Low-cost Publication and Querying of Dynamic Linked Data

Ruben Taelman - @rubensworks
imec - Ghent University
RDF Stream

*stream of <RDF statements, timestamp>*

e.g. Thermometer measures every minute:

`:thermometer1 :hasValue “19,05°C” [11:00]
:thermometer1 :hasValue “19,06°C” [11:01]
:thermometer1 :hasValue “19,11°C” [11:02]
:thermometer1 :hasValue “19,08°C” [11:03]
...

RDF Stream Processing mainly happens server-side

Clients send query to server: e.g. *What is the current temperature?*

Server continuously evaluates the query

→ Server does all of the work

  Cause of low public endpoint availability!
  ½ have availability of $< 95\%$ (Buil-Aranda 2013)

→ Clients just wait for results
What if we moved continuous query evaluation to the client?

→ to lower server load
Triple Pattern Fragments does this for static data!

**Triple pattern fragments (TPF)** (Verborgh 2016):

Servers can only respond to triple pattern queries
Clients need to evaluate queries locally
→ Lowers server load
Triple Pattern Fragments does this for static data!

Triple pattern fragments (TPF) (Verborgh 2016):

Servers can only respond to triple pattern queries
Clients need to evaluate queries locally
→ Lowers server load

Can we do the same for dynamic data?
Overview

RDF Stream representation

TPF-QS engine

Evaluation
Overview

RDF Stream representation

TPF-QS engine

Evaluation
Expose RDF Stream through TPF interface

Assumption: incoming statements must have **time intervals**

→ Time intervals represent how long they remain valid

→ Client can derive the time at which data can change! *(smart polling)*

→ Serialize RDF statements with **time intervals**
Time labeling types

<table>
<thead>
<tr>
<th>Time interval</th>
<th>Start- and endtime</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Good for maintaining a history of elements</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Expiration time</th>
<th>Endtime</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>When only the latest version is required</td>
</tr>
</tbody>
</table>
Time-annotated statement example

:thermometer1 :hasValue "19,06°C".

Graph-annotation: [ 11:30, 11:31 ]
Time-annotated statement example

:thermometer1 :hasValue "19,06°C".

Graph-annotation: [ 11:30, 11:31 ]

GRAPH _:g1 {
  :thermometer1 :hasValue "19,06°C".
}
_:g1 tmp:interval _:interval_1.
_:interval_1 tmp:initial "2016-12-09T11:30:00"^^xsd:dateTime.
_:interval_1 tmp:final "2016-12-09T11:31:00"^^xsd:dateTime.
Overview

RDF Stream representation

TPF-QS engine

Evaluation
Client-side TPF-QS engine
Evaluation times *only* when needed

```
:thermometer1 :hasValue "19,05°C" [11:00, 11:01]
:thermometer1 :hasValue "19,06°C" [11:01, 11:02]
:thermometer1 :hasValue "19,11°C" [11:02, 11:03]
:thermometer1 :hasValue "19,08°C" [11:03, 11:04]
...
```

```
11:00  19,05°
11:01  19,06°
11:02  19,11°
11:03  19,08°
...
```
Evaluation times *only* when needed

```plaintext
:thermometer1 :hasValue "19,05°C" [11:00, 11:01]
:thermometer1 :hasValue "19,06°C" [11:01, 11:02]
:thermometer1 :hasValue "19,11°C" [11:02, 11:03]
:thermometer1 :hasValue "19,08°C" [11:03, 11:04]
...

11:00  19,05°
11:01  19,06°
11:02  19,11°
11:03  19,08°
...
```
Evaluation times *only* when needed

:thermometer1 :hasValue “19,05°C” [11:00, 11:01]
:thermometer1 :hasValue “19,06°C” [11:01, 11:02]
:thermometer1 :hasValue “19,11°C” [11:02, 11:03]
:thermometer1 :hasValue “19,08°C” [11:03, 11:04]

...
Evaluation times *only* when needed

:thermometer1 :hasValue "19,05°C"   [11:00, 11:01]
:thermometer1 :hasValue "19,06°C"   [11:01, 11:02]
:thermometer1 :hasValue "19,11°C"   [11:02, 11:03]
:thermometer1 :hasValue "19,08°C"   [11:03, 11:04]

...
Overview

RDF Stream representation

TPF-QS engine

Evaluation
Evaluating scalability

Measure server CPU usage for increasing # clients

Query: “All trains with their delay in station X within the next hour”
Frequency: 10 seconds
Clients: 1 → 200
Engines: TPF-QS, C-SPARQL (Barbieri 2012) and CQELS (Le-Phuoc 2011)

Time labeling types: expiration time
TPF-QS has better lower server load
TPF-QS moves load from server to client
Conclusions

Further evaluation: Different query types, larger datasets …?

Formalization using RSP-QL

Improve historical querying

Promising approach for publishing RDF streams at a low cost