th}}$ of August 2007

```
SELECT ?h ?hGeo ?hAcqTime ?hConfidence ?hProvider ?hConfirmation ?hSensor
WHERE {
    ?h  a noa:Hotspot ;
```

```
        noa:hasGeometry ?hGeo ;
        noa:hasAcquisitionTime ?hAcqTime ;
        noa:hasConfidence ?hConfidence ;
        noa:isProducedBy ?hProvider ;
        noa:hasConfirmation ?hConfirmation ;
        noa:isDerivedFromSensor ?hSensor ;
        FILTER( "2007-08-23T00:00:00" <= str(?hAcqTime)
              && str(?hAcqTime) <= "2007-08-26T23:59:59" ) .
        FILTER(  strdf:contains("POLYGON((21.027 38.36, 23.77 38.36,
  23.77 36.05, 21.027 36.05, 21.027 38.36))"^^strdf:WKT, ?hGeo) ) .}
```

This query retrieves all hotspots that where detected during 23$^{\text{rd}}$-26$^{\text{th}}$ of August 2007. Along with uris every information about hotspots are retrieved, e.g., time of acquisition (`?hAcqTime`), provider (`?hProvider`) and their geometry (`?hGeo`), as well.

Get all areas with a characterized land use and the respective label of land use in Peloponnese

```
SELECT ?a ?aGeo ?aLandUseType
WHERE {
    ?a  a clc:Area ;
        clc:hasLandUse ?aLandUse ;
        noa:hasGeometry ?aGeo .
    ?aLandUse a ?aLandUseType .
    FILTER( ?aLandUseType = clc:AgriculturalAreas
    || ?aLandUseType = clc:ArtificialSurfaces
    || ?aLandUseType = clc:ForestsAndSemiNaturalAreas
    || ?aLandUseType = clc:Water_Bodies
    || ?aLandUseType = clc:Wetlands ) .
    FILTER(  strdf:contains("POLYGON((21.027 38.36, 23.77 38.36, 23.77 36.05,
                            21.027 36.05, 21.027 38.36))"^^strdf:WKT, ?aGeo) ) . }
```

This query retrieves all areas with a determined land use and the according first level categorization (remember that Corine Land Cover nomenclature has 3 levels of categorization)

Get all primary roads in Peloponnese

```
SELECT ?r ?rGeo
WHERE {
    ?r  a ?rType ;
        noa:hasGeometry ?rGeo .
        FILTER( ?rType = lgdo:Primary ) .
        FILTER(  strdf:contains("POLYGON((21.027 38.36, 23.77 38.36, 23.77 36.05,
                 21.027 36.05, 21.027 38.36))"^^strdf:WKT, ?rGeo) ) .}
```

This query utilizes information from LinkedGeoData dataset and retrives all primary roads of Pelloponese

Get all seats of a first-order administrative division in Peloponnese

```
SELECT ?n ?nName ?nGeo
WHERE {
    ?n  a gn:Feature ;
```

```
    noa:hasGeography ?nGeo ;
    gn:name ?nName ;
    gn:featureCode ?featureCode .
    FILTER( ?featureCode = gn:P.PPLA || ?featureCode = gn:P.PPLA2 ) .
    FILTER(  strdf:contains("POLYGON((21.51 36.41, 22.83 36.41,
    22.83 37.69, 21.51 37.69, 21.51 36.41))"^^strdf:WKT, ?nGeo)). }
```

This query asks for every feature in GeoNames that is contained in Peloponnese and its `gn:featureCode` is `gn:P.PPLA` that is the code for seats of first-order administrative divisions. Apart from information about every node (variables `?n` and `?nGeo`), the geometry (variable `?nGeo`) of the feature is also returned so it can be depicted in a map.

Get all Municipality boundaries in Peloponnese

```
SELECT ?g ?gYpesCode ?gContainer ?gLabel ( strdf:boundary(?gGeog) as ?gBoundary )
WHERE {
    ?g  a gag:Dhmos ;
        noa:hasYpesCode ?gYpesCode ;
        gag:isPartOf ?gContainer ;
        rdfs:label ?gLabel ;
        strdf:hasGeometry ?gGeog .
    FILTER(  strdf:contains("POLYGON((21.027 38.36, 23.77 38.36, 23.77 36.05,
                              21.027 36.05, 21.027 38.36))"^^strdf:WKT, ?gGeog) ) . }
```

This query makes use of information from dataset about Greek Administrative Geography. Especially, it retrieves all lowest administrative divisions (municipalities) along with some information about them (`?gYpesCode`, `?gContainer` and `?gLabel`) and their boundaries

The resulting map is shown if Figure 5.2.

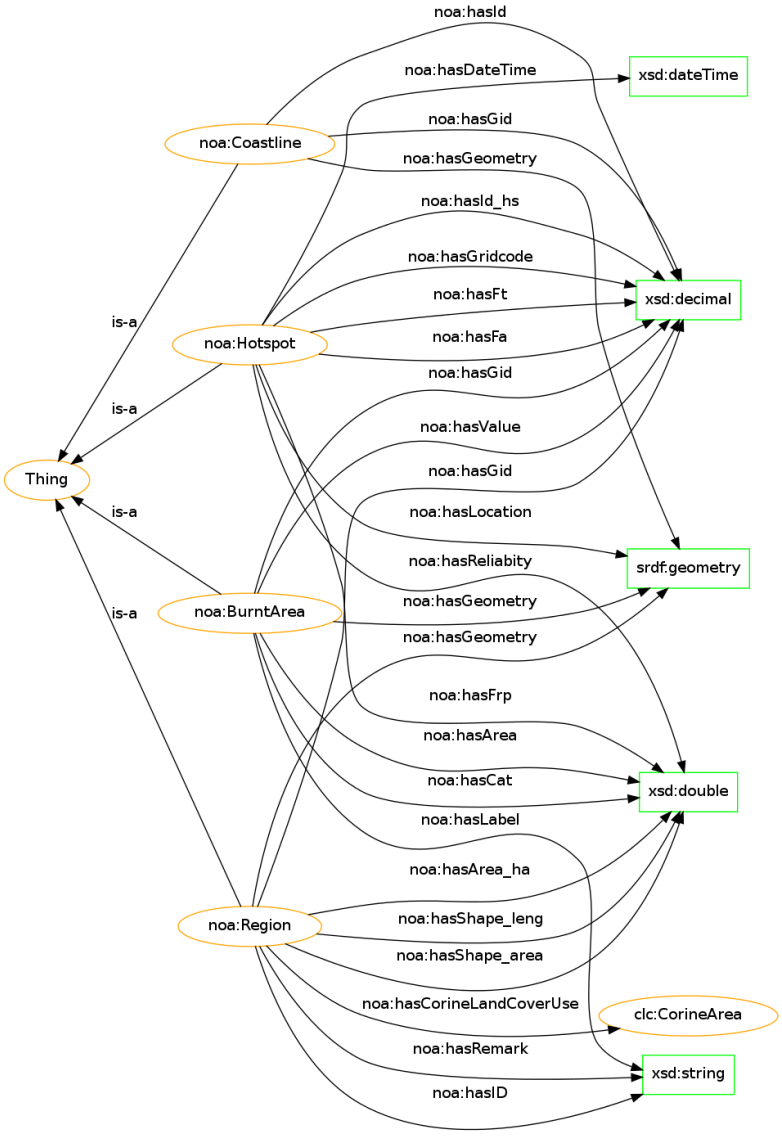


Figure 5.1: Classes in NOA's ontology

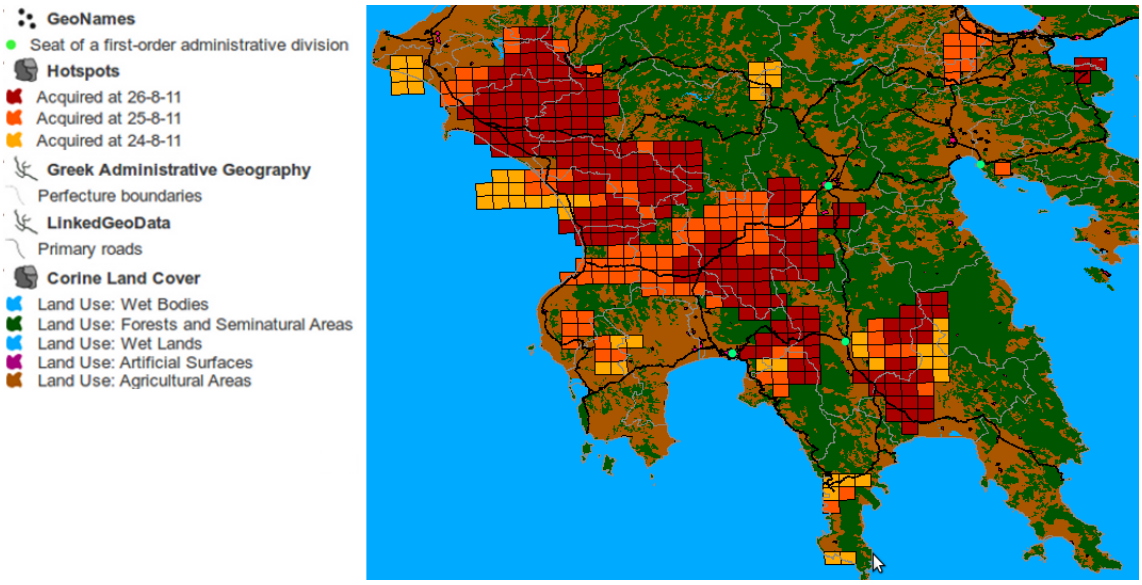


Figure 5.2: A map example that can be created by overlaying data retrieved by many queries on EO and Linked Geospatial Data

Bibliography

[BNM10] Andreas Brodt, Daniela Nicklas, and Bernhard 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*, 2010.

[CBGG97] Anthony G. Cohn, Brandon Bennett, John Gooday, and Nicholas Mark Gotts. Qualitative Spatial Representation and Reasoning with the Region Connection Calculus. *Geoinformatica*, 1(3):275–316, 1997.

[CFO93] Eliseo Clementini, Paolino Di Felice, and Peter van Oosterom. A small set of formal topological relationships suitable for end-user interaction. In *Proceedings of the Third International Symposium on Advances in Spatial Databases*, SSD '93, pages 277–295, London, UK, UK, 1993. Springer-Verlag.

[CSE94] Eliseo Clementini, Jayant Sharma, and Max J. Egenhofer. Modelling topological spatial relations: Strategies for query processing. *Computers & Graphics*, 18(6):815 – 822, 1994.

[Ege89] Max J. Egenhofer. A Formal Definition of Binary Topological Relationships. In *FODO*, volume 367 of *Lecture Notes in Computer Science*, pages 457–472. Springer, 1989.

[GDH08] J. Goodwin, C. Dolbear, and G Hart. Geographical Linked Data: The Administrative Geography of Great Britain on the Semantic Web. *Transactions in GIS*, 12:19–30, 2008.

[KK10] Manolis Koubarakis and Kostis Kyzirakos. Modeling and Querying Metadata in the Semantic Sensor Web: The Model stRDF and the Query Language stSPARQL. In Lora Aroyo, Grigoris Antoniou, Eero Hyvönen, Annette ten Teije, Heiner Stuckenschmidt, Liliana Cabral, and Tania Tudorache, editors, *ESWC (1)*, volume 6088 of *Lecture Notes in Computer Science*, pages 425–439. Springer, 2010.

[KKN+11] Manolis Koubarakis, Kostis Kyzirakos, Babis Nikolaou, Michael Sioutis, and Stavros Vassos. A data model and query language for an extension of rdf with time and space. Deliverable D2.1, TELEIOS, 2011. Available from: `http://www.earthobservatory.eu/deliverables/FP7-257662-TELEIOS-D2.1_revised.pdf`.

[LGMR05] Paul A. Longley, Michael F. Goodchild, David J. Maguire, and David W. Rhind. *Geographic Information Systems and Science*. John Wiley & Sons, 2005.

[ME01] Jim Melton and Andrew Eisenberg. SQL Multimedia and Application Packages (SQL/MM). *SIGMOD Record*, 30(4):97–102, 2001.

[OGC07] Open Geospatial Consortium Inc OGC. OpenGIS Geography Markup Language(GML) Encoding Standard. OpenGIS Implementation Standard, 08 2007. `http://portal.opengeospatial.org/files/?artifact_id=20509`.

[OGC10a] Open Geospatial Consortium Inc OGC. Geography Markup Language (GML) simple features profile. OpenGIS Implementation Standard Profile, 07 2010. `http://portal.opengeospatial.org/files/?artifact_id=39853`.

[OGC10b] Open Geospatial Consortium Inc OGC. GeoSPARQL - A geographic query language for RDF data. Proposal for an OGC Draft Candidate Standard, 04 2010.

[OGC10c] Open Geospatial Consortium Inc OGC. OpenGIS Implementation Specification for Geographic information - Simple feature access - Part 2: SQL option. OpenGIS Implementation Standard, 08 2010. `http://portal.opengeospatial.org/files/?artifact_id=25354`.

[OGC10d] Open Geospatial Consortium Inc OGC. OpenGIS Implementation Standard for Geographic information - Simple feature access - Part 1: Common Architecture. OpenGIS Implementation Standard, 08 2010. `http://portal.opengeospatial.org/files/?artifact_id=25355`.

[Per08] Matthew Perry. *A Framework to Support Spatial, Temporal and Thematic Analytics over Semantic Web Data*. PhD thesis, Wright State University, 2008.

[Sto03] Knut Stolze. SQL/MM Spatial - The Standard to Manage Spatial Data in a Relational Database System. In *BTW*, pages 247–264, 2003.