Semantic Processing of Sensor Event Stream by Using External Knowledge Bases

Short Paper

Kia Teymourian and Adrian Paschke

Freie Universitaet Berlin, Berlin, Germany
{kia, paschke}@inf.fu-berlin.de

Abstract. Usage of domain background knowledge about sensor data can improve the expressiveness and flexibility of event processing in sensor network applications. Huge amount of domain background knowledge stored in external knowledge bases can be processed in combination with sensor data stream in order to achieve more knowledgeable event processing. In this paper, we discuss the benefits of background knowledge for event processing and describe a simple classification of event query rules.[1]

Keywords: Knowledge-Based Complex Event Processing

1 Motivation

Detection, prediction and mastery of complex situations are crucial to the competitiveness of networked businesses, the efficiency of Internet of Services and dynamic distributed infrastructures in manifold domains such as logistics, automotive, telecommunication, e-health and life sciences. Event Processing is an emerging technology to achieve actionable, situational knowledge from large-scale event streams in real-time.

Semantic models of events can improve event processing quality by using event meta-data in combination with ontologies and rules (knowledge bases). The successes of the knowledge representation research community in building standards and tools for technologies such as formalized and declarative rules, are opening novel research and application areas. The combination of event processing and knowledge representation can lead to novel semantic-rich event processing engines. The identification of critical events and situations requires processing vast amounts of data and metadata within and outside the systems.

Using external background knowledge about sensor data stream is one of the promising approaches for detection of real-world complex events. Knowledge about event types and their hierarchies, i.e., specialization, generalization, or

[1] This work has been partially supported by the "InnoProfile-Corporate Semantic Web" project funded by the German Federal Ministry of Education and Research (BMBF) and the BMBF Innovation Initiative for the New German Länder - Entrepreneurial Regions.

other forms of relations between events can be useful. Huge amount of background knowledge about sensor event data can be integrated with the main data stream to improve the expressiveness of event processing by inferencing on background knowledge. Semantic inference is used to infer relations between events such as, e.g., transitivity or equality between event types and their properties. Temporal and spatial reasoning on events can be done based on their data properties.

In the rest of this paper, we describe in Section 2 the benefits of using external knowledge bases for event processing and in Section 3 different event query categories are discussed.

2 Knowledge-Based Complex Event Processing

Previously, we proposed in [7,6] a new approach for semantic enabled complex event processing (SCEP). We proposed that the usage of the background knowledge about events and other related concepts can improve the quality of event processing. We described how to formalize complex event patterns based on a logical knowledge representation (KR) interval-based event/action algebra, namely the interval-based Event Calculus [2,3,4].

The fusion of background knowledge with data from an event stream can help the event processing engine to know more about incoming events and their relationships to other related concepts. We propose to use one or several *Knowledge Bases (KB)* which can provide background knowledge (conceptual and assertional, T-Box and A-Box of an ontology) about the events and other non-event resources. This means that events can be detected based on reasoning on their type hierarchy, temporal/spatial relationships, or their relationship to other objects in the application domain. The connections to other relevant concepts and objects means for example the relationship of a food item to a particular drug.

The benefits of using background knowledge in complex event processing can be seen as two major advantages over the state of the art CEP systems. The first benefit is its higher *expressiveness* and the second one its *flexibility*. Expressiveness means that an event processing system can precisely express complex event patterns and reactions to events which can be directly translated into business operations. Flexibility means that a CEP system is able to integrate new business changes into the systems in a fraction of time rather than changing the whole event processing rules. Complex event patterns are independent of current businesses and are defined in a higher level of abstraction based on business strategies. When something is changed in the business environment, it can be considered simply as an update in the background knowledge and the complex event detection patterns which are defined based on the business plans should not be changed.

In many event processing use cases, the amount of background knowledge about events and the relevant objects can be very high, so that they can not be included as rule sets in the main memory of event processing agents for reason-

ing. Therefore, we propose to use external KBs for the storage and reasoning on background knowledge. The background knowledge about events and other non-event concepts/objects is described in description logic. The knowledge in the KB can be stored in the Resource Description Framework (RDF) data format[2] in an external triple store (special kind of databases for storage and management of RDF data). This knowledge can be queried from the event processing agents based on the demands of the event query rules. The external KB includes a description logic inference engine to reason on the relations between events and other relevant non-event objects in the application domain. The KB can be queried by using SPARQL [5] queries and the results are then included in the event processing engine.

3 Event Query Rules and Their Categories

Event query rules are declarative rules which are used to detect complex events from streams of raw events. The aggregated knowledge from event streams and background KB can be queried by different types of event queries. These event queries have a hybrid semantic, because they use event operation algebra to detect events and they use SPARQL queries to include background knowledge about these events and their relationships.

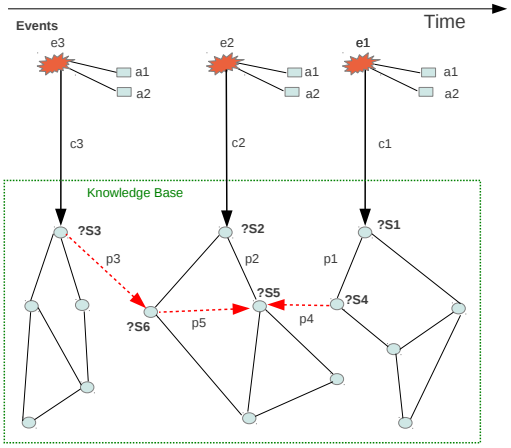
Lets consider an Event type E_1 which can be instantiated with n (attribute, value) tuples like: $e_1^1((a_1, v_1), \ldots, (a_n, v_n))$. The figure 1 shows the event stream and the relationships of events to resources in the background knowledge. An event instance e_1 can be connected to one or more resources in the background knowledge by using a connecting predicate c_1 using one or more attribute value pairs of the event instance.

We use attribute-value tuples of an incoming event instance to query an external KB about the relevant background knowledge of that event instance. These tuples are used to build basic graph patterns (BGPs) which are used in SPARQL queries as sets of triple patterns defined in SPARQL queries. The usage of SPARQL queries allows the event processing agent to include external knowledge and combine it with the event stream to know more about the incoming events.

Our event query rules allow simple event algebra operations, similar to Snoop [1] (i.e. event operations like AND, SEQ, OR, NOT), to query the event stream as well as higher interval-based event operations like (BEFORE, MEETS, OVERLAP, ...). Our event query rules can include SPARQL query predicate to query external KBs. The results of SPARQL queries are used in combination with event stream to detect complex events. This means that a complex event pattern is defined based on the event operation algebra in combination with SPARQL queries (basic graph patterns plus inferencing on knowledge graph).

One event detection pattern of the relationship shown in the figure 1 can be represented by the given pseudocode. The event e1 is connected to the resource

[2] http://www.w3.org/RDF/




```
{ { (?e1, c1, ?s1) .
    (?s1, p*, ?s) . }
% SEQ event algebra operation
[?e1 SEQ ?e2][Within 5 min.]

{ (?e2, c2, ?s2) .
  (?s2, p*, ?s) . } }

[?e2 SEQ e3] [Within 5 min.]

{ (?e3, c3, ?s3) .
  (?s3, p*, ?s) . } }
```

Fig. 1. Relation of Events to Resources in the Background Knowledge and Pseudocode of Event Detection Pattern

s1 in the background knowledge by the predicate c1. In the same way the event e2 is connected to the resource s2 by predicate c2. The predicate p4 connect the two resources s4 and s5, so that it connects the two sub-graphs.

The above query can written in declarative rules for event detection. The precise semantics of event query rules is based on interval-point semantics [6] which allows us to include event algebra operators based on time-point semantics. This kind of event query rules can also include SPARQL query predicate to query external KBs. The semantics of the whole event query is a hybrid semantic of description logic and event operation algebra which defines the semantics of event detection.

Event query rules can be categorized into several categories based on the usage of knowledge queries (SPARQL queries) inside the event query rule. Our criteria for these categorization are based on the following different factors: 1. Number of SPARQL queries in each event processing step are sent to the KB *(Single SPARQL or Several SPARQLs)* 2. Whether the SPARQL query depends on incoming event data and is generated based on their attributes or is independent and describes only some attributes or the type of events. *(Generated or Not Generated)* 3. By Generated SPARQL queries from event attributes, the number of event attributes used for generating SPARQL query *(Single attribute or Several attributes)* 4. By generated SPARQL queries whether they are generated from several events in a sliding window or they are generated from a single event instance. *(Sliding Window or Single Event)*

In this paper, we describe the most important and interesting categories of event query rules. This categorization is not a complete classification of all possible rule combinations, our aim is more to emphasize interesting rule com-

binations which can be processed using different event processing approaches. Our implementation of these event query rules and our initial experiments with these rules are described in [8].

Category A - Single SPARQL Query: In this category, the event query rule includes only one single knowledge query and uses its results in one or more variables within the event detection rule. A SPARQL query is used to import knowledge about event instances or types. One or more attributes of events are used to build the basic triple pattern inside the SPARQL query. Category A event query rules can be categorized into three subcategories:

Category A1 - Raw SPARQL: This category of event query rule is the simplest form of these event query rule. The included SPARQL query is only about the resources in the background knowledge. The background knowledge query is independent from the event stream, however the complex event detection is defined on the results of this query in combination with the event stream. In some cases, on each event the SPARQL query should be resent to the KB to update the latest results from the KB.

Category A2 - Generated SPARQL: In this category of event query rules with each incoming event a different SPARQL query is generated and sent to the target knowledge base. The attribute/values of an event instance are used to generate basic triple patterns of a SPARQL query. Based on user definitions some of the tuples (attribute, value) of an event instance are selected and used to generate a single SPARQL Query.

Category A3 - Generated SPARQL from Multiple Events: The query is similar to A2, but the SPARQL query is generated from multiple events. Within a data window (e.g., a sliding time window) from two or more events a single SPARQL query is generated. Multiple events are used to generate the single SPARQL query, the event processing waits for receiving some new events and then generate a SPARQL query based on the emitted events, and query for the background knowledge about them.

Category B - Several SPARQL Queries: Queries of this category include several SPARQL queries and combine them with event detection rules. This means that several A category rules are combined together which can build a category B. The category B of rules are able to combine results from KBs with events using event operation algebra.

Category B1 - Several SPARQL Queries in AND, OR and SEQ Operations: The category B1 is based on the category B, but the results from the SPARQL query predicates are combined with AND, OR, SEQ or similar event algebra operations. The whole query is evaluated on sliding windows of event streams. The SPARQL query predicates are not depending on each other, i.e., the results from one is not used in another SPARQL predicate, so that they are not depending on the results of the other SPARQL query.

Category B2 - Chaining SPARQL Queries: In category B2 several SPARQL queries are generated and executed in sequence. They can be generated based on the results of the previous SPARQL query. Each SPARQL query can be generated from a set of events (e.g., included in a slide of event stream by means of a sliding

window, a counting or timing window). This means that different data windows can be defined to wait until some events happened and then a SPARQL query is executed. SPARQL queries might be defined in a sequence chain. The results are directly used for event processing or used in another following SPARQL query.

Category B3 - Chained and Combined SPARQL Queries: In this category SPARQL queries are used in combination with all possible event algebra operations like, AND $\bigwedge$, OR $\bigvee$, SEQ $\bigoplus$, Negation $\neg$, etc. The event operations are used for combining the results from several SPARQL queries or several SPARQL queries are used in combination with event algebra operations like: $((sparql_1 \bigoplus sparql_2) \bigwedge sparql_3 \bigvee \neg sparql_4)$. This category of event query rules is the general form of queries and has the highest possible complexity, because the results from external KBs are used in combination with event operations or the attribute/values from incoming events are used for generation of complex SPARQL queries.

4 Conclusion and Future Work

In this paper, we have the different categories of event query rules which use special rule predicates for importing data from external KBs and its combination with event algebra operations. Our future steps are to work on details of different event processing algorithms for each of the different rule categories, e.g. by rewriting complex event query to several simple queries which can be distributed over an event processing network.

References

1. Chakravarthy, S., Mishra, D.: Snoop: an expressive event specification language for active databases. *Data Knowl. Eng.*, 14:1–26, November 1994.
2. Paschke, A.: Eca-lp/eca-ruleml: A homogeneous event-condition-action logic programming language. In *RuleML-2006*, Athens, Georgia, USA, 2006.
3. Paschke, A.: Eca-ruleml: An approach combining eca rules with temporal interval-based kr event/action logics and transactional update logics. *CoRR*, abs/cs/0610167, 2006.
4. Paschke, A., Bichler, M.: Knowledge representation concepts for automated sla management. *Decis. Support Syst.*, 46(1):187–205, 2008.
5. Prud'hommeaux, E., Seaborne, A.: SPARQL Query Language for RDF. W3C Recommendation, 2008.
6. Teymourian, K., Paschke, A.: Semantic rule-based complex event processing. In *RuleML 2009: Proceedings of the International RuleML Symposium on Rule Interchange and Applications*, 2009.
7. Teymourian, K., Paschke, A.: Towards semantic event processing. In *DEBS '09: Proceedings of the Third ACM International Conference on Distributed Event-Based Systems*, pages 1–2, New York, NY, USA, 2009. ACM.
8. Teymourian, K., Rohde, M., Paschke, A.: Fusion of background knowledge and streams of events. In *Proceedings of the 6th ACM International Conference on Distributed Event-Based Systems.* ACM.