The Data Model stRDF and the Query Language stSPARQL

Kostis Kyzirakos

Dept. of Informatics and Telecommunications
National and Kapodistrian University of Athens

January 28, 2011

Outline

Motivation

The vision of the **Semantic Sensor Web**: annotate sensor data and services to enable discovery, integration, interoperability etc. (Sheth et al. 2008, SemsorGrid4Env).

Sensor annotations involve **thematic, spatial** and **temporal metadata.** Examples:

- The sensor measures temperature. (thematic)
- The sensor is located in the location represented by point (A, B). (spatial)
- The sensor measured -3^o Celsius on 27/01/2011 at 23:00. (temporal)

How about using RDF?

Good idea. But **RDF can represent only thematic metadata** properly. What can we do about spatial and temporal metadata?

Answer: Extend RDF to represent spatial and temporal metadata.

Our Approach

- Use ideas from **constraint databases** (Kanellakis, Kuper and Revesz, 1991).
  Slogan: What's in a tuple? Constraints.

- Extend RDF to a constraint database model.
  Slogan: What's in a triple? Constraints.

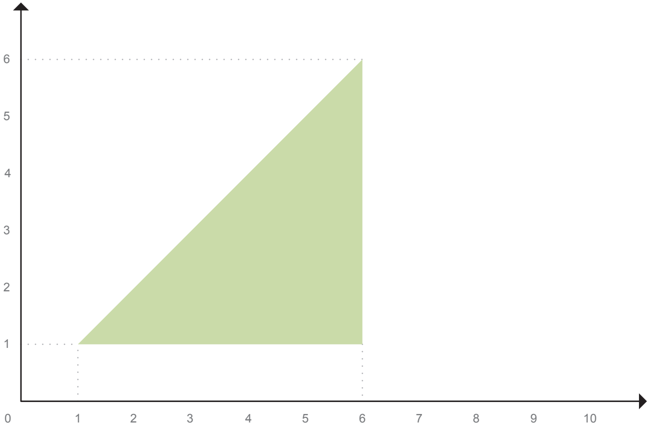
- Extend SPARQL to a constraint query language.

Our Approach - More Details

- Follow exactly the approach of CSQL (Kuper et al., 1998).
  - Nested relational model with **one level of nesting to represent point sets.**
  - Use **linear constraints** to encode these point sets (that are used to represent spatial and temporal objects).

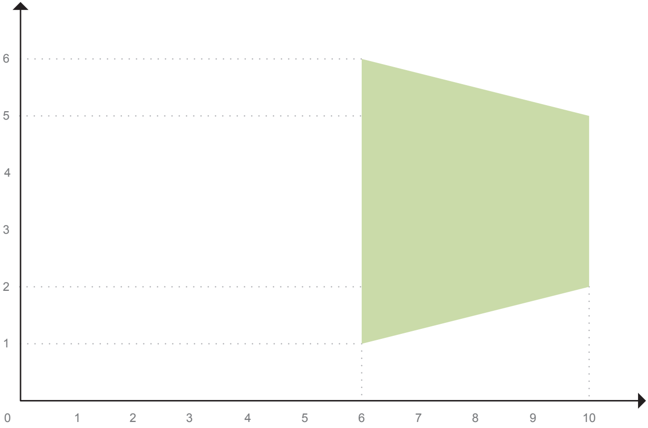
- Follow the approach of Gutierrez et al., 2005, for temporal metadata, but use **linear constraints to represent temporal intervals**.

Spatial Metadata Using Linear Constraints - Example



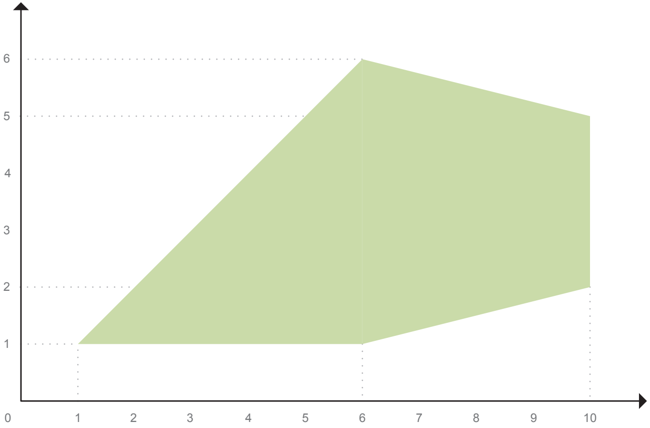
$$x <= 6 \wedge y >= 1 \wedge y <= x$$

Spatial Metadata Using Linear Constraints - Example



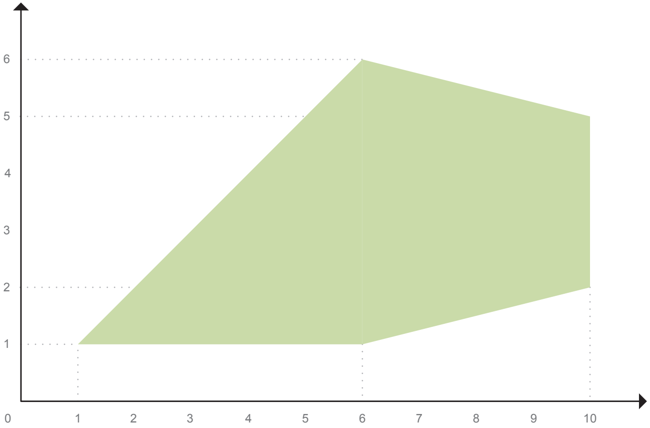
$$x \geq 6 \wedge x \leq 10 \wedge x - 4y \leq 2 \wedge x + 4y \leq 18$$

Spatial Metadata Using Linear Constraints - Example



$$(x <= 6 \wedge y >= 1 \wedge y <= x) \vee (x \geq 6 \wedge x \leq 10 \wedge x - 4y \leq 2 \wedge x + 4y \leq 18)$$

Spatial Metadata Using Well Known Text - Example




```
POLYGON((1 1, 6 1, 10 2, 10 5, 6 6, 1 1))
```

Spatial Metadata Using Linear Constraints - Definitions

- We start with a FO language $\mathcal{L} = \{\leq, +\} \cup \mathbb{Q}$ over the structure

$$\mathcal{Q} = \langle \mathbb{Q}, \leq, +, (q)_{q \in \mathbb{Q}} \rangle$$

- **Atomic formulae:** linear equations and inequalities of the form

$$(\sum_{i=1}^{p} a_i x_i) \Theta a_0$$

  where Θ is one of $=$, $\leq$ or $<$.

- Geometric objects are represented by **semi-linear point sets**: sets that can be defined by **quantifier-free formulas** of $\mathcal{L}$.

Outline

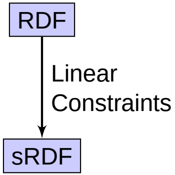
From RDF to sRDF - Example

```
ex:sensor1 rdf:type ex:Sensor .
ex:sensor1 ex:measures ex:Temperature .
ex:sensor1 ex:hasLocation ex:location1 .
```

RDF

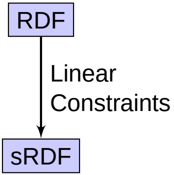
From RDF to sRDF - Example

```
ex:sensor1 rdf:type ex:Sensor .
ex:sensor1 ex:measures ex:Temperature .
ex:sensor1 ex:hasLocation ex:location1 .

  ex:location1 strdf:hasGeometry
    "x=40 and y=15"^^strdf:SemiLinearPointSet .
```



From RDF to sRDF - Example

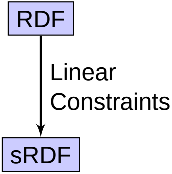
```
ex:sensor1 rdf:type ex:Sensor .
ex:sensor1 ex:measures ex:Temperature .
ex:sensor1 ex:hasLocation ex:location1 .

  ex:location1 strdf:hasGeometry
   "POINT(40 15)"^^ogc:WKT .
```



From RDF to sRDF - Example

```
ex:sensor1 rdf:type ex:Sensor .
ex:sensor1 ex:measures ex:Temperature .
ex:sensor1 ex:hasLocation ex:location1 .

  ex:location1 strdf:hasGeometry
   "POINT(40 15)"^^ogc:WKT .
```

New kind of typed literals



The sRDF data model

- Let I, B and L be the sets of IRIs, blank nodes and literals.
- Let C_k be the set of quantifier-free formulae of $\mathcal{L}$ with k free variables ($k = 1, 2, \ldots$).
- Let C be the infinite union $C_1 \cup C_2 \cup \cdots$.

Definition

- An **sRDF triple** is an element of the set $(I \cup B) \times I \times (I \cup B \cup L \cup C)$.
- An **sRDF graph** is a set of sRDF triples.

- sRDF can be realized as an extension of RDF with a new kind of **typed literals**: quantifier-free formulae with linear constraints. The datatype of these literals is `strdf:SemiLinearPointSet`.

Temporal Metadata Using Constraints - Example

$$t = 1 \vee (t \geq 5 \wedge t \leq 10) \vee (t \geq 12 \wedge t \leq 15)$$

Temporal Metadata Using Constraints - Definitions

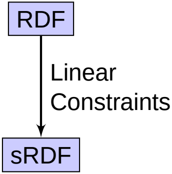
- **Time structure**: the set of rational numbers $\mathbb{Q}$ (i.e., time is assumed to be linear, dense and unbounded).
- Temporal constraints are expressed by **quantifier-free formulas** of the language $\mathcal{L}$ defined earlier, but their syntax is limited to elements of the set C_1.

Definition

- **Atomic temporal constraints**: 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**: Boolean combinations of atomic temporal constraints using a single variable.

From RDF to sRDF to stRDF - Example

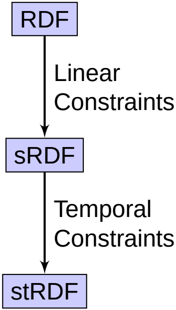
```
ex:sensor1 rdf:type ex:Sensor .
ex:sensor1 ex:measures ex:Temperature .
ex:sensor1 ex:hasLocation ex:location1 .

ex:location1 strdf:hasGeometry
      "x=40 and y=15"^^strdf:SemiLinearPointSet .
```



From RDF to sRDF to stRDF - Example

```
ex:sensor1 rdf:type ex:Sensor .
ex:sensor1 ex:measures ex:Temperature .
ex:sensor1 ex:hasLocation ex:location1 .


ex:location1 strdf:hasGeometry
      "x=40 and y=15"^^strdf:SemiLinearPointSet
      "t = 11"^^strdf:SemiLinearPointSet .
```

Valid time



The stRDF data model

stRDF extends sRDF with the ability to represent the **valid time** of a triple following the approach of Gutierrez et al., 2005:

Definition

- An **stRDF quad** (a, b, c, τ) is an sRDF triple (a, b, c) with a fourth component τ which is a temporal constraint.
- An **stRDF graph** is a set of sRDF triples and stRDF quads.

Outline

Example - Dataset I

Sensor metadata using the CSIRO/SSN ontology (Neuhaus and Compton, 2009):

```
ex:sensor1 rdf:type ssn:Sensor .
ex:sensor1 ssn:measures ex:temperature .
ex:temperature rdf: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=40 and y=15"^^strdf:SemiLinearPointSet .

ex:sensor2 rdf:type ssn:Sensor .
```

Example - Dataset I

Sensor metadata using the CSIRO/SSN ontology (Neuhaus and Compton, 2009):

```
ex:sensor1 rdf:type ssn:Sensor .
ex:sensor1 ssn:measures ex:temperature .
ex:temperature rdf: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=40 and y=15"^^strdf:SemiLinearPointSet .

ex:sensor2 rdf:type ssn:Sensor .
```

Spatial selection. Find the URIs of the 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") }
```

Queries - Dataset I

Spatial selection. Find the URIs of the 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") }
```

Queries - Dataset I

Spatial selection. Find the URIs of the 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") }
```

Answer

?S
ex:sensor1

Queries - Dataset I

Spatial selection with OPTIONAL. Find the URIs of the sensors that are optionally located inside the rectangle R(0, 0, 100, 100)?

```
select  ?S ?GEO
where { ?S rdf:type ssn:Sensor .
    optional {
        ?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")}}
```

Queries - Dataset I

Spatial selection with OPTIONAL. Find the URIs of the sensors that are optionally located inside the rectangle R(0, 0, 100, 100)?

```
select  ?S ?GEO
where { ?S rdf:type ssn:Sensor .
    optional {
        ?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")}}
```

Answer

?S	?GEO
ex:sensor1	"x=40 and y=15" ^^strdf:SemiLinearPointSet
ex:sensor2	

Example - Dataset II

Sensor observation metadata using the O&M ontology (Henson et al., 2009):

```
ex:sensor1 rdf:type ex:TemperatureSensor .
ex:TemperatureSensor rdfs:subClassOf om:Sensor .
ex:obs1 rdf:type om:Observation .
ex:obs1 om:procedure ex:sensor1 .
ex:obs1 om:observedProperty ex:temperature .
ex:temperature rdf:type om:Property .

ex:obs1 om:observationLocation ex:obslocation1 .
ex:obslocation1 rdf:type om:Location .
ex:obslocation1 strdf:hasGeometry
          "x=40 and y=15"^^strdf:SemiLinearPointSet .

ex:obs11 om:result ex:obs1Result .
ex:obs1Result rdf:type om:ResultData .
ex:obs1Result om:uom ex:Celcius .
ex:obs1Result om:value "27"
          "(10 <= t <= 11)"^^strdf:SemiLinearPointSet .
```

Example - Dataset II

Sensor observation metadata using the O&M ontology (Henson et al., 2009):

```
ex:sensor1 rdf:type ex:TemperatureSensor .
ex:TemperatureSensor rdfs:subClassOf om:Sensor .
ex:obs1 rdf:type om:Observation .
ex:obs1 om:procedure ex:sensor1 .
ex:obs1 om:observedProperty ex:temperature .
ex:temperature rdf:type om:Property .

ex:obs1 om:observationLocation ex:obslocation1 .
ex:obslocation1 rdf:type om:Location .
ex:obslocation1 strdf:hasGeometry
          "x=40 and y=15"^^strdf:SemiLinearPointSet .

ex:obs11 om:result ex:obs1Result .
ex:obs1Result rdf:type om:ResultData .
ex:obs1Result om:uom ex:Celcius .
ex:obs1Result om:value "27"
          "(10 <= t <= 11)"^^strdf:SemiLinearPointSet .
```

Example - Dataset II

Sensor observation metadata using the O&M ontology (Henson et al., 2009):

```
ex:sensor1 rdf:type ex:TemperatureSensor .
ex:TemperatureSensor rdfs:subClassOf om:Sensor .
ex:obs1 rdf:type om:Observation .
ex:obs1 om:procedure ex:sensor1 .
ex:obs1 om:observedProperty ex:temperature .
ex:temperature rdf:type om:Property .

ex:obs1 om:observationLocation ex:obslocation1 .
ex:obslocation1 rdf:type om:Location .
ex:obslocation1 strdf:hasGeometry
          "x=40 and y=15"^^strdf:SemiLinearPointSet .

ex:obs11 om:result ex:obs1Result .
ex:obs1Result rdf:type om:ResultData .
ex:obs1Result om:uom ex:Celcius .
ex:obs1Result om:value "27"
          "(10 <= t <= 11)"^^strdf:SemiLinearPointSet .
```

Example - Dataset II (cont'd)

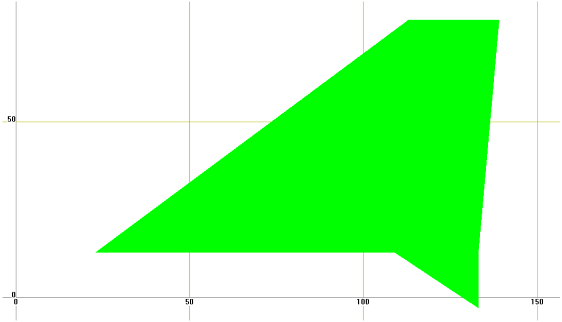
Metadata about geographical areas:

```
ex:area1 rdf:type ex:SuburbanArea .
ex:area1 ex:hasName "Dudweiller" .
ex:area1 strdf:hasGeometry
           "(-x+1.4y<=-5.3 and y<=79 and -y<=-13 and
            x-0.1y<=132) or (y<=13 and x<=133 and
           -x-1.5y<=-128)"^^strdf:SemiLinearPointSet .
```

Example - Dataset II (cont'd)

Metadata about geographical areas:

```
ex:area1 rdf:type ex:SuburbanArea .
ex:area1 ex:hasName "Dudweiller" .
ex:area1 strdf:hasGeometry
           "(-x+1.4y<=-5.3 and y<=79 and -y<=-13 and
            x-0.1y<=132) or (y<=13 and x<=133 and
           -x-1.5y<=-128)"^^strdf:SemiLinearPointSet .
```



Queries - Dataset II

Spatial and temporal selection. Find the values of all observations that were valid at time 11 and the suburban area they refer to.

```
select  ?V ?RA
where { ?OBS rdf:type om:Observation .
        ?LOC rdf:type om:Location .
        ?R rdf:type om:ResultData .
        ?RA rdf:type ex:SuburbanArea .
        ?OBS om:observationLocation ?LOC .
        ?LOC strdf:hasGeometry ?OBSLOC .
        ?OBS om:result ?R .
        ?R om:value ?V ?T .
        ?RA strdf:hasGeometry ?RAGEO .
        filter(?T contains "t = 11" &&
               ?RAGEO contains ?OBSLOC) }
```

Queries - Dataset II

Spatial and temporal selection. Find the values of all observations that were valid at time 11 and the suburban area they refer to.

```
select  ?V ?RA
where { ?OBS rdf:type om:Observation .
        ?LOC rdf:type om:Location .
        ?R rdf:type om:ResultData .
        ?RA rdf:type ex:SuburbanArea .
        ?OBS om:observationLocation ?LOC .
        ?LOC strdf:hasGeometry ?OBSLOC .
        ?OBS om:result ?R .
        ?R om:value ?V ?T .
        ?RA strdf:hasGeometry ?RAGEO .
        filter(?T contains "t = 11" &&
               ?RAGEO contains ?OBSLOC) }
```

Queries - Dataset II

Spatial and temporal selection. Find the values of all observations that were valid at time 11 and the suburban area they refer to.

```
select  ?V ?RA
where { ?OBS rdf:type om:Observation .
        ?LOC rdf:type om:Location .
        ?R rdf:type om:ResultData .
        ?RA rdf:type ex:SuburbanArea .
        ?OBS om:observationLocation ?LOC .
        ?LOC strdf:hasGeometry ?OBSLOC .
        ?OBS om:result ?R .
        ?R om:value ?V ?T .
        ?RA strdf:hasGeometry ?RAGEO .
        filter(?T contains "t = 11" &&
               ?RAGEO contains ?OBSLOC) }
```

Answer

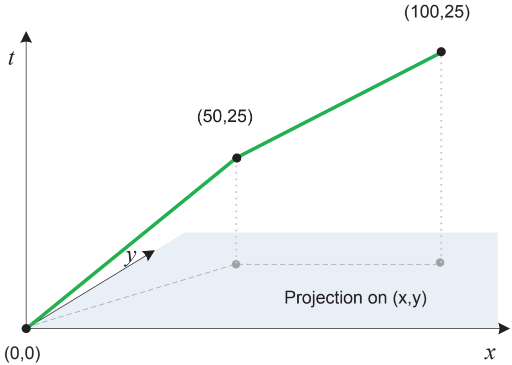
?V	?RA
"27"	ex:area1

Example - Dataset III

Moving sensor metadata using the CSIRO/SSN ontology:

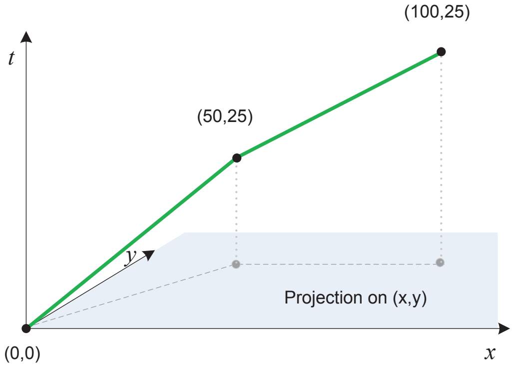
```
ex:sensor2 rdf:type ssn:Sensor .
ex:sensor2 ssn:measures ex:temperature .
ex:sensor2 ssn:supports ex:grounding2 .
ex:grounding2 rdf:type ssn:SensorGrounding .
ex:grounding2 ssn:hasLocation ex:location2 .
ex:location2 rdf:type ssn:Location .
```

Example - Dataset III (cont'd)




```
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.
```

Example - Dataset III (cont'd)




```
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.
```

Example - Dataset III (cont'd)

Metadata about geographical areas:

```
ex:area1 rdf:type ex:SuburbanArea .
ex:area1 ex:hasName "Dudweiller" .
ex:area1 strdf:hasGeometry
        "(-x+1.4y<=-5.3 and y<=79 and -y<=-13
         and x-0.1y<=132) or (y<=13 and x<=133
         and -x-1.5y<=-128)"
         ^^strdf:SemiLinearPointSet .
```

Queries - Dataset III

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

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

Queries - Dataset III

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

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

Queries - Dataset III

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

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

Answer

?SENSEDAREA	?T1
"((y=0.5x and 0<=x<=50) or (x=50 and 25<=y<=50)) and ((y<=79 and -y<=-13 and -x+1.4y<=-5.2 and x-0.1y<=132) or (y<=13 and x<=133 and -x-1.5y<=-128))" ^^strdf:SemiLinearPointSet	"0 <= t <= 10"^^strdf: SemiLinearPointSet

What is new in stSPARQL syntax?

- *k*-ary spatial terms
  - quantifier-free formulas (**constants**)
  - **spatial variables**
  - **projections** of *k*-ary spatial terms
  - the result of **set operations** on *k*-ary spatial terms: *intersection, union, difference*
  - the result of **geometric operations** on *k*-ary spatial terms: *boundary, buffer, minimum bounding box*
- Metric spatial terms
  - *VOL, AREA, LEN, MAX, MIN*
- **Select clause**: construction of new spatial terms
  - *intersection, union, difference, projection of spatial terms*
- **Where clause**: Quad patterns to refer to the valid time of a triple
- **Filter clause**:
  - **Spatial predicates** (topological): *disjoint, touch, equals, inside, coveredby, contains, covers, overlap*
  - **Temporal predicates**: *before, equal, meets, overlaps, during, starts, finishes*
  - a linear equation or inequality of $\mathcal{L}$ with metric spatial terms in the place of variables

What is new in stSPARQL syntax?

- *k*-ary spatial terms
  - quantifier-free formulas (**constants**)
  - **spatial variables**
  - **projections** of *k*-ary spatial terms
  - the result of **set operations** on *k*-ary spatial terms: *intersection, union, difference*
  - the result of **geometric operations** on *k*-ary spatial terms: *boundary, buffer, minimum bounding box*
- Metric spatial terms
  - *VOL, AREA, LEN, MAX, MIN*
- **Select clause**: construction of new spatial terms
  - *intersection, union, difference, projection of spatial terms*
- **Where clause**: Quad patterns to refer to the valid time of a triple
- **Filter clause**:
  - **Spatial predicates** (topological): *disjoint, touch, equals, inside, coveredby, contains, covers, overlap*
  - **Temporal predicates**: *before, equal, meets, overlaps, during, starts, finishes*
  - a linear equation or inequality of $\mathcal{L}$ with metric spatial terms in the place of variables

What is new in stSPARQL syntax?

- k-ary spatial terms
  - quantifier-free formulas (**constants**)
  - **spatial variables**
  - **projections** of k-ary spatial terms
  - the result of **set operations** on k-ary spatial terms: *intersection, union, difference*
  - the result of **geometric operations** on k-ary spatial terms: *boundary, buffer, minimum bounding box*
- Metric spatial terms
  - *VOL, AREA, LEN, MAX, MIN*
- **Select clause**: construction of new spatial terms
  - *intersection, union, difference, projection of spatial terms*
- **Where clause**: Quad patterns to refer to the valid time of a triple
- **Filter clause**:
  - **Spatial predicates** (topological): *disjoint, touch, equals, inside, coveredby, contains, covers, overlap*
  - **Temporal predicates**: *before, equal, meets, overlaps, during, starts, finishes*
  - a linear equation or inequality of $\mathcal{L}$ with metric spatial terms in the place of variables

What is new in stSPARQL syntax?

- *k*-ary spatial terms
  - quantifier-free formulas (**constants**)
  - **spatial variables**
  - **projections** of *k*-ary spatial terms
  - the result of **set operations** on *k*-ary spatial terms: *intersection, union, difference*
  - the result of **geometric operations** on *k*-ary spatial terms: *boundary, buffer, minimum bounding box*
- Metric spatial terms
  - *VOL, AREA, LEN, MAX, MIN*
- **Select clause**: construction of new spatial terms
  - *intersection, union, difference, projection of spatial terms*
- **Where clause**: Quad patterns to refer to the valid time of a triple
- **Filter clause**:
  - **Spatial predicates** (topological): *disjoint, touch, equals, inside, coveredby, contains, covers, overlap*
  - **Temporal predicates**: *before, equal, meets, overlaps, during, starts, finishes*
  - a linear equation or inequality of $\mathcal{L}$ with metric spatial terms in the place of variables

What is new in stSPARQL syntax?

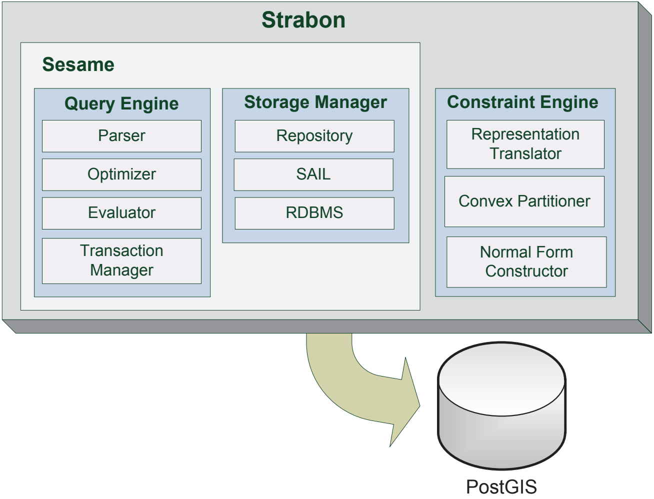
- *k*-ary spatial terms
  - quantifier-free formulas (**constants**)
  - **spatial variables**
  - **projections** of *k*-ary spatial terms
  - the result of **set operations** on *k*-ary spatial terms: *intersection, union, difference*
  - the result of **geometric operations** on *k*-ary spatial terms: *boundary, buffer, minimum bounding box*
- Metric spatial terms
  - *VOL, AREA, LEN, MAX, MIN*
- **Select clause**: construction of new spatial terms
  - *intersection, union, difference, projection of spatial terms*
- **Where clause**: Quad patterns to refer to the valid time of a triple
- **Filter clause**:
  - **Spatial predicates** (topological): *disjoint, touch, equals, inside, coveredby, contains, covers, overlap*
  - **Temporal predicates**: *before, equal, meets, overlaps, during, starts, finishes*
  - a linear equation or inequality of $\mathcal{L}$ with metric spatial terms in the place of variables

stSPARQL Semantics

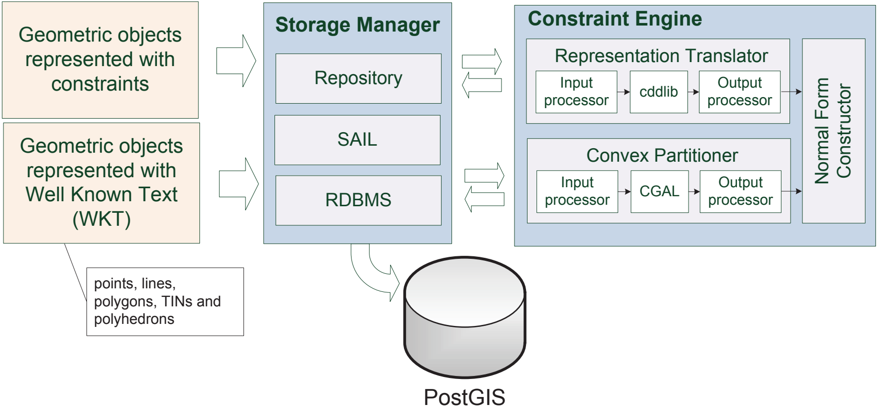
- Extension of the SPARQL semantics of (Perez et al., 2006).
  - Extend the concept of mapping
    - A variable is mapped to an element of C (quantifier-free formulas of $\mathcal{L}$ with $1,2,\ldots k$ free variables).
  - The semantics of AND, OPT, UNION remain the same.
  - We need to define carefully the evaluation of spatial terms and the semantics of spatial and temporal filters.
  - Closure property
    - The output of any operation or query is representable in stRDF.

Outline

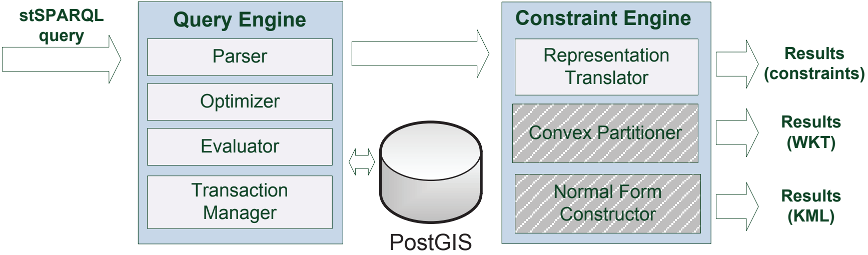
The System Strabon



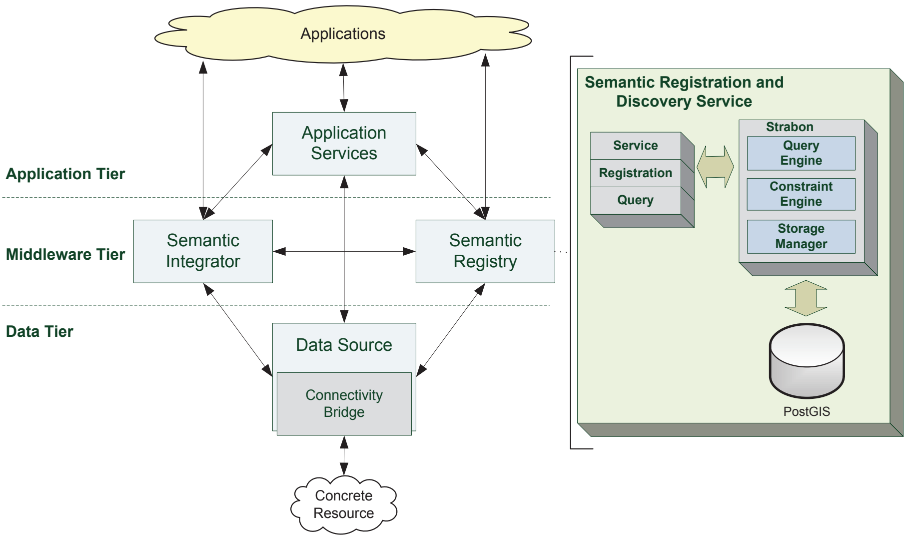
Storing stRDF Data



Evaluating stSPARQL Queries



SemsorGrid4Env Architecture and the Semantic Registry



Outline

Related Work

- Introducing Time into RDF (Gutierrez et al, 2005)
- SPARQL-ST (Perry, 2008)
- SPAUK (Kollas, 2007)
- Deep integration of spatial query processing into native RDF triple stores (Brodt et al, 2010)

Future Work

- Study the complexity of stSPARQL query processing.
- Port the implementation to MonetDB.
- Demonstrate that our approach can be implemented efficiently in comparison with competitive approaches.
- Use our implementation to publish public spatial datasets as Linked Open Data.

Thank you!

Thank you for your attention!

Fragen?