Stream Reasoning
For Linked Data
J-P Calbimonte, D. Dell'Aglio,
E. Della Valle, A. Intizar and A. Mileo
http://streamreasoning.org/events/sr4ld2015

RSP Extensions for RDF and SPARQL
Jean-Paul Calbimonte
Jean-paul.Calbimonte@epfl.ch
Share, Remix, Reuse — Legally

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

- **Your 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 a credits slide stating
    - These slides are partially based on “Streaming Reasoning for Linked Data 2015” by J-P Calbimonte, D. Dell'Aglio, E. Della Valle, M. I. Ali and A. Mileo

- To view a copy of this license, visit [http://creativecommons.org/licenses/by/3.0/](http://creativecommons.org/licenses/by/3.0/)
1. Continuous RDF model extensions
   - RDF Streams, timestamps

2. Continuous extensions of SPARQL
   - Continuous evaluation
   - Additional operators

3. Overview of existing systems
   - Features
   - Comparison
Outline

1. Continuous RDF model extensions
   - RDF Streams, timestamps

2. Continuous extensions of SPARQL
   - Continuous evaluation
   - Additional operators

3. Overview of existing systems
   - Features
   - Comparison
Continuous extensions of RDF

- As you know, “RDF is a standard model for data interchange on the Web” (http://www.w3.org/RDF/)

  \[
  \text{<sub}_1 \text{ pred}_1 \text{ obj}_1> \\
  \text{<sub}_2 \text{ pred}_2 \text{ obj}_2>
  \]

- We want to extend RDF to model data streams
- A data stream is an (infinite) ordered sequence of data items
- A data item is a self-consumable informative unit
With **data item** we can refer to:

1. A **triple**

   ```
   <:alice :isWith :bob>
   ```

2. A **graph**

   ```
   <:alice :posts :p>  
   <:p :who :bob>  
   <:p :where :redRoom>
   ```

   ```
   :graph1
   ```
Data items and time

- Do we need to associate the time to data items?
  - It depends on what we want to achieve (see next!)

- If yes, how to take into account the time?
  - Time should not (but could) be part of the schema
  - Time should not be accessible through the query language
  - Time as object would require a lot of reification

- How to extend the RDF model to take into account the time?
Application time

- A timestamp is a temporal identifier associated to a data item
- The **application time** is a set of one or more timestamps associated to the data item
- Two data items can have the same application time
  - Contemporaneity
- Who does assign the application time to an event?
  - The one that generates the data stream!
Missing application time

A RDF stream without timestamp is an ordered sequence of data items

The order can be exploited to perform queries
- Does Alice meet Bob before Carl?
- Who does Carl meet first?
Application time: point-based extension

- One timestamp: the time instant on which the data item occurs
- We can start to compose queries taking into account the time
  - How many people has Alice met in the last 5m?
  - Does Diana meet Bob and then Carl within 5m?
Application time: interval-based extension

- Two timestamps: the time range on which the data item is valid (from, to)
- It is possible to write even more complex constraints:
  - Which are the meetings the last less than 5m?
  - Which are the meetings with conflicts?
Outline

1. Continuous RDF model extensions
   - RDF Streams, timestamps

2. Continuous extensions of SPARQL
   - Continuous evaluation
   - Additional operators

3. Overview of existing systems
   - Features
   - Comparison
Continuous query evaluation

- From SPARQL
  - One query, one answer
  - The query is sent after that the data is available

- To a continuous query language
  - One query, multiple answers
  - The query is **registered** in the query engine
  - The registration usually happens before that the data arrives
  - Real-time responsiveness is usually required
Let’s process the RDF streams!

- In literature there are two different main approaches to process streams
- Data Stream Management Systems (DSMSs)
  - Roots in DBMS research
  - Aggregations and filters
- Complex Event Processors (CEPs)
  - Roots in Discrete Event Simulation
  - Search of relevant patterns in the stream
  - Non-equi-join on the timestamps (after, before, etc.)
- Current systems implements feature of both of them
  - EPL (e.g. Esper, ORACLE CEP)
- Now we focus on the CQL/STREAM model
  - Developed in the DSMS research
  - C-SPARQL (and others) is inspired to this model
Our assumptions

In the following we will consider the following setting

- A RDF triple is an event
- Application time: point-based

\[
\begin{align*}
& < :\text{alice} :\text{isWith} : \text{bob} > : [1] \\
& < :\text{alice} :\text{isWith} : \text{carl} > : [3] \\
& < :\text{bob} :\text{isWith} : \text{diana} > : [6] \\
& \ldots
\end{align*}
\]
Querying data streams – The CQL model

Streams

Relations

*Stream operators

Sliding windows

stream-to-relation

relation-to-stream

Relational algebra

relation-to-relation

Mapping: $T \rightarrow R$

Relation $R(t)$

finite bag

infinite unbounded sequence

Stream

$\langle S, T \rangle$

$\langle S_1 \rangle$

$\langle S_2 \rangle$

$\langle S_3 \rangle$
CQL extension for querying RDF data streams

- RDF Streams
- RDF Mappings
- S2R operators
- SPARQL operators
- R2S operators
- Sliding windows
- *Stream operators
Time-based sliding window

\[ W(\omega, \beta) \]

- **slide**
- **width**
Time-based sliding window - tumbling

R2R operator

slide

width

\( W(\omega,\beta) \)

\( S_1 \) \( S_3 \) \( S_6 \) \( S_8 \) \( S_{10} \) \( S_{11} \) \( S_{12} \)
Tuple-based sliding window

- \( \omega \) tuples in the window
- Slide of \( \beta \) tuples
- R2R operator
- Contemporaneity implies a non-deterministic selection

\[ W(\omega, \beta) \]
SPARQL: a quick recap
Which is the format of the answer?

We can distinguish two cases

1. No R2S operator: the output is a relation (that changes during the time)
2. R2S operator: a stream.
   - An RDF stream? It depends by the Query Form
SELECT ?a ?b ...
FROM ....
WHERE ....

CONSTRUCT {?a :prop ?b }
FROM ....
WHERE ....
R2S operator: stream

- R2S operators

CONSTRUCT RSTREAM {?a :prop ?b } FROM .... WHERE ....

- Three operators:
  - Rstream: streams out all data in the last step
  - Istream: streams out data in the last step that wasn’t on the previous step, i.e. streams out what is new
  - Dstream: streams out data in the previous step that isn’t in the last step, i.e. streams out what is old
Brief overview on the CEP operators

- Sequence operators and CEP world

  - **SEQ**: joins $e_{t_1, t_f}$ and $e'_{t_1', t_f'}$ if $e'$ occurs after $e$
  
  - **EQUALS**: joins $e_{t_i, t_f}$ and $e'_{t_i', t_{f'}}$ if they occur simultaneously

  - **OPTIONALSEQ, OPTIONALEQUALS**: Optional join variants
1. Continuous RDF model extensions
   - RDF Streams, timestamps

2. Continuous extensions of SPARQL
   - Continuous evaluation
   - Additional operators

3. Overview of existing systems
   - Features
   - Comparison
Existing RSP systems (oversimplified!)

- **C-SPARQL**: RDF Store + Stream processor
  - Combined architecture

- **CQELS**: Implemented from scratch. Focus on performance
  - Native + adaptive joins for static-data and streaming data
Existing RSP systems (oversimplified!)

- **SPARQL\textsubscript{stream}:** Ontology-based stream query answering
  - Virtual RDF views, using R2RML mappings
  - SPARQL stream queries over the original data streams.

- **EP-SPARQL:** Complex-event detection
  - SEQ, EQUALS operators

- **Instans:** RETE-based evaluation
## Classification of existing systems

<table>
<thead>
<tr>
<th>Model</th>
<th>Model</th>
<th>Continuous execution</th>
<th>Union, Join, Optional, Filter</th>
<th>Aggregates</th>
<th>Time window</th>
<th>Triple window</th>
<th>R2S operator</th>
<th>Sequence, Co-occurrence</th>
</tr>
</thead>
<tbody>
<tr>
<td>TA-SPARQL</td>
<td>TA-RDF</td>
<td>✗</td>
<td>✔</td>
<td>Limited</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>tSPARQL</td>
<td>tRDF</td>
<td>✗</td>
<td>✔</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Streaming SPARQL</td>
<td>RDF Stream</td>
<td>✔</td>
<td>✗</td>
<td>✗</td>
<td>✔</td>
<td>✔</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>C-SPARQL</td>
<td>RDF Stream</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✗</td>
</tr>
<tr>
<td>CQELS</td>
<td>RDF Stream</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>❗</td>
<td>❗</td>
</tr>
<tr>
<td>SPARQLStream</td>
<td>(Virtual) RDF Stream</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>❗</td>
<td>✗</td>
<td>❗</td>
</tr>
<tr>
<td>EP-SPARQL</td>
<td>RDF Stream</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Instans</td>
<td>RDF</td>
<td>✔</td>
<td>✔</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
</tbody>
</table>

Disclaimer: other features may be missing
Similar models, similar (not equals!) query languages

```sparql
SELECT ?sensor
FROM NAMED STREAM <http://www.cwi.nl/SRBench/observations>
WHERE {
  ?observation om-owl:procedure ?sensor;
  om-owl:observedProperty weather:WindSpeed;
  om-owl:result [ om-owl:floatValue ?value ].
}
GROUP BY ?sensor
HAVING ( AVG(?value) >= "74"^^xsd:float )
```

```c-sparql
SELECT ?sensor
FROM STREAM <http://www.cwi.nl/SRBench/observations>
WHERE {
  ?observation om-owl:procedure ?sensor;
  om-owl:observedProperty weather:WindSpeed;
  om-owl:result [ om-owl:floatValue ?value ].
}
GROUP BY ?sensor
HAVING ( AVG(?value) >= "74"^^xsd:float )
```

```cqeels
SELECT ?sensor
WHERE {
  STREAM <http://www.cwi.nl/SRBench/observations> {
    ?observation om-owl:procedure ?sensor;
    om-owl:observedProperty weather:WindSpeed;
    om-owl:result [ om-owl:floatValue ?value ].
  }
}
GROUP BY ?sensor
HAVING ( AVG(?value) >= "74"^^xsd:float )
```
The correctness problem (1)

Where are Alice and Bob, when they are together?

Let’s consider a tumbling window $W(\omega=\beta=5)$

Let’s execute the experiment 4 times on C-SPARQL

<table>
<thead>
<tr>
<th>Execution</th>
<th>1° answer</th>
<th>2° answer</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>- [7]</td>
<td>- [12]</td>
</tr>
</tbody>
</table>

Which is the correct answer?
RSP output correctness: the $t_0$ parameter

Exec | 1° answer | 2° answer
--- | --- | ---
4 | - [7] | - [12]

Window | 1° answer | 2° answer
--- | --- | ---
t_0=0 | :hall [5] | :kitchen [10]
t_0=2 | - [7] | - [12]
Which system behaves in the correct way? Both!
The window operator (through SECRET)

\[ t_0 \]: When does the window start? (internal window param)

\textbf{TICK}: When are data stream elements added to the window? \textit{Triple-based vs graph-based}

\[ \omega \]

\[ \beta \]

\textbf{REPORT}: When is the window content made available to the R2R operator? \textit{Non-empty content, Content-change, Window-close, Periodic}

\[ W(\omega, \beta) \]
Understanding the RSPs

- They share similar models, but they behave in different ways
- The C-SPARQL, CQELS and SPARQLstream models does not allow to determine in a unique way which should be the answer given the inputs and the query
  - There are missing parameters (encoded in the implementations)
- Why is it important to understand those behaviours?
  - To assess the correct implementation of the systems
  - To improve the comprehension of the benchmarking
- W3C RDF stream processor community group started to jointly work out a recommendation in 2014
  - http://www.w3.org/community/rsp/
References

- **DSMSs and CEPs**

- **RDF Stream Processors**

- **Benchmarks and RSP comparison**
  - Danh Le Phuoc, Minh Dao-Tran, Minh-Duc Pham, Peter A. Boncz, Thomas Eiter, Michael Fink: Linked Stream Data Processing Engines: Facts and Figures. International Semantic Web Conference (2) 2012: 300-312