Semantic Processing of Sensor Event Stream by using External Knowledge Bases

Kia Teymourian and Adrian Paschke
Freie Universität Berlin

Presenter: **Kia Teymourian**
Workshop SSN 2012
Motivation

• In some of the use cases huge amount of Background Knowledge about Sensor Events are available.

• Fusion of external Knowledge Bases with the event stream can improve the expressiveness, agility and flexibility of event processing systems.
Example – Semantic Enrichment of Events

Query:
Select food products, which include substances capable of causing cancer (Carcinogen) and are produced in the Europe.

Stream of Product IDs

\{
  (Product_id, “X1234”), (ProductionDate, 5/1/2012)
, 
  (Product_id, “X1235”), (ProductionDate, 6/1/2012)
\}

Knowledge Base

\{(
  \text{Company}_1, \text{produce}, \text{Product}_X1),
  (\text{Product}_X1, \text{includes}, \text{Tartrazine (E102)}),
  (\text{Product}_X1, \text{includes}, \text{Erythrosine (E127)}),
  (\text{Company}_1, \text{production_facilities_in}, \text{Berlin}),
  (\text{Berlin, is_in}, \text{Germany}),
  (\text{Germany, is_in}, \text{Europe})\}
Knowledge-based Event Processing

Event Stream

Event Processing

User Event Query

Complex Events

A-Box Update Stream

Knowledge Base

T-Box

A-Box

DL Reasoning

Knowledge Base
**Event Detection Pattern**

1. **Event Algebra Operation**
   - Sequence, Disjunction, Conjunction, Simultaneous, Negation, etc.

2. **SPARQL Query**
   - Operations to detect events based on their semantics (related meaning in background knowledge)

3. **Stream Windowing Operation**
   - Operations to slide event stream

\[
\{ \text{[SPARQL Query]} , \\
\text{[Event Algebra Operation]}, \\
\text{[SPARQL Query]}, \\
\text{[Event Algebra Operation]}, \\
\text{[SPARQL Query]}, \ldots \\
\} \text{ [Sliding Window Operation]}
\]
**Example of Complex Event Pattern**

**Knowledge Base**

**Query**

\[
\{ \\
\{ (\text{?e1}, c1, ?s1) .  \\
(\text{?s1}, p^*, ?s) . \} \\
\text{[?e1 SEQ ?e2]} \\
\text{[Within 2 min.]} \\
\{ (\text{?e2}, c2, ?s2) .  \\
(\text{?s2}, p^*, ?s) . \} \} \\
\text{[?e2 SEQ ?e3]} \\
\text{[Within 5 min.]} \\
\{ (\text{?e3}, c3, ?s3) .  \\
(\text{?s3}, p^*, ?s) . \} \\
\} \]
Event Query Rule Categories

- Categorization are based on the following factors:
  1) **Number of SPARQL queries** on KB in each event processing step
  2) Whether the SPARQL query **depends on incoming event data** and is generated based on their attributes
  3) **Number of event attributes** used for generating SPARQL queries
  4) **Number of events** used to generate SPARQL queries (Events in a Window of Stream or Single Event)
Event Query Rule Categories

- Classification of most relevant and interesting event query rules based on embedding form of SPARQL predicates inside event query rule.

  - **Category – A:** Single SPARQL query inside the rule

  - **Category – B:** Several SPARQL queries are embedded in an event query rule and combined with event algebra operations.

- Not a complete categorization of all possible rules.
Pseudocode Example of Category A1

```
stream(CEvents) :-
    SResults = sparql_select(KB_id, SQuery),
    eProcessing(SResults, EStream, EQuery).
```

CEvents = Detected Complex Events
EStream = Raw event stream
EQuery = Event pattern query
SResult = Result set of the SPARQL query
SQuery = SPARQL query part of event sQuery rule
KB_id = ID of the target KB
sparql_select = Rule predicate used for querying the external KB
Pseudocode Example of Category A2

1. \text{stream}(CEvents) : -
2. \text{ETuple} = \text{getSingelEvent}(EStream, \text{Udef}),
3. \text{SQuery} = \text{generateSPARQL}(UQuery, \text{ETuple}),
4. \text{SResults} = \text{sparql_select}(\text{KB_id}, \text{SPQuery}),
5. \text{eProcessing}(\text{SResults}, \text{EStream}, \text{EQuery}).

\text{CEvents} = \text{Detected Complex Events}
\text{EStream} = \text{Raw event stream}
\text{EQuery} = \text{Event pattern query}
\text{SResult} = \text{Result set of the SPARQL query}
\text{SQuery} = \text{SPARQL query part of event sQuery rule}
\text{KB_id} = \text{ID of the target KB}
\text{UDef} = \text{Event type tuples defined by users}
\text{ETuple} = \text{Event instance tuples defined by UDef}
\text{sparq_select} = \text{Rule predicate used for querying the external KB}
Pseudocode Example of Category B1

1. \textbf{stream}(CEvents) : –
2. \textbf{ETuples1} = getEvents(EStream, UDef1),
3. \textbf{ETuples2} = getEvents(EStream, UDef2),
4. \textbf{SQuery1} = generateSPARQL(UQuery, \textbf{ETuples1}),
5. \textbf{SQuery2} = generateSPARQL(UQuery, \textbf{ETuples2}),
6. \textbf{SResults1} = sparql\_select(KB\_id, \textbf{SQuery1}),
7. \textbf{SResults2} = sparql\_select(KB\_id, \textbf{SQuery2}),
8. \textbf{eProcessing}(\textbf{SResults1, SResults2, EStream, EQuery}).

\textbf{CEvents} = Detected Complex Events
\textbf{EStream} = Raw event stream
\textbf{EQuery} = Event pattern query
\textbf{SResult} = Result set of the SPARQL query
\textbf{SQuery} = SPARQL query part of event sQuery rule
\textbf{KB\_id} = ID of the target KB
\textbf{UDef} = Event type tuples defined by users
\textbf{ETuple} = Event instance tuples defined by UDef
\textbf{sparq\_select} = Rule predicate used for querying the external KB
Pseudocode Example of Category B2

```prolog
1 stream(cEvents):-
2   ETuples1 = getEvents(EStream, UDef1),
3   SQuery1 = generateSPARQL(UQuery, ETuples1),
4   SResults1 = sparql_select(KB_id, SQuery1),

% Wait until CEvents1 is happened!
5   CEvents1 = eProcessing(SResults1, EStream, EQuery),

6   ETuples2 = getEvents(EStream, UDef2),
7   SQuery2 = generateSPARQL(UQuery, ETuples2, CEvents1),
8   SResults2 = sparql_select(KB_id, SQuery2),

9   eProcessing(SResults2, CEvents2, EStream, EQuery).
```

Ce\textit{events} = Detected Complex Events  
E\textit{stream} = Raw event stream  
E\textit{query} = Event pattern query  
S\textit{result} = Result set of the SPARQL query  
S\textit{query} = SPARQL query part of event s\textit{query} rule  
KB\_id = ID of the target KB  
U\textit{def} = Event type tuples defined by users  
E\textit{tuple} = Event instance tuples defined by U\textit{def}  
sparql\_select = Rule predicate used for querying the external KB
Example: Implementation in Prova rule language (http://prova.ws)

:- eval(server()).

server() :-
  sparqlrule(QueryID),
  rcvMult(XID, Protocol, Sender, event, {url->URL}) [testrule(QueryID, URL)],
  sendMsg(XID, Protocol, Sender, testrule, {url->URL}).

testrule(QueryID, URL) :-
  sparql_results(QueryID, URL, CompanyEmployees),
  CompanyEmployees > 50000.

sparqlrule(QueryID) :-
  Query = '
    PREFIX DBPPROP: <http://dbpedia.org/property/> 
    PREFIX DBPEDIA: <http://dbpedia.org/resource/> 

    SELECT ?company ?employees WHERE {
      ?company DBPPROP:industry DBPEDIA:Computer_software .
      ?company DBPPROP:industry DBPEDIA:Retail . }
',
  sparql_select(Query, QueryID, [], 'http://dbpedia.org/sparql').
Experiments

Experimental Performance Results on Different Query Rule Categories

Installation on two machines, both Quad Core Intel(R) Xeon(R) CPU E31245 @ 3.30GHz with 16 GB RAM
Dedicated network

DBpedia 3.7
Complete Mirror
288 Million RDF triples
Virtuoso Triple Store

<table>
<thead>
<tr>
<th>Category of sQuery Rules</th>
<th>Throughput (Events/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>A1 (Caching)</td>
<td>280000</td>
</tr>
<tr>
<td>A2</td>
<td>2200</td>
</tr>
<tr>
<td>A3</td>
<td>1300</td>
</tr>
<tr>
<td>B1, B2, B3</td>
<td>500-4000</td>
</tr>
</tbody>
</table>
Conclusion & Outlook

- **Semantic Enrichment of Events**
- **Different categories** of event query rules for data fusion from external KBs.

**Future Work**

- Algorithms for efficient processing of events based on background knowledge
  - Enrichment of events
  - Preprocessing
  - Planning

AG Corporate Semantic Web
http://www.inf.fu-berlin.de/groups/ag-csw/
Thank you!

http://www.corporate-semantic-web.de

AG Corporate Semantic Web
Freie Universität Berlin
http://www.inf.fu-berlin.de/groups/ag-csw/