Querying Semantically Enriched Sensor Observations

Jean-Paul Calbimonte[1],
Hoyoung Jeung[2], Oscar Cocho[1]

[1] Universidad Politécnica de Madrid
[2] Ecole Polytechinque Fédérale de Lausanne

ESWC Semantic BPM Workshop
Heraklion, 29 May 2011

Speaker: Jean-Paul Calbimonte

Outline

- Motivation
- Sensor Streaming Data
- Ontology-based data access
- Mapping streams to RDF
- SPARQL-Stream queries
- Semantic Integration
- Implementation

Introduction & Scope

- Sensor technologies
  - Ubiquitous data capture
  - Data processing
  - Cheap
  - Noisy, Unreliable
  - Low computational, power resources, storage

- Streaming Data
  - Continuously appended data
  - Potentially infinite
  - Time-stamped tuples
  - Continuous queries
  - Changes of values over time
  - Latest used in queries

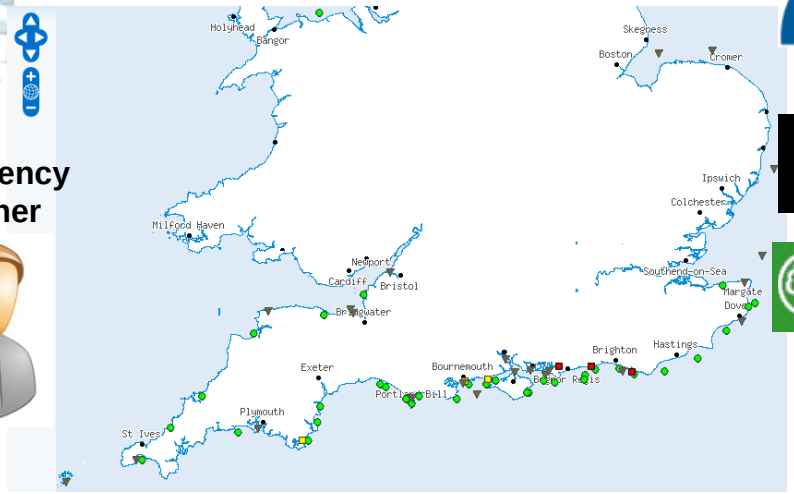
Streaming Data

(t9, a1, a2, ... , an)
(t8, a1, a2, ... , an)
(t7, a1, a2, ... , an)
...
...
(t1, a1, a2, ... , an)
...
...

• Applications in security surveillance, healthcare provision, environmental monitoring, you name it.

Motivation

Flood risk alert:
South East England



Emergency planner

Channel Coastal Observatory → Real-time data: Wave, Wind, Tide

Met Office → Meteorological forecasts

Environment Agency → Flood defences data

... → Other sources

- Detect conditions likely to cause a flood
- Present data model in terms of the user domain: e.g. Flood risk assessment

Example:

- “provide me with the wind speed observations average over the last minute in the Solent region, if it is higher than the average of the last 2 to 3 hours”

Motivation

Geo Researcher

Environmental and GeoScience research Swiss Alps

→ Real-time data { Snow, Wind, Radiation

→ Meteorological forecasts

→ Other sources

...
...
...

- Collect massive amounts of environmental data for researchers
- Heterogeneous sources
- Ad-hoc, specific schemas in every sensor
- No interoperability
- No common model, no agreed terminology
- Unable to find the relevant data

Motivation

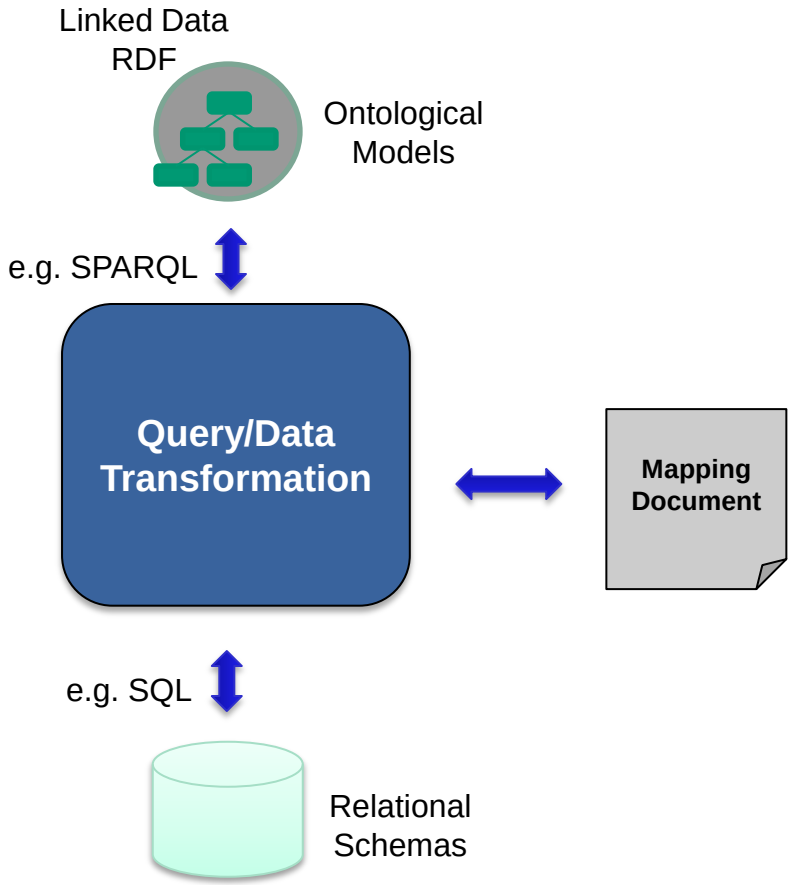
- Ontologies can be used as a common model
  - Achieve logical transparency in access to data.
  - Hide to the user where and how data are stored.
  - Present to the user a conceptual view of the data.
  - Use a semantically rich formalism for the conceptual view.

- Need to provide solution for:
  - Establish mappings between ontological models and streaming data source schemas
  - Access streaming data sources through queries over ontology models

Ontology-based Data Access

Generate Semantic Web content from existing relational data sources

Ontology-based Data Access

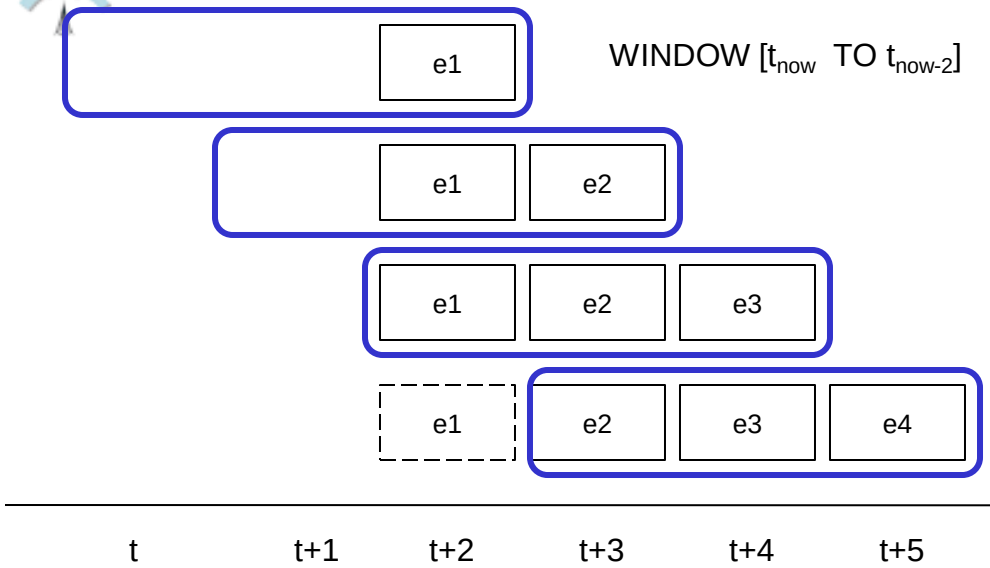


- R_2O + ODEMapster
- D2RQ
- SquirrelRDF
- RDBToOnto
- Relational.OWL
- SPASQL
- Virtuoso
- MASTRO

Background – Querying Relational Data Streams

Streaming Data

Event Streams/Acquisitional Streams



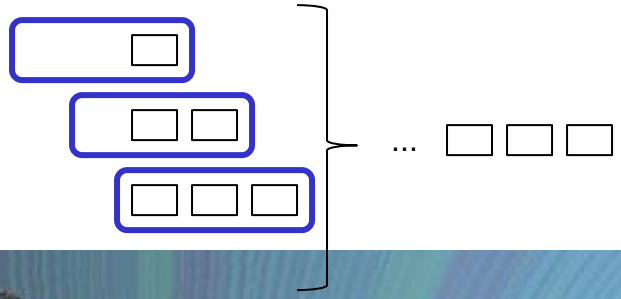
- STREAM
- Aurora/Borealis
- Cougar
- TinyDB
- SNEE

Query engines

- CQL
- SNEEql
- TinyQL

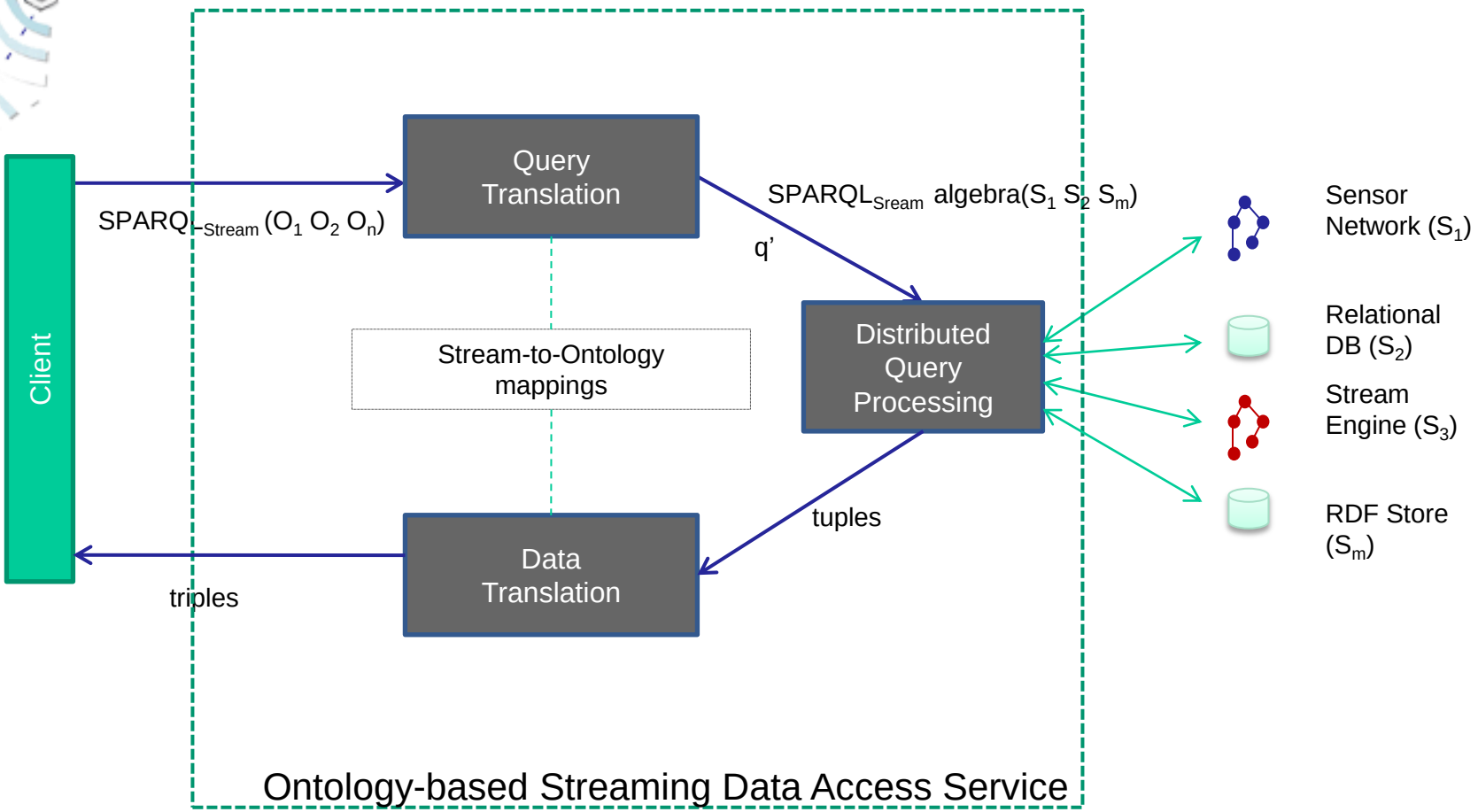
Query languages

Transform infinite sequence of tuples to bounded bag



Window-to-Stream operators: convert stream of windows to stream of tuples

Ontology-based Streaming Data Access



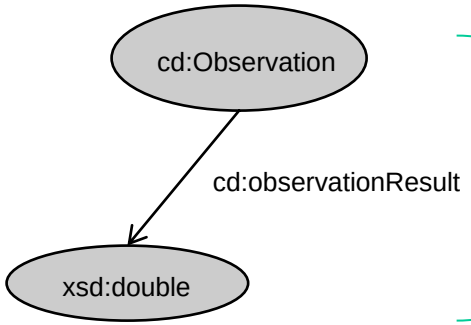
$SPARQL_{Stream}$

Example:

- "provide me with the wind speed observations over the last minute in the Solent Region "

- RDF-Stream

...
...
($<s_{i-1}, p_{i-1}, o_{i-1}>$, t_{i-1}),
($<s_i, p_i, o_i>$, t_i),
($<s_{i+1}, p_{i+1}, o_{i+1}>$, t_{i+1}),
...
...



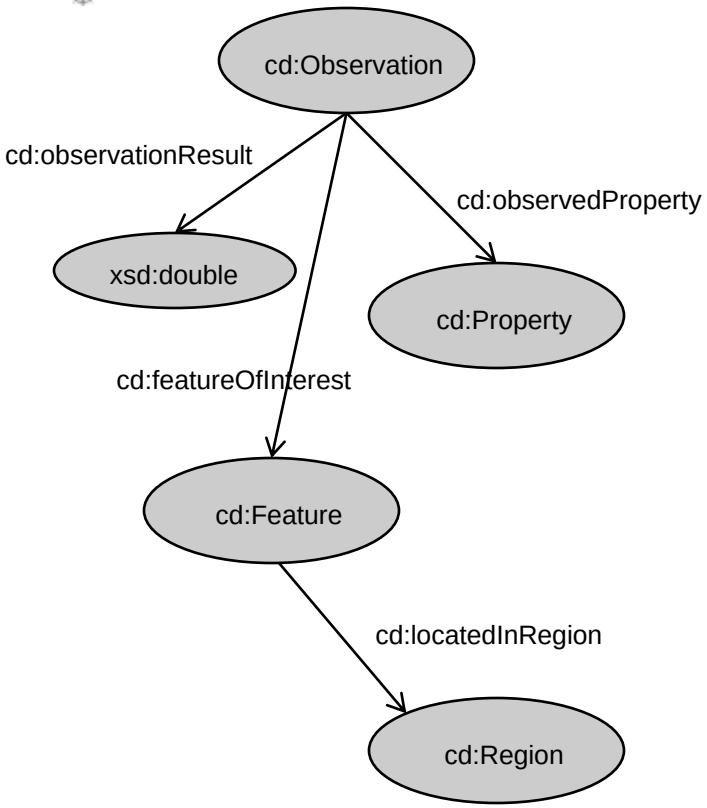
STREAM
<http://www.semsorgrid4env.eu/ccometeo.srdf>

...
...
(<ssg4e:Obs1,rdf:type, cd:Observation>, t_i),
(<ssg4e:Obs1,cd:observationResult,"34.5">, t_i),
(<ssg4e:Obs2,rdf:type, cd:Observation>, t_{i+1}),
(<ssg4e:Obs2,cd:observationResult,"20.3">, t_{i+1}),
...
...

SPARQL$_{Stream}$

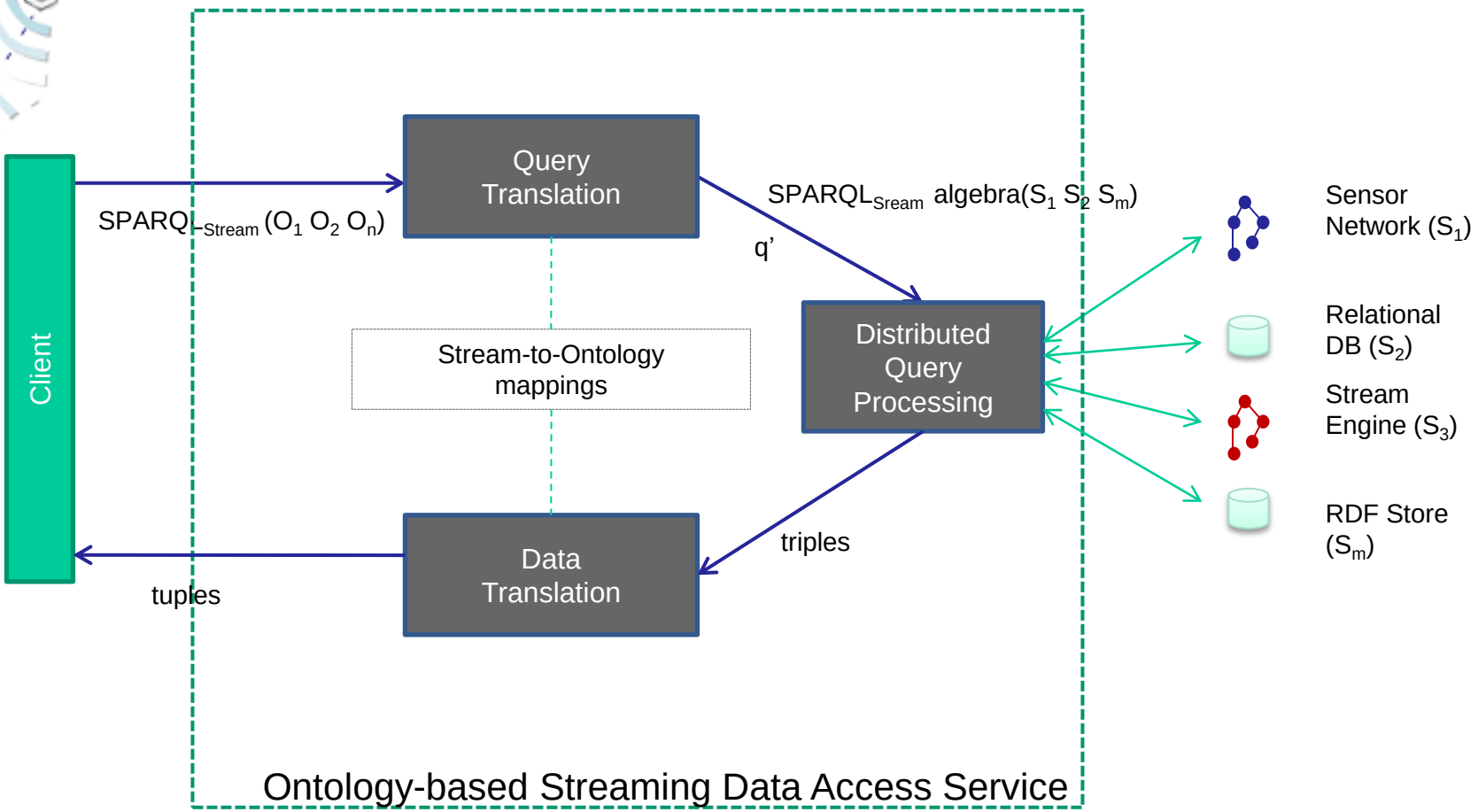
Example:

- “provide me with the wind speed observations over the last minute in the Solent Region ”




```
PREFIX cd: <http://www.semsorgrid4env.eu/ontologies/CoastalDefences.owl#>
PREFIX sb: <http://www.w3.org/2009/SSN-XG/Ontologies/SensorBasis.owl#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT  ?waveheight ?wavets
FROM NAMED STREAM <http://www.semsorgrid4env.eu/waves.srdf>
[ NOW – 1 MINUTE TO NOW – 0 MINUTES ]
WHERE
{
 ?WaveObs a cd:Observation;
    cd:observationResult ?waveheight;
    cd:observationResultTime ?wavets;
    cd:observedProperty ?waveProperty;
    cd:featureOfInterest ?waveFeature.
 ?waveFeature a cd:Feature;
    cd:locatedInRegion cd:SouthEastEnglandCCO.
 ?waveProperty a cd:WaveHeight.
 }
```

Ontology-based Streaming Data Access



R2RML

- W3C RDB2RDF working group
- Standardize RDB2RDF mappings

```
:BoscombeWaveHeight a rr:TriplesMapClass;
    rr:tableName "envdata_boscombe";
    rr:subjectMap [ rr:template
       "http://semsorgrid4env.eu/ns#WaveHeight/CCO/{DateTime}";
        rr:class ssn:ObservationValue; rr:graph ssg:waves.srdf ];
    rr:predicateObjectMap [ rr:predicateMap [ rr:predicate ssn:hasQuantityValue ];
    rr:objectMap[ rr:column "Hs" ] ];        .
```

```
<http://semsorgrid4env.eu/ns#/WaveHeight/CCO/2011-05-20:20:00 >
a  ssn:ObservationValue
<http://semsorgrid4env.eu/ns#/WaveHeight/CCO/2011-05-20:20:00 >
ssn:hasQuantityValue " 4.5"
```

Query Translation

Queries:

```
SELECT ?y
WHERE
{ ?x a cd:Observation;
    cd:observationResult ?y. }
```

$\Leftrightarrow$ $q(y) \leftarrow Observation(x) \wedge observationResult(x, y)$

$$q(\vec{x}) \leftarrow \phi(\vec{x}, \vec{y}))$$

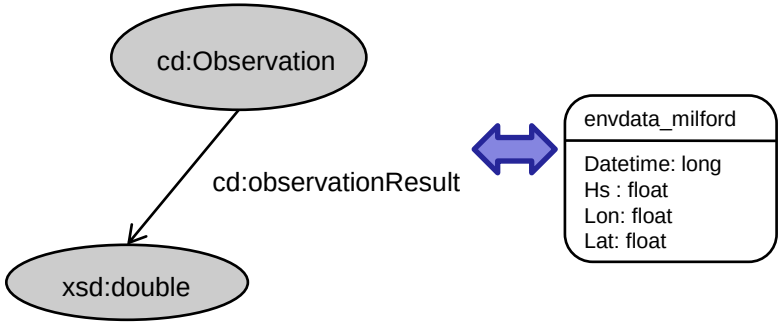
```
SELECT ?y
FROM STREAM < STREAM <http://www.semsorgrid4env.eu/waves.srdf>
[ NOW – 1 MINUTE TO NOW MINUTES ] >
WHERE
{ ?x a cd:Observation;
    cd:observationResult ?y. }
```

$$q(y)[t_i, t_f, \delta] \leftarrow (Observation(x) \wedge observationResult(x, y))[t_i, t_f, \delta]$$

Query Translation

Mappings:

query → $\Psi \rightsquigarrow \Phi$ ← algebra expression over sources



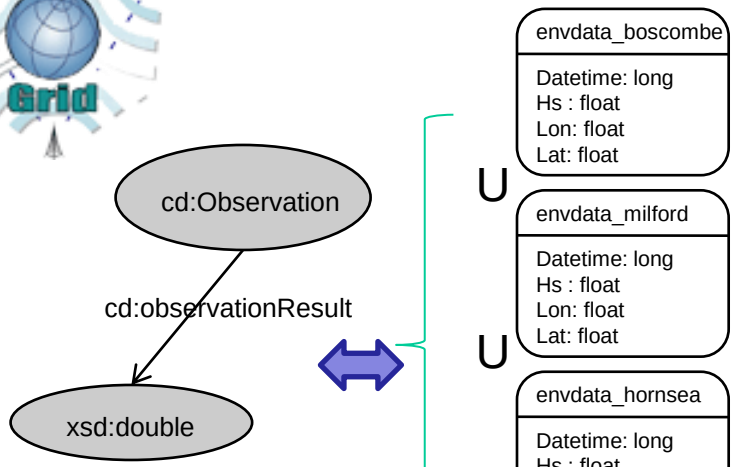
$$(Observation(x) \wedge observationResult(x, y))[t_i, t_f, \delta] \rightsquigarrow$$

Π Datetime Hs

$\omega_{[t_i, t_f, \delta]}$

envdata_milford

Query Translation



envdata_boscombe
Datetime: long
Hs : float
Lon: float
Lat: float

U

envdata_milford
Datetime: long
Hs : float
Lon: float
Lat: float

U

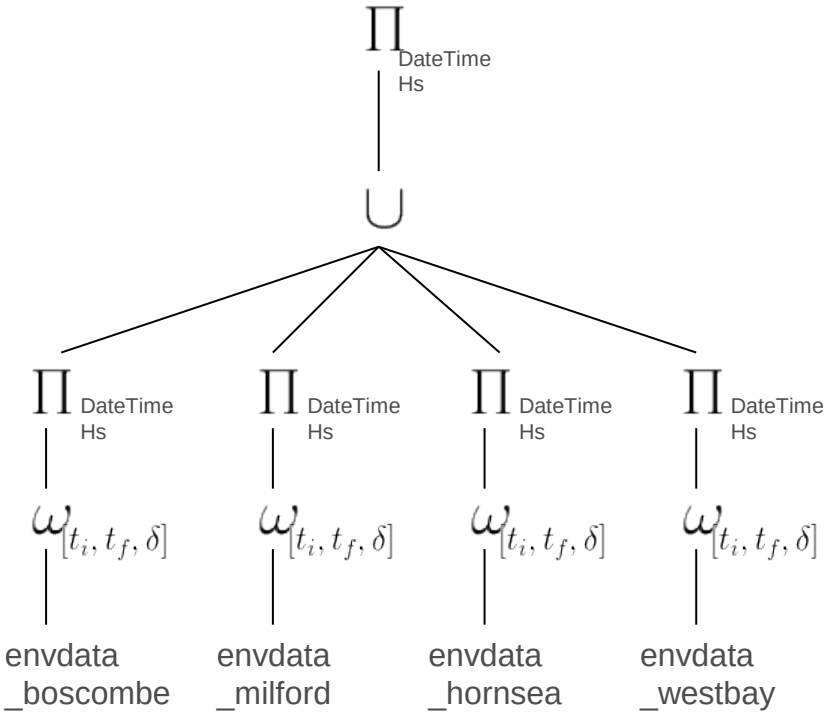
envdata_hornsea
Datetime: long
Hs : float
Lon: float
Lat: float

U

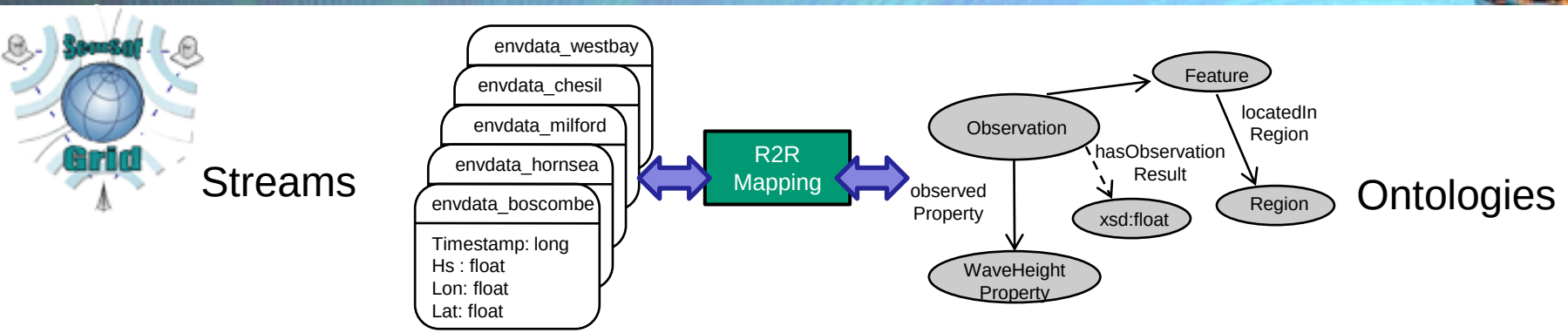
envdata_westbay
Datetime: long
Hs : float
Lon: float
Lat: float

$$(Observation(x) \wedge observationResult(x, y))[t_i, t_f, \delta]$$

$\rightsquigarrow$



Query Translation



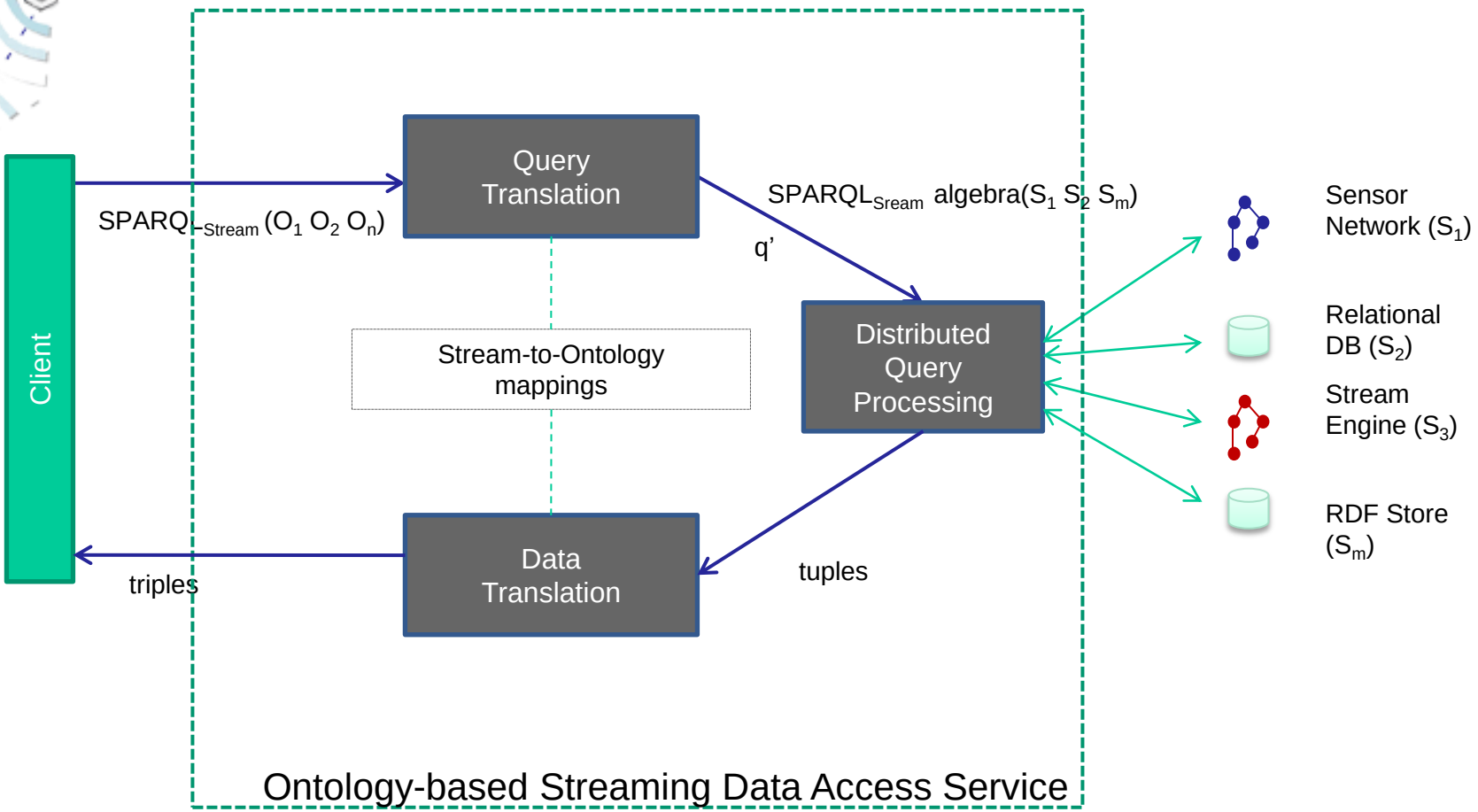

```
PREFIX cd: <http://www.semsorgrid4env.eu/ontologies/CoastalDefences.owl#>
PREFIX sb: <http://www.w3.org/2009/SSN-XG/Ontologies/SensorBasis.owl#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT  ?waveheight ?wavets
FROM NAMED STREAM <http://www.semsorgrid4env/waves.srdf>
WHERE
{
 ?WaveObs a cd:Observation;
    cd:observationResult ?waveheight;
    cd:observationResultTime ?wavets;
    cd:observedProperty ?waveProperty;
    cd:featureOfInterest ?waveFeature.
 ?waveFeature a cd:Feature;
    cd:locatedInRegion cd:SouthEastEnglandCCO.
 ?waveProperty a cd:WaveHeight.
 }
```

```
(SELECT timestamp,Hs FROM envdata_boscombe) UNION
(SELECT timestamp,Hs FROM envdata_hornsea) UNION
(SELECT timestamp,Hs FROM envdata_milford) UNION
(SELECT timestamp,Hs FROM envdata_chesil) UNION
(SELECT timestamp,Hs FROM envdata_perranporth) UNION
(SELECT timestamp,Hs FROM envdata_westbay) UNION
(SELECT timestamp,Hs FROM envdata_pevenseybay)
```

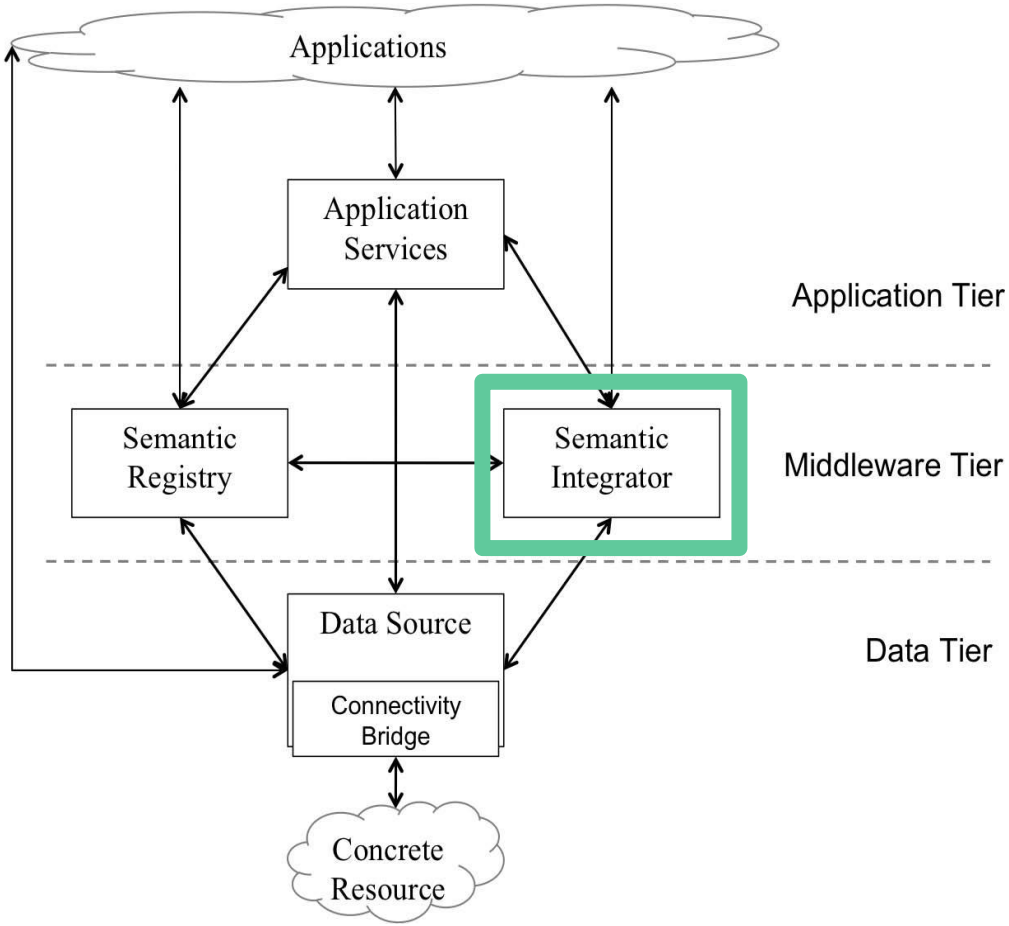
$SPARQL_{STR}$

SNEEql

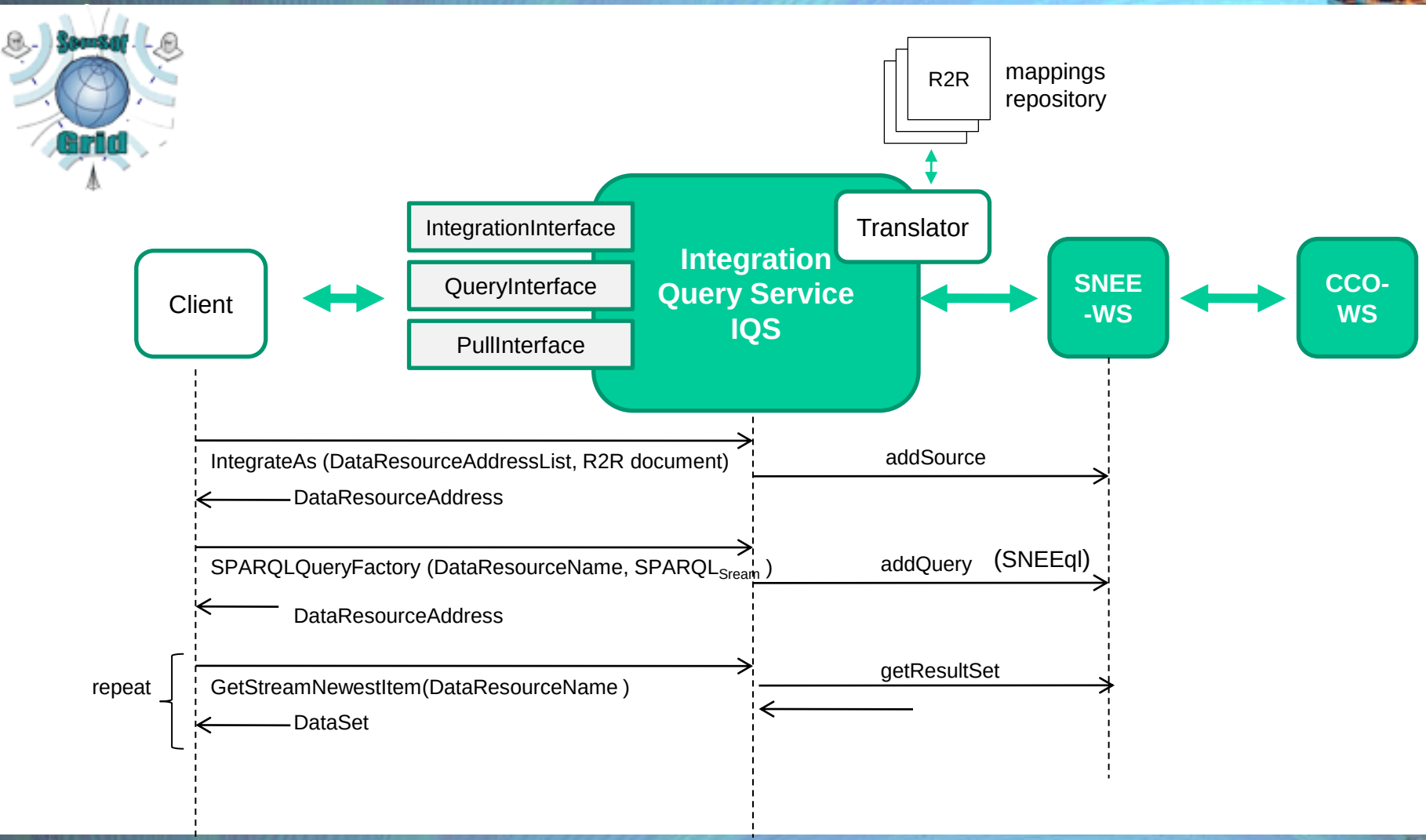
Ontology-based Streaming Data Access



Semantic Integrator in SemSorGrid4Env



Semantic Integrator IQS



IQS

- Integrating and Querying streaming data through Ontologies
- Use the SemSorGrid4Env Integration and Query Service (IQS)
  - Create an integrated resource from existing streams
    - Create a mapping document in R2RML
    - Use the IntegrateAs operation
  - Query the integrated resource
    - Write SPARQL-Stream queries over an ontology
    - Launch continuous queries (create a pull data resource)
  - Pull data periodically from a resource

Use Case: Waves at Boscombe?

- Wave data sources
  - CCO
  - WaveNet

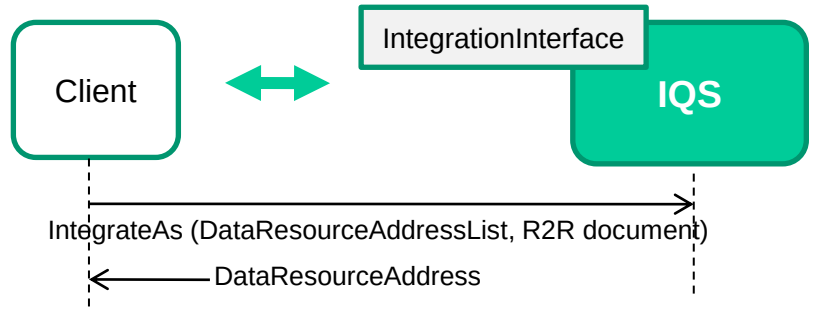
```
<stream name="envdata_boscombe" type="push">
        <column name="timestamp">
                <type class="timestamp"/>
        </column>
        <column name="stream_name">
                <type class ="string"/>
        </column>
        <column name="Hs">
                <type class ="float"/>
        </column>
        <column name="Tsea">
                <type class ="float"/>
        </column>
        <column name="Lat">
                <type class ="float"/>
        </column>
        <column name="Lon">
                <type class ="float"/>
        </column>
    </stream>
```

R2RML Mappings

```
:MilfordWaveHeight a rr:TriplesMapClass;
    rr:tableName "envdata_boscombe";
    rr:subjectMap [ rr:template
       "http://semsorgrid4env.eu/ns#WaveHeight/CCO/{DateTime}";
        rr:class ssn:ObservationValue; rr:graph ssg:ccometeo.srdf ];
    rr:predicateObjectMap [ rr:predicateMap [ rr:predicate ssn:hasQuantityValue ];
    rr:objectMap[ rr:column "Hs" ] ];        .
```

```
<http://semsorgrid4env.eu/ns#/WaveHeight/CCO/2011-05-20:20:00 >
a  ssn:ObservationValue
<http://semsorgrid4env.eu/ns#/WaveHeight/CCO/2011-05-20:20:00 >
ssn:hasQuantityValue " 4.5"
```

Integrating the sources



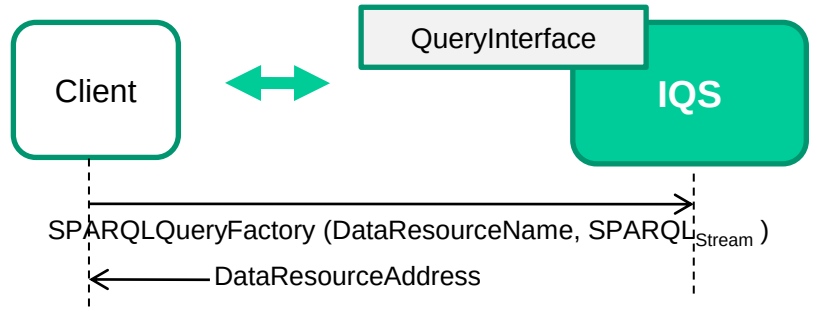
- Integrate configuration:

```
integrated.resource.name = obsResource
integrated.resource.mapping = mappingSimple.ttl
integrated.resource.list =
    http://webgis1.geodata.soton.ac.uk:8080/CCO/services/PullStream?wsdl,
http://webgis1.geodata.soton.ac.uk:8080/WaveNet/services/PullStream?wsdl
```

Integrated resources

- After IntegrateAs:
  - Virtual integrated resource created
  - Queryable using SPARQL
  - Underlying data resources mapped through R2RML

Querying the new Resource

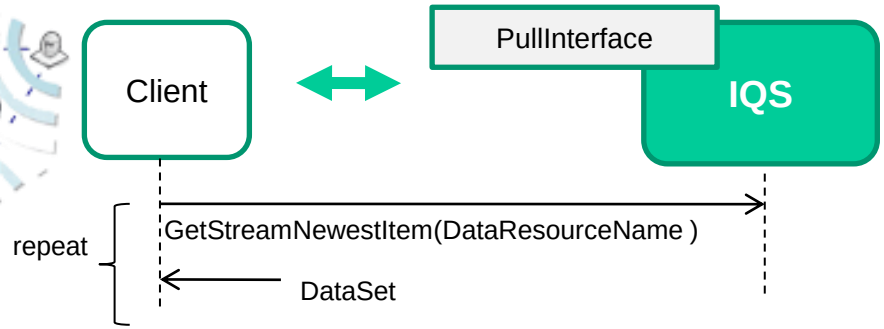


- Sample query

```
SELECT ?obs ?waveHeight ?wavetime
WHERE
{
    ?obs a ssn:Observation;
         ssn:observationResult ?waveHeight;
         ssn:observedProperty cd:WaveHeight;
           ssn:observationResultTime ?wavetime.
    FILTER (?waveHeight > 0.2)
}
```

- Launch continuous query

Pulling Data



- Pulling data from the new resource

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<sparql xmlns="http://www.w3.org/2007/SPARQL/results#">
    <head>
        <variable name="obs"/>
    </head>
    <results>
        <result>
            <binding name="obs">
                <uri>http://semsorgrid4env.eu/ns#Obs/WaveHeight/boscombe/1306497537858</uri>
            </binding>
.....
```

Mapping other Sources, other Properties

- Sea temperature observations:

```
:boscombeTseaObsMapping   a rr:TriplesMapClass;
   rr:SQLQuery "";
   rr:subjectMap [         rr:template
     "http://semsorgrid4env.eu/ns#Obs/Temperature/boscombe/{timestamp}";
                                        rr:class ssn:Observation; rr:graph ssg:waves.srdf];
     rr:tableName "envdata_boscombe";
   rr:predicateObjectMap
     [          rr:predicateMap [ rr:predicate ssn:observationResult ];
                rr:objectMap    [ rr:column "Tsea" ]],
     [          rr:predicateMap [ rr:predicate ssn:observationResultTime ];
                rr:objectMap    [ rr:column "timestamp" ]],
     [          rr:predicateMap [ rr:predicate ssn:observedProperty ];
                rr:objectMap    [ rr:object cd:SeaTemperature ]];
.
```

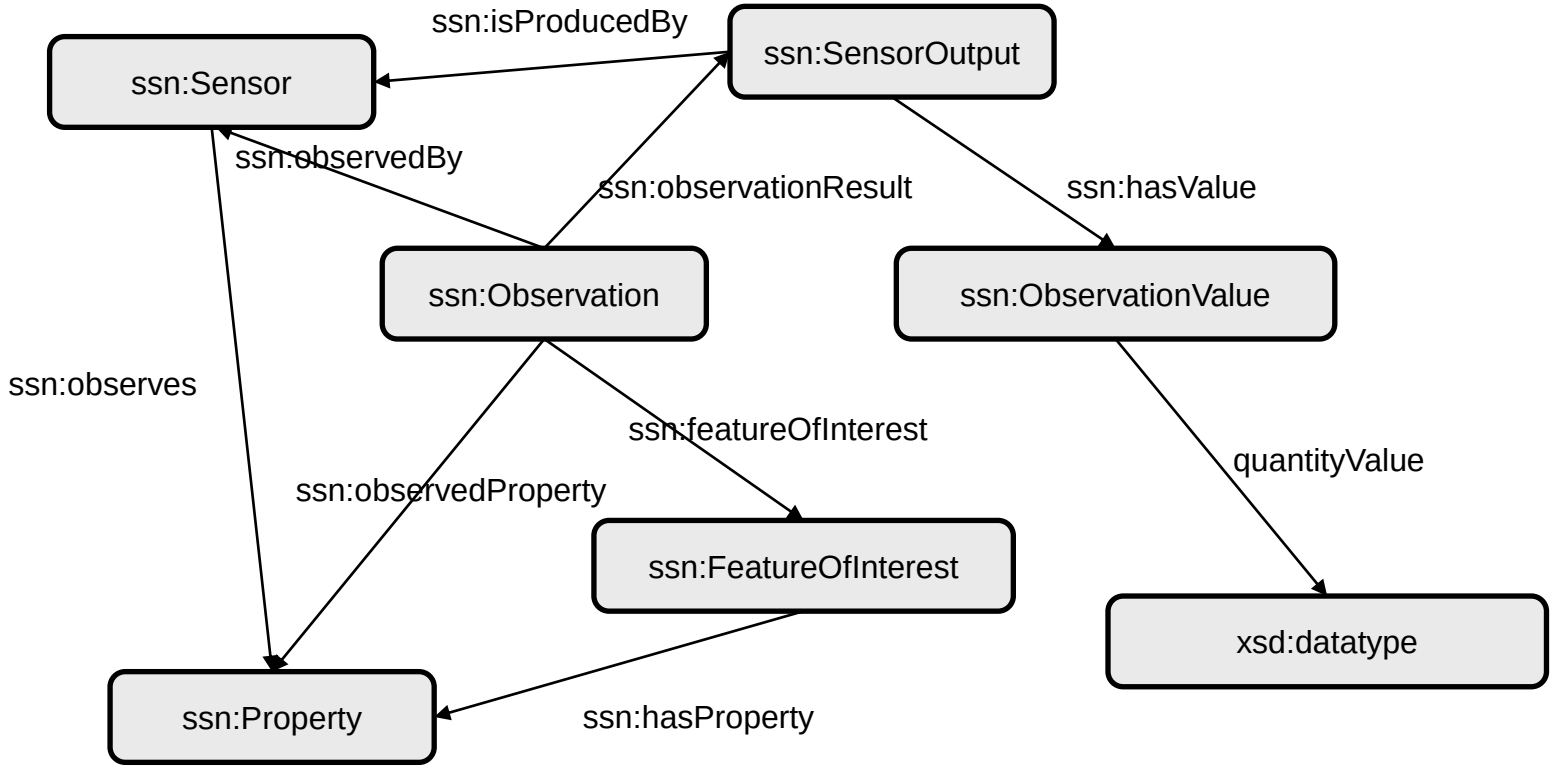
Modify the Query: Get Temperature

- Sample query

```
SELECT ?obs ?value ?time
WHERE
{
    ?obs a ssn:Observation;
         ssn:observationResult ?value;
         ssn:observedProperty cd:SeaTemperature;
           ssn:observationResultTime ?time.
    FILTER (?value > 0.2)
}
```

- Launch continuous query
- Pull data

Refining the output: RDF Observations



Construct Queries

- Sample query

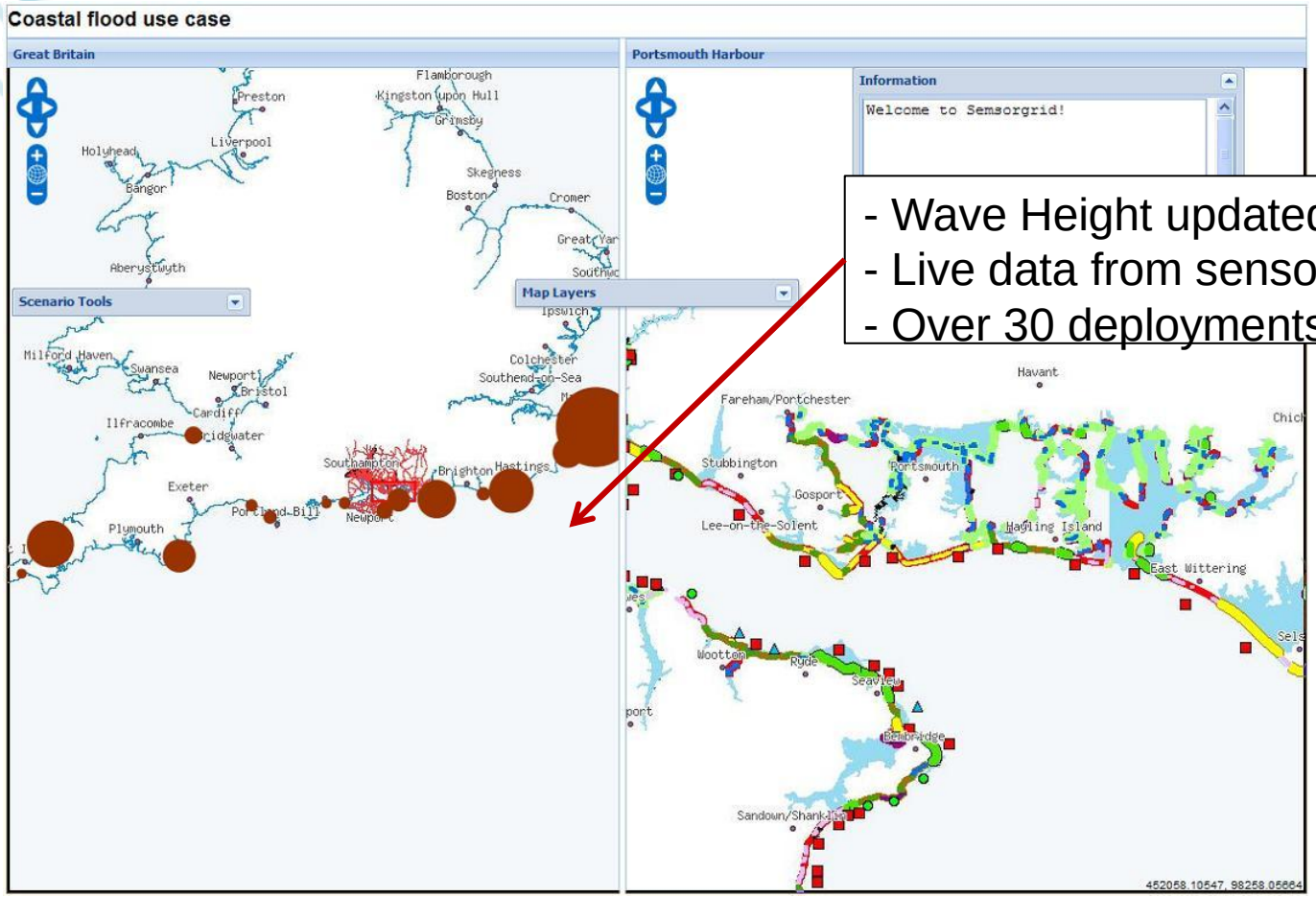
```
CONSTRUCT
{
   ?obs a ssn:Observation;
        ssn:observationResult [
                    a ssn:SensorOutput;
                    ssn:hasValue [ a ssn:ObservationValue;
                                   ssg:hasQuantityValue ?waveHeight ]];
        ssn:observedProperty cd:WaveHeight;
        ssn:observationResultTime ?wavetime.
}
WHERE
{
    ?obs a ssn:Observation;
        ssn:observationResult ?waveHeight;
        ssn:observedProperty cd:WaveHeight;
        ssn:observationResultTime ?wavetime.
    FILTER (?waveHeight > 0.9)
}
```

Pull RDF Observations

- Pull data

```
<rdf:Description rdf:about="http://semsorgrid4env.eu/ns#Obs/WaveHeight/boscomb
e/1306624735493-19162413495275452O7">
    <j.1:observationResultTime>1306624735493</j.1:observationResultTime>
    <j.1:observedProperty rdf:resource="http://www.semsorgrid4env.eu/ontologies/
CoastalDefences.owl#WaveHeight"/>
    <j.1:observationResult rdf:nodeID="A2"/>
    <rdf:type rdf:resource="http://purl.oclc.org/NET/ssnx/ssn#Observation"/>
  </rdf:Description>
 <rdf:Description rdf:nodeID="A2">
    <j.1:hasValue rdf:nodeID="A1"/>
    <rdf:type rdf:resource="http://purl.oclc.org/NET/ssnx/ssn#SensorOutput"/>
  </rdf:Description>
  <rdf:Description rdf:nodeID="A1">
    <j.0:hasQuantityValue>0.9518465</j.0:hasQuantityValue>
    <rdf:type rdf:resource=http://purl.oclc.org/NET/ssnx/ssn#ObservationValue/>
  </rdf:Description>
```

Implementation in SemSorGrid4Env



- Wave Height updated every 10 min
- Live data from sensor in buoys
- Over 30 deployments

Implementation in Swiss-Experiment

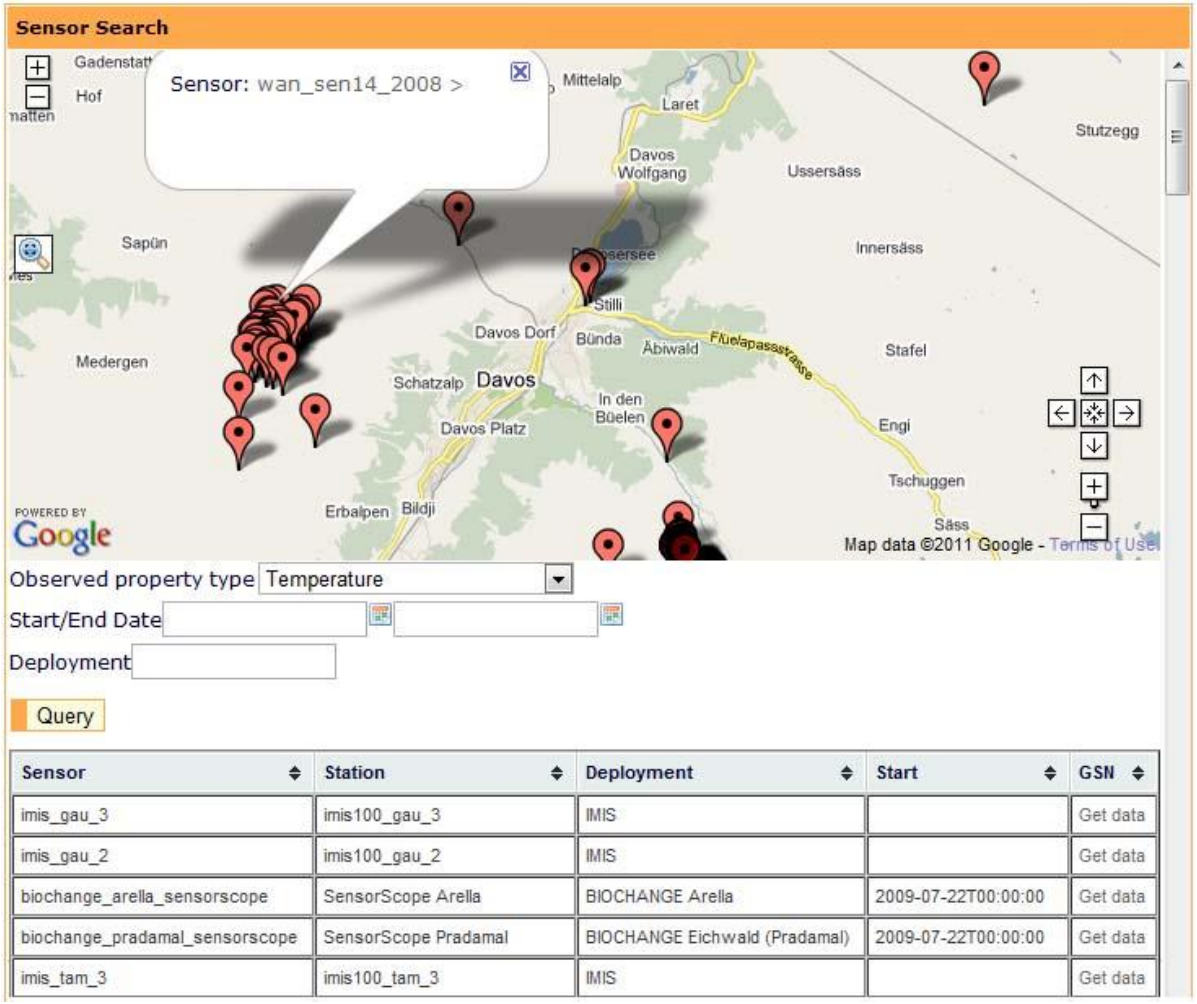
- GSN Stream processing Engine
- GSN has a WebService interface. No query language available:

```
GetLatestMultiData request = new GetLatestMultiData();
GSNWebService_FieldSelector[] selector = new GSNWebService_FieldSelector[1];
selector[0] = new GSNWebService_FieldSelector();
selector[0].setVsname("wan5");

request.setFieldSelector(selector );
GetLatestMultiDataResponse response = gsn.getLatestMultiData(request);
```

- Hundreds of sensors deployed in the Swiss-Ex project.
- Apply Ontology-based search

Implementation in Swiss-Experiment



Sensor	Station	Deployment	Start	GSN
imis_gau_3	imis100_gau_3	IMIS		Get data
imis_gau_2	imis100_gau_2	IMIS		Get data
biochange_arella_sensorscope	SensorScope Arella	BIOCHANGE Arella	2009-07-22T00:00:00	Get data
biochange_pradamal_sensorscope	SensorScope Pradamal	BIOCHANGE Eichwald (Pradamal)	2009-07-22T00:00:00	Get data
imis_tam_3	imis100_tam_3	IMIS		Get data

Ontology-based Streaming Data Querying

- SPARQL interface for Sensor data
- User queries over the SSN Ontology
- Transparently adding new sources
- User unaware of underlying source schema details
- Query translation mechanism and delegation of query execution to streaming data processor
- Integrated virtual sources available for higher tier applications