Tutorial on RDF Stream Processing 2016
M.I. Ali, J-P Calbimonte, D. Dell'Aglio, E. Della Valle, and A. Mauri
http://streamreasoning.org/events/rsp2016

RSP models
Daniele Dell’Aglio

dellaglio@ifi.uzh.ch
http://dellaglio.org
@dandellaglio
Share, Remix, Reuse — Legally

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

- **Your are free:**

  - ![Icon](http://streamreasoning.org/events/rsp2016) **to Share** — to copy, distribute and transmit the work
  - ![Icon](http://streamreasoning.org/events/rsp2016) **to Remix** — to adapt the work

- **Under the following conditions**

  - ![Icon](http://streamreasoning.org/events/rsp2016) **Attribution** — You must attribute the work by inserting
    - "[source http://streamreasoning.org/events/rsp2016]" at the end of each reused slide
    - a credits slide stating
      - These slides are partially based on "Tutorial on RDF Stream Processing 2016" by M.I. Ali, J-P Calbimonte, D. Dell'Aglio, E. Della Valle and Andrea Mauri
        http://streamreasoning.org/events/rsp2016

- To view a copy of this license, visit 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
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/)

\[
\langle \text{sub}_1 \, \text{pred}_1 \, \text{obj}_1 \rangle
\]

\[
\langle \text{sub}_2 \, \text{pred}_2 \, \text{obj}_2 \rangle
\]

- 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
Data items

- 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>`
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

http://streamreasoning.org/events/rsp2016
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-equijoin 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
In this session we consider the following setting:

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

\[ \langle :alice :isWith :bob \rangle [1] \]
\[ \langle :alice :isWith :carl \rangle [3] \]
\[ \langle :bob :isWith :diana \rangle [6] \]

\[ \ldots \]
Querying data streams – The CQL model

Streams

- Infinite unbounded sequence
- ...<s,T>...<s_t>...
- Stream

Relation R(t)
- Mapping: T → R

Relational algebra
- Relation-to-relation

Sliding windows
- Stream-to-relation

Stream operators

Relation R(t)
- Finite bag

*Stream operators
CQL extension for querying RDF data streams

- Sliding windows
- S2R operators
- SPARQL operators
- R2S operators
- *Stream operators

RDF Streams

Mappings
Time-based sliding window

- Slide: \( \omega \)
- Width: \( \beta \)
- R2R operator

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

\( S_1, S_2, S_3, S_4, S_5, S_6, S_7, S_8, S_9, S_{10}, S_{11}, S_{12} \)
Time-based sliding window - tumbling

- slide
- width

R2R operator

$W(\omega, \beta)$

$S_1 \quad S_3 \quad S_5 \quad S_7 \quad S_9 \quad S_2 \quad S_4 \quad S_6 \quad S_8 \quad S_{10} \quad S_{11} \quad S_{12}$

$t$
Tuple-based sliding window

- Slide of $\beta$ tuples
- R2R operator
- Contemporaneity implies a non-deterministic selection

$W(\omega, \beta)$
SPARQL: a quick recap

CONSTRUCT | DESCRIBE | SELECT | ASK

Query Form

Dataset
FROM
FROM NAMED

Clause

Where Clause
Triple pattern
OPTIONAL
AND
UNION

Dataset

TRUE - FALSE

+ SPARQL1.1
  e.g. aggregates
The query output

- 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
No R2S operator: relation

SELECT ?a ?b ...
FROM ....
WHERE ....

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

RSP

queries

bindings

triples
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

\[
\begin{align*}
&<... :prop ... > [t\to 1] \\
&<... :prop ... > [t\to 1] \\
&<... :prop ... > [t\to 3] \\
&<... :prop ... > [t\to 5] \\
&<... :prop ... > [t\to 7] \\
&\ldots
\end{align*}
\]
CEP operators

- Sequence operators and CEP world

- **SEQ**: joins $e_i$ and $e_j$ if $e_j$ occurs after $e_i$
- **EQUALS**: joins $e_i$ and $e_j$ if they occur simultaneously
- **AND**: joins $e_i$ and $e_j$ if they both occur
- **NOT**: check if $e_i$ does not exist
- …
CEP operators: examples

- B SEQ A
  - not matches

- A AND C SEQ D
  - matches!

- A SEQ NOT B SEQ C
  - not matches
CEP operators: intervals

P₁ SEQ P₃
P₂ AND P₃
P₂ OR P₃
P₁ PAR P₂
P₃ STARTS P₁
P₁ EQUALS P₃
NOT(P₃).[P₁, P₁]
P₃ FINISHES P₂
P₂ MEETS P₃

http://streamreasoning.org/events/rsp2016
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
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>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>X</td>
<td>✓</td>
<td>Limited</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>tSPARQL</td>
<td>tRDF</td>
<td>X</td>
<td>✓</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Streaming SPARQL</td>
<td>RDF Stream</td>
<td>✓</td>
<td>✓</td>
<td>X</td>
<td>✓</td>
<td>✓</td>
<td>X</td>
</tr>
<tr>
<td></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>Rstream only</td>
</tr>
<tr>
<td></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>Istream only</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>SPARQLStream</td>
<td>(Virtual) RDF Stream</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>X</td>
<td>✓</td>
</tr>
<tr>
<td></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>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td></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>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Disclaimer: other features may be missing
Tutorial on RDF Stream Processing 2016
M.I. Ali, J-P Calbimonte, D. Dell'Aglio, E. Della Valle, and A. Mauri
http://streamreasoning.org/events/rsp2016

RSP models
Daniele Dell’Aglio
✉ dellaglio@ifi.uzh.ch  🌐 http://dellaglio.org  @dandellaglio
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

http://streamreasoning.org/events/rsp2016