UNIVERSITÄT ZU LÜBECK

An Engine for Ontology-Based Stream Processing

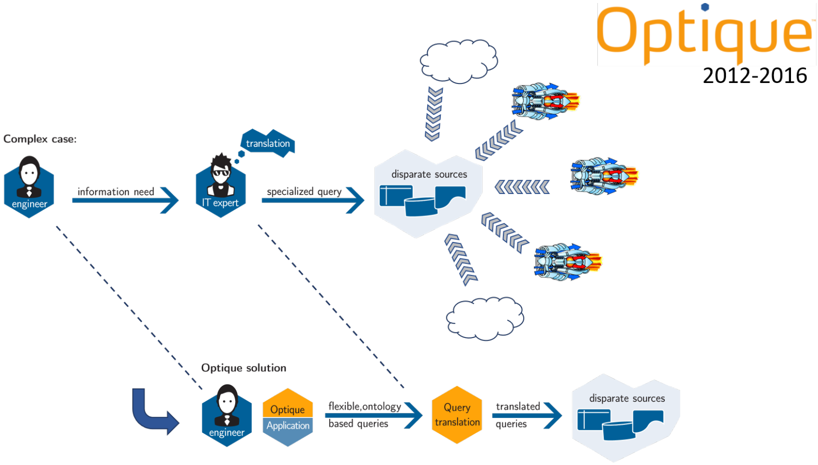
Theory and Implementation

Christian Neuenstadt

6. Februar 2018

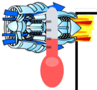
Lübeck

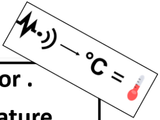
Motivation - Use Case



Query Answering

SPARQL





Sensor1	**Type**	**Sensor .**
Sensor1	**observedProperty**	**Temperature .**
Sensor1	**mountedAt**	*GasTurbine1*

Ontology

SELECT ***?x*** WHERE

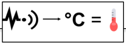
GasTurbine1	**type**	**Turbine .**
x?	**type**	**TempSensor .**
x?	**mountedAt**	*GasTurbine1*

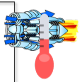
BGP

→ *Sensor1*

Query Transformation

From SPARQL to SQL





SELECT ***?x*** WHERE

x?	**type**	**TempSensor .**
GasTurbine1	**type**	**Turbine .**
x?	**mountedAt**	*GasTurbine1*

BGP

SPARQL

Mappings

SELECT ***SensorName*** FROM

Sensors

SensorID	SensorName	Property
1	Sensor1	Temperature

⋈

Mountings

SensorID	Turbine
1	Gasturbine1

WHERE ...

SQL

Query Transformation

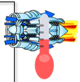
From SPARQL to SQL

SELECT ***?x*** WHERE

~~*x?*~~	~~**type**~~	~~**TempSensor .**~~
x?	**type**	**Sensor .**
x?	**observesProp**	**Temperature .**
GasTurbine1	**type**	**Turbine .**
x?	**mountedAt**	*GasTurbine1*

BGP

SPARQL



Mappings

SELECT ***SensorName*** FROM

Sensors

SensorID	SensorName	Property
1	Sensor1	Temperature

⋈

Mountings

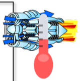
SensorID	Turbine
1	Gasturbine1

WHERE ...

SQL

Query Transformation

From SPARQL to SQL



SPARQL

SELECT ***?x*** WHERE

BGP

~~*x?*~~	~~**type**~~	~~**TempSensor .**~~
x?	**type**	**Sensor .**
x?	**observesProp**	**Temperature .**
GasTurbine1	**type**	**Turbine .**
x?	**mountedAt**	*GasTurbine1*

Mappings

x?* type Sensor** ⟵ ***SELECT SensorName AS ?x FROM Sensors

SQL

SELECT ***SensorName*** FROM

Sensors

SensorID	SensorName	Property
1	Sensor1	Temperature

⋈

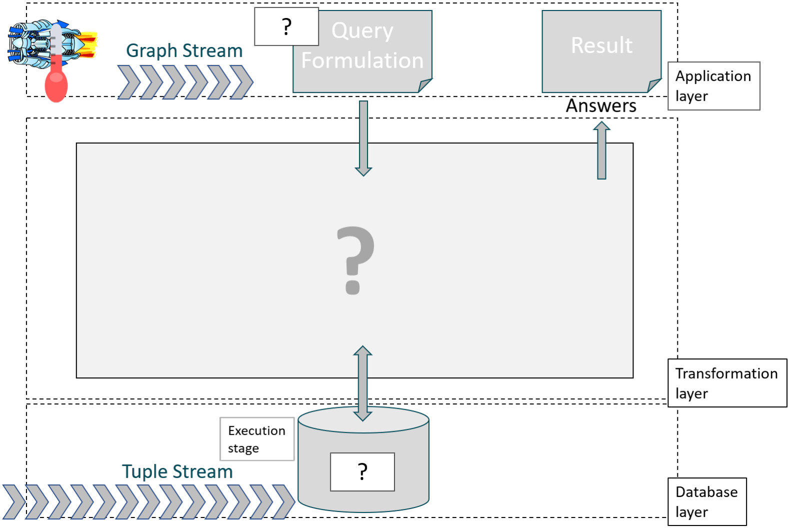
Mountings

SensorID	Turbine
1	Gasturbine1

WHERE ...

Transformation of Temporal Queries

Ontology-Based Stream Processing

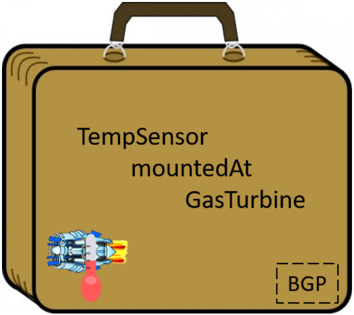


The Idea of STARQL

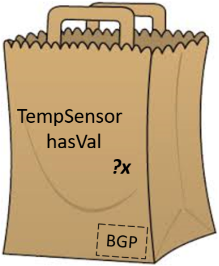
[S]treaming and [T]emporal ontology [A]ccess with a [R]easoning-based [Q]uery [L]anguage

- 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

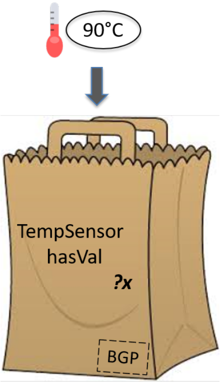
A static graph pattern



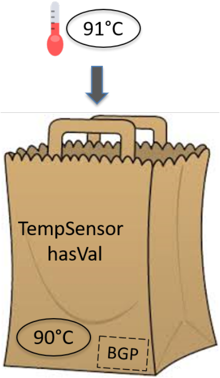
A temporal graph pattern



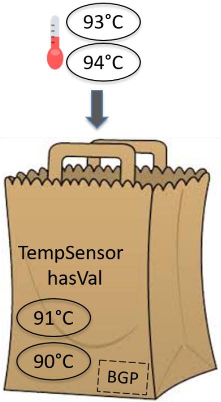
Graph pattern and streaming data



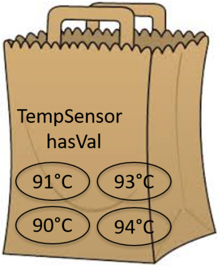
Graph pattern and streaming data



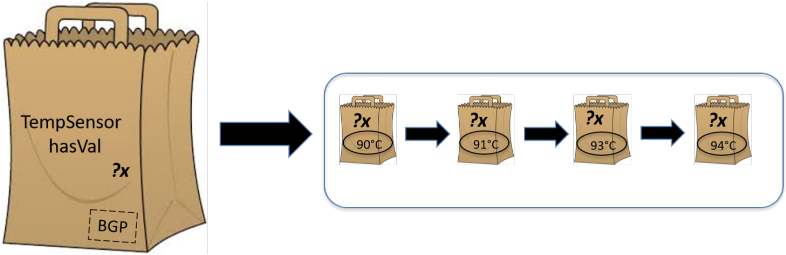
Graph pattern and streaming data



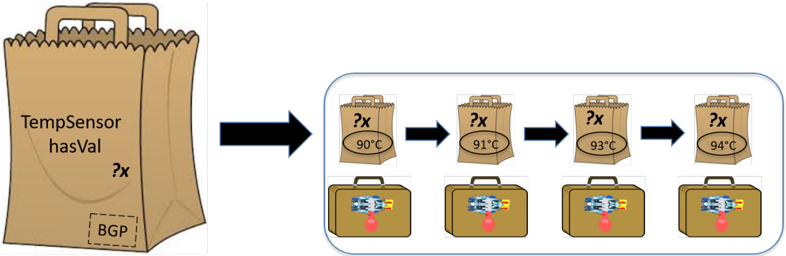
Graph pattern and streaming data



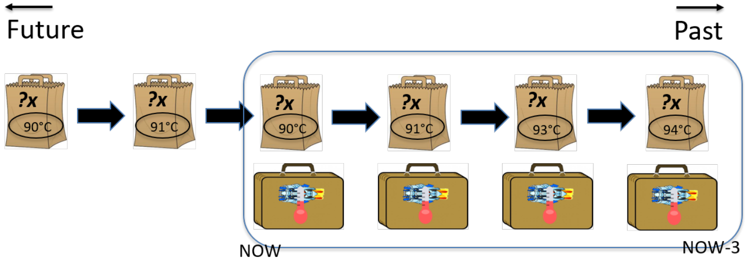
From temporal graphs to temporal states



From temporal graphs to temporal states

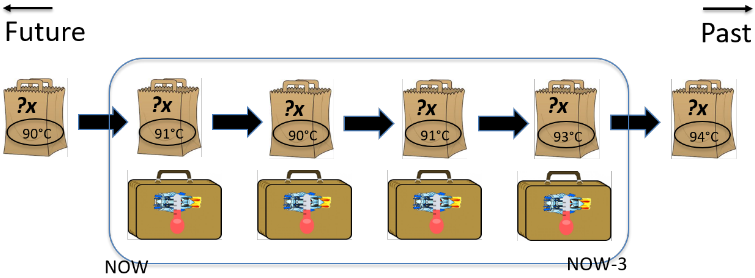


A window operator



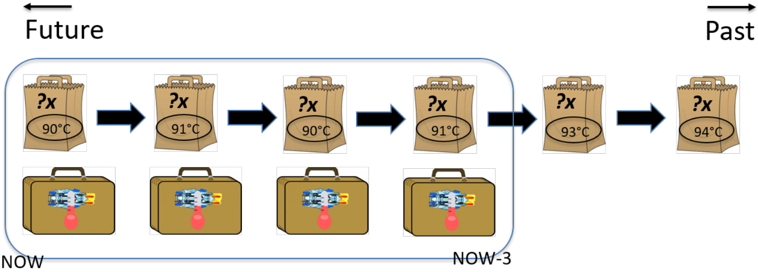
$$S_{Msmt}[NOW - 3s, NOW] \rightarrow 1s$$

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

```
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

$$\forall i, j, x, y(R_1(sens, x, i) \wedge R_2(sens, y, j) \wedge i < j \rightarrow x \leq y))$$

Example

```
CONSTRUCT GRAPH NOW {:tempSensor rdf:type MonInc }
FROM S_Msmt [NOW-3s, NOW]-> 1s
WHERE {  :tempSensor :mountedAt :GasTurbine }
HAVING FORALL ?i, ?j, ?x, ?y IN (
    IF GRAPH ?i { :tempSensor :hasVal ?x }
        AND GRAPH ?j { :tempSensor :hasVal ?y }
        AND ?i < ?j
    THEN ?x <= ?y )
```

Transformation of temporal Graph Patterns with STARQL

Static mapping example

$$?sens\ :type\ :Sensor \leftarrow \text{SELECT SensorName AS ?sens FROM Sensors} \tag{1}$$

Time based mapping example

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

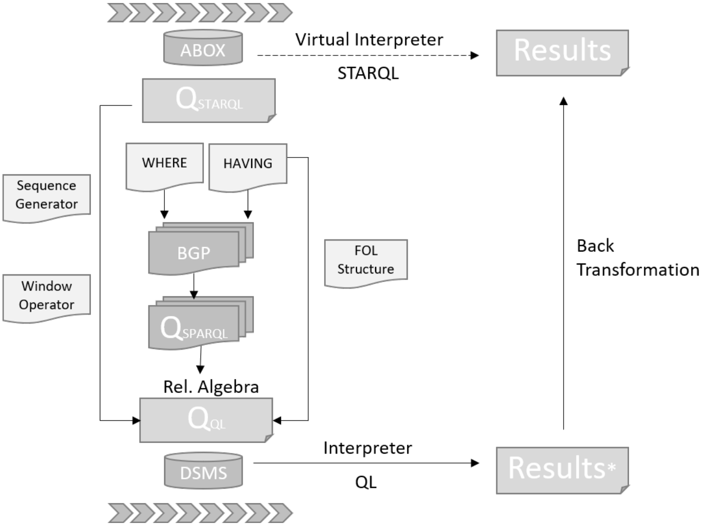
i: index of the specific temporal state

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



Comparison of implemented backend examples

	PostgreSQL	PipelineDB	Exareme	Spark
Live Streams	No	Yes	Yes	Yes
Static Data	Yes	Yes	Yes	Yes
Historic Streams	Yes	No	Yes	Yes
API	JDBC	JDBC	REST API	REST API / built in

"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

	PostgreSQL	PipelineDB	Exareme	Spark
Live Streams	No	Yes	Yes	Yes
Static Data	Yes	Yes	Yes	Yes
Historic Streams	Yes	No	Yes	Yes
API	JDBC	JDBC	REST API	REST API / built in

"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

	PostgreSQL	PipelineDB	Exareme	Spark
Live Streams	No	Yes	Yes	Yes
Static Data	Yes	Yes	Yes	Yes
Historic Streams	Yes	No	Yes	Yes
API	JDBC	JDBC	REST API	REST API / built in

"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

	PostgreSQL	PipelineDB	Exareme	Spark
Live Streams	No	Yes	Yes	Yes
Static Data	Yes	Yes	Yes	Yes
Historic Streams	Yes	No	Yes	Yes
API	JDBC	JDBC	REST API	REST API / built in

"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 volumns
- 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

Language	SPARQLStream	C-SPARQL	CQELS	STARQL
Supported queries	17	17	11	11

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 volumns 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