An Engine for Ontology-Based Stream Processing
Theory and Implementation

Christian Neuenstadt

6. Februar 2018
Lübeck
**Motivation - Use Case**

**Optique**

**2012-2016**

Complex case:
- **engineer** information need
- **IT expert** specialized query

Optique solution:
- **engineer**
- **Optique Application**
- **flexible, ontology based queries**
- **Query translation**
- **translated queries**
- **disparate sources**
Query Answering

SPARQL

SELECT ?x WHERE {
  ?x ?type Turbine .
  ?x ?type TempSensor .
  GasTurbine1 ?type GasTurbine1 .
  Sensor1 ?type Sensor1 .
  Sensor1 observedProperty mountedAt GasTurbine1 .
  Sensor1 type Sensor1 .
  Temperature ?type Temperature .
  Ontology
}

C. Neuenstadt
An Engine for Ontology-Based Stream Processing
Query Transformation
From SPARQL to SQL

SELECT ?x WHERE
   {?x a ?type .
    tempSensor .
    Turbine .
    ?x a mountedAt .
    gasTurbine1 .
    BGP
   }

SELECT SensorName FROM Sensors
WHERE ...

Mappings

SELECT SensorName FROM Sensors
WHERE ...

Mountings

C. Neuenstadt
An Engine for Ontology-Based Stream Processing
Query Transformation
From SPARQL to SQL

**SELECT ?x WHERE**

```sparql
(SELECT x
WHERE {x type TempSensor .
   x type Sensor .
   x observesProp Temperature .
   GasTurbine1 type Turbine .
   x mountedAt GasTurbine1 .
})
```

**Motivation**

**STARQL Transformation**

**Evaluation**
Query Transformation
From SPARQL to SQL

SELECT ?x WHERE

x? type TempSensor .

x? type Sensor .

x? observesProp Temperature .

GasTurbine1 type Turbine .

x? mountedAt GasTurbine1

SELECT SensorName AS ?x FROM Sensors

SELECT SensorID SensorName Property
WHERE ...

<table>
<thead>
<tr>
<th>SensorID</th>
<th>SensorName</th>
<th>Property</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Sensor1</td>
<td>Temperature</td>
</tr>
</tbody>
</table>

Mountings

<table>
<thead>
<tr>
<th>SensorID</th>
<th>Turbine</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Gasturbine1</td>
</tr>
</tbody>
</table>

BGP

Motivation

C. Neuenstadt
An Engine for Ontology-Based Stream Processing
Transformation of Temporal Queries
Ontology-Based Stream Processing

Graph Stream

Tuple Stream

Query Formulation

Result

Application layer

Transformation layer

Database layer

Answers
The Idea of STARQL


- We have developed a new query language for ontology-based streams
  1) Uses temporal operators on state sequences
  2) Adopts current ontology standards
  3) Evaluates multiple streams
- We have implemented a query transformation strategy
- We execute transformed STARQL queries in modern database environments
  - DBMS for historic data
  - DSMS for live streams
A static graph pattern

TempSensor

mountedAt

GasTurbine

BGP
A temporal graph pattern
Graph pattern and streaming data
Graph pattern and streaming data
Graph pattern and streaming data

93°C

94°C

TempSensor

hasVal

91°C

90°C

BGP
Graph pattern and streaming data

TempSensor

hasVal

91°C
90°C
93°C
94°C
From temporal graphs to temporal states
From temporal graphs to temporal states
A window operator

\[ S_{Msmt}[NOW - 3s, NOW] \rightarrow 1s \]
A window operator

\[ S_{Msmt}[NOW - 3s, NOW] \rightarrow 1s \]
A window operator

\[ S_{Msmt}[NOW - 3s, NOW] \rightarrow 1s \]
STARQL Example 1 - Threshold

\[ \exists i, x(R_1(x, i) \land x > 93) \]

Example

```sql
SELECT ?x
FROM S_Msmt [NOW-3s, NOW]-> 1s
WHERE { :tempSensor :mountedAt :GasTurbine }
HAVING EXISTS ?i IN (GRAPH ?i { :tempSensor :hasVal ?x }
  AND ?x > 93)
```
STARQL Example 2 - Monotonic Increase

\[ \forall i, j, x, y ( R_1(sens, x, i) \land R_2(sens, y, j) \land i < j \rightarrow x \leq y ) \]
Transformation of temporal Graph Patterns with STARQL

Static mapping example

\[ ?\text{sens} : \text{type} : \text{Sensor} \leftarrow \text{SELECT SensorName AS } ?\text{sens} \]
\[ \text{FROM Sensors} \quad (1) \]

Time based mapping example

\[ \text{GRAPH } i \{ \ ?\text{sens hasVal } ?y \} \leftarrow \text{SELECT sId as } ?\text{sens, val as } ?y \]
\[ \text{FROM Slice(Measurement,} i, r, sl, st \text{).} \quad (2) \]

- **i**: index of the specific temporal state
- **r**: range of the window operator
- **sl**: slide parameter of the window operator
- **st**: sequencing strategy of the sequence generator
Schematic Transformation of STARQL queries

1. **ABOX**
   - Virtual Interpreter
   - STARQL

2. **Q_{STARQL}**
   - WHERE
   - HAVING

3. **BGP**
   - **Q_{SPARQL}**
   - Rel. Algebra

4. **Q_{QL}**
   - Interpreter
   - QL

5. **DSMS**
   - **Results**

6. **Results**

7. **Back Transformation**
Comparison of implemented backend examples

<table>
<thead>
<tr>
<th></th>
<th>PostgreSQL</th>
<th>PipelineDB</th>
<th>Exareme</th>
<th>Spark</th>
</tr>
</thead>
<tbody>
<tr>
<td>Live Streams</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Static Data</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Historic Streams</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>API</td>
<td>JDBC</td>
<td>JDBC</td>
<td>REST API</td>
<td>REST API / built in</td>
</tr>
</tbody>
</table>

“Semantic access to streaming and static data at Siemens”
Journal of Web Semantics 2017

“Towards Analytics Aware Ontology Based Access to Static and Streaming Data” ISWC 2016

“OBDA for Temporal Querying and Streams” HiDeSt@KI 2015

“A Stream-Temporal Query Language for Ontology Based Data Access”
DL 2014 / KI 2014
Comparison of implemented backend examples

<table>
<thead>
<tr>
<th></th>
<th>PostgreSQL</th>
<th>PipelineDB</th>
<th>Exareme</th>
<th>Spark</th>
</tr>
</thead>
<tbody>
<tr>
<td>Live Streams</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Static Data</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Historic Streams</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>API</td>
<td>JDBC</td>
<td>JDBC</td>
<td>REST API</td>
<td>REST API / built in</td>
</tr>
</tbody>
</table>

“Semantic access to streaming and static data at Siemens”
Journal of Web Semantics 2017

“Towards Analytics Aware Ontology Based Access to Static and Streaming Data” ISWC 2016

“OBDA for Temporal Querying and Streams” HiDeSt@KI 2015

“A Stream-Temporal Query Language for Ontology Based Data Access”
DL 2014 / KI 2014
Comparison of implemented backend examples

<table>
<thead>
<tr>
<th></th>
<th>PostgreSQL</th>
<th>PipelineDB</th>
<th>Exareme</th>
<th>Spark</th>
</tr>
</thead>
<tbody>
<tr>
<td>Live Streams</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Static Data</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Historic Streams</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>API</td>
<td>JDBC</td>
<td>JDBC</td>
<td>REST API</td>
<td>REST API / built in</td>
</tr>
</tbody>
</table>

“Semantic access to streaming and static data at Siemens”
Journal of Web Semantics 2017

“Towards Analytics Aware Ontology Based Access to Static and Streaming Data” ISWC 2016

“OBDA for Temporal Querying and Streams” HiDeSt@KI 2015

“A Stream-Temporal Query Language for Ontology Based Data Access”
DL 2014 / KI 2014
Comparison of implemented backend examples

<table>
<thead>
<tr>
<th></th>
<th>PostgreSQL</th>
<th>PipelineDB</th>
<th>Exareme</th>
<th>Spark</th>
</tr>
</thead>
<tbody>
<tr>
<td>Live Streams</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Static Data</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Historic Streams</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>API</td>
<td>JDBC</td>
<td>JDBC</td>
<td>REST API</td>
<td>REST API / built in</td>
</tr>
</tbody>
</table>

“**Semantic access to streaming and static data at Siemens**”
Journal of Web Semantics 2017

“**Towards Analytics Aware Ontology Based Access to Static and Streaming Data**” ISWC 2016

“**OBDA for Temporal Querying and Streams**” HiDeSt@KI 2015

“**A Stream-Temporal Query Language for Ontology Based Data Access**”
DL 2014 / KI 2014
Experimental Evaluation

Prototypical Implementation

Experiment 1: PostgreSQL / Spark (Historic Data)

- Threshold and MonInc query executed on different data volumes
- Time scales for larger dataset with INTRAstate comparison
- But INTERstate comparisons are expensive!!

Experiment 2: Multi Core Evaluation

- Prototypical implementation per window execution based on pl/pgSQL
- Reduces data set per execution dramatically for interstate queries
- Scales by number of cores
- Overhead for each window execution is not applicable to Spark
Related Work

SRBenchmark Evaluation

<table>
<thead>
<tr>
<th>Language</th>
<th>SPARQLStream</th>
<th>C-SPARQL</th>
<th>CQELS</th>
<th>STARQL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Supported queries</td>
<td>17</td>
<td>17</td>
<td>11</td>
<td>11</td>
</tr>
</tbody>
</table>

Missing functionalities of STARQL are: ASK queries(1) and Property Paths(6)

Overall comparison

Query Language:
- All other three languages handle incoming triples as one graph per window.
- Only C-SPARQL accesses timestamps or temporal ordering directly

Transformation:
- Only SPARQLStream and STARQL can be transformed to relational algebra
- C-SPARQL / CQELS use their own execution environment
Summary/Outlook

- We have shown how we can query intra/inter state-based temporal sequences with temporal analytics in a new query language with syntax and semantics.
- We defined a new extended query transformation strategy that allows for an execution on relational DB and streaming systems.
- We executed the transformed queries on large volumes of batch and streamed data successfully and showed their scalability regarding distributed window execution.
- Future extensions:
  1) Extend temporal operators and aggregation functions
  2) Optimize window execution on backend systems
  3) Extend ontology language