Enabling Knowledge-Based Complex Event Processing

Kia Teymourian

Supervisor: Prof. Adrian Paschke
Freie Universitaet Berlin, Berlin, Germany
{kia, paschke}@inf.fu-berlin.de

Abstract. Usage of background knowledge about events and their relations to other concepts in the application domain can improve the expressiveness and flexibility of complex event processing systems. Huge amounts of domain background knowledge stored in external knowledge bases can be used in combination with event processing in order to achieve more knowledgeable complex event processing. In this dissertation, I address the challenges of adding formalized vocabularies/ontologies and declarative rules to the area of event processing for enabling more intelligent event processors which can understand the semantics of events.

1 Motivation

In many business organizations some of the important complex events cannot be used in process management, because they are not detected from the workflows and decision makers can not be informed about them. Detection of events is one of the critical factors for the event-driven systems and business process management. Because of current successes in business process management (BPM) and enterprise application integration (EAI), many organizations know a lot about their own activities, but this huge amount of event information can not be used in the decision making process. The permanent stream of low level events in business organizations needs an intelligent real-time event processor. The detection of occurrence of complex events in the organization can be used to optimize the management of business processes.

Semantic models of events can improve event processing quality by using event meta-data in combination with ontologies and rules (knowledge bases). The combination of event processing and knowledge representation can lead to novel semantic-rich event processing engines. These intelligent event processing engines can understand what is happening in terms of events, can (process) state and know what reactions and processes it can invoke, and furthermore what new events it can signal. The identification of critical events and situations requires processing vast amounts of data and metadata within and outside the systems. Knowledge about event types and their hierarchies i.e. specialization, generalization, or other forms of relations between events can be useful. Semantic (meta) models of events can improve the quality of event processing

by using event metadata in combination with ontologies and rules (knowledge bases). Event knowledge bases can represent complex event data models which link to existing semantic domain knowledge such as domain vocabularies / ontologies and existing domain data. 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, e.g. a time ontology describing temporal quantities.

The usage of background knowledge in event processing can have several use cases such as: e-health, business activity monitoring, fraud detection, etc.

Use Case - High Level Stock Market Monitoring: Companies have some business dependencies to each other, e.g., a company C_1 produces raw material M_1, the business of another company C_2 depends on this raw material for its production and might have big troubles if they can not supply the material. A third company C_3 financed the company C_2 and might have some financial problems if the company C_2 have some material troubles. Let's consider that Mr. Smith is a stock broker and has access to a stock exchange event stream like: $(Name, \text{“}VOW''), (Price, 20.24), (Volume, 8, 835)$ Mr. Smith might be interested in this dependency chain and can define a complex event detection pattern for this special complex event without even knowing what these companies are. He might be interested to know when the prices for these three companies have started falling.

Mr. Smith might also be interested in special kinds of stocks and would like to be informed if there are some interesting stocks available for sale. His special interest or his special stock handling strategy can be described in high level language which describes the interest using background knowledge about companies. Mr. Smith would like to start a query on the event stream similar to the following query: **Buy** Stocks of Companies, **Who** have *production facilities in Europe* **and** *produce products from Iron* **and** have more than *10,000 employees* **and** are at the moment in *reconstruction phase* **and** their *price/volume increased stable in the past 5 minutes*.

As we can see, the above query cannot be processed without having background knowledge which can define the concepts in this query. Mr. Smith needs an intelligent system which can use background knowledge about companies. A background knowledge like the following should be integrated and processed together with the event data stream in a real-time manner so that interesting complex events can be timely detected.

```
{ (OPEL, belongsTO, GM),(OPEL, isA, automobilCompany),
  (automobilCompany, produce, Cars), (Cars, areFrom, Iron),
  (OPEL, hatProductionFacilitiesIn, Germany), (Germany, isIn, Europe),
  (OPEL, isA, MajorCorporation), (MajorCorporation, have, over10,000employees),
  (OPEL, isIn, reconstructionPhase), ... }
```

2 Research Problem

The existing event processing approaches are dealing primarily with the syntactical processing of low-level signals, constructive event database views, streams, and primitive actions. They provide only inadequate expressiveness to describe

the ontological semantics of events, actions, agents, states, processes, temporal/spatial concepts and other related concepts. They also do not provide adequate description methods for the complex decisions, behavioral logics including expressive situations, pre- and post-conditions, complex transactional (re-) actions, and work-flow like executions. All of these are needed to declaratively represent many real-world domain problems on a higher level of abstraction. My dissertation will address the following two main problems of the existing event processing approaches:

1. Lacking Knowledge Representation Methods: Event processing needs a knowledge (metadata) representation methodology. The current event processing systems do not provide any knowledge representation methods for events, and there is no precise logical semantics about events and other related concepts. There is a need for methods which can include ontological semantics of all related concepts to the event processing without affecting the scalability and real-time processing. A formal specification can build a stable foundation which is needed for any describing and reasoning about a system. It is also needed for comparing different systems without misunderstandings. Event processing needs as its basis a formalization and specification which can describe events, event patterns, situations, pre- and post-conditions, (re-) actions etc. Definition of events by logic is not addressed in current complex event processing solutions.

In this dissertation, I will address the challenge of knowledge representation for complex event processing (CEP) which integrates the domain and application specific ontologies for events, complex events, situations, actions and other concepts related to CEP.

2. Limited Processing and Integration Method of Background Knowledge with Event Stream: The processing approach of current event processing engines often rely on processing of simple event signals. They do not implement any usage of metadata about events or other related concepts from the application domain. The existing on-the-fly in-memory processing methods do not address the challenges of integration of background knowledge and semantic enrichment of events or event queries (complex event definitions/patterns). In this dissertation, I will address the nature of the trade-off real-time high-performance processing of events and expressiveness reasoning on background information. The advantages and disadvantages of alternative processing methods for the fusion of event stream and background knowledge should be investigated which can be used without effecting the real-time processing or scalability.

3 Fusion of Events and Background Knowledge

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. I propose to use a *Knowledge Base (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 stock market event (price change) to the products or services of a company.

The benefits of using background knowledge in CEP are higher *expressiveness* and *flexibility*. Expressiveness means that an event processing system can precisely express CEP patterns and reactions. 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. Furthermore, complex event processing can benefit from the knowledge representation and semantic web technologies, because a central problem of event processing is information integration for which these technologies have already been proven to be a valid solution.

I 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[1] 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 also includes a description logic 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[2] queries and the results are then included in the event processing engine.

4 Knowledge Representation for CEP

Ontologies play an important key role in the knowledge-based CEP. They should be the conceptualization of the application domain to allow reasoning on events and other non-event concepts. I propose that event processing domain should be described by a modular and layered ontology model which can be reused in different application areas. Important general concepts such as event, action, situation, space/place, time, agent and process should be defined based on meta-models and pluggable ontologies which are in a modularized ontological top-level structure. These general concepts defined in the top-level ontologies can be further specialized with existing domain ontologies and ontologies for generic tasks and activities. The applications ontologies for specialize these domain and task concepts with respect to a specific application, often on a more technical platform specific level.

Event Query Rules: Event query rules (Complex event Patterns) can be considerd as declarative rules which are used to detect complex events from streams of raw events. 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.

The event query rules allow simple event algebra operations, similar to Snoop [6] i.e. event operations like Sequence (Ordered), Disjunction (Or), Xor (Mutually Exclusive), Conjunction (And), Concurrent (Parallel), Any, Aperiodic,

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

[2] SPARQL http://www.w3.org/TR/rdf-sparql-query/

Periodic, Operator (generic Operator). Futher higher interval-based event operations like (BEFORE, MEETS, OVERLAP, ...) can also be used. My event query rules also include SPARQL query predicate to query external KBs, the SPARQL queries are used in a rule in combination with event operation algebra. This hybrid use of SPARQL query with event operation algebra can be categorized into several categories.

4.1 Categorize of Event Query Rules

Event query rules can be categorized into several categories based on the usage of knowledge queries (SPARQL queries) inside the query rule. As previously described, 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. In this section we describe the most important and interesting categories of event sQuery rules. This categorization is not a complete classification of all possible rule combinations, our aim is more to emphasize interesting rule combinations which can be processed using different event processing approaches.

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 sQuery rules can be categorized into three subcategories:

Category A1 - Raw SPARQL: This category of sQuery 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 sQuery 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.

5 Integration and Processing of Event Stream with KB

Based on the above discussed categories of event query rules, different event processing approaches are possible to satisfy the requirements of event processing agent (EPA), e.g., high performance, scalability and elasticity. In the following different processing approaches are disscused:

Polling the Knowledge Base: The basic approach is to execute a query on the KB on each incoming event. After events are emitted and received in EPA, the EPA sends one or more queries to KB for every event. The problems of this approach, scalability and real-time processing, makes it impossible to use it for time-sensitive use cases like algorithmic trading or fraud-detection systems.

Knowledge Query First (KQF): For the processing of some rule categories, it is possible to execute the SPARQL query in advance and offline to the live event stream, i.e. execuction of SPARQL query, before the events are emitted to the system. The results of knowledge queries can be cached in the main memory and be processed together with the events. Nevertheless, the results of the knowledge query can be old results from the knowledge base, hence they should be updated from time to time, e.g., by executing the whole query or pushing the result differences to the event processing agent.

Plan-Based Processing (PBP) This approach is about processing of event query based on an optimal plan for its sub-queries to avoid any unnecessary costs or losses of time. Some rule categories like category B1 rules, have several SPARQL queries which use multistep knowledge acquisition from external KBs. These SPARQL queries are combined in AND, OR, SEQ or similar event operations and the whole query should be evaluated in a time window. This makes it possible that the SPARQL queries can be executed in a sequence one after another or in a parallel setting. An execution plan can be generated to find out which execution plan is the low cost plan and which execution plan can be considered as high performance execution plan.

Event Query Preprocessing (EQPP) Event Query Preprocessing (EQPP) means that the complex query is preprocessed before the query is executed against the incoming stream event data. The original complex event query can be preprocessed by use of a KB and rewritten into several simple *new queries*. The original complex event query Q_a is preprocessed under the usage of a KB and divided into a set of simple event queries like $\{q_1, ...q_n\}$. A simple query is here a query which can be processed only with the information from the event stream and there is no need for using background knowledge. In the next step, these *new queries* can be syntactically processed on a network of event processing agents. The complex query Q_a can be considered as a propositional formula which can be converted to conjunctive normal form (CNF) $Q_a \leftarrow q_1 \wedge ... \wedge q_n$, i.e. if all of the simple queries are given, then the complex event query is satisfied. The preprocessing is done by a processing agent which can access the KB and divide the complex query into several simple queries. The complex query Q_a can also be mapped in disjunctive normal form (DNF) $Q_a \leftarrow q_1 \vee ... \vee q_n$, i.e. when one of the simple queries is triggered, then the complex event query will be satisfied and triggered.

6 Related Work

The state of the art approaches for event processing can be distinguished into two categories, rule-based approaches and non-rule-based approaches. Some of the event processing systems use non-deterministic finite state automata like Cayuga[4] or ESPER[3]. Many event processing languages have been proposed like, Snoop [6], Cayuga Event Language (CEL)[4], XChangeEQ [5]. Also several data stream processing systems have been proposed like Telegraph[8] which are targeted at handling continuous queries over high-throughput data streams. These systems are also related to the event processing systems[7].

Some stream reasoning languages and processing approaches are also proposed. Barbieri et.al. propose Continuous SPARQL (C-SPARQL) [3] as a language for continuous query processing and Stream Reasoning. Stream reasoning approaches for reasoning on RDF stream are not designed for fusion of background KBs and event stream. One of the recent rule-based systems is ETALIS [2]. ETALIS is a rule-based stream reasoning and complex event processing (CEP). ETALIS is implemented in Prolog and uses Prolog inference

[3] Esper: http://esper.codehaus.org , May 2012

engine for event processing. EP-SPARQL [1] is a language for complex events and stream reasoning. The formal semantics of EP-SPARQL is along the same lines as SPARQL. EP-SPARQL can be used in ETALIS for reasoning on RDF triple stream (event stream can be mapped to RDF stream). I have discussed CEP approaches which are most related for our knowledge-based CEP.

7 Future Work

My future steps are to work on more details of knowledge representation for events, situations, actions, and other related concepts. One of my tasks is to work on details of event query preprocessing algorithms for rewriting of complex event queries to several simple queries which can be distributed over an event processing network. Furthermore, I have to work on the described plan-based approach and specification of heuristics which can be used for selection of the optimized processing plan for a given query.

References

1. Darko Anicic, Paul Fodor, Sebastian Rudolph, and Nenad Stojanovic. Ep-sparql: a unified language for event processing and stream reasoning. In *Proceedings of the 20th international conference on World wide web*, WWW '11, pages 635–644, New York, NY, USA, 2011. ACM.
2. Darko Anicic, Paul Fodor, Sebastian Rudolph, Roland Stühmer, Nenad Stojanovic, and Rudi Studer. Etalis: Rule-based reasoning in event processing. In Sven Helmer, Alexandra Poulovassilis, and Fatos Xhafa, editors, *Reasoning in Event-Based Distributed Systems*, volume 347 of *Studies in Computational Intelligence*, pages 99–124. Springer Berlin / Heidelberg, 2011.
3. Davide Francesco Barbieri, Daniele Braga, Stefano Ceri, and Michael Grossniklaus. An execution environment for c-sparql queries. In *Proceedings of the 13th International Conference on Extending Database Technology*, EDBT '10, pages 441–452, New York, NY, USA, 2010. ACM.
4. Lars Brenna, Alan Demers, Johannes Gehrke, Mingsheng Hong, Joel Ossher, Biswanath Panda, Mirek Riedewald, Mohit Thatte, and Walker White. Cayuga: a high-performance event processing engine. In *SIGMOD '07: Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, pages 1100–1102, New York, NY, USA, 2007. ACM.
5. François Bry and Michael Eckert. Rule-based composite event queries: The language xchangeeq and its semantics. In *Proceedings of First International Conference on Web Reasoning and Rule Systems, Innsbruck, Austria (7th–8th June 2007)*, volume 4524 of *LNCS*, pages 16–30, 2007.
6. S. Chakravarthy and D. Mishra. Snoop: an expressive event specification language for active databases. *Data Knowl. Eng.*, 14:1–26, November 1994.
7. Sharma Chakravarthy and Qingchun Jiang. *Stream Data Processing: A Quality of Service Perspective Modeling, Scheduling, Load Shedding, and Complex Event Processing*. Springer Publishing Company, Incorporated, 1st edition, 2009.
8. Sirish Chandrasekaran, Owen Cooper, Amol Deshpande, Michael J. Franklin, Joseph M. Hellerstein, Wei Hong, Sailesh Krishnamurthy, Samuel R. Madden, Fred Reiss, and Mehul A. Shah. Telegraphcq: continuous dataflow processing. In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, SIGMOD '03, pages 668–668, New York, NY, USA, 2003. ACM.