How to Build a Stream Reasoning Application
D. Dell'Aglio, E. Della Valle, T. Le-Pham, A. Mileo, and R. Tommasini
http://streamreasoning.org/events/streamapp2017

Continuous queries
Daniele Dell’Aglio
email: dellaglio@ifi.uzh.ch
http://dellaglio.org
@dandellaglio
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 “How to Build a Stream Reasoning Application 2017” by D. Dell'Aglio, E. Della Valle, T. Le-Pham, A. Mileo, and R. Tommasini available online at http://streamreasoning.org/events/streamapp2017

To view a copy of this license, visit http://creativecommons.org/licenses/by/3.0/
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
Querying data streams – The CQL model

Streams

Relations

Sliding windows

stream-to-relation

Relation $R(t)$

Mapping: $T \rightarrow R$

relation-to-stream

relation-to-relation

finite bag

*Stream operators

infinite unbounded sequence

... $<s,T>$ ...

Stream

<s₁>

<s₂>

<s₃>
CQL extension for querying RDF data streams

RDF Streams

Mappings

S2R operators

Sliding windows

SPARQL operators

R2S operators

*Stream operators
Time-based sliding window

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

- **slide**
- **width**

\[ \omega \]

\[ \beta \]
SPARQL: a quick recap

+ SPARQL1.1 e.g. aggregates
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

a→ ... b→... [t→1]
a→ ... b→...

a→ ... b→... [t→3]
a→ ... b→...

a→ ... b→... [t→5]
a→ ... b→...

a→ ... b→... [t→7]

<... :prop ... > [t→1]
<... :prop ... >

<... :prop ... > [t→3]
<... :prop ... >

<... :prop ... > [t→5]
<... :prop ... >

<... :prop ... > [t→7]
### R2S operators

- **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**
Some existing RSP systems (oversimplified!)

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

  ![Diagram](http://streamreasoning.org/events/streamapp2017)

- **CQELS:** Implemented from scratch. Focus on performance
  - Native + adaptive joins for static-data and streaming data

  ![Diagram](http://streamreasoning.org/events/streamapp2017)
Some 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.

- **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>
</tr>
</thead>
<tbody>
<tr>
<td>TA-SPARQL</td>
<td>TA-RDF</td>
<td>✓</td>
<td></td>
<td>Limited</td>
<td></td>
<td></td>
</tr>
<tr>
<td>tSPARQL</td>
<td>tRDF</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>
</tr>
<tr>
<td>C-SPARQL</td>
<td>RDF Stream</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>
</tr>
<tr>
<td>SPARQLStream</td>
<td>(Virtual) RDF Stream</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>
</tr>
</tbody>
</table>

Disclaimer: only a partial view
SELECT ?sensor
FROM NAMED STREAM <http://www.cwi.nl/SRBench/observations> [NOW-3 HOURS SLIDE 10 MINUTES]
WHERE {
  ?observation om-owl:procedure ?sensor ;
  om-owl:observedProperty weather:WindSpeed ;
}
GROUP BY ?sensor HAVING ( AVG(?value) >= "74"^^xsd:float )

SELECT ?sensor
FROM STREAM <http://www.cwi.nl/SRBench/observations> [RANGE 1h STEP 10m]
WHERE {
  ?observation om-owl:procedure ?sensor ;
  om-owl:observedProperty weather:WindSpeed ;
}
GROUP BY ?sensor HAVING ( AVG(?value) >= "74"^^xsd:float )

SELECT ?sensor
WHERE {
  STREAM <http://www.cwi.nl/SRBench/observations> [RANGE 10800s SLIDE 600s] {
    ?observation om-owl:procedure ?sensor ;
    om-owl:observedProperty weather:WindSpeed ;
  }
}
GROUP BY ?sensor HAVING ( AVG(?value) >= "74"^^xsd:float )

Similar models, similar (not equals!) query languages

SPARQLstream

C-SPARQL

CQELS
The 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

<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?
The problem (2)

Which system behaves in the correct way?

<table>
<thead>
<tr>
<th>Executions</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>

CSPARQL

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

CQELS
Understanding the RSPs

- They share similar models, but they behave in different ways
- The C-SPARQL, CQELS and SPARQL\textsubscript{stream} models do 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/](http://www.w3.org/community/rsp/)
The problem (3)

- 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
  - Study RDF Stream Processing related problems
  - Standard RSP query language
A reference model that formally defines the semantics of RDF Stream Processing engines.

- Event Pattern detection operators
- Model to express continuous queries
- BGP evaluation over background data
- BGP evaluation over streams
RSEP-QL
From SPARQL to RSEP-QL
RSEP-QL: Dataset
Time-based Sliding Window: \( W(\omega, \beta, t_0) \)

- \( t_0 \): When does the window start? (internal window param)
- Sequence of timestamped graphs (stream items)

\[ W(3, 1, 1) \]
RSEP-QL: Dataset
Landmark window: $\mathbb{L}(t_0)$

Sequence of timestamped graphs (stream items)
RSEP-QL: Dataset
From SPARQL to RSEP-QL dataset

RESP-QL dataset

Time-Varying Graph
$G: T \rightarrow R$

Instantaneous Graph
$G(t_1) \in R$

SPARQL dataset

$T \subseteq \mathbb{N}$

$R = \{ \text{RDF graph} \}$

$W(S)$

$G(t_1)$

$H$

$S_1, S_3, S_4, S_5, S_6, S_7, S_8, S_9, S_{10}, S_{11}, S_{12}$
RSEP-QL: Operators

- Stream Processing operators (RSP-QL)
  - SPARQL operators
  - 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)
  - Not covered in this tutorial
RSEP-QL: Evaluation Semantics
Stream Processing Evaluation Semantics

- The SPARQL evaluation function is defined as
  
  
  \[ \left\{ P \right\}_{DS(G)} \]

- The RSEP-QL evaluation function extends the SPARQL one by introducing the evaluation time instant
  
  \[ \left\langle P \right\rangle_{SDS(A)}^{t} \]

- SPARQL operators are straight extended to the new evaluation function

- Example: JOIN

  - \[ \left\{ JOIN(P_{1}, P_{2}) \right\}_{DS(G)} = \left\{ P_{1} \right\}_{DS(G)} \bowtie \left\{ P_{2} \right\}_{DS(G)} \]

  - \[ \left\langle JOIN(P_{1}, P_{2}) \right\rangle_{SDS(A)}^{t} = \left\langle P_{1} \right\rangle_{SDS(A)}^{t} \bowtie \left\langle P_{2} \right\rangle_{SDS(A)}^{t} \]
The main difference is on the BGP evaluation:

\[ \langle BGP \rangle_{SDS(A)}^t = [BGP]_{SDS(A,t)} \]

SDS \((A, t)\) is:

- \(SDS(G, t) = SDS(G(t))\) if \(A\) is a time-varying graph \(G\)
- \(SDS(\mathbb{W}(S), t) = SDS(m(\mathbb{W}(S, t)))\) if \(A\) is from a sliding window \(\mathbb{W}\)
- \(SDS(\mathbb{L}(S), t) = SDS(m(\mathbb{L}(S, t)))\) if \(A\) is from a landmark window \(\mathbb{L}\)

where \(m\) denotes a **merge** function

\[ m(\mathbb{W}(S, t)) = \bigcup_{(d_i, t_i) \in \mathbb{W}(S, t)} d_i \]

- takes as input a window content i.e. a sequence of timestamped RDF graphs
- produces an RDF graph
How to Build a Stream Reasoning Application
D. Dell’Aglio, E. Della Valle, T. Le-Pham, A. Mileo, and R. Tommasini
http://streamreasoning.org/events/streamapp2017

Continuous queries
Daniele Dell’Aglio

✉ dellaglio@ifi.uzh.ch  ➨ http://dellaglio.org  ➤ @dandellaglio
We use a new evaluation function \((\cdot)^t_{(o,c)}\):

- \(t\) is the evaluation time instant (as in \(\langle \cdot \rangle^t_{SDS(A)}\)),
- \((o, c]\) is an additional window to identify the portion of the data on which the event may happen.

Event pattern evaluation produces event mappings \((\mu, t_1, t_2)\):

- \(\mu\) is a solution mapping,
- \(t_1\) and \(t_2\) denote the time interval justifying \(\mu\).
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
- ...

Sequence Simultaneous

1 3 6 9

A B C D

S
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

- $P_1$ SEQ $P_3$
- $P_2$ AND $P_3$
- $P_2$ OR $P_3$
- $P_1$ PAR $P_2$
- $P_3$ STARTS $P_1$
- $P_1$ EQUALS $P_3$
- NOT($P_3$).[$P_1$, $P_1$]
- $P_3$ FINISHES $P_2$
- $P_2$ MEETS $P_3$
RSEP-QL: Evaluation Semantics

Evaluation semantics - Example

- \( (\text{FIRST EVENT } w_1 P_1 \text{ SEQ EVERY EVENT } w_2 P_2)^t_{(9,16)} \)
RSEP-QL: Evaluation Semantics
MATCH graph pattern

- Event patterns are eclosed in MATCH graph patterns
- Event mappings exist only in the context of event patterns
- The evaluation of a MATCH graph pattern produces a bag of solution mappings
  \[ \langle \text{MATCH } E \rangle_{SDS(A)}^t = \{ \mu | (\mu, t_1, t_2) \in \langle E \rangle_{(0,t]}^t \} \]
- It is possible to combine the MATCH graph pattern with other SPARQL graph patterns
RSEP-QL: Evaluation Semantics
Continuous evaluation

- For each evaluation time $t \in ET$: $\langle SE \rangle^t_{SDS(A)}$
  - The continuous evaluation is a sequence of instantaneous evaluations

- It is not always possible to compute ET a priori
  - Can be data dependent
  - ET is expressed through a Report Policy
A Report Policy is a set of conditions to one or more window operators in SDS
- Initially defined in SECRET for Stream Processing engines

Report Policy examples:
- **P Periodic**: the window reports only at regular intervals
- **WC Window Close**: the window reports if the active window closes
- **CC Content Change**: the window reports if the content changes.
RSEP-QL in action
Correctness assessment

Execution

<table>
<thead>
<tr>
<th></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></th>
<th>1° answer</th>
<th>2° answer</th>
</tr>
</thead>
<tbody>
<tr>
<td>t₀=2</td>
<td>- [7]</td>
<td>- [12]</td>
</tr>
</tbody>
</table>
RSEP-QL in action
Correctness assessment

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

Window-close vs content-change

Executions | 1° answer | 2° answer
---|---|---
2 | | No answers
4 | | No answers

Empty relation notification (yes|no)
CSR Bench

- **CSR-bench** is an extension of the SRbench benchmark that focuses on correctness

- A test suite
  - LinkedSensorData as dataset
  - 8 parametric queries to tests the RSP engines in different conditions

- An **oracle** (an automatic correctness validator)
  - Based on RSP-QL
CSR Bench
Oracle

Q
(E, SDS, ET, QF)

Online

Data importer

DS

SPARQL engine

Result matcher

Offline

Query transformer

RSP-QL model of R

Correctness assessment
All the three systems that we considered in our experiments showed wrong behaviours.

The defects we identified are related to:
- the window operator
  - Initialization
  - Slide parameter
  - Window contents
- timestamps of the triples
  - Internal timestamp management
RSEP-QL: Results

- **RSEP-QL captures** the evaluation semantics of **existing RSP engines**
- RSEP-QL can determine which are the **expected correct answers** of an RSP engine, given the input data and query
  - At the basis of CSR Bench
- 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