Fusion of Background Knowledge and Streams of Events

Kia Teymourian, Malte Rohde and Adrian Paschke

Freie Universität Berlin

Presenter: Kia Teymourian

DEBS2012, Berlin 16-20 July 2012
Motivation

- High complex *processes, actions, events, ...*
- Knowledge at the right time
- **Missing background knowledge about events**
Knowledge-Based CEP

• The use of Background Knowledge in Complex Event Processing (CEP) offers
  • precise and expressive description of complex events/situations and reactions,
  • Enhancement of CEP systems to more agile systems and improves flexibility to business changes.
  • Simplicity of event pattern rule definition

• The event processing engines can understand what is happening in terms of events.
Example – Semantic Event Processing

Query:
Buy stocks of companies, who have in Europe production facilities and produce products from Metal and more than 10,000 employees and are at the moment in reconstruction phase and their price/volume increased stable in the past 5 minutes.

Event Stream
{(Name, “OPEL”)(Preis, 45)(Volumen, 2000)}
{(Name, “SAP”)(Preis, 65)(Volumen, 1000)}

Knowledge Base
{(OPEL, is_a, automobil_company),
 (automobil_company, build, Cars),
 (Cars, are_build_from, Metal),
 (OPEL, hat_production_facilities_in, Germany),
 (Germany, is_in, Europe),
 (OPEL, is_a, Major_corporation),
 (Major_corporation, have, over_10,000_employees),
 (OPEL, is_in, reconstruction_phase)}
Example of Complex Event Pattern

Events
E3: Stock_ID3
 □ Price
 □ Volume

E2: Stock_ID2
 □ Price
 □ Volume

E1: Stock_ID1
 □ Price
 □ Volume

Knowledge
Company_X3
finance
Company_Y2

has

M1
uses

produce

Company_X1

has

Company_Y1
Knowledge-based Event Processing

Event Stream

Knowledge Base

A-Box Update Stream

T-Box
A-Box

Event Processing

Event Query

Events

Complex Events
Fusion of Knowledge-base with event stream

- Fusion of huge background knowledge
  - Not RDF stream processing
  - Enrichment of events with background knowledge
  - Enrichment of event query pattern

- **Processing of events** in timely manner and inferencing on background knowledge
  - Expressiveness levels vs. high complexity
  - Trade-off between expressive reasoning and real time processing
Event Detection Pattern

- **Event Algebra Operation**
  - Operations to detect events based on their sequence

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

- **Sliding Window Operation**
  - Operations to slide event stream (time, count, semantic)
Event Algebra Operators

- **Sequence Operator (;)**: $(E_1;E_2)$
- **Disjunction Operator ($\lor$)**: $(E_1 \lor E_2)$, at least one
- **Conjunction Operator ($\land$)**: $(E_1 ; E_2) \lor E_3$, in any order
- **Simultaneous Operator ($=$)**: $(E_1 = E_2)$
- **Negation Operator ($\neg$)**: $(E_1 \land \neg E_2)$
- **Quantification (Any)**: Any(n) $E_1$, when n events of type $E_1$ occurs
- **Aperiodic Operator (Ap)**: $Ap(E_2, E_1, E_3)$, $E_2$ Within $E_1$ & $E_3$
- **Periodic Operator (Per)**: $Per(t, E_1, E_2)$, every t time-steps in between $E_1$ and $E_2$
Event Query Rule

{ [SPARQL Query],
  [Event Algebra Operation],
  [SPARQL Query],
  [Event Algebra Operation],
  [SPARQL Query],
} [Sliding Window Operation]
Example: Event Pattern

\[
\{ \[ (e_1 \ p:\text{stockOf} \ \text{Company}_X1), \\
(\text{Company}_X1 \ p:\text{affiliates} \ \text{Company}_Y1) \\
(\text{Company}_Y1 \ p:\text{supplierOf} <\text{Metal}>) \] \\
; \text{sequence} \quad \%\text{(or other event algebra Operations)} \\
\]

\[
[ (e_2 \ p:\text{stockOf} \ \text{Company}_X2), \\
(\text{Company}_X2 \ p:\text{affiliates} \ \text{Company}_Y2), \\
(\text{Company}_Y2 \ p:\text{hasResourcesDemand} <\text{Metal}>)]
\]

; sequence \quad \%\text{(or other event algebra Operations)}

\[
[ (e_3 \ p:\text{stockOf} \ \text{Company}_X3), \\
(\text{Company}_X3 \ p:\text{finance} \ \text{Company}_Y2)]
\}
\{\text{within 10 minutes}\}
Definitions

• Event Pattern Rule
  • *is a event detection rule without using any SPARQL query predicates*. With this term we mean only the event detection part which includes event algebra operations.

• Event sQuery Rule
  • *is a declarative rule which defines detection pattern for user-defined complex events and can include one or more SPARQL query predicates to query external KBs.*
Sparql_select in Prova

`sparql_select(QueryString, QueryID, [SearchToken(ReplacementToken), ...], SPARQL_Endpoint).`

`sparql_results(Id, Predicate, Object).`

Example:

`exampleSparqlEPAndReplacement(Predicate, Object, URL) :-`

```prolog
QueryString = 'SELECT ?p ?object
WHERE {
    $url ?p ?object . }
',
```

`sparql_select(QueryString, Id, [url(URL)], "http://dbpedia.org/sparql"),
sparql_results(Id, Predicate, Object).
```

`:— solve(testSparqlEPAndReplacement(Predicate, Object,
"<http://dbpedia.org/resource/Berlin>" )).`
Example: Event Query Rule in Prova

% Query Example Two - Companies in industry sector Computer Software who
% have more than 10000 employees AND the last share price is more than 80
% Complex Event Processing with Cached Query Results from the Knowledge Base

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

testrule(QueryID, URL, Lastprice) :-
sparql_results(QueryID, URL, CompanyEmployees),
  CompanyEmployees > 10000,
  Lastprice > 80,
  println(['"Complex event: ", URL, " ", Lastprice']).

sparqlrule(QueryID) :-
  Query = \SELECT ?company ?employees
WHERE {
  ?company DBPROP:industry DBPEDIA:Computer_software .
},
sparql_select(Query, QueryID, [], 'http://dbpedia.org/sparql'),
sparql_results(QueryID, URL, CompanyEmployees),
  println([URL]).
sQuery Rule Categories

- Classification of most relevant and interesting event query rules by using 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 operations.

- Not a complete categorization of all possible rules.
Category A

- **Category – A**: Single SPARQL query inside the rule
  - **A1**: 
    - **Simplest form** of sQuery
    - SPARQL query is *independent* from the event stream, and only about the resources in KB.
    - Complex event detection is defined on the results of the SPARQL query in *combination* with the event stream.
    - To **update results** from the KB, re-execute the SPARQL query on each event.
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
Category A2

• Category – A: Single SPARQL query inside the rule
  • A2 :
    • A SPARQL query is *generated* and sent to the KB
    • Attribute/Values of an event instance are used to generate basic triple patterns of a SPARQL query.
    • Based on user definitions some of event instance tuples are selected and used to *generate* a single SPARQL Query.
Pseudocode Example of Category A2

```plaintext
stream(CEvents) : -
ETuple = getSingelEvent(EStream, Udef),
SQuery = generateSPARQL(UQuery, ETuple),
SResults = sparql_select(KB_id , SPQuery),
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
UDef = Event type tuples defined by users
ETuple = Event instance tuples defined by UDef
sparql_select = Rule predicate used for querying the external KB
Category A3

- Category – A: Single SPARQL query inside the rule
  - A3:
    - Similar to the category A2, but the SPARQL query is **generated from multiple events**.
    - Within a **sliding window** from two or more event instances a single SPARQL query is generated.
Category B

- **Category – B**: Several SPARQL queries
  - Include several SPARQLs

- **Several “Category A”** rules are combined together.

- It is possible to **map between background knowledge and the relations of events** to each other.
Pseudocode Example of Category B1

\[\text{stream(CEvents) : – }
\]
\[
\text{ETuples1} = \text{getEvents(EStream, UDef1)}, \\
\text{ETuples2} = \text{getEvents(EStream, UDef2)},
\]

\[
\text{SQuery1} = \text{generateSPARQL(UQuery, ETuples1)}, \\
\text{SQuery2} = \text{generateSPARQL(UQuery, ETuples2)},
\]

\[
\text{SResults1} = \text{sparql\_select(KB\_id, SQuery1)}, \\
\% \text{ execute between event operations} \\
\text{SResults2} = \text{sparql\_select(KB\_id, SQuery2)},
\]

\[
\text{eProcessing(SResults1, SResults2, EStream, EQuery}).
\]

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 
UDef = Event type tuples defined by users 
ETuple = Event instance tuples defined by UDef 
sparq\_select = Rule predicate used for querying the external KB
**Category B2**

- SPARQL(s) are generated and executed in sequence.

- Generated based on the results of the previous SPARQL query.

- Each SPARQL query can be generated from event instances included in a sliding window.

- Waiting for events and then execute SPARQL queries

- SPARQL results are used for event processing or for generation of a following SPARQL query
Pseudocode Example of Category B2

```
stream(cEvents):-
    ETuples1 = getEvents(EStream, UDef1),
    SQuery1 = generateSPARQL(UQuery, ETuples1),
    SResults1 = sparql_select(KB_id, SQuery1),

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

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

    eProcessing(SResults2, CEvents2, 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
UDef = Event type tuples defined by users
ETuple = Event instance tuples defined by UDef
sparql_select = Rule predicate used for querying the external KB
Event Processing Approaches
Event Processing Approaches

1) Polling the Knowledge Base
2) **Knowledge Query First (KQF)**
3) **Plan-Based Processing**
4) Semantic Enrichment of Event Stream (SEES)
5) **Event Query Pre-Processing (EQPP)**

Complementary approaches
Plan-Based Processing (PBP)

- Processing of an event query based on an optimal plan for its sub-queries to avoid any unnecessary costs or losses of time.

- The SPARQL queries are combined in AND, OR, SEQ or similar event operations and the whole query should be evaluated in a sliding window.

- SPARQL queries can be executed in a sequence one after another or in a parallel setting.

- A Optimal plan can be found for the execution of the single SPARQL queries.
Semantic Enrichment of Event Stream

Event Stream: e1 e2 e1

Raw Events

Semantic Enrichment

Derived Events: e3 e4 e3

e1 e2 e1

Knowledge Base

Final Processing on EPN

Complex Events
Event-Query Pre-Processing

- The **complex query** is pre-processed and rewritten in several **simple queries**.
- **Simple query** is a query which can be processed **without** the external KB.
- New **simple queries** can be **generated using the knowledge base**.

\[
Q \quad KB \rightarrow q_1, q_2, q_3, q_4, \ldots
\]

- Simple queries can be in conjunction and disjunction.
- Queries are processed by several Event Processing Agents.
- Results are jointed together by EPAs.
Event Query Pre-Processing

Event Stream
\[ e_1 \ e_2 \ e_1 \]

Raw Events

Complex Query
\( Q \)

Event Query Pre-Processing

Rewrite
Simple Queries
\[ q_1 \ q_2 \ q_3 \]

Knowledge Base

Final Processing distributed on a network of processing Agents

Event Processing Network
Experiments
Experiment Setup

Knowledge Base

Stock Market Data Stream

Event Stream

Prova Rule Engine

Different sQuery Rule Categories

Complex Events

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
Experiments

- Highest throughput for simple event processing without KB is between **450000 - 500000 events/s**

- Each SPARQL query has between **1ms - 100ms response time**

- Performance of polling approach (generating of SPARQL query and execution on each event) is about **2200 events/s** depending on SPARQL Query
### Experimental Performance Results on Different sQuery Rule Categories

<table>
<thead>
<tr>
<th>Category of sQuery Rules</th>
<th>Throughput (Events/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>A1 (KQF)</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>
Experiments

Throughput in Relation to Number of SPARQL Query Results in Cache
Improvement by PBP & EQPP

• **Improvement of PBP**
  - A category B2 event query including two SPARQL queries, Q1 \(\text{and Q2}\), connected with an AND operator
  - Q1 has an execution time of 5ms while Q2 takes 700ms
  - **Plan1:** Execute the query Q2 first
    - Total performance of the query execution → 55 events/sec
  - **Plan2:** Execute the query Q1 first
    - Total performance of the query execution → 600 events/sec

• **Improvement of EQPP**
  - A B1 query with execution throughput of about 1000 events/s
  - Manually rewritten the B1 query in two A1 Queries with the total execution performance of 230000 events/s
Conclusion & Outlook

• Approach for fusion of background knowledge with event stream

• **Different categories** of event query rules which use special predicates for importing data from KBs.

• **Different processing approaches** are proposed

Outlook:

• Details of event query preprocessing algorithm

• Details of plan-based approach and specification of heuristics which can be used for selection

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/