On unifying query languages for RDF streams

Daniele Dell’Aglio

dellaglio@ifi.uzh.ch  http://dellaglio.org  @dandellaglio

Stream Processing and RDF Stream Processing

In literature there are two different main approaches to process streams

Data Stream Management Systems (DSMSs)

- Aggregations and filters
- Relies on the notion of sliding window
- DSMS-inspired RSPs: C-SPARQL, CQELS, SPARQL_{stream}

Complex Event Processors (CEPs)

- Search of relevant patterns in the stream
- Non-equi-join on timestamps (after, before, etc.)
- CEP-inspired RSP: EP-SPARQL
- C-SPARQL offers some CEP features through a $timestamp$ function; CQELS has some initial extension on this
The problem
Unpredictable behaviour

\{ :a :isIn :rRoom \} \ \{ :b :isIn :rRoom \}
\{ :a :talksIn :bRoom \} \ \{ :b :talksIn :bRoom \}

Where are :a and :b, when they are together?
\{ :a :isIn ?room , :b :isIn ?room \}

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

Let’s execute the experiment 4 times on CSPARQL

<table>
<thead>
<tr>
<th>Exec.</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>
The problem
Different behaviours (1)

CSPARQL

<table>
<thead>
<tr>
<th>Exec.</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>

CQELS

<table>
<thead>
<tr>
<th>Exec.</th>
<th>1° answer</th>
<th>2° answer</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>No answers</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>No answers</td>
<td></td>
</tr>
</tbody>
</table>

08.12.2016  Daniele Dell’Aglio - Stream Reasoning workshop 2016 4/19
The problem
Different behaviours (2)

Who is listening to who?

{?att :isIn ?room}
SEQ
{?spe :talksIn ?room}

<table>
<thead>
<tr>
<th>Engine</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>C-SPARQL (sl. win. 7,1)</td>
<td>:b, :d [3,7]</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Research Question

In the context of **continuous query answering** over RDF streams, how can the **behaviour** of existing systems be captured, compared and contrasted?

Why do we need it?

- Comparison and contrast
- Interoperability
- Standard RSP query language
- Study RDF Stream Processing and Stream Reasoning related problems
A reference model that formally defines the semantics of RDF Stream Processing engines.
RSEP-QL
From SPARQL to RSEP-QL
RSEP-QL
From SPARQL to RSEP-QL

\[ Q \quad (E, SD, QF) \]

\[ E \]

Query Interface

SDS

Evaluator

Data layer

RDF graphs

RDF streams

Result Formatter

Ans(Q)
RSEP-QL
From SPARQL to RSEP-QL

Query Interface

Continuous Evaluator

SQL graphs

RDF graphs

Evaluator

Data layer

Result Formatter

Ans(Q)
RSEP-QL
From SPARQL to RSEP-QL

Q (E, DS, QF)

Query Interface

ET

Continuous Evaluator

SE

Evaluator

SDS

Data layer

RDF graphs

RDF streams

Result Formatter

Ans(Q)

Q (E, SDS, ET, QF)
RSEP-QL: Data
RDF Streams and windows

An RDF stream is an (infinite) ordered sequence of time-annotated data items (RDF graphs)
RSEP-QL: Operators

- SPARQL operators

- Stream Processing operators (RSP-QL)
  - WINDOW to specify that the active element is a window (similar to GRAPH)
  - RStream, IStream, DStream to create the output stream

- Event Processing operators (RSEP-QL)
  - MATCH to describe an event pattern
  - EVENT w P Basic Event Pattern
  - FIRST E
  - LAST E
  - E₁ SEQ E₂
RSEP-QL: Instantaneous Evaluation Semantics

The SPARQL evaluation function is defined as

$$\llbracket P \rrbracket_{DS(G)}$$

Two evaluation functions in RSEP-QL:

- For SPARQL and RSP-QL operators: $$\llbracket P \rrbracket_{SDS(A)}^t$$
- For event processing operators: $$\llbracket E \rrbracket_{(o,c)}^t$$

The two functions co-exist
RSEP-QL: Continuous Evaluation
Evaluation time instants

The evaluations usually do not occur at every time instant, but only at some of them (evaluation time instants)

• We indicate with ET the set of evaluation time instants

It is not always possible to define ET a priori

• ET is expressed through a Report Policy (from the SECRET framework)

Report Policy examples:

• WC Window Close: an evaluation occurs every time the active window closes
  • C-SPARQL, SPARQL\textsubscript{stream}
• CC Content Change: an evaluation occurs every time the stream content changes.
  • CQELS, EP-SPARQL
RSEP-QL: Continuous Evaluation
Consumption policies

The evaluations in a sequence are not always independent
  • Some data may be involved in the current evaluation only if it was not before

We need a mechanism to move this information between evaluations
  • Consumption policies are functions to describe the input for the next evaluation

Two policies for our needs
  • $P^u$: the input is the one selected by the window
  • $P^n$: the input is the one selected by the window and that was not used to produce results in the previous evaluations
RSEP-QL in action
Unpredictable behaviours

<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>

Window

<table>
<thead>
<tr>
<th>t₀=0</th>
<th>1° answer</th>
<th>2° answer</th>
</tr>
</thead>
</table>
RSEP-QL in action
Unpredictable behaviours

Execution | $1^\circ$ answer | $2^\circ$ answer
--- | --- | ---
4 | - [7] | - [12]

Window | $1^\circ$ answer | $2^\circ$ answer
--- | --- | ---
t_0=0 | :rRoom [5] | :bRoom [10]
RSEP-QL in action
Unpredictable behaviours

<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>

<table>
<thead>
<tr>
<th>Window</th>
<th>1° answer</th>
<th>2° answer</th>
</tr>
</thead>
<tbody>
<tr>
<td>t₀=0</td>
<td>:rRoom [5]</td>
<td>:bRoom [10]</td>
</tr>
<tr>
<td>t₀=2</td>
<td>- [7]</td>
<td>- [12]</td>
</tr>
</tbody>
</table>
RSEP-QL in action
Different behaviours (1)

{:a :isIn :rRoom}  
{:b :isIn :rRoom}  
{:a :talksIn :bRoom}  
{:b :talksIn :bRoom}

CSPARQL

CQELS

<table>
<thead>
<tr>
<th>Exec.</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>

<table>
<thead>
<tr>
<th>Exec.</th>
<th>1° answer</th>
<th>2° answer</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>No answers</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>No answers</td>
<td></td>
</tr>
</tbody>
</table>

08.12.2016    Daniele Dell’Aglio - Stream Reasoning worksho 2016
RSEP-QL in action
Different behaviours (2)

SEQ operators with different selection policies

- $E_1 \text{SEQ} E_2$
  - EP-SPARQL (unr.)
- $E_1 \text{SEQ}^c E_2$
  - EP-SPARQL (chr.)
- $E_1 \text{SEQ}^r E_2$
  - EP-SPARQL (rec.)
- $E_1 \text{SEQ}^n E_2$
  - C-SPARQL

<table>
<thead>
<tr>
<th>Engine</th>
<th>7</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>:b, :d [3,5]</td>
<td></td>
</tr>
<tr>
<td>EP-SPARQL (chronological)</td>
<td>:a, :c [1,5]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>:b, :d [3,5]</td>
<td></td>
</tr>
<tr>
<td>EP-SPARQL (recent)</td>
<td>:b, :d [3,5]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>:b, :d [3,7]</td>
<td></td>
</tr>
<tr>
<td>C-SPARQL (sl. win. 7,1)</td>
<td>:b, :d [3,7]</td>
<td></td>
</tr>
</tbody>
</table>

Different consumption policies

08.12.2016 Daniele Dell’Aglio - Stream Reasoning workshop 2016
## Captured behaviours

<table>
<thead>
<tr>
<th>System</th>
<th>Report policy</th>
<th>Streaming operator</th>
<th>Window operator</th>
<th>Sequencing operator</th>
<th>Consumption policy</th>
</tr>
</thead>
<tbody>
<tr>
<td>CQELS</td>
<td>Content change</td>
<td>Istream</td>
<td>Sliding</td>
<td>SEQ</td>
<td>( p^u )</td>
</tr>
<tr>
<td>SPARQL\text{_stream}</td>
<td>Window close</td>
<td>Rstream, Dstream, Istream</td>
<td>Sliding</td>
<td>-</td>
<td>( p^u )</td>
</tr>
<tr>
<td>C-SPARQL</td>
<td>Window close</td>
<td>Rstream</td>
<td>Sliding</td>
<td>( \text{SEQ}^n ) (timestamp function)</td>
<td>( p^u )</td>
</tr>
<tr>
<td>EP-SPARQL\text{_unrestricted}</td>
<td>Content change</td>
<td>Rstream</td>
<td>Landmark</td>
<td>SEQ</td>
<td>( p^u )</td>
</tr>
<tr>
<td>EP-SPARQL\text{_chronological}</td>
<td>Content change</td>
<td>Rstream</td>
<td>Landmark</td>
<td>( \text{SEQ}^c )</td>
<td>( p^n )</td>
</tr>
<tr>
<td>EP-SPARQL\text{_recent}</td>
<td>Content change</td>
<td>Rstream</td>
<td>Landmark</td>
<td>( \text{SEQ}^r )</td>
<td>( p^n )</td>
</tr>
</tbody>
</table>
Conclusions

The dynamics introduced in the continuous query evaluation process have not been totally understood

- Not fully captured by existing models

RSEP-QL captures those dynamics, and the behaviours of existing RDF Stream Processing engines

- DSMS-based (RSP-QL)
- CEP-based (RSEP-QL)

What’s next

- Continuous query answering and reasoning
- Further investigation about the relation of RSEP-QL with other models (SPARQL, LARS)
Thank you! Questions?

On unifying query languages for RDF streams

Daniele Dell’Aglio
dellaglio@ifi.uzh.ch
http://dellaglio.org
@dandellaglio