Developing Registries for the Semantic Sensor Web using stRDF and stSPARQL *

Kostis Kyzirakos, Manos Karpathiotakis and Manolis Koubarakis

Dept. of Informatics and Telecommunications
National and Kapodistrian University of Athens
Greece
{kkyzir,mk,koubarak}@di.uoa.gr

Abstract. We address the problem of registering and discovering sensors, sensor services and other related resources in the Semantic Sensor Web. We show how to develop a semantic registry for the storage and manipulation of metadata describing such resources using an implementation of the data model stRDF and the query language stSPARQL. stRDF extends RDF with the ability to represent spatial and temporal metadata using linear constraints. stSPARQL extends SPARQL for querying stRDF metadata. Together they provide a natural modeling framework for the problem at hand, since spatial and temporal metadata figure prominently in the description of sensors and related resources or services. We present Strabon, a storage and query evaluation module for stRDF/stSPARQL for the architecture of the EU FP7 project Semsor-Grid4Env and we show how to use Strabon to implement a semantic registry.

1 Introduction

Millions of sensors are currently been deployed in sensor networks around the globe and are actively collecting an enormous amount of data. Together with legacy data sources, specialized software modules (e.g., modules performing mathematical modeling and simulation) and current Web 2.0 technologies such as mashups, deployed sensor networks give us the opportunity to develop unique applications in a variety of sectors (environment, agriculture, health, transportation, surveillance, public security etc.). The term *Semantic Sensor Web (SSW)* [1] has recently been used to refer to the combination of sensor network, Web and semantic technologies with the view of addressing the opportunity that we have to unify the real and the virtual world. The SSW vision is currently shared by several research projects world-wide [1,2,3,4,5] with work concentrating on ontologies for describing sensors [6], integrating sensor information [7], linked sensor data [8], data models and query languages for capturing sensor-related metadata [9] etc.

* This work was supported in part by the European Commission project Semsor-Grid4Env (http://www.semsorgrid4env.eu/).

Another important activity in this area is the standardization activities of the Sensor Web Enablement (SWE) Working Group of the Open Geospatial Consortium (OGC) [10]. Since the standards of this working group do not emphasize semantics, it is interesting to investigate how SSW efforts can be coupled with or can extend the OGC SWE proposals [1].

An important component of SSW architectures is the *registry* which is used for discovering sensors, sensor services and other related resources. Registries store metadata about SSW resources and make this metadata available to client applications. In this paper we discuss the design and implementation of a *semantic registry* that is currently being developed in the context of the European FP7 SSW project SemsorGrid4Env[3]. In SemsorGrid4Env resource metadata is modeled using *stRDF*, a constraint-based extension of RDF, that can be used to represent *thematic*, *spatial* and *temporal* metadata. Resource metadata are queried using stSPARQL, an extension to SPARQL for querying stRDF data [9]. After introducing stRDF and stSPARQL, we present Strabon, a storage and query evaluation module for stRDF/stSPARQL. Strabon is built on top of the well-known RDF store Sesame [11] and extends Sesame's components to be able to manage thematic, spatial and temporal metadata that are stored in PostGIS. Then, we show how to use Strabon to develop a semantic registry for the SSW architecture of SemsorGrid4Env.

The organization of this paper is the following. In Section 2 we present the data model stRDF and the query language stSPARQL by means of examples. In Section 3 we present our current implementation of stRDF/stSPARQL and in Section 4 we present our implementation of a semantic registry. Comparison with related work is presented in Section 5 and in Section 6 we present our conclusions and discuss future work.

2 A Data Model and Query Language for SSW Resource Metadata

The first two questions that needed to be answered while developing a registry for the SSW are: (i) what data model should be used to encode SSW resource metadata and (ii) what query language should be used to facilitate the discovery of SSW resouces. As first pointed out in [1], SSW resource metadata can be distinguished into *thematic* (e.g., the sensor measures windspeed), *spatial* (e.g., the sensor is located in Athens) and *temporal* (e.g., the sensor was dead throughout the last two weeks). We have chosen to use RDF(S) as the base of our registry metadata model and SPARQL as the base of our query language. The motivation for this choice is pragmatic; we sacrifice the expressivity of ontology languages such as OWL in order to concentrate on efficient and scalable query evaluation which is possible in RDF(S). However, RDF(S) can only represent thematic metadata and needs to be extended if we want to model *spatial* and *temporal* information.

In [9] we have presented such an extension of RDF and SPARQL, called stRDF and stSPARQL respectively. In the rest of this section we present this data model and query language mostly by means of examples. Material in this section comes directly from [9] where the reader can also find a formal definition of the syntax and the semantics of stSPARQL query evaluation.

2.1 Data model

To develop stRDF, we follow closely the ideas of constraint databases [12,13] and especially the work on CSQL [14]. First, we define the formulae that we allow as constraints. Then, we develop stRDF in two steps. The first step is to define the model sRDF which extends RDF with the ability to represent spatial data. Then, we extend sRDF to stRDF so that thematic and spatial data with a temporal dimension can be represented.

Linear constraints Constraints will be expressed in the first-order language $\mathcal{L} = \{\leq, +\} \cup \mathbb{Q}$ over the structure $\mathcal{Q} = \langle \mathbb{Q}, \leq, +, (q)_{q \in \mathbb{Q}} \rangle$ of the linearly ordered, dense and unbounded set of the rational numbers, denoted by $\mathbb{Q}$, with rational constants and addition. The atomic formulae of this language are *linear equations* and *inequalities* of the form: $\sum_{i=1}^{p} a_i x_i \Theta a_0$, where Θ is a predicate among $=$, or $\leq$, the x_i's denote variables and the a_i's are integer constants. Note that rational constants can always be avoided in linear equations and inequalities. The multiplication symbol is used as an abbreviation i.e., $a_i x_i$ stands for $x_i + \cdots + x_i$ (a_i times).

We now define semi-linear subsets of $\mathbb{Q}^k$, where k is a positive integer.

Definition 1. *Let S be a subset of $\mathbb{Q}^k$. S is called* semi-linear *if there is a quantifier-free formula $\phi(x_1, \ldots, x_k)$ of $\mathcal{L}$ where $x_1, \ldots, x_k$ are variables such that $(a_1, \ldots, a_k) \in S$ iff $\phi(a_1, \ldots, a_k)$ is true in the structure $\mathcal{Q}$.*

We will use $\varnothing$ to denote the empty subset of $\mathbb{Q}^k$ represented by any inconsistent formula of $\mathcal{L}$.

The sRDF data model As in theoretical treatments of RDF [15], we assume the existence of pairwise-disjoint countably infinite sets I, B and L that contain IRIs, blank nodes and literals respectively. In sRDF, we also assume the existence of an infinite sequence of sets $C_1, C_2, \ldots$ that are pairwise-disjoint with I,B and L. The elements of each $C_k, k = 1, 2, \ldots$ are the quantifier-free formulae of the first-order language $\mathcal{L}$ with k free variables. We denote with C the infinite union $C_1 \cup C_2 \cup \cdots$.

Definition 2. *An* sRDF triple *is an element of the set $(I \cup B) \times I \times (I \cup B \cup L \cup C)$. If (s, p, o) is an sRDF triple, s will be called the subject, p the predicate and o the object of the triple. An* sRDF graph *is a set of sRDF triples.*

In the above definition, the standard RDF notion of a triple is extended, so that the object of a triple can be a quantifier-free formula with linear constraints. According to Definition 1 such a quantifier-free formula with k free variables is a finite representation of a (possibly infinite) semi-linear subset of $\mathbb{Q}^k$. Semi-linear subsets of $\mathbb{Q}^k$ can capture a great variety of spatial geometries, e.g., points, lines, line segments, polygons, k-dimensional unions of convex polygons possibly with holes, thus they give us a lot of expressive power. However, they cannot be used to represent other geometries that need higher-degree polynomials e.g., circles.

Example 1. The following are sRDF triples:

```
ex:s1 rdf:type, ex:Sensor .
ex:s1 ex:has_location "x=10 and y=20"^^strdf:SemiLinearPointSet
```

The above triples define a sensor and its location using a conjunction of linear constraints. The last triple is not a standard RDF triple since its object is an element of set C.

In terms of the W3C specification of RDF, sRDF can be realized as an extension of RDF with a new kind of *typed literals*: quantifier-free formulae with linear constraints (we will usually call them spatial literals). The datatype of these literals is e.g., `strdf:SemiLinearPointSet` (see Example 1 above) and can be defined using XML Schema. Alternatively, linear constraints can be expressed in RDF using MathML[1] and serialized as `rdf:XMLLiterals` as in [16]. [16] specifies a syntax and semantics for incorporating linear equations in OWL 2. We now move on to define stRDF.

The stRDF Data Model We will now extend sRDF with time. Database researchers have differentiated among user-defined time, valid time and transaction time. RDF (and therefore sRDF) supports user-defined time since triples are allowed to have as objects literals of the following XML Schema datatypes: texttxsd:`dateTime`, `xsd:time`, `xsd:date`, `xsd:gYearMonth`, `xsd: gYear`, `xsd:gMonthDay`, `xsd:gDay`, `xsd:gMonth`.

stRDF extends sRDF with the ability to represent the *valid time* of a triple (i.e., the time that the triple was valid in reality) using the approach of Gutierrez et al. [17] where the a fourth component is added to each sRDF triple.

The *time structure* that we assume in stRDF is the set of rational numbers $\mathbb{Q}$ (i.e., time is assumed to be linear, dense and unbounded). Temporal constraints are expressed by quantifier-free formulas of the language $\mathcal{L}$ defined earlier, but their syntax is limited to elements of the set C_1. *Atomic* temporal constraints are formulas of $\mathcal{L}$ of the following form: $x \sim c$, where x is a variable, c is a rational number and $\sim$ is $<, \leq, \geq, >, =$ or $\neq$. *Temporal constraints* are Boolean combinations of atomic temporal constraints using a *single* variable.

The following definition extends the concepts of triple and graph of sRDF so that thematic and spatial data with a temporal dimension can be represented.

Definition 3. *An* stRDF quad *is an sRDF triple* (a, b, c) *with a fourth component* τ *which is a temporal constraint. For quads, we will use the notation*

[1] `http://www.w3.org/Math/`

(a, b, c, τ), where the temporal constraint τ defines the set of time points that the fact represented by the triple (a, b, c) is valid in the real world. An stRDF graph *is a set of sRDF triples and stRDF quads.*

2.2 Query language

We present the syntax of stSPARQL by means of examples involving sensor networks. More examples of stSPARQL are given in [18,9].

We consider a dataset that describes static and moving sensors and uses the CSIRO/SSN Ontology [6] to describe them. The main classes of interest in the SSN ontology is the class *Feature* that describes the observed domain, the class *Sensor* that describes the sensor, the class *SensorGrounding* that describes the physical characteristics and the location of the sensor and the class *Location* that is self explained. We extend the aforementioned ontology with the properties `strdf:hasGeometry` and `strdf:hasTrajectory` with range `strdf:SemiLinearPointSet`.

The stRDF description of a static sensor that measures temperature and has a certain location is the following (`ssn` is the namespace of the CSIRO/SSN ontology and `ex` an example ontology):

```
ex:sensor1 rdf:type ssn:Sensor .
ex:sensor1 ssn:measures ex:temperature .
ex:temperature ssn:type ssn:PhysicalQuality .
ex:sensor1 ssn:supports ex:grounding1 .
ex:grounding1 rdf:type ssn:SensorGrounding .
ex:grounding1 ssn:hasLocation ex:location1 .
ex:location1 rdf:type ssn:Location .
ex:location1 strdf:hasGeometry
               "x=10 and y=10"^^strdf:SemiLinearPointSet .
```

Let us now present an example of modeling *moving sensors* in stRDF. Trajectories of moving sensors are represented by semi-linear sets of dimension 3 in variables `x`, `y`, and `t`.

```
ex:sensor2 rdf:type ssn:Sensor .
ex:sensor2 ssn:measures ex:temperature .
ex:sensor2 ssn:supports ex:grounding2 .
ex:grounding2 rdf:type ssn:SensorGrounding .
ex:grounding2 ssn:hasLocation ex:location2 .
ex:location2 rdf:type ssn:Location .
ex:location2 strdf:hasTrajectory
          "(x=10t and y=5t and 0<=t<=5) or
           (x=10t and y=25 and 5<=t<=10)"^^strdf:SemiLinearPointSet.
```

Finally, we assume that we have the stRDF descriptions of some rural area where the sensors are deployed. The stRDF description of such an area called Paros is the following:

```
ex:area1 rdf:type ex:RuralArea .
ex:area1 ex:hasName "Paros" .
ex:area1 strdf:hasGeometry
           "(-10x+13y<=-50 and y<=79 and y>=13 and
              x<=133) or (y<=13 and x<=133 and
              x+2y>=129)"^^strdf:SemiLinearPointSet .
```

Example 2. Spatial selection. Find the URIs of the static sensors that are inside the rectangle R(0,0,100,100)?

```
select ?S
where {?S rdf:type ssn:Sensor . ?G rdf:type ssn:SensorGrounding .
       ?L rdf:type ssn:Location . ?S ssn:supports ?G .
       ?G ssn:haslocation ?L . ?L strdf:hasGeometry ?GEO .
       filter(?GEO inside "0<=x<=100 and 0<=y<=100")}
```

Let us now explain the new features of stSPARQL by referring to the above example. In stSPARQL, variables can be used in basic graph patterns to refer to spatial literals denoting semi-linear point sets. They can also be used in *spatial filters*, a new kind of filter expressions introduced by stSPARQL that is used to compare *spatial terms* using spatial predicates. Spatial terms include spatial constants (finite representations of semi-linear sets e.g., `"0<=x<=10 and 0<=y<=10"`), spatial variables and complex spatial terms (e.g., `?GEO INTER "x=10 and y=10"` which denotes the intersection of the value of spatial variable `?GEO` and the semi-linear set `"x=10 and y=10"`). There are several types of spatial predicates such as topological, distance, directional, etc. that one could introduce in a user-friendly spatial query language. In the current version of stSPARQL only the topological relations of [19] can be used as predicates in a spatial filter expression e.g., `filter(?GEO1 inside ?GEO2)`.

Example 3. Intersection of an area with a trajectory. Which areas of Paros were sensed by a moving sensor and when?

```
select (?TR[1,2] INTER ?GEO) as ?SENSEDAREA  ?GEO[3] as ?T1
where {?SN rdf:type ssn:Sensor . ?RA rdf:type ex:RuralArea.
       ?X rdf:type ssn:SensorGrounding . ?Y rdf:type ssn:Location.
       ?SN ssn:supports ?X . ?X ssn:hasLocation ?Y.
       ?Y strdf:hasTrajectory ?TR . ?RA ex:hasName "Paros".
       ?RA strdf:hasGeometry ?GEO . filter(?TR[1,2] overlap ?GEO)}
```

The above query demonstrates the projection of spatial terms. Projections of spatial terms (e.g., `?TR[1,2]`) denote the projections of the corresponding point sets on the appropriate dimensions, and are written using the notation `Variable "[" Dimension1 "," ... "," DimensionN "]"`.

Example 4. Projection and spatial function application. Find the URIs and the location of the sensors that are north of Paros. Encode the sensor locations in WKT.

```
select ?SN ToWKT(?SN_LOC) AS ?WKT_LOC
where {?SN rdf:type ssn:Sensor . ?X rdf:type
ssn:SensorGrounding .
       ?Y rdf:type ssn:Location . ?RA rdf:type ex:RuralArea .
       ?RA ex:hasName "Paros" . ?RA strdf:hasGeometry ?GEO .
       ?SN ssn:supports ?X . ?X ssn:hasLocation ?Y .
       ?Y strdf:hasGeometry ?SN_LOC .
       filter(MAX(?GEO[2])<MIN(?SN_LOC[2]))}
```

The above query demonstrates the projection of spatial terms and the application of metric spatial functions to spatial terms. We allow expressions like `MAX(?GEO[2])` that return the maximum value of the unary term `?GEO[2]`. Since we cater for the case that the user would like to retrieve spatial geometries expressed in different representations, we introduced the function `ToWKT` that returns the WKT representation of a spatial geometry.

3 Implementation

In this section we present Strabon[2], a storage and query evaluation module for stRDF/stSPARQL which is currently under development by our group in the context of project SemsorGrid4Env [3]. In the first prototype of Strabon, we are supporting only the model sRDF (i.e., there is no support for time) and semi-linear sets of dimension at most 2 (i.e., only spatial data in 2 dimensions are supported). This is a realistic scenario that allows us to capture all the spatial data of SemsorGrid4Env (e.g., spatial coverage for data sources exposing UK's national Spatial Data Infrastructure datasets, programmatic data and commercial product datasets). Once our first prototype is complete and has been thoroughly evaluated, we will turn our attention to the full data model.

3.1 The Strabon system

Strabon is built on top of the well-known RDF store Sesame [11] and extends Sesame's components to be able to manage thematic and spatial metadata that are stored in PostGIS. We chose to base our system on Sesame since its layered architecture allows implementations on top of a wide variety of repositories without changing any of Sesame's other components. In Strabon, the repository is the PostGIS DBMS that gives us many of the needed functionalities for spatial data[3]. The Sesame architecture consists of the following layers (some of

[2] Strabon (Greek: Στράβων) historian, geographer and philosopher, see `http://en.wikipedia.org/wiki/Strabo/`

[3] Ideally, our repository should have been a constraint database such as CSQL [14], Dédale [20], or MLPQ [21]. Unfortunately, CSQL has never been implemented, the Dédale code is not readily available as we learned from Philippe Rigaux, and MLPQ, although it is available from Peter Revesz, is not an open source system something that is a requirement for us in the context of project SemsorGrid4Env.

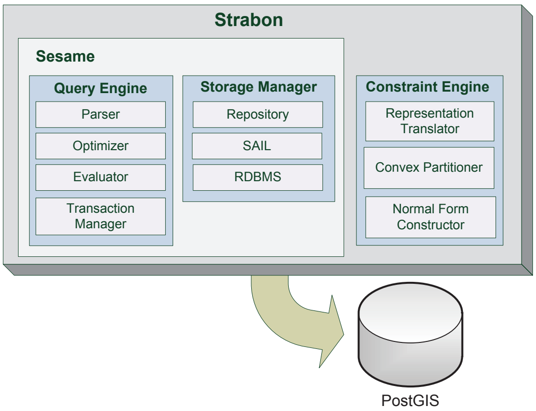

Fig. 1. The Strabon system architecture

which are shown in Figure 1 in the context of the Strabon architecture): Access APIs and Storage and Inference Layer (SAIL) APIs. The most important Access API is the Repository API that is used for querying and updating data. Storage and Inference Layer (SAIL) is the layer below the Access APIs. SAIL API's role is to offer storage support to the entire application, while the Repository API mentioned above just provides transparent access to SAIL. SAIL provides RDF-specific methods to access RDF data that are translated to calls to the underlying database.

We would like to stress that although a new generation of RDF stores recently proposed by the database community [22,23,24,25] has been shown to be more efficient than Sesame, the features of the software architecture of Sesame outlined above, its maturity as an open source RDF store, and its wide user base affected our decision to adopt it for the first implementation of Strabon. In the future, we plan to experiment with implementations which are based on more recently proposed RDF stores.

Figure 1 presents the system architecture of Strabon which consists of the following modules:

- *Query Engine*: This module is used to evaluate stSPARQL queries posed by clients. The Query Engine receives all the stSPARQL queries, parses them, optimizes them, prepares an execution plan and returns the appropriate response to the client. We have extended Sesame's Parser and Evaluator to handle stSPARQL queries and use Sesame's Optimizer and Transaction Manager as is. In Section 3.3 we describe how the Query Engine evaluates stSPARQL queries.
- *Storage Manager*: This module is responsible for storing data in the underlying PostGIS database. The *Repository*, *SAIL* and *RDBMS* layers are Sesame's modules that were described above. We have extended the *RDBMS* layer to be able to store spatial literals in PostGIS. In Section 3.2 we provide more details about the Storage Manager.

- *PostGIS*: Strabon uses PostGIS to store stRDF data and to perform part of the query evaluation process as we will show in Section 3.3.
- *Constraint Engine*: This module is responsible for processing the part of the stSPARQL queries that deal with geometries and the conversions that take place during data loading. Since we want our implementation to cater for the case that input spatial objects are expressed in other spatial data models (e.g., using OGC standards), the *Representation Translator* component has been introduced to translate spatial objects between the constraint and other equivalent representations. Spatial objects that represent non-convex geometries are convexified prior to storage to take advantage of efficient computational geometry algorithms for convex geometries. This is the job of the *Convex Partitioner* module. The *Normal Form Constructor* takes the output of the Representation Translator or the Convex Partitioner and constructs a geometry in a normal form that we describe in Section 3.2 below.

3.2 Storing stRDF data

When a user wants to store stRDF data, she makes them available in the form of an stRDF document. The document is decomposed into stRDF triples and each triple is stored in the underlying repository as we explain below.

Sesame supports two storage schemes for pure RDF data. A "monolithic" scheme where all triples are stored in a giant *Triple* table, and a vertical partitioning scheme that stores triples using one table per predicate. In both cases, the data is stored using *dictionary encoding* i.e., each URI or literal is encoded by a unique positive integer to reduce space. The mapping between the original RDF term and its encoding is stored in a separate table.

In Strabon, the storage scheme of Sesame is extended with an extra table that stores detailed information about spatial literals. In addition, spatial literals are encoded using the same dictionary encoding techniques. If the "monolithic" storing scheme is used, all stRDF triples are stored in the *Triple* table. The schema of the *Triple* table is given below.

Schema of table *Triple*

Attribute	**Type**	**Comment**
subj_id	integer	The unique encoding for the subject.
pred_id	integer	The unique encoding for the predicate.
obj_id	integer	The unique encoding for the object.

Each tuple in the table *Triple* has a *subj_id* (resp. *pred_id*, *obj_id*) that is the unique encoding of the triple's subject (resp. predicate, object).

An additional table *SpatialLiteral* is used to store information about spatial literals. The schema of the table *SpatialLiteral* is given below.

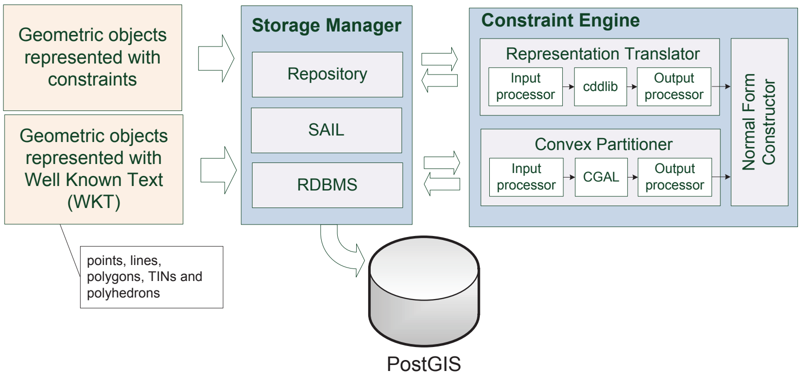
Geometric objects represented with constraints
Geometric objects represented with Well Known Text (WKT)
points, lines, polygons, TINs and polyhedrons
Storage Manager
Repository
SAIL
RDBMS
PostGIS
Constraint Engine
Representation Translator
Input processor
cddlib
Output processor
Convex Partitioner
Input processor
CGAL
Output processor
Normal Form Constructor

Fig. 2. Storing a spatial geometry in Strabon

Schema of table *SpatialLiteral*

Attribute	**Type**	**Comment**
id	integer	The unique encoding for the spatial literal.
value	varchar	The string representation of the spatial literal.
strdfgeo	geometry	The serialized object of the vector representation of a semi-linear point set in normal form.

Each tuple in the *SpatialLiteral* table has an *id* that is the unique encoding of the spatial literal. The string representation of the spatial geometry is stored in the *value* attribute. The attribute *strdfgeo* is a spatial column whose data type is the PostGIS-defined geometry type and is used to store the geometric object that is described by the spatial literal. The stored object is the vector representation of a semi-linear point set in normal form. In the case that the vertical partitioning scheme of Sesame is used, the same additions are done to it to facilitate the storage of stRDF data.

In the rest of this section we will describe in detail the conversions that take place during data loading and how we populate the table *SpatialLiteral*. Prior to storing geometries, we process the input geometries provided by the users so that each geometric object that we consequently store in PostGIS is in a *normal form* that satisfies the following requirements:

- Constraints have been transformed to the equivalent vector representation.
- The geometry is expressed as the union of convex components.
- There are neither redundant nor inconsistent geometries.
- The geometries are stored in a specific order to speed-up the conversion between the vector and constraint representation.

Let us now describe in detail how data are converted in normal form. When a user wants to store a spatial literal, the RDBMS layer of the Storage Manager initiates the procedure of converting the input geometry to normal form and storing it to PostGIS. When the input geometry is represented with constraints, the Storage Manager communicates with the Representation Translator to convert these constraints to disjunctive normal form (DNF) so that each geometry is expressed as a disjunction of conjunctions of constraints. Since each conjunction of constraints represents a convex spatial object, the DNF form is equivalent to a union of convex spatial objects. Subsequently, the Representation Translator converts the constraint representation of the geometry to the equivalent vector representation using Fukuda's cddlib library [26] that is an implementation of the Double Description Method of Motzkin et al. [27] for generating all vertices and extreme rays of a general convex polyhedron in R^d given a system of linear inequalities. Afterwards, the Normal Form Constuctor of the Constraint Engine expresses the geometry in PostGIS' Enhanced Well Known Binary (EWKB) format, a superset of OGC's WKB format. For optimization purposes we store the vertices of each spatial object in a predefined order, so that we can later compute the constraint representation of the geometry with a simple scan over the geometry's vertices. Finally, the normalized geometry is stored in the *strdfgeo* attribute of the SpatialLiteral table described previously in this section.

Our implementation also supports geometries expressed in the OGC Well-Known Text (WKT) specification[4]. In this case, the RDBMS layer of the Storage Manager communicates with the Constraint Engine, which normalizes the input geometry, but a different procedure is followed. Specifically, the Convex Partitioner simplifies the user input so that non-simple polygons are converted to a list of simple polygons. Each non-convex simple polygon is partitioned into convex sub-partitions using CGAL [28]. The Convex Partitioner uses CGAL's implementation of the simple approximation algorithm of Hertel and Mehlhorn [29] that requires $O(n)$ time and space to construct a decomposition of the initial polygon into no more than four times the optimal number of convex pieces. Finally, the Normal Form Constructor of the Constraint Engine expresses the geometry in EWKB, just like in the previous case, and the normalized geometry is stored in the *strdfgeo* attribute of the *SpatialLiteral* table.

The implementation described above has been heavily influenced by the implementation of Dédale [30]. For example, we use the same normal for storing semi-linear sets, and we also cater for the storage of non-convex geometries imported from data sources that use the vector model.

3.3 Evaluating stSPARQL queries

Another important module of Strabon is the query engine. The design of the query engine is classical and is illustrated in Figure 3. It consists of a parser, an optimizer, a query processor (evaluator) and a transaction manager. The parser extends Sesame's parser to parse stSPARQL queries. The parser generates a

[4] Although stRDF/stSPARQL is based on linear constraints, it is our intention that Strabon enables the processing of spatial data represented in common OGC formats, supported by popular DBMS or GIS. In fact, this need has arisen in SemsorGrid4Env where we have to import geometries expressed in WKT that described the spatial coverage of stored and stream data sources.

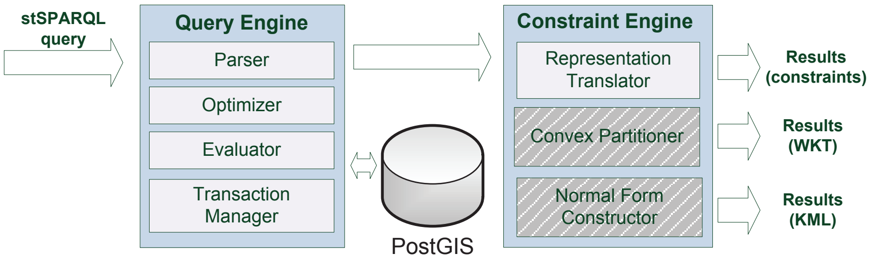

Fig. 3. Query evaluation

query model[5] that will be optimized later on by the optimizer. In the version of Strabon currently under development, the Sesame optimizer and transaction manager are used essentially as is. Later versions will deal with the new features of stSPARQL. Sesame's optimizer consists of a set of heuristics that rearranges the order of the query's triple patterns so that the query will be evaluated more efficiently. Sesame's evaluator has been extended so that it takes as input the optimized model of the query and evaluates it in a streaming fashion by taking into account the new features introduced by stSPARQL.

An stSPARQL query is evaluated as follows: In the first step, in which most of the processing takes place, the stSPARQL query is transformed to an SQL query by the evaluator depending on the storage being used. For example, the spatial filters of the stSPARQL query are mapped to the equivalent built-in functions and operators of PostGIS. Afterwards, Sesame evaluates the SQL query via PostGIS. The results of this query are returned to the evaluator which performs some post-processing to the results. For example, it calculates the result of expressions that may exist in the SELECT clause of the stSPARQL query, such as the construction of new spatial terms, using the results that have been retrieved from PostGIS in the previous step. The last step of the query processing involves displaying the final results in the format specified by the user. In the default case, results are encoded according to the SPARQL Query Results XML Format recommendation[6] and the constraint representation is used for spatial data. The user can also request that the spatial literals are encoded using OGC's Well Known Text representation. Finally, the user can also retrieve the results encoded in Keyhole Markup Language (KML)[7] which is an OGC standard and is widely used in the mapping industry.

We now move on to describe how Strabon is being used to implement a semantic registry for the SSW.

[5] "Query model" is the terminology of Sesame. Seasoned database researchers could understand this to be a query graph.

[6] `http://www.w3.org/TR/rdf-sparql-XMLres/`

[7] `http://www.opengeospatial.org/standards/kml/`

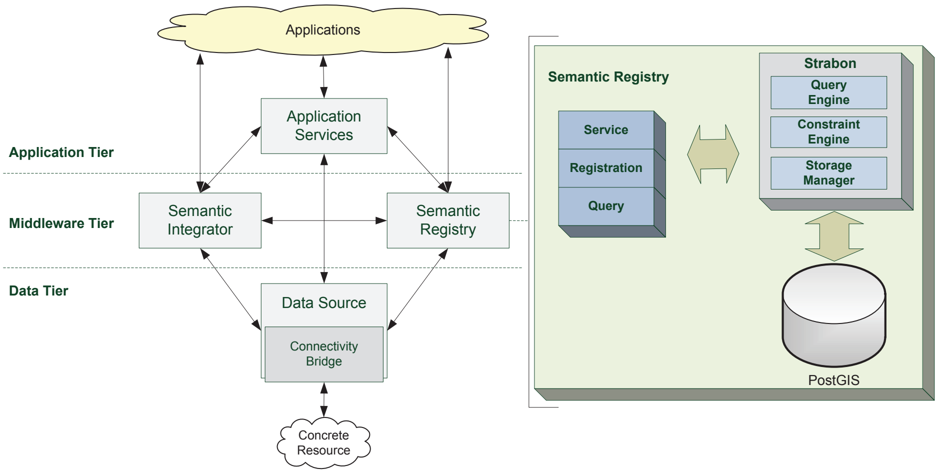

Fig. 4. SemsorGrid4Env Architecture and the Semantic Registry

4 Using Strabon to Implement a Semantic Registry

In this section, we provide a brief description of the project SemsorGrid4Env and a synopsis of the proposed conceptual Web service architecture. Besides that, our main goal is showing how Strabon can be used to implement the Semantic Registry of the SemsorGrid4Env architecture.

4.1 The SemSorGrid4Env Web service architecture

The project "Semantic Sensor Grids for Rapid Application Development for Environmental Management (SemsorGrid4Env)", aims to realize the SSW vision by developing a software platform which allows the rapid development of open large-scale semantics-based applications over data sources such as sensor networks that provide real-time information, and traditional data sources such as relational databases that may provide historical data, simulation data etc. The results of the project are demonstrated in two use cases: "Fire Risk Monitoring and Warning in NorhtWest Spain" and "Coastal and Estuarine Flood Warning in Southern UK".

The SemsorGrid4Env service-oriented architecture [31] shown in Figure 4 has three major classes of services that can be characterized as an Application Tier, a Middleware Tier and a Data Tier. The Application Tier comprises services that provide domain specific functionality to consumers and applications. For example, an application may enable a user to visualize a layer on a map, displaying temperature or tide height in a selected area. The Middleware Tier consists of two main services: the *Semantic Registry* (also called the *Semantic Registration and Discovery service*), a service that can be accessed by prospective clients who want to manage thematic and spatial metadata about sensors, sensor networks, data sources, etc. and the *Semantic Integration service* that virtualizes multiple data sources with the use of semantic technologies [7]. The Data Tier comprises services that provide access to *streaming* data sources e.g., access to real-time sensor data, and *stored* data sources e.g., sensed data stored in a relational database. Let us now give more details about the Semantic Registry.

The Semantic Registry enables the registration of SSW resources, such as sensors, sensor networks, streaming data sources with real-time data and stored data sources with historical data. These resources can then be discovered using thematic and spatial criteria, and subsequently be used in other services or applications like mashups.

Figure 4 depicts how Strabon is used to realize the Semantic Registry of the SemsorGrid4Env arhitecture. Strabon lies between clients i.e., metadata providers or metadata consumers, and PostGIS, that stores the metadata provided by metadata providers. Metadata providers intend to make public their resources by registering them with the registry, while metadata consumers aim to discover specific data resources based on criteria specific to them. The components of the SemsorGrid4Env architecture play the roles of metadata providers or consumers. A data source is a metadata provider that may produce metadata about the stream or stored data exposed via its interfaces. The Semantic Integrator may be either a metadata consumer or a metadata provider. It may query the registry to discover data sources with specific characteristics, semantically integrate them in a new virtual data source and then store the description of the new virtual data source to the registry. Applications may also be metadata providers or metadata consumers. Applications may use the registry to discover data sources that meet their needs, or publish data sources that reside outside the context of SemsorGrid4Env, such as OGC's SWE services.

The list of interfaces exposed by the Semantic Registry that are mandatory for its operation are depicted in Figure 4. Specifically the service consists of the following interfaces:

- *Service Interface.* This interface enables clients to "make well-informed and well-formed interactions" [31] with the service. For example, a client may access the Service Interface to be informed about the interfaces that are offered by the registry and the declarative query languages supported by the registry.
- *Registration Interface.* This interface enables a metadata provider to register an RDF(S) document about SSW resources (e.g. sensors, sensor networks, data sources, etc.). If a metadata provider wants to store an stRDF description of a data resource, he must use an appropriate message from the ones defined in the specification of *WS-DAI-RDF(S)-Query*[8]. The *Web Services Data Access and Integration Interface* (WS-DAI)[9] is a flexible framework defined by the Global Grid Forum, that ensures that a user dealing with grid or Web applications does not become exposed to the complexity of underlying data storage mechanisms. WS-DAI-RDF(S)-Query is a realization of

[8] `http://forge.ogf.org/sf/go/doc14074`

[9] `http://www.ogf.org/documents/GFD.74.pdf`

this interface, that is specialized for accessing resources containing RDF(S) data.

- *Query Interface*. The *WS-DAI-RDF(S)-Query* specification has been adopted for this interface as well. The Query Interface provides operations allowing a metadata consumer to query the contents of the registry in order to discover relevant resources using stSPARQL. The consumer's query is encapsulated inside a `SPARQLExecute` message, which is defined in the WS-DAI-RDF(S) realization of WS-DAI.

We have not so far defined any interface that allows harvesting of resource metadata as in other approaches [32] since no such functionality is needed by the use cases of SemsorGrid4Env. Other useful interfaces are defined in [31] that enable clients to subscribe to the registry their request to be notified whenever resources with certain characteristics are published (e.g., allowing for subscriptions to alerts regarding the status of a sensor). This has been left for future work depending on whether the functionality is needed in the project and when the relevant functionality of Strabon (e.g., continuous queries) becomes available.

4.2 Example orchestrations in the SemsorGrid4Env architecture

Let us now give examples of the SemsorGrid4Env software platform functionality, focusing on the interactions of various components with the Semantic Registry for registering and discovering data sources.

Suppose that a user wants to expose sensed data of wind speed and predicted values of wave height in the area of Southampton using a Stored Data Service, and another user wants to register a service providing the scheduled ship departure times from the ports of England. The users use the ontologies that have been developed in the context of SemsorGrid4Env to create semantically enriched property documents that contain thematic and spatial metadata about the exposed data sources. These ontologies include, among others, a sensor network and observation ontology, a domain-specific ontology describing the concept of flood and an ontology describing the content of a dataset exposed by a service (e.g., the dataset's spatial coverage, its structure, etc) [33]. Afterwards, the users invoke the registration interface of the Semantic Registry to register the semantically enriched property documents.

Let us suppose now that an end-user is looking for a Stored Data Service that observes tide height or wave height and partially covers South England. The user may utilize a mashup application that transforms the user's request in an stSPARQL query with the appropriate thematic and spatial restrictions. This query is transmitted through the Semantic Registry to Strabon encapsulated inside a `SparqlExecute` request, where it is evaluated. The result of this query is returned to the mashup application and is presented to the user.

Similarly, a domain expert may be interested in exposing a source targeted at more specialized audiences. Specifically, she may want to associate the wave height with the ship departure times from Southampton, so that potential hazards for the ships may be discovered. In order to accommodate these specialized needs, she can use the Semantic Integrator to create a “virtual” data source that unifies the data source concerning wave height and the data source related to ship departure times. Afterwards, the new data source is registered with the registry in the same way, and can be discovered in a way identical to the scenario above.

5 Related Work

There have been several approaches to developing registries for SSW architectures. Below we discuss the most relevant to our work.

In the context of OGC SWE, the functionality of a registry can be realized by some component that implements the OGC catalogue service specification[10]. The common query language developed for interoperability in this specification is based on attribute-value pairs (data model) and Boolean attribute operator value expressions (queries). In SemsorGrid4Env, we rely on stRDF and stSPARQL i.e., a semantics-based data model and query language that is more expressive than the ones offered by OGC catalogues.

In their initial paper on the SSW [1], Sheth and colleagues do not address registries explicitly but their work on extending RDF with a spatial and temporal dimension and the development of the query language SPARQL-ST [34] is a solid foundation for a data model and query language for SSW registries, exactly like our work on stRDF and stSPARQL. A comparison of SPARQL-ST and stSPARQL has already been given in [9].

IrisNet (Internet-scale Resource-Intensive Sensor Network Service) is a past project of Intel Research that was probably the first research effort to envision a world-wide sensor web built from inexpensive common computing hardware [35]. In IrisNet, data produced by the sensors and metadata describing the available sensors belong conceptually to the same XML database and are queried with the same query language (XPath).

The SenseWeb/SensorMap project of Microsoft Research provides a common platform for users to easily publish their sensor data and enable queries over live sensor data sources. The SenseWeb platform uses as a registry a relational database, called GeoDB, to store sensor metadata (e.g., publisher name, sensor location, sensor name, sensor type, data type, unit, sensor data access methods and free text descriptions) [36,37].

The Swiss Experiment (*SwissExp*) [4,38] is a collaboration of environmental science and technology research projects of various research institutions across Switzerland which has been created to provide a platform for large scale sensor network deployment, querying and exploitation. In [38], the Sensor Metadata Repository of SwissExp is introduced. Through a semantic wiki, all data and metadata are integrated and stored in the form of RDF and then different data sources can be queried through a Netapi SPARQL endpoint to the wiki.

[10] `http://portal.opengeospatial.org/files/?artifact_id=20555`

The SANY Sensor Anywhere - IST FP6 Integrated Project[11] is developing a service-oriented infrastructure for sensor networks and services specific for environmental applications. The registry of the SANY architecture behaves as a cascading catalogue where various underlying data sources may be accessed. It is based on the ORCHESTRA Catalogue Service which is a geospatial catalogue service with a semantically-enriched interface which can be adjusted to any application-specific domain [39]. The ORCHESTRA catalogue service acts like a broker that mediates the client requests to the underlying catalogue protocols such as an other SANY or ORCHESTRA catalogue, a web search engine or various OGC protocols.

The European FP6 project OSIRIS has developed a Sensor Instance Registry (SIR) that offers operations for inserting, harvesting and querying information about sensor instances [32]. SensorML is used to describe sensor instances and queries are posed using a combination of spatial constraints, temporal constraints and keywords. Query answering is carried out by using three indices, one for each of the three kinds of criteria allowed. There is another component called Sensor Observable Registry (SOR) where semantic information about phenomena observed by sensors can be stored using an ontology. SIR can use SOR to exploit semantic information to offer better answers to queries (e.g., to determine equivalent phenomena). [32] also discusses how to link the OSIRIS discovery framework with OGC catalogues.

6 Conclusions

We presented the design and implementation of a registry for registering and discovering sensors, sensor networks and other related SSW resources in the context of project SemsorGrid4Env. Our future work concentrates on evaluating our implementation of Strabon and comparing it with competitive approaches such as [40]. Finally, we will be extending Strabon to cover all of stSPARQL.

References

1. Sheth, A., Henson, C., Sahoo, S.S.: Semantic Sensor Web. Internet Computing, IEEE **12**(4) (2008) 78–83
2. Semantic Reality Project. `http://www.semanticreality.org/`
3. Semantic Sensor Grids for Rapid Application Development for Environmental Management (SemsorGrid4Env). `http://www.semsorgrid4env.eu`
4. The Swiss Experiment initiative. `http://www.swiss-experiment.ch/`
5. W3C Semantic Sensor Network Incubator Group. `http://www.w3.org/2005/Incubator/ssn/`
6. Neuhaus, H., Compton, M.: The Semantic Sensor Network Ontology: A Generic Language to Describe Sensor Assets. In: AGILE 2009 Pre-Conference Workshop Challenges in Geospatial Data Harmonisation. (2009)

[11] `http://www.sany-ip.org/`

7. Calbimonte, J.P., Corcho, O., Gray., A.J.G.: Enabling Ontology-based Access to Streaming Data Sources. In: ISWC 2010 (to appear). (2010)
8. Page, K., Roure, D.D., Martinez, K., Sadler, J., Kit, O.: Linked Sensor Data: RESTfully serving RDF and GML. In: Proceedings of the 2nd International Workshop on Semantic Sensor Networks (SSN09). (2009) 49–63
9. Koubarakis, M., Kyzirakos, K.: Modeling and Querying Metadata in the Semantic Sensor Web: The Model stRDF and the Query Language stSPARQL. In: Proceedings of the 7th Extended Semantic Web Conference, ESWC. (2010) 425–439
10. Sensor Web Enablement Working Group. `http://www.opengeospatial.org/projects/groups/sensorweb/`
11. Sesame RDF framework. `http://www.openrdf.org/`
12. Kanellakis, P., Kuper, G., Revesz, P.: Constraint Query Languages. In: PODS. (1990)
13. Revesz, P.Z.: Introduction to Constraint Databases. Springer (2002)
14. Kuper, G., Ramaswamy, S., Shim, K., Su, J.: A Constraint-based Spatial Extension to SQL. In: Proceedings of the 6th International Symposium on Advances in Geographic Information Systems. (1998)
15. Pérez, J., Arenas, M., Gutierrez, C.: Semantics and Complexity of SPARQL. In: ISWC. (2006) 30–43
16. Parsia, B., Sattler, U.: OWL 2 Web Ontology Language, Data Range Extension: Linear Equations. W3C Working Group Note (October 2009) , `http://www.w3.org/TR/2009/NOTE-owl2-dr-linear-20091027/`, last accessed February 20, 2010.
17. Gutierrez, C., Hurtado, C., Vaisman, A.: Introducing Time into RDF. IEEE TKDE (2007)
18. Kyzirakos, K., Koubarakis, M., Kaoudi, Z.: Data models and languages for registries in SemsorGrid4Env. Deliverable D3.1, SemSorGrid4Env (2009)
19. Cui, Z., Cohn, A.G., Randell, D.A.: Qualitative and Topological Relationships in Spatial Databases. In: Advances in Spatial Databases. (1993)
20. Grumbach, S., Rigaux, P., Segoufin, L.: The DEDALE System for Complex Spatial Queries. In: ACM SIGMOD. (1998)
21. Revesz, P., Chen, R., Kanjamala, P., Li, Y., Liu, Y., Wang, Y.: The MLPQ/GIS constraint database system. ACM SIGMOD **29**(2) (2000)
22. Abadi, D.J., Marcus, A., Madden, S., Hollenbach, K.J.: Scalable semantic web data management using vertical partitioning. In: VLDB. (2007)
23. Sidirourgos, L., Goncalves, R., Kersten, M.L., Nes, N., Manegold, S.: Column-store support for RDF data management: not all swans are white. VLDB **1**(2) (2008)
24. Weiss, C., Karras, P., Bernstein, A.: Hexastore: sextuple indexing for semantic web data management. VLDB **1**(1) (2008)
25. Neumann, T., Weikum, G.: RDF-3X: a RISC-style engine for RDF. VLDB **1**(1) (2008)
26. Fukuda, K.: cddlib library. `http://www.ifor.math.ethz.ch/~fukuda/cdd_home/cdd.html`
27. T.S. Motzkin, H. Raifa, G.T., Thrall, R.: The double description method. In Kuhn, H., A.W.Tucker, eds.: Contributions to theory of games. Volume 2., Princeton University Press (1953)
28. Cgal, C.G.A.L. `http://www.cgal.org`
29. Hertel, S., Mehlhorn, K.: Fast triangulation of simple polygons. In: Foundations of Computation Theory. Volume 158 of Lecture Notes in Computer Science., Springer Berlin / Heidelberg (1983) 207–218

30. Rigaux, P., Scholl, M., Segoufin, L., Grumbach, S.: Building a constraint-based spatial database system: model, languages, and implementation. Information Systems **28**(6) (2003) 563–595
31. Gray, A.J.G., Galpin, I., Fernandes, A.A.A., Paton, N.W., Page, K., Sadler, J., Koubarakis, M., Kyzirakos, K., Calbimonte, J.P., Corcho, O., Garcia, R., Diaz, V.M., Libana, I.: SemSorGrid4Env Architecture Phase I. Deliverable D1.3v1, SemSorGrid4Env (2009)
32. Jirka, S., Broring, A., Stasch, C.: Discovery Mechanisms for the Sensor Web. Sensors **9**(4) (2009) 2661–2681
33. Garcia-Castro, R., Rucabado-Rucabado, G., Hill, C., Izquierdo, A., Corcho, O.: Sensor Network Ontology Suite. Deliverable D4.3v1, SemSorGrid4Env (2009)
34. Perry, M.: A Framework to Support Spatial, Temporal and Thematic Analytics over Semantic Web Data. PhD thesis, Wright State University (2008)
35. Gibbons, P., Karp, B., Ke, Y., Nath, S., Seshan, S.: IrisNet: An Architecture for a World-Wide Sensor Web. IEEE Pervasive Computing **2**(4) (October-December 2003)
36. Nath, S., Liu, J., Zhao, F.: Challenges in Building a Portal for Sensors World-Wide. In: First Workshop on World-Sensor-Web: Mobile Device Centric Sensory Networks and Applications (WSW 2006), Boulder CO (October 2006)
37. Santanche, A., Nath, S., Liu, J., Priyantha, B., Zhao, F.: Senseweb: Browsing the physical world in real time. In: Demo Abstract, Nashville, TN (April 2006)
38. Dawes, N., Kumar, K.A., Michel, S., Aberer, K., Lehning, M.: Sensor Metadata Management and its Application in Collaborative Environmental Research. In: 4th IEEE International Conference on e-Science. (2008)
39. Uslander, T.: Specification of the Sensor Service Architecture V1. Deliverable D2.3.2, SANY Sensor Anywhere - IST FP6 Integrated Project
40. Perry, M., Hakimpour, F., Sheth, A.P.: Analyzing Theme, Space, and Time: an Ontology-based Approach. In: Proceedings of the 14th annual ACM international symposium on Advances in geographic information systems (GIS'06). (2006) 147–154