Towards Efficient Semantically Enriched Complex Event Processing and Pattern Matching (Position Paper)

Syed Gillani[1 2], Gauthier Picard[1], Frédérique Laforest[2], and Antoine Zimmermann[1]

[1] Institute Henri Fayol, EMSE, Saint Etienne, France
firstname.lastname@emse.fr,
[2] Telecom Saint Etienne, Université Jean Monnet, Saint Etienne, France
frederique.laforest@telecom-st-etienne.fr

Abstract. Management and recognition of event patterns is becoming thoroughly ingrained in many application areas of Semantically enabled Complex Event Processing (SCEP). However, the reliance of state-of-the-art technologies on relational and RDF triple model without having the notion of time has severe limitations. This restricts the system to employ temporal reasoning at RDF level and use historical events to predict new situations. Additionally, the semantics of traditional query languages makes it quite challenging to implement distributed event processing. In our vision, SCEP needs to consider RDF as a first class citizen and should implement parallel and distributed processing to deal with large amount of data streams. In this paper, we discuss various challenges and limitations of state-of-the-art technologies and propose a possible solution to extend RDF data model for stream processing and pattern matching. Furthermore, we describe a high-level query design that enables efficient parallel and distributed pattern matching through query rewriting.

1 Introduction

Complex event processing (CEP) has proven to be the key technique for handling and analysing complex relations over high volume of dynamic data [1]. An additional property of data produced from diverse sources is the heterogeneity and potentially the large number of distributed event sources. CEP systems are characterised by real-time response and complex events are typically described by a set of temporally and spatially distributed patterns [5]. The Relational database community has proposed various solutions to capture the streaming nature of data [1, 2, 16]. But due to the limitations in their underlying data model, the handling of heterogeneous data with variable semantics and dynamic information inference is still an open issue. Thus, graph based data models, such as RDF have emerged to be an accepted solution for such problem and proved to be a vital component of the Semantic Web. RDF data model shows great promise but lacks the notion of time. This constraints most of the existing Semantically enabled Complex Event Processing (SCEP) systems (CQELS [11], C-SPARQL [6], EP-SPARQL [4]) to base their roots on the work of relational CEP. Such models cut down on the syntactic diversity and lack in customisable selection policies [12]. Additionally, existing SCEP solutions commonly assume centralised availability and processing of all relevant events, and thus incur significant overheads in distributed settings. Our vision is to extend the design of stream reasoning and pattern matching, while introducing RDF as a first class citizen in modelling and reasoning on incoming streams.

In this paper, we propose a new event data model for SCEP that allows temporal reasoning at RDF level. We also propose the design of new query language that enables parallel and distributed pattern matching, and stream reasoning by employing query rewriting. The event data model underlying the system allows filters and predicates to be expressed with respects to the events in specified patterns. The integration of event model and query semantics allows the use of event detection graphs (EDG) and non-deterministic automata (NFA) in the context of Semantic Web. In particular, Section 2 reports some challenges and limitations of traditional SCEP. Section 3 describes the detailed event model and its advantages, Section 4 presents the semantics of our high-level pattern matching query. Section 5 provides the details of patterns detection and finally, Section 6 outlines some conclusive remarks.

2 Foundations

This section highlights some of the foundational challenges faced by the traditional SCEP.

Distributed Event Processing: The detection of complex event patterns is computing intensive and the system needs to scale in a concurrent distributed manner to detect patterns by utilising a set of distributed machines or existing resources [7]. Traditionally, streams processing for pattern matching is based on a centralised, push-based event acquisition and processing model [2]. The limitations of such centralised model are two folded: first, it inevitably limits SCEP scalability and performance to deal with high rate event sources; second, it requires communicating all base events to the main processing engine, while only a small fraction of all base events eventually make up complex events. This motivates the design of a distributed SCEP system in order to utilise the resources in an efficient way with optimised performance.

Language for Pattern Queries: The ability to recognise complex patterns is critical for various system settings. On one hand, most of the relational languages for event processing systems are inspired from regular languages and therefore have automata based semantics [1]. While on the other hand SCEP mostly relies on rule-based processing due to the inability of reasoning capability for automata-based semantics. For instance, ETAILS (EP-SPARQL) converts queries to prolog rules and evaluates them as a single thread [12]. This limits the use of various operators including kleene closure, negation, union/intersection of patterns. Thus, this requires a model that can borrow the useful operators from relational CEP and embed them in a semantically enriched CEP.

Predictive Event Processing: Most of the early SCEP research is primary focussed on pattern matching techniques over real-time event streams. However, querying historical events has fundamental importance in many SCEP applications and has so far not received much attention [10]. The presence of stream archive enables a new class of SCEP applications, which can correlate patterns matched over live and archived event streams. This builds the foundation of forecasting, prediction of future events occurrence and identification of casual relationships among complex events across multiple time scales. The analysis of historical patterns is a particularly challenging task. It involves in processing of potentially large amount of historical data and requires efficient query semantics, storage, retrieval and load shedding of time-annotated information. Currently, no solution fully integrates stream processing and historical data management [12].

RDF Graph Based Streaming: Generally most of the existing SCEP assumed that incoming data is delivered as individual RDF triples and implicitly ordered by arrival time, or explicitly ordered by timestamps [12]. This triple based model suffers from various limitations, as event objects generally consist of RDF graph (e.g., data enacting from sensors), rather than individual RDF triples. When event objects are decomposed into a list of RDF triples for streaming purpose, firstly, it becomes essential that the boundaries of the event objects can be respected within queries, since a query that observes only a partial RDF graph may return false results. Secondly, RDF is structured as graph. Thus, in order to distribute it into triples for the sake of allowing the consumption in triples, the system has to shred the general joined structure of RDF. This results in extra computation to rejoin the triples for graph manipulation. The use of RDF graph for streaming purpose instead of individual triples would greatly simplify the system design. It would enable the event producers to group the triples from the same source with the same timestamps into one graph, while implementing fair boundaries for the streaming queries to operate in.

3 Event and Stream Data Model

Our vision for SCEP is to use RDF as a first class citizen for continuous query processing and pattern matching. The standard RDF model consists of sets of atomic statements called triples (subject, predicate, object). In order to integrate temporal relatedness and to deal with the problem of diachronic identity [9] (i.e., how do we logically account for the fact that the "same" entity appears to be "different" at different times), it should be extended to associate time as metadata. This timestamp could be a single temporal identifier to represent the arrival time, but in order to assimilate facts, system requires an interval or time range on which data item is valid. The temporal relations between events can be captured through a set of operators, such as operators from Allens's interval algebra [3]. Hence, we put forward a new event and stream model, where each incoming event is a temporally annotated RDF Named Graph and bounds a set of RDF triples. Our data model is analogous to temporally annotated RDF [9] with the integration of Named Graphs. The serialisation of such a model can be achieved by relying on existing proposals such as reification or TriG[3] format.

Definition: Temporally Annotated RDF Named Graph A Named Graph NG is a pair (u, G), where u is an IRI and G is a RDF graph. A temporally annotated Named graph is a tuple $\langle NG, A_t \rangle$, Where NG is a Named Graph and A_t is a time interval to describe the validity of Named Graph.

Definition : Event An event E $\langle NG, A_t \rangle$ consists of a temporally annotated RDF Named graph and temporal interval $A_t = [t_s, t_e]$, where $t_s, t_e \in A_t$ is a time interval such that $x \in A_t$ iff $x \geq t_s$ and $x \leq t_e$.

Definition : Event Stream An Event stream S is a totally ordered sequence of events $S = (E_1, E_2..., E_n)$. Each event instance E_i belongs to a particular event type $c_i \in \sum$, and

[3] TriG (http://www.w3.org/TR/2014/REC-trig-20140225/)

has a unique annotated time interval A_t, where $\sum = \{c_i\}$ is the domain of all possible event types.

In this work we make the simplifying assumption that events in the sequence are in strictly increasing time order, and there are no out-of-order events in the sequence.

Definition : Complex Event A complex event is an event that result from the application of a function that takes as input a sequence S of events, a set of query patterns Q_i of the form $Q_i = (p_1, p_2.., p_n)$ with $p_i \in \sum$ being an event type (resulted from a certain sub-query, see section 4), and that outputs a set $\mathcal{M} = \bigcup_{q_i \in Q} M(q_i, S)$, where $M(q_i, S)$ is the set of all query matches of q_i over S.

Annotated RDF graph allows to capture the diachronic nature of events. It describes temporally valid facts, where validity is indicated by specified time intervals. For instance, if *James* was a CEO of *IBM* from June 2011 to July 2013 and later he was CEO of *Oracle* from July 2013 to January 2014, then this generates two facts with their interval and validities. There are numerous cases where streaming of valid facts is essential for complex pattern matching. The status of these facts should be maintained by the stream processing platform and later saved for archived query processing.

In the context of event processing, the use of RDF Named Graphs has the following advantages including data partitioning, stream data summarising, event body definition, access control and provenance tracking.

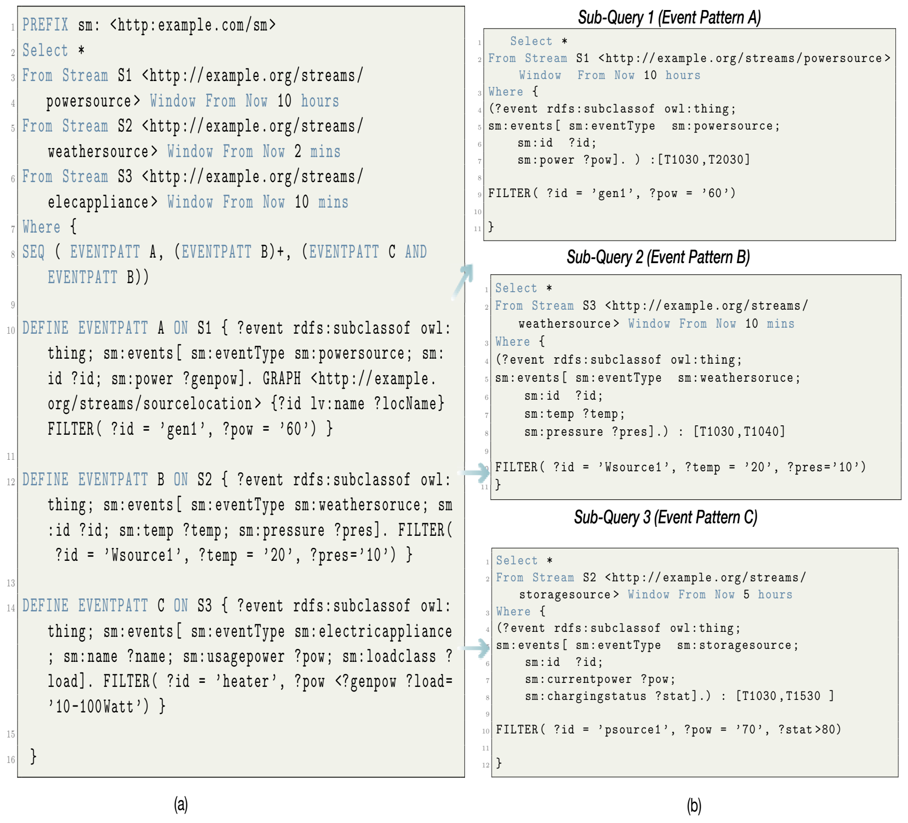
Data Partitioning: Efficiently storage and management of historical stream archives on a single machine could be unrealistic, due to the large volume of data streams and their heterogeneity. Traditionally, triple stores use hashing approaches to allocate triples to different data stores. However, complex pattern queries usually involve multiple joins and would result in heavy communication costs [15]. Hence, the use of logical partitioning with RDF Named Graphs can realise a vertical partitioning of data using Named Graph indexing. This makes it easier to identify and query subsets within an aggregation of events.

Summarising Stream Data: High volume and frequent sampling aspect of streaming data propose a reasonable assumption, that two identical values that are reported within some time threshold indicate unchanged state between these values. In our case, summarisation equates the merging of two Named Graphs, where each represents an observation. Summarisation of events can be achieved by merging two events into a single one, whose validity spans the duration of both events. This will further decrease the size of data stored and increase the query performance [14].

Defining Event Boundary, Access control and Provenance Tracking: As discussed earlier, Named Graphs are essential to define the boundary to a set of triples emanating from the same source with the same temporal information (e.g., sensor data with values of current temperature, device ID, location). Further, Named Graphs aid in tracking provenance and managing the access control of stream processing in a restricted environment.

4 Query Model and Language Specification

Complex pattern queries cover a considerable amount of literature in the relational community. But in the context of the Semantic Web EP-SPARQL is considered to be the



```
PREFIX sm: <http:example.com/sm>
Select *
From Stream S1 <http://example.org/streams/
  powersource> Window From Now 10 hours
From Stream S2 <http://example.org/streams/
  weathersource> Window From Now 2 mins
From Stream S3 <http://example.org/streams/
  elecappliance> Window From Now 10 mins
Where {
SEQ ( EVENTPATT A, (EVENTPATT B)+, (EVENTPATT C AND
  EVENTPATT B))

DEFINE EVENTPATT A ON S1 { ?event rdfs:subclassof owl:
  thing; sm:events[ sm:eventType sm:powersource; sm:
  id ?id; sm:power ?genpow]. GRAPH <http://example.
  org/streams/sourcelocation> {?id lv:name ?locName}
  FILTER( ?id = 'gen1', ?pow = '60') }

DEFINE EVENTPATT B ON S2 { ?event rdfs:subclassof owl:
  thing; sm:events[ sm:eventType sm:weathersoruce; sm
  :id ?id; sm:temp ?temp; sm:pressure ?pres]. FILTER(
   ?id = 'Wsource1', ?temp = '20', ?pres='10') }

DEFINE EVENTPATT C ON S3 { ?event rdfs:subclassof owl:
  thing; sm:events[ sm:eventType sm:electricappliance
  ; sm:name ?name; sm:usagepower ?pow; sm:loadclass ?
  load]. FILTER( ?id = 'heater', ?pow <?genpow ?load=
  '10-100Watt') }

 }
```

(a)

Sub-Query 1 (Event Pattern A)

```
   Select *
From Stream S1 <http://example.org/streams/powersource>
    Window  From Now 10 hours
Where {
(?event rdfs:subclassof owl:thing;
sm:events[ sm:eventType  sm:powersource;
    sm:id  ?id;
    sm:power ?pow]. ) :[T1030,T2030]

FILTER( ?id = 'gen1', ?pow = '60')

}
```

Sub-Query 2 (Event Pattern B)

```
Select *
From Stream S3 <http://example.org/streams/
   weathersource> Window From Now 10 mins
Where {
(?event rdfs:subclassof owl:thing;
sm:events[ sm:eventType  sm:weathersoruce;
    sm:id  ?id;
    sm:temp ?temp;
    sm:pressure ?pres].) : [T1030,T1040]

FILTER( ?id = 'Wsource1', ?temp = '20', ?pres='10')
}
```

Sub-Query 3 (Event Pattern C)

```
Select *
From Stream S2 <http://example.org/streams/
   storagesource> Window From Now 5 hours
Where {
(?event rdfs:subclassof owl:thing;
sm:events[ sm:eventType  sm:storagesource;
    sm:id  ?id;
    sm:currentpower ?pow;
    sm:chargingstatus ?stat].) : [T1030,T1530 ]

FILTER( ?id = 'psource1', ?pow = '70', ?stat>80)

}
```

(b)

Fig. 1: Conceptual High-level Pattern Matching Query and Resulted Sub-queries after Query-rewriting

only core language for such case. However, it suffers from various drawbacks including reliance on triple-based data model, no support for multiple and heterogeneous event sources, event source based filtering, use of black-box approach to delegate the processing to other engines and overhead in query and data translation for underlying model [12]. Our vision is to introduce a new core language for pattern matching. It covers various limitations of previous approaches and includes constructs that are useful in real-world applications. Our core language employs the event model described above and is applicable to distributed event streams. It seeks for a series of events that occur in required temporal order, while satisfying other constraints. Some of the constructs are borrowed from SCEP [12], while many are new to this context. The newly introduced constructs and overall structure of a pattern query are as follows.

```
<Query>::= SELECT | CONSTRUCT
FROM STREAM <Stream ID> <Source Uri> WINDOW <Time Window>
WHERE
SEQ <Pattern Matching Condition>
---------------------------------------------------------------
<Pattern Matching Condition> ::= <Pattern> (,<Pattern>)*|(,!<
    Pattern>)*|(,+<Pattern>)*|(,AND<Pattern>)*|(,OR<Pattern>)*
<Pattern> ::= DEFINE EVENTPATT ON <Stream ID> (,<Stream ID>)*
    <Group Graph Pattern>
```

```
<Group Graph Pattern> ::= <Triple Block>  FILTER (<Filter
    Specification>)*  RETURN (<Output Specification>)*
```

Sequencing (`SEQ`)**:** It lists the required event type in temporal order and assigns a variable to refer to each event selected in the network. e.g., `SEQ{EventPatt A,EventPatt B, EventPatt C}.`

Kleene Closure (`+`), ***Negation***(`!`), ***Union/Intersection*** (`AND/OR`)**:** Kleene closure collects a finite yet unbounded number of events, negation verifies the absence of certain event, while union/intersection can be applied to two patterns. These are used as component in sequencing block. e.g., `SEQ {Eventpatt A+, Eventpatt B AND Eventpatt C, !Eventpatt D }.`

Event Graph Pattern (`EVENTPATT`)**:** This describes the RDF graph pattern to be matched on incoming streams. It includes a filter block to constraint graph matching. It is used as a component of sequencing construct. e.g., `EventPatt A ON Stream 1,Stream 2 {⟨⟨ S,P,O⟩...⟩}`

Return (`RETURN`)**:** This generates the aggregated values of matched results. For instance, to compute the average of certain values. It is a part of event graph pattern construct. e.g., `Return avg(?val1)`

The detection of complex event patterns in high rate stream system requires a substantial amount of resources. A solution for scalable SCEP would be to distribute resources across the set of machines or to use existing resources more efficiently through query rewriting [5]. However, the majority of current SCEP systems only support execution on a single machine and semantics of their query language makes automated query rewriting quite challenging [12, 7]. Our vision for distributed and concurrent SCEP is to distribute event detection with the combination of query rewriting and resource distribution across multiple machines. The user can utilise a high-level language to write pattern queries with the set of operators described above. These patterns are translated into stateless/stateful sub-queries or sub-patterns. This enables the system to consume temporally annotated event graphs and to process each sub-query in a distributed and parallel manner by employing either NFA states [7] or rules for EDG [8]. Figure 1-b depicts a set of sub-queries that are translated from high-level query described in Figure 1-a.

5 Complex Pattern Matching

The main computation involved in stream processing and pattern matching is to incrementally compute continuous queries against incoming streams, meet QoS requirements for each query and apply or map rules to detect patterns. Several approaches have been discussed in literature for the detection of patterns including, EDG [8], NFA [7], RETE algorithm [13] and rule engines [12]. However, SCEP systems have mostly focussed on logical rules engines and RETE algorithm. One of the main advantage of our pattern query algebra is to allow query rewriting, which enables the use of any of the above mentioned techniques. But most importantly, it allows efficient pattern matching techniques like EDG and NFA to be used in the context of SCEP. Figure 2 describes an integrated SCEP architecture that consists of four stages. The first stage involves in the selection of streams from the set of incoming streams as according to the high-level

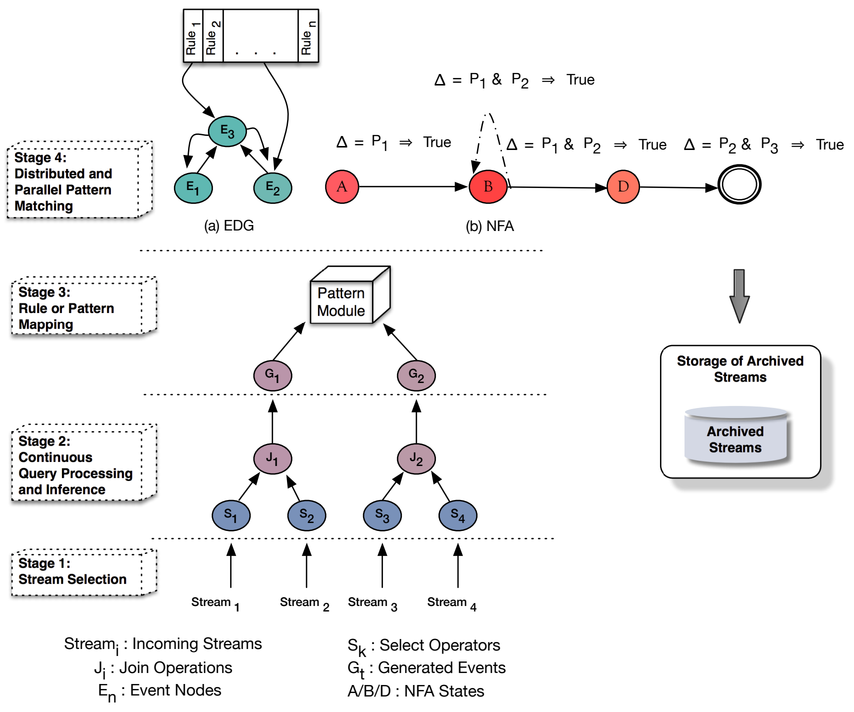


Fig. 2: Conceptual Architecture of Event Stream Processing and Pattern Matching

query. The second stage consist of two operations: first, applying backward chaining to infer more data that can satisfy various sub-queries and second, processing continuous sub-queries along with certain operators (selection, join, construct, filter) to generate meaningful events. The third stage glues together the stream processing and pattern matching part. In this stage, pattern rules/operators described in high-level language are either mapped as a predicate for NFA states or as rules for EDG. The fourth stage determines the final patterns computed after the application of rules/operators. After these processing steps, the system decides whether to store the event in archive storage or simply to drop it. This decision is based on system's event consumption policies that are pre-defined for each type of events. The seamless nature of our proposed integrated model enables the parallel and distributed processing of patterns and the integration of stream processing with various models or techniques (NFA, EDG) for pattern matching.

6 Discussion and Conclusion

Adding annotation to logical Named Graphs enables temporal reasoning at RDF level and makes it especially tailored for the management of streaming data. Our initial approach is restricted to temporal annotation, but our data model can be extended to a more generic case to handle multiple dimensions of annotation. This generic extension would enable the system to utilise various other forms of meta-information, such as provenance and uncertainty measure of incoming streams. EDG and NFA are the proven techniques for pattern matching problem in the relational community. But no one has yet employed it in the case of the Semantic Web due to its limitations in reasoning process. However, by utilising query rewriting, we can get the best of both worlds. This not only can devise a pattern matching environment but also enables the distributed and parallel processing of heterogeneous streams. The three parallelism techniques that could be employed to the system's design include, parallelism by input, parallelism by state and parallelism by run. The right combination of these techniques can further optimise query computation, especially in case of stateless sub-queries.

In this paper we explored the idea of distributed and scalable design for SCEP. We propose a new data model for events that enables the temporal reasoning of streams at RDF level including past and recent events. It also uses Named Graph approach for the distribution and summarisation of event data. We also provide a design for high-level query algebra. This enables the system to rewrite the high-level queries into sub-queries and process them in distributed and parallel manner. Our future endeavours involve the detail analysis of query cost models and realisation of such an open source SCEP engine.

References

1. Agrawal, J., Diao, Y., Gyllstrom, D., Immerman, N.: Efficient pattern matching over event streams. In: Proceedings of the 2008 ACM SIGMOD, USA
2. Akdere, M., Çetintemel, U., Tatbul, N.: Plan-based complex event detection across distributed sources. Proc. VLDB Endow. **1**(1) (August 2008) 66–77
3. Allen, J.F.: Maintaining Knowledge about Temporal Intervals. Communications of the ACM **26**(11) (1983) 832–843
4. Anicic, D., Fodor, P., Rudolph, S., Stojanovic, N.: Ep-sparql: A unified language for event processing and stream reasoning. In: Proceedings of the 20th WWW, New York, USA (2011)
5. Arasu, A., Babcock, B., Babu, S., McAlister, J., Widom, J.: Characterizing memory requirements for queries over continuous data streams. In: Proceedings of the Twenty-first ACM SIGMOD-SIGACT-SIGART, New York,USA, ACM (2002)
6. Barbieri, D.F., Braga, D., Ceri, S., Valle, E.D., Grossniklaus, M.: Querying rdf streams with c-sparql. SIGMOD Rec. **39**(1) (September 2010) 20–26
7. Brenna, L., Gehrke, J., Hong, M., Johansen, D.: Distributed event stream processing with non-deterministic finite automata. DEBS '09, New York, NY, USA, ACM (2009) 3:1–3:12
8. Ghanem, T.M., Aref, W.G., Elmagarmid, A.K.: Exploiting predicate-window semantics over data streams. SIGMOD Rec. **35**(1) (March 2006) 3–8
9. Gutierrez, C., Hurtado, C., Vaisman, A.: Temporal rdf. In: Proceedings of the Second European Conference on The Semantic Web: Research and Applications. ESWC'05, Berlin
10. Keskisrkk, R., Blomqvist, E.: Semantic complex event processing for social media monitoring- a survey. Proceedings of SMILE Co-located with 10th ESWC (2013)
11. Le-Phuoc, D., Xavier Parreira, J., Hauswirth, M.: Linked stream data processing. In: Reasoning Web. Semantic Technologies for Advanced Query Answering. Berlin (2012)
12. Margara, A., Urbani, J., van Harmelen, F., Bal, H.: Streaming the web: Reasoning over dynamic data. Web Semantics: Science, Services and Agents on WWW (2014)
13. Rinne, M., Nuutila, E., Trm, S.: Instans: High-performance event processing with standard rdf and sparql. In: International Semantic Web Conference. (2012)
14. Stevenson, G., Ye, J., Dobson, S.: Managing the temporal and structural semantics of sensor data in pervasive environments. (2010) Technical Report.
15. Wang, R., Chiu, K.: A graph partitioning approach to distributed rdf stores. In: Parallel and Distributed Processing with Applications (ISPA), 2012 IEEE 10th International Symposium
16. Zhang, H., Diao, Y., Immerman, N.: Recognizing patterns in streams with imprecise timestamps. Proc. VLDB Endow. **3**(1-2) (September 2010) 244–255