SPARQLStream: Ontology-based access to data streams

Jean-Paul Calbimonte, Oscar Corcho
jp.calbimonte@upm.es, ocorcho@fi.upm.es
http://www.oeg-upm.net/
Share, Remix, Reuse — Legally

- This work is licensed under the Creative Commons Attribution 3.0 Unported License.

- **You are free:**
  - **to Share** — to copy, distribute and transmit the work
  - **to Remix** — to adapt the work

- **Under the following conditions**
  - **Attribution** — You must attribute the work by inserting
    - “[source http://streamreasoning.org/sr4ld2013]” at the end of each reused slide
    - a credits slide stating
      - These slides are partially based on “Streaming Reasoning for Linked Data 2013” by M. Balduini, J-P Calbimonte, O. Corcho, D. Dell'Aglio, E. Della Valle, and J.Z. Pan http://streamreasoning.org/sr4ld2013

- To view a copy of this license, visit http://creativecommons.org/licenses/by/3.0/
- Virtual RDF views over data streams
- Ontology-based access to data streams
  - Examples
  - Architecture
  - Underlying query processors
- SPARQLStream language
- Query rewriting
  - R2RML mappings
- Resources
Querying RDF Streams

users, applications

query processing

RDF Stream Processor

Where is the data coming from?

Existing streaming data sources

DSMS, CEP, Sensor middleware, ...

Wrap Import Load Continuously

http://streamreasoning.org/sr4ld2013
Virtual RDF views over data streams

users, applications

query processing

RDF Stream Processor

queries

data layer

DSMS

CEP

Sensor middleware

Morph-streams

SPARQLStream

Virtual RDF Stream

Virtual RDF views over data streams

http://streamreasoning.org/sr4ld2013
Already seen somewhere...?
Stream Processor Implementations

Data Stream Management Systems (DSMS)
- CQL/Stream
- Borealis
- StreamMill
- NiagaraCQ
- TelegraphCQ
- Esper
- Oracle CEP
- IBM InfoSphere
- Microsoft StreamInsight
- Sybase CEP
- StreamBase

Complex Event Processors (CEP)
- GEM
- Rapide
- Cayuga
- CEDR
- Hourglass
- Cosm
- SStreamWare
- GSN
- Microsoft StreamInsight

Stream Data Middleware

Diverse query languages
Different query capabilities
Different query models
Morph-streams: Overview

SELECT ?proximity
WHERE {
  ?obs a ssn:ObservationValue;
  qudt:numericalValue ?proximity;
  FILTER (?proximity>10) }

SELECT prox
FROM sens.win:time(5 sec)
WHERE prox >10

Morph-streams processing SPARQL_{Stream} queries

http://streamreasoning.org/sr4ld2013

https://github.com/jpcik/morph-streams
SPARQLStream Language

Query Form

CONSTRUCT

DESCRIBE

SELECT

ISTREAM

DSTREAM

RSTREAM

FROM NAMED STREAM

WINDOW

FROM NAMED STREAM

WHERE Clause (Graph Pattern)

Underlying data source restrictions

Filter

OPTIONAL

AND

UNION

Triple pattern

Dataset

Restrictions

TRUE - FALSE
SPARQLStream: examples

**SPARQL Stream**

```sparql
PREFIX sr4ld: <http://www.streamreasoning.org/ontologies/socialsensor,owl#>
SELECT ?room
FROM NAMED STREAM <http://www.streamreasoning.org/streams/socialsensor.srdf> [NOW-10 S]
WHERE {
  ?obs sr4ld:observedBy ?sensor.
}
```

**All rooms where something was observed in the last 10s**

```sparql
PREFIX sr4ld: <http://www.streamreasoning.org/ontologies/socialsensor,owl#>
SELECT (COUNT(?person) AS ?nmb) ?room
FROM NAMED STREAM <http://www.streamreasoning.org/streams/socialsensor.srdf> [NOW-10 S]
WHERE {
}
GROUP BY ?room
```

**Number of persons observed in each room in the last 10s**
SPARQLStream Language

- **NamedStream** → ‘FROM’ [‘NAMED’] ‘STREAM’ StreamIRI [‘ Window ’]
- **Window** → ‘NOW-’ Integer TimeUnit [UpperBound] [Slide]
- **UpperBound** → ‘TO NOW-’ Integer TimeUnit
- **Slide** → ‘SLIDE’ Integer TimeUnit
- **TimeUnit** → ‘MS’ | ‘S’ | ‘MINUTES’| ‘HOURS’ | ‘DAY’

SELECT ISTREAM ?room
FROM NAMED STREAM <http://www.streamreasoning.org/streams/socialsensor.srdf> [NOW-10 S]
WHERE {...}

- **Select** → ‘SELECT’ [Xstream] [Distinct | Reduced] ...
- **Xstream** → ‘RSTREAM’ | ‘ISTREAM’ | ‘DSTREAM’
Morph-streams: Overview

SELECT ?proximity
WHERE {
  ?obs a ssn:ObservationValue;
      qudt:numericalValue ?proximity;
  FILTER (?proximity > 10)
}

SELECT prox
FROM sens.win:time(5 sec)
WHERE prox > 10

Morph-streams processing SPARQL\textsubscript{Stream} queries
Now, where is the data?

(sensor) \[(alice, room1)_{i-1} \rightarrow (carl, room1)_{i} \rightarrow (person, room, ...)_{i} \rightarrow (bob, room2)_{i} \rightarrow (alice, room3)_{i+1} \rightarrow (luke, room1)_{i+1} \]

Stream Schema

DSMS, CEP, middleware can evaluate queries over this model.
## Underlying Query Processors

<table>
<thead>
<tr>
<th><strong>Esper</strong></th>
<th><strong>SNEE</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>• CEP/DSMS</td>
<td>• DSMS/Sensor Network Query Evaluator</td>
</tr>
<tr>
<td>• EPL language</td>
<td>• Compile queries to sensor code</td>
</tr>
<tr>
<td>SELECT prox FROM sensors.win:time(5 minute) WHERE prox &gt;10</td>
<td>SELECT prox FROM sensors [FROM NOW-5 MINUTES TO NOW] WHERE prox &gt;10</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>GSN</strong></th>
<th><strong>Cosm/Xively</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>• Sensor middleware</td>
<td>• Sensor middleware</td>
</tr>
<tr>
<td>• REST API</td>
<td>• Open platform</td>
</tr>
<tr>
<td>• REST API</td>
<td>• REST API</td>
</tr>
<tr>
<td><a href="http://montblanc.slf.ch:22001/multidata?vs%5B0%5D=sensors&amp;field%5B0%5D=proximity_field&amp;c_min%5B0%5D=10&amp;from=15/05/2012+05:00:00&amp;to=15/05/2012+10:00:00">http://montblanc.slf.ch:22001/multidata?vs[0]=sensors&amp;field[0]=proximity_field&amp;c_min[0]=10&amp;from=15/05/2012+05:00:00&amp;to=15/05/2012+10:00:00</a></td>
<td><a href="http://api.cosm.com/v2/feeds/14321/datastreams/4?start=2012-05-15T05:00:00Z&amp;end=2012-05-15T10:00:00Z">http://api.cosm.com/v2/feeds/14321/datastreams/4?start=2012-05-15T05:00:00Z&amp;end=2012-05-15T10:00:00Z</a></td>
</tr>
</tbody>
</table>
Underlying Query Processors

SPARQLStream

SELECT ?proximity
FROM STREAM <http://streamreasoning.org/SensorReadings.srdf>
[NOW–5 S]
WHERE {
  ?obs a ssn:ObservationValue;
  qudt:numericalValue ?proximity;
  FILTER (?proximity>10) 
}

R2RML

SELECT prox FROM sensors [FROM NOW-5 MINUTES TO NOW]
WHERE prox >10

Query rewriting

SELECT prox FROM sensors.win:time(5 minute)
WHERE prox >10

SNEE (DSMS)

SELECT prox FROM sensors [FROM NOW-5 MINUTES TO NOW]
WHERE prox >10

Esper (CEP)

http://montblanc.slf.ch:22001/multidata?
vs[0]=sensors&field[0]=proximity_field&c_min[0]=10&
from=15/05/2012+05:00:00&to=15/05/2012+10:00:00

GSN (middlwr)

http://api.cosm.com/v2/feeds/14321/datastreams/4?
start=2012-05-15T05:00:00Z&end=2012-05-15T10:00:00Z

Cosm Xively

5 Seconds
### Underlying query processors

<table>
<thead>
<tr>
<th>Features</th>
<th>Esper</th>
<th>SNEE</th>
<th>GSN</th>
<th>Cosm/Xively</th>
</tr>
</thead>
<tbody>
<tr>
<td>Projection</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>Fixed</td>
</tr>
<tr>
<td>Proj expression</td>
<td>✔</td>
<td>✔</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Joins</td>
<td>✔</td>
<td>✗ ✗ only window</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Union</td>
<td>✗</td>
<td>✗ ✗ not windows</td>
<td>✔</td>
<td>✗</td>
</tr>
<tr>
<td>Selection</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✗ ✗ limited</td>
</tr>
<tr>
<td>Aggregates</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✗</td>
</tr>
<tr>
<td>Time window</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Tuple window</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✗</td>
</tr>
<tr>
<td>R2S</td>
<td>✔</td>
<td>✔</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Conjunction, Disj</td>
<td>✔</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Repetition pattern</td>
<td>✔</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Sequence</td>
<td>✔</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
</tbody>
</table>
Configuring Morph-streams

- Main ingredients:
  1. Data streams
  2. Ontology (network)
  3. R2RML mappings

Link both models
SSN Ontology with other ontologies

W3C SSN Ontology

modeling our streaming data

combine with domain ontologies
Our simpler ontology...

We can use different ontologies for the same data.
Our simpler ontology...

- **Observation**
  - **observes**
  - **subClassOf**
  - **Person**

- **Room**
  - **where**

Define mappings

- **detections**
  - \( (\text{person}, \text{room}, ...) \)

**ISWC 2013**
Sydney, Australia
R2RML - Overview

R2RML Mapping

- TriplesMap
- LogicalTable
- SubjectMap
- PredicateObjectMap
  - PredicateMap
  - ObjectMap
  - RefObjectMap
R2RML - Overview
Encoding in R2RML

Mapping definition

:triplesMap a rr:TriplesMap;
  rr:logicalTable [ rr:tableName "sensors"; ]
  rr:subjectMap [
    rr:template "http://streamreasoning.org/data/Observation/{person}{timed}";
    rr:class sr4ld:Observation; rr:graph sr4ld:socialstream.srdf ];
  rr:predicateObjectMap [
    rr:predicate sr4ld:who ;
    rr:objectMap [ rr:template “http://streamreasoning.org/data/Person/{person}” ]];

the stream name

stream attributes

subject URI

triple predicate + object

the object (a URI in this case)
Now some code

Morph-streams:

- Coded in **Scala**
- JAR bundle, use it from Scala or **Java** code
- Maven, Sbt

**Examples**
- One off query
- Register continuous query
- Pull data
- Push
- Basic REST

- [https://github.com/jpcik/morph-streams](https://github.com/jpcik/morph-streams)
### Code examples

- **Parse SPARQLStream**

```scala
val query = "PREFIX sr4ld: <...>. SELECT ?a ..."
val syntax = StreamQueryFactory.create(query);
```

- **Execute One-off query**

```scala
val query = "PREFIX sr4ld: <...>. SELECT ?a ..."
mapping = Mapping(new URI(mappings/social.ttl))
val adapter: QueryEvaluator = Application.adapter(system)
val results = adapter.executeQuery(query, mapping)
```
### Register and Pull

val queryid = adapter.registerQuery(query, mapping)
val results1 = adapter.pull(queryid)
val results2 = adapter.pull(queryid)

### Register and Push

class ExampleReceiver extends StreamReceiver{
  override def receiveData(s:SparqlResults): Unit =
    Logger.debug("got: "+res)
}
val receiver = new ExampleReceiver
val queryid = adapter.listenToQuery(query, mapping, receiver)
SPARQLStream from command line


  Just encoding query

- curl "http://streams.linkeddata.es/emt/sparqlstream?query=$encoded_value"

Disclaimer: Simplistic, not implementing all of the SPARQL protocol
Sample result

```json
{
    "head": {
        "vars": [ "timeto", "obs" ]
    },
    "results": {
        "bindings": [
            {
                "timeto": { "datatype": "http://www.w3.org/2001/XMLSchema#string", "type": "typed-literal", "value": "0" },
            }
        ]
    }
}
```

Bindings in JSON
Resources

- **Morph-Streams**
  - [https://github.com/jpcik/morph-streams](https://github.com/jpcik/morph-streams)

- **See demos**
  - Check our Madrid buses demo at SSN2013 workshop tomorrow

- **Read out more**

- **Contact point**
  - [jp.calbimonte@upm.es](mailto:jp.calbimonte@upm.es)
  - [ocorcho@fi.upm.es](mailto:ocorcho@fi.upm.es)
SPARQLStream: Ontology-based access to data streams

Jean Paul Calbimonte, Oscar Corcho
jp.calbimonte@upm.es, ocorcho@fi.upm.es
http://www.oeg-upm.net/
RDF Streams

\[ \langle s, p, o \rangle \]

\[ \langle \text{aemet:observation1}, \text{ssn:observedBy}, \text{aemet:Sensor3} \rangle \]

\[ \langle \text{aemet:observation1}, \text{qudt:hasNumericValue}, \text{"15.5"} \rangle \]

For streams?

\[ (\langle s, p, o \rangle, \tau) \]

\[ (\langle \text{aemet:observation1}, \text{qudt:hasNumericValue}, \text{"15.5"} \rangle, 34532) \]

timestamped triples

- Gutierrez et al. (2007) Introducing time into RDF. IEEE TKDE
- Rodríguez et al. (2009) Semantic management of streaming data. SSN
Streaming SPARQL execution approaches

- Extend RDF for streaming data
- Extend SPARQL for streaming RDF
- Use a SPE internally for evaluation
- Logic-programming based query evaluation
- RDF Streaming engine from scratch

~Similarities

Divergence

Query rewriting to SPEs

DSMSs
CEPs
Middleware