Part 2:
Spatial and Temporal Data in RDF: stRDF/stSPARQL and GeoSPARQL
Common Approach

• The two proposals (stRDF/stSPARQL and GeoSPARQL) offer constructs for:
  o Developing **ontologies** for spatial and temporal data.
  o Encoding **spatial and temporal data** that use these ontologies **in RDF**.
  o **Extending SPARQL** to query spatial and temporal data.
Two Proposals

• stRDF/stSPARQL

• GeoSPARQL
The data model stRDF

- An extension of RDF for the representation of geospatial information that changes over time.

- Geospatial dimension:
  - Spatial data types are introduced.
  - Geospatial information is representing using spatial literals of these datatypes.
  - OGC standards WKT and GML are used for the serialization of spatial literals.

- Temporal dimension (later)

- Proposed independently and around the same time as GeoSPARQL (starting with an ESWC 2010 paper by Koubarakis and Kyzirakos).
Spatial Datatypes

strdf:geometry rdf:type rdfs:Datatype;
   rdfs:subClassOf rdfs:Literal.

strdf:WKT rdf:type rdfs:Datatype;
   rdfs:subClassOf strdf:geometry.

strdf:GML rdf:type rdfs:Datatype;
   rdfs:subClassOf strdf:geometry.
Example Ontology: Administrative Geography of Greece

Legend:
- rdfs:subClassOf
- gag:belongsTo

Geometry property

gag:hasGeometry
strdf:WKT

strdf:geometry
strdf:GML
strdf:WKT

Country
Decentralized Administration
Region
Regional Unit
Municipality
Municipality Unit
Municipal Community
Local Community

Administrative Area
Example Ontology: Administrative Geography of Greece

Geometry property

Legend
- rdfs:subClassOf
- gag:belongsTo

Administrative Area
- gag:hasGeometry
- strdf:WKT
- gag:hasOfficialName
- xsd:string
- gag:hasPopulation
- xsd:integer

Country
Decentralized Administration
Region
Regional Unit
Municipality
Municipality Unit
Municipal Community
Local Community

strdf:geometry
strdf:GML
strdf:WKT
Example Data in stRDF

gag:Olympia

gag:name "Ancient Olympia";
rdf:type gag:MunicipalCommunity .

gag:Olympia gag: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
Example (cont’d)

```
gag:Olympia
  rdf:type gag:MunicipalCommunity;
  gag:name "Ancient Olympia";
  gag:population "184"^^xsd:int;
  gag:hasGeometry "POLYGON (((25.37 35.34, ...)))"^^strdf:WKT.

gag:OlympiaMUnit
  rdf:type gag:MunicipalityUnit;
  gag:name "Municipality Unit of Ancient Olympia".

gag:OlympiaMunicipality
  rdf:type gag:Municipality;
  gag:name "Municipality of Ancient Olympia".

gag:Olympia gag:belongsTo gag:OlympiaMUnit .
gag:OlympiaMUnit gag:belongsTo gag:OlympiaMunicipality.
```
More Examples

- Corine Land Use/Land Cover
  (http://www.eea.europa.eu/publications/COR0-landcover)

- Burnt Area Products (project TELEIOS,
  http://www.earthobservatory.eu/)
Corine Land Use/Land Cover
Corine Land Use/Land Cover in stRDF (http://www.linkedopendata.gr)

clc:Area_24015134
   rdf:type clc:Area ;
   clc:hasCode "312"^^xsd:decimal;
   clc:hasID "EU-203497"^^xsd:string;
   clc:hasArea_ha "255.5807904"^^xsd:double;
   clc:hasGeometry "POLYGON((15.53 62.54, ...))"^^strdf:WKT;
   clc:hasLandUse clc:ConiferousForest .

Geometry Property
Burnt Area Products
(http://www.earthobservatory.eu/ontologies/noaOntology.owl)
Burnt Area Products

noa:ba_15
  rdf:type noa:BurntArea;
  noa:isProducedByProcessingChain "static thresholds"^^xsd:string;
  noa:hasAcquisitionTime "2010-08-24T13:00:00"^^xsd:dateTime;
  noa:hasGeometry "MULTIPOLYGON(((
    393801.42 4198827.92, ..., 393008 424131)));
  <http://www.opengis.net/def/crs/EPSG/0/2100>"^^strdf:WKT.

Geometry Property
We define a SPARQL extension function for each function defined in the OpenGIS Simple Features Access standard

Basic functions

- Get a property of a geometry
  \texttt{xsd:int strdf:dimension(strdf:geometry A)}
  \texttt{xsd:string strdf:geometryType(strdf:geometry A)}
  \texttt{xsd:int strdf:srid(strdf:geometry A)}

- Get the desired representation of a geometry
  \texttt{xsd:string strdf:asText(strdf:geometry A)}
  \texttt{xsd:string strdf:asGML(strdf:geometry A)}

- Test whether a certain condition holds
  \texttt{xsd:boolean strdf:isEmpty(strdf:geometry A)}
  \texttt{xsd:boolean strdf:isSimple(strdf:geometry A)}
stSPARQL: Geospatial SPARQL 1.1

Functions for testing topological spatial relationships

- **OGC Simple Features Access**

  xsd:boolean strdf:equals(strdf:geometry A, strdf:geometry B)
  xsd:boolean strdf:disjoint(strdf:geometry A, strdf:geometry B)
  xsd:boolean strdf:intersects(strdf:geometry A, strdf:geometry B)
  xsd:boolean strdf:touches(strdf:geometry A, strdf:geometry B)
  xsd:boolean strdf:crosses(strdf:geometry A, strdf:geometry B)
  xsd:boolean strdf:within(strdf:geometry A, strdf:geometry B)
  xsd:boolean strdf:contains(strdf:geometry A, strdf:geometry B)
  xsd:boolean strdf:overlaps(strdf:geometry A, strdf:geometry B)
  xsd:boolean strdf:relate(strdf:geometry A, strdf:geometry B,
                            xsd:string intersectionPatternMatrix)

- **Egenhofer**

- **RCC-8**
stSPARQL: Geospatial SPARQL 1.1

Spatial analysis functions

- **Construct new geometric objects from existing geometric objects**

  strdf:geometry strdf:boundary(strdf:geometry A)
  strdf:geometry strdf:envelope(strdf:geometry A)
  strdf:geometry strdf:convexHull(strdf:geometry A)
  strdf:geometry strdf:intersection(strdf:geometry A, strdf:geometry B)
  strdf:geometry strdf:union(strdf:geometry A, strdf:geometry B)
  strdf:geometry strdf:difference(strdf:geometry A, strdf:geometry B)
  strdf:geometry strdf:symDifference(strdf:geometry A, strdf:geometry B)
  strdf:geometry strdf:buffer(strdf:geometry A, xsd:double distance, xsd:anyURI units)

- **Spatial metric functions**

  xsd:float strdf:distance(strdf:geometry A, strdf:geometry B, xsd:anyURI units)
  xsd:float strdf:area(strdf:geometry A)

- **Spatial aggregate functions**

  strdf:geometry strdf:union(set of strdf:geometry A)
  strdf:geometry strdf:intersection(set of strdf:geometry A)
  strdf:geometry strdf:extent(set of strdf:geometry A)
stSPARQL: Geospatial SPARQL 1.1

Select clause
- Construction of new geometries (e.g., strdf:buffer(geo, 0.1, uom:metre))
- Spatial aggregate functions (e.g., strdf:union(geo))
- Metric functions (e.g., strdf:area(geo))

Filter clause
- Functions for testing topological spatial relationships between spatial terms (e.g., strdf:contains(G1, strdf:union(G2, G3)))
- Numeric expressions involving spatial metric functions (e.g., strdf:area(G1) ≤ 2*strdf:area(G2)+1)
- Boolean combinations

Having clause
- Boolean expressions involving spatial aggregate functions and spatial metric functions or functions testing for topological relationships between spatial terms (e.g., strdf:area(strdf:union(geo))>1)
Return the names of local communities that have been affected by fires

```
SELECT  ?name
WHERE  {
  ?comm rdf:type gag:LocalCommunity;
  gag:name ?name;
  gag:hasGeometry ?commGeo .
  ?ba rdf:type noa:BurntArea;
  noa:hasGeometry ?baGeo .

  FILTER (strdf:overlaps(?commGeo, ?baGeo))
}
```

Spatial Function
Find all burnt forests near local communities

```sparql
SELECT ?ba ?baGeom
WHERE {
  ?r rdf:type clc:Region;
  clc:hasGeometry ?rGeom;
  clc:hasCorineLandUse ?f.
  ?f rdfs:subClassOf clc:Forest.
  ?c rdf:type gag:LocalCommunity;
  gag:hasGeometry ?cGeom.
  ?r RDF:type clc:HasCorineLandUse.

  ?ba rdf:type noa:BurntArea;
  noa:hasGeometry ?baGeom.

  FILTER ( strdf:intersects(?rGeom, ?baGeom) &&
  strdf:distance(?baGeom, ?cGeom, uom:metre) < 200 )
}
```

**Spatial Functions**
Compute the parts of burnt areas that lie in coniferous forests.

```
SELECT ?burntArea
(strdf:intersection(?baGeom,
    strdf:union(?fGeom))
AS ?burntForest)
WHERE {

?burntArea rdf:type noa:BurntArea;
  noa:hasGeometry ?baGeom.

?forest rdf:type clc:Region;
  clc:hasLandCover clc:ConiferousForest;
  clc:hasGeometry ?fGeom.

FILTER(strdf:intersects(?baGeom,?fGeom))
}
GROUP BY ?burntArea ?baGeom
```
Time dimensions in Linked Data

**User-defined time**: A time value (literal) with no special semantics.

**Valid time**: The time when a fact (represented by a triple) is true in the modeled reality.

**Transaction time**: The time when the triple is current in the database.
The time dimension of stRDF: The valid time of triples

The following extensions are introduced in stRDF:

- **Timeline**: the (discrete) value space of the datatype `xsd:dateTime` of XML-Schema.

- Two kinds of time primitives are supported: **time instants** and **time periods**.
  - A **time instant** is an element of the time line.
  - A **time period** is an expression of the form \([B, E)\) or \([B, E]\) or \((B, E]\) or \((B, E)\) where \(B\) and \(E\) are time instants called the beginning and ending time of the period.

- The new datatype `strdf:period` is introduced.
The time dimension of stRDF (cont’d)

• **Triples** are extended to **quads**.

• A **temporal triple (quad)** is an expression of the form 
  \[ s p o \ t. \]
  where \( s p o \) is an RDF triple and \( t \) is a time instant or time period called the **valid time** of the triple.

• The **temporal constants** **NOW** and **UC** (“until changed”) are introduced.
An example with valid time

25/08/06 Forest 25/08/07 25/08/09
An example with valid time

clc:region1 clc:hasLandCover clc:Forest
"[2006-08-25T11:00:00+02, "UC")"^^strdf:period .
An example with valid time

\[
\text{clc:region1 clc:hasLandCover clc:Forest} \\
\text{"[2006-08-25T11:00:00+02, "UC")"} \text{^^strdf:period}.
\]
An example with valid time

```
clc:region1 clc:hasLandCover clc:Forest
  "[2006-08-25T11:00:00+02, "UC"]"^^strdf:period .

noa:bal rdf:type noa:BurntArea
  "[2007-08-25T11:00:00+02, "UC"]"^^strdf:period .
```
An example with valid time

clc:region1 clc:hasLandCover clc:Forest
  "[2006-08-25T11:00:00+02,2007-08-25T11:00:00+02)"^^strdf:period .

noa:bal rdf:type noa:BurntArea
  "[2007-08-25T11:00:00+02, "UC")"^^strdf:period .
An example with valid time

clc:region1 clc:hasLandCover clc:Forest
"[2006-08-25T11:00:00+02,2007-08-25T11:00:00+02)"^^strdf:period .

noa:ba1 rdf:type noa:BurntArea
"[2007-08-25T11:00:00+02,2009-08-25T11:00:00+02)"^^strdf:period .

clc:region1 clc:hasLandCover clc:AgriculturalArea
"[2009-08-25T11:00:00+02, "UC")"^^strdf:period .
The time dimension of stSPARQL

The following extensions are introduced:

- **Triple patterns** are extended to **quad patterns** (the last component is a **temporal term**: variable or constant)

- **Temporal extension functions** are introduced:
  - Allen's temporal relations (e.g., `strdf:after`)
  - Period constructors (e.g., `strdf:period_intersect`)
  - Temporal aggregates (e.g., `strdf:maximalPeriod`)

Example Query

- Find the **current** land cover of all areas in the dataset

```
SELECT ?clc
WHERE {
  ?R rdf:type clc:Region .
  FILTER(strdf:duration ("NOW", ?t1))
}
```
Two Proposals

- stRDF/stSPARQL
- GeoSPARQL
GeoSPARQL

GeoSPARQL is an OGC standard.

Functionalities similar to stRDF/stSPARQL:

- Geometries are represented using literals of spatial datatypes.
- Literals are serialized using WKT and GML.
- The same families of functions are offered for querying geometries.

Functionalities beyond stSPARQL:

- High level ontologies inspired from GIS terminology.
- Topological relations can now be asserted as well so that reasoning and querying on them is possible.
- A query rewriting mechanism.

Functionalities of stSPARQL that are not included in GeoSPARQL:

- Geospatial aggregate functions
- Temporal dimension
GeoSPARQL Components

Parameters
- Serialization
  - WKT
  - GML
- Relation Family
  - Simple Features
  - RCC-8
  - Egenhofer
GeoSPARQL Core

Defines two **top level classes** that can be used to organize geospatial data.
GeoSPARQL Geometry Extension

Provides vocabulary for asserting and querying data about the geometric attributes of a feature.
Example Ontology: Greek Administrative Geography
Greek Administrative Geography
Greek Administrative Geography

Legend:
- rdfs:subClassOf
- gag:belongsTo

Diagram:
- geo:Feature
- geo:hasGeometry
- geo:hasDefaultGeometry
- Administrative Unit
  - gag:code
  - gag:name
  - gag:population
  - xsd:integer
  - xsd:string

Countries:
- Country
- Decentralized Administration
- Region
- Regional Unit
- Municipality
- Municipality Unit
- Municipal Community
- Local Community
Example Data

gag:Olympia
  rdf:type gag:MunicipalCommunity;
  gag:name "Ancient Olympia";
  gag:population "184"^^xsd:int;
  geo:hasGeometry ex:polygon1.

ex:polygon1
  rdf:type geo:Geometry;
  geo:asWKT "http://www.opengis.net/def/crs/OGC/1.3/CRS84
  POLYGON((21.5 18.5, 23.5 18.5, 23.5 21, 21.5 21, 21.5 18.5))"
  ^^sf:wktLiteral.
Non-Topological Query Functions of the Geometry Extension

- The following non-topological query functions are also offered:
  - `geof:distance`
  - `geof:buffer`
  - `geof:convexHull`
  - `geof:intersection`
  - `geof:union`
  - `geof:difference`
  - `geof:symDifference`
  - `geof:envelope`
  - `geof:boundary`
GeoSPARQL Topology Vocabulary Extension

- The extension is parameterized by the family of topological relations supported.
  - Topological relations for simple features
    - The Egenhofer relations e.g., `geo:ehMeet`
    - The RCC-8 relations e.g., `geo:rcc8ec`
gag:Olympia
  rdf:type  gag:MunicipalCommunity;
  gag:name  "Ancient Olympia".

gag:OlympiaMUnit
  rdf:type  gag:MunicipalityUnit;
  gag:name  "Municipality Unit of Ancient Olympia".

gag:OlympiaMunicipality
  rdf:type  gag:Municipality;
  gag:name  "Municipality of Ancient Olympia".

gag:Olympia geo:sfWithin gag:OlympiaMUnit.

gag:OlympiaMUnit geo:sfWithin gag:OlympiaMunicipality.

Simple Features
topological relation
GeoSPARQL: An example

Find the **municipality unit** that contains the community of Ancient Olympia

```sparql
SELECT ?m
WHERE {
  ?m rdf:type gag:MunicipalityUnit.
  ?m geo:sfContains gag:Olympia.
}
```

**Answer:** ?m = gag:OlympiaMUnit
GeoSPARQL: An example

Find the **municipality** that contains the community of Ancient Olympia

```sparql
SELECT  ?m
WHERE  {
?m rdf:type gag:Municipality.
?m geo:sfContains gag:Olympia.
}
```

Answer?
Example (cont’d)

The answer to the previous query is

\[ ?m = \text{gag:OlympiaMunicipality} \]

GeoSPARQL does not tell you how to compute this answer which needs reasoning about the transitivity of relation \text{geo:sfContains}.

Options:
- Use rules
- Use constraint-based techniques
The Geometry Topology Extension

- Offers vocabulary for **querying topological properties** of geometry literals.

- **Simple Features**
  - `geof:relate`
  - `geof:sfEquals`
  - `geof:sfDisjoint`
  - `geof:sfIntersects`
  - `geof:sfTouches`
  - `geof:sfCrosses`
  - `geof:sfWithin`
  - `geof:sfContains`
  - `geof:sfOverlaps`

- **Egenhofer** (e.g., `geof:ehDisjoint`)
- **RCC-8** (e.g., `geof:rcc8dc`)
Example Query

Return the names of local communities that have been affected by fires

```sparql
SELECT ?name
WHERE { 
  ?comm rdf:type gag:LocalCommunity;
  gag:name ?name;
  geo:hasGeometry ?commGeo .
  ?ba rdf:type noa:BurntArea;
  geo:hasGeometry ?baGeo .

  FILTER (geof:sfOverlaps(?commGeo,?baGeo))
}
```
GeoSPARQL Query Rewrite Extension

- Provides a collection of RIF rules that use topological extension functions to establish the existence of topological predicates.

- Example: given the RIF rule named `geor:sfWithin`, the serializations of the geometries of `gag:Athens` and `gag:Greece` named `AthensWKT` and `GreeceWKT` and the fact that

  \[
  \text{geof:sfWithin}(\text{AthensWKT}, \text{GreeceWKT})
  \]

  returns true from the computation of the two geometries, we can derive the triple

  \[
  \text{gag:Athens geo:sfWithin gag:Greece}
  \]

- One possible implementation is to re-write a given SPARQL query.
RIF Rule

Forall ?f1 ?f2 ?g1 ?g2 ?g1Serial ?g2Serial
(?f1[geo:spatialWithin->?f2] :-
  Or(
    And (?f1[geo:defaultGeometry->?g1]
      ?f2[geo:defaultGeometry->?g2]
      ?g1[ogc:asGeometryLiteral->?g1Serial]
      ?g2[ogc:asGeometryLiteral->?g2Serial]
      External(geof:spatialWithin (?g1Serial,?g2Serial)))
    And (?f1[geo:defaultGeometry->?g1]
      ?g1[ogc:asGeometryLiteral->?g1Serial]
      ?f2[ogc:asGeometryLiteral->?g2Serial]
      External(geof:spatialWithin (?g1Serial,?g2Serial)))
    And (?f2[geo:defaultGeometry->?g2]
      ?f1[ogc:asGeometryLiteral->?g1Serial]
      ?g2[ogc:asGeometryLiteral->?g2Serial]
      External(geof:spatialWithin (?g1Serial,?g2Serial)))
    And (?f1[ogc:asGeometryLiteral->?g1Serial]
      ?f2[ogc:asGeometryLiteral->?g2Serial]
      External(geof:spatialWithin (?g1Serial,?g2Serial))))

Example

Find all features that are inside the municipality of Ancient Olympia

```sparql
SELECT ?feature
WHERE {
}
```
Rewritten Query

```
SELECT ?feature
WHERE { {?feature geo:sfWithin geonames:Olympia } }
UNION
  FILTER (geof:sfWithin (?featureSerial, ?olSerial)) } }
  geonames:Olympia geo:asWKT ?olSerial .
  FILTER (geof:sfWithin (?featureSerial, ?olSerial)) } }
  FILTER (geof:sfWithin (?featureSerial, ?olSerial)) } }
  geonames:Olympia geo:asWKT ?olSerial .
  FILTER (geof:sfWithin (?featureSerial, ?olSerial)) } }
```
GeoSPARQL RDFS Entailment Extension

- Specifies the RDFS entailments that follow from the class and property hierarchies defined in the other components e.g., the Geometry Extension.

- Systems should use an implementation of RDFS entailment to allow the derivation of new triples from those already in a graph.
Example

Given the triples

\[
\text{ex:f1 geo:hasGeometry ex:g1.}
\]
\[
\text{geo:hasGeometry rdfs:domain geo:Feature.}
\]

we can infer the following triples:

\[
\text{ex:f1 rdf:type geo:Feature .}
\]
\[
\text{ex:f1 rdf:type geo:SpatialObject .}
\]
Readings

• Material from the Strabon web site (http://strabon.di.uoa.gr).

• The following tutorial paper which introduces to the topic of linked geospatial data:

• The following paper which introduces stSPARQL and Strabon:

• The following paper which introduces the temporal features of stSPARQL and Strabon:

• The GeoSPARQL standard found at http://www.opengeospatial.org/standards/geosparql
• The following paper which introduces the RDF\(^i\) framework:
  Charalampos Nikolaou and Manolis Koubarakis. Incomplete Information in RDF.

• The following paper which introduces the benchmark Geographica:
  \[\text{http://cgi.di.uoa.gr/~koubarak/publications/Geographica.pdf}\]