Freie Universität Berlin
Department of Mathematics and Computer Science
Institute for Computer Science

DISSERTATION

A Framework for Knowledge-Based Complex Event Processing

in partial fulfillment of the requirements for the degree of
doctor rerum naturalium

by

Kia Teymourian

August 14, 2014

Supervisor:
Prof. Dr. Adrian Paschke
Freie Universität Berlin
Institute for Computer Science
Corporate Semantic Web Research Group

Second Supervisor:
Prof. Dr. Oscar Corcho
Universidad Politécnica de Madrid
Departamento de Inteligencia Artificial
Facultad de Informática

Submission: August 14, 2014
Defense: November 28, 2014

Eidesstattliche Erklärung
Ich erkläre hiermit an Eides Statt, dass ich die vorliegende Dissertation selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe. Diese Arbeit wurde bisher noch nicht in einem früheren Promotionsverfahren eingereicht. (§7 Abs. 4 – Promotionsordnung des Fachbereichs Mathematik und Informatik der Freien Universität Berlin, Stand 2007)

Statement in Lieu of an Oath

I hereby confirm that I have written this thesis on my own and that I have not used any other media or materials than the ones referred to in this dissertation.

Berlin, August 14, 2014

Kia Teymourian

I also declare that parts of this thesis have already been published in the following papers:

- TEYMOURIAN, Kia and PASCHKE, Adrian: Plan-Based Semantic Enrichment of Event Streams, in: 11th Extended Semantic Web Conference (ESWC 2014), Crete, Greece
- TEYMOURIAN, Kia; ROHDE, Malte and PASCHKE, Adrian: Fusion of background knowledge and streams of events, in: 6th ACM International Conference on Distributed Event-Based Systems DEBS2012, Berlin, Germany, July 16-20, 2012, ACM, pp. 302–313
- TEYMOURIAN, Kia; ROHDE, Malte and PASCHKE, Adrian: Knowledge-based processing of complex stock market events, in: 15th International Conference on Extending Database Technology, EDBT2012, Berlin, Germany, March 27-30, 2012, ACM, pp. 594–597
- TEYMOURIAN, Kia and PASCHKE, Adrian: Semantic Processing of Sensor Event Stream by Using External Knowledge Bases (Short Paper), in: Proceedings of the 5th International Workshop on Semantic Sensor Networks, SSN12, Boston, Massachusetts, USA, November 12, 2012, vol. 904 of CEUR Workshop Proceedings, CEUR-WS.org, pp. 121–126
- TEYMOURIAN, Kia; COŞKUN, Gökhan and PASCHKE, Adrian: Modular Upper-Level Ontologies for Semantic Complex Event Processing, in: Proceedings of the Fourth International Workshop, WOMO 2010, Toronto, ON, Canada, May 11, 2010, vol. 210 of Frontiers in Artificial Intelligence and Applications, IOS Press, pp. 81–93
- TEYMOURIAN, Kia and PASCHKE, Adrian: Enabling knowledge-based complex event processing, in: Proceedings of the 2010 EDBT/ICDT Workshops, Lausanne, Switzerland, March 22-26, 2010, ACM International Conference Proceeding Series, ACM
- TEYMOURIAN, Kia; STREIBEL, Olga; PASCHKE, Adrian; ALNEMR, Rehab and MEINEL, Christoph: Towards Semantic Event-Driven Systems, in: 3rd International Conference on New Technologies, Mobility and Security, 20-23 December 2009, Cairo, Egypt, IEEE, pp. 1–6
- TEYMOURIAN, Kia. Enabling Knowledge-Based Complex Event Processing. in: Proceedings of the VLDB PhD Workshop, co-located with 37th Intl. Conference on Very Large Databases (VLDB2011), 2011
- TEYMOURIAN, Kia; ROHDE, Malte; HASSAN-HAIDAR, Ahmad and PASCHKE, Adrian: Processing of Complex Stock Market Events Using Background Knowledge, in: Proceedings of the Workhop Detection, Representation, and Exploitation of Events in the Semantic Web Workshop in conjunction with the 10th International Semantic Web Conference 2011 23 October 2011, Bonn, Germany.
- TEYMOURIAN, Kia; ROHDE, Malte and PASCHKE, Adrian: Processing of Complex Stock Market Events Using Background Knowledge, in: Proceedings of RuleML2011@BRF Challenge, co-located with the 5th International Rule Symposium, Fort Lauderdale, Florida, USA.
- TEYMOURIAN, Kia and PASCHKE, Adrian: Semantic Rule-Based Complex Event Processing, in: RuleML 2009: Proceedings of the International RuleML Symposium on Rule Interchange and Applications, RuleML, vol. 5858 of Lecture Notes in Computer Science, Springer, pp. 82–92
- TEYMOURIAN, Kia and PASCHKE, Adrian: Towards semantic event processing, in: DEBS ’09: Proceedings of the Third ACM International Conference on Distributed Event-Based Systems, ACM, New York, NY, USA.

Abstract

Existing event processing approaches deal primarily with the syntactic processing of low-level primitive events. Usage of ontological background knowledge regarding events and their relations to other resources improves the quality of event processing.

The research problem of this dissertation is the utilization of background knowledge about events and other relevant concepts for enabling knowledge-based complex event processing. This integration enhances the expressiveness of event processing semantics and facilitates the specification of event patterns based on the relations of events in the background knowledge graph.

We propose a framework for the fusion and processing of event streams and background knowledge. The first problem that we faced is the lack of knowledge representation methods for the representation of events and other relevant concepts. We propose using modular upper-level ontologies for semantic-enabled complex event processing. We discuss why modularity is needed and how these ontologies should be modularly built up based on ontology engineering aspects so as to be able to address expressiveness and reuse of ontologies. For the representation of complex event patterns, a combinatorial pattern specification is proposed based on background knowledge patterns and event algebra operators.

The second challenge is the limitation of processing methods for the fusion of event streams and huge amounts of background knowledge. Typically, there is a trade-off between the high expressiveness of the background knowledge used, which leads to higher levels of computational complexity, and the efficiency and performance requirements in real-time event processing. Although some of the existing approaches work directly on stream reasoning, they do not address event detection and inference on the basis of huge amounts of background knowledge.

We propose different event processing approaches for the fusion of background knowledge with real-time event streams. The first one is semantic enrichment of event streams which is an approach for the enrichment of events prior to the event processing step. Semantic enrichment is a multi-step event enrichment and detection process so that events are processed by multi-passing through several event processing steps. We provide algorithms for planning the enrichment process in order to achieve near real-time processing and optimize the cost of event enrichment. Our aim is to find low-cost event detection plans while meeting user latency expectations.

The enrichment of complex patterns is another approach that we propose for the integration of background knowledge. Complex patterns are converted to patterns that can be processed without requiring knowledge retrieval from external knowledge bases. Event patterns can be rewritten based on event operators and knowledge based results of subqueries.

One further approach for reducing the event processing load is to sample the event stream so that the event processing engine can observe only a subset of the original stream. Approaches for sampling data streams are mostly designed for computation with aggregation functions and not for pattern detection. We propose an approach for type-based sampling of event streams based on the entropy of event streams and properties of the user-defined event detection pattern. Our approach samples the original event stream by calibrating the sampling rate of each event type to achieve the requested throughput rate of raw events or detected complex events.

Acknowledgments

Foremost, I would like to express my sincere gratitude to my thesis adviser Prof. Adrian Paschke for supervising my research during the past several years. He has not only been a Ph.D supervisor to me, but also a mentor, a dear friend and a very nice colleague.

I am especially thankful to Prof. Oscar Corcho for accepting the external reviewing of this thesis. I also thank Prof. Manfred Hauswirth for his advices on my thesis.

I am indebted to Prof. Robert Tolksdorf who has made all of this possible and hired me for different research projects as research associate.

It has been a great pleasure to work with *Lyndon Nixon, Philipp Obermeier, Olga Streibel, Marko Harasic, Gökhan Coskun, Ralf Heese, Markus Lukzac-Rösch, Mario Rothe, Hannes Mühleisen, Malte Rohde, Moritz von Hoffen, Alexandru Todor, Ralph Schäfermeier, Wojciech Lukasiewicz, Shashishekar Ramakrishna and many others* in the research working group of networked information system including the corporate semantic web research group at the Freie Universität Berlin.

I thank the anonymous reviewers for their comments on the early versions of my scientific publications during my work at the Freie Universität Berlin as research associate and Ph.D student.

Especially, I would like to give my special thanks to my wife Mitra for being my guiding light, for her patience and for her moral support in the course of this research.

I am deeply thankful to my parents, my sister and brother for their support and encouragement.

Kia Teymourian
Berlin, August 14, 2014

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

In recent years the expanding capabilities of information systems characterized by Moore's law[1] and the current successes in business process management and enterprise application integration technologies, have made it possible that organizations can acquire huge amounts of real-time event data about business activities. In today's business processes, complex issues happen in real-time and the decision makers can profit from real-time data processing and on time knowledge acquisitions in order to be aware of the latest situations to trigger optimal reactions.

The detection of complex events in organization can be used for optimized management of business processes. 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 finance, logistics, automotive, telecommunication and life sciences.

A critical part of organization knowledge is real-time knowledge which should be better utilized. Huge amounts of an organization's real-time data that flow into different systems as data stream can be filtered, mapped, enriched and processed in real-time so that the decision makers can access *the right information at the right time and at the right place* so as to be able to make *the right decisions*. One of the crucial success factors for today's enterprises is the timely delivery of information to trigger reactions to potential challenges and opportunities.

Some complex events cannot be considered in business process management because they stem from complex factors and cannot be simply detected in the business work-flows by using low-level definitions of event detection rules. The specification of complex event detection patterns is a challenging task and requires expertise and in-depth knowledge about the business application. The permanent streams of low level events in different business sectors need an intelligent real-time event processor that can profit from the large amounts of knowledge stored in the knowledge bases.

In computer science, especially in event-based systems, an event can be defined as "Anything that happens, or is contemplated as happening."[DL11]. In most cases events are any transient occurrence of a happening of interest that can be observed from within a computer system [GM06]. An event instance is a concrete semantic object containing data describing the event. A common model for an event instance is, e.g., a structure consisting of name/value pairs like a stock market event: $\{(type, StockQuote), (name, APPL''), (price, 80)\}$.

[1]The prediction of Gordon Moore, co-founder of the Intel company, in 1965 about the progress of capabilities computer systems that the number of components in integrated circuits would double each year. During the last 15 years we can observe an exponential growth.

Complex Event Processing (CEP) [Luc01] is an emerging enabling technology used to achieve actionable, situational knowledge from huge amount of events in real-time or almost close to real-time. One of the fundamental requirements of CEP systems is the time-critical processing of events. The processing of events can be seen as fundamental quality of service (QoS) requirements of event processing systems [Cha09].

CEP has different use cases such as Business Activity Monitoring (BAM), Healthcare, Fraud Detection, Smart Offices/Cities, Logistics and Cargo, Information Dissemination, Event-driven Adaptive Systems, Supply Chain Management [Etz10b, Luc01].

Semantic models of events can improve event processing quality by using event metadata 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, have been to break new grounds on novel research and application. In recent years, huge amounts of public background knowledge are available on the Web as Linked Open Data (LOD)[2]. Organizations can access these background data, filter, enrich and integrate them with organization's internal knowledge to build high quality and trustworthy background knowledge (metadata) about their specific business application domain.

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 happens 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.

One of the promising areas of application is *Semantic Complex Event Processing (SCEP)* (or synonym Knowledge-Based Complex Event Processing)[3] which is the main subject of this dissertation. Complex events are events which are not explicitly defined before event processing, but a system user can describe them in a high level and abstract language based on background knowledge of the target application domain.

In most of the CEP use cases, usage of background knowledge can improve the quality of event processing. The CEP system can profit from using semantics of events in combination with existing knowledge about the target application domain. The benefits of the fusion of ontological background knowledge with the event processing are the same as using metadata in data-driven applications like extracting more relevant and useful complex events[4] and flexibility that enables interoperability and extensibility [5]. Complex events, patterns and reactions can be precisely expressed and can be directly translated into business operations. New business changes can be integrated into CEP systems in a fraction of the time, while complex event patterns are independent of actual business and are defined based on abstract business strategies.

Some of the motivating examples of SCEP use cases are presented below to illustrate the benefits of using background knowledge in the domain of complex event processing. More details about the use cases of SCEP and their requirements are described in Chapter 4.

[2]Linked Open Data (LOD) `http://linkeddata.org`

[3]The term semantic complex event processing can be confusing. It can be confused with the formal semantics of event detection rules. With the term semantic we refer to the usage of ontological background knowledge about events and other resources in the application area.

[4] 5 Things To Know About Semantic Technologies `https://www.ibm.com/developerworks/community/blogs/5things/entry/5_things_to_know_about_the_semantic_technologies?lang=en` retrieved April, 2014

[5]Case Study: Open Services Lifecycle Collaboration framework based on Linked Data by Arnaud Le Hors and Steve Speicher, IBM `http://www.w3.org/2001/sw/sweo/public/UseCases/IBM/` retrieved April, 2014

One example of highly complex events can be found in eHealth. A patient in a critical, life-threatening situation can be seen as one such complex events. Patient can be monitored at hospitals [Wie10] or at home using different body sensing devices, RFID readers. These technologies makes it possible to capture huge amount of stream data about the vital signs of patients. The statement whether a patient is in critical situation can only be done by the medical staff. However, the medical staff can be informed about these critical situations so that they can react faster. A system can support these decisions and detect complex events based on the event stream of sensor data which are emitted from different body sensors and other tracking devices.

A classic example in the area of event processing and data stream processing is the real-time analysis of stock market events generated by the stock market exchange. The decision to buy or sell company shares depends on real-time information and background knowledge about different companies. Stock brokers develop their negotiation strategies based on chunks of information that they gather from their resources. Their strategy may depend not only on stock market prices and the volumes handled, but more on a combination of the companies' attributes and relationships (e.g., to other organizations or to their customers) in a specific temporal context. A buy or sale decision can be based on market status, company products, company staff and all of the situational attributes of companies. Most of today's monitoring and automated handling systems work on the basis of the syntactic processing of price/volume event stream, but not based on background knowledge about the companies.

They can only deal with it in the case of increasing/decreasing of prices/volume around pre-defined levels. The brokers might be able to describe it in a higher level and more abstract language, which can be processed by intelligent processing systems.

For example, a stock market broker might be interested in the shares of *companies*, which

have *production facilities in Europe* **and**
produce products from *iron* **and**
have more than *10,000 employees* **and**
are at the moment in *restructuring phase* **and**
their price/volume *have been increasing continuously* in the *past 5 minutes*.

1.2 Problem Description

The state of the art event processing approaches (reviewed in Chapter 3) deal primarily with the syntactic processing of low-level primitive events, constructive event database views, streams and primitive actions. The processing approach of current event processing engines often rely on the processing of simple event signals and events are merely implementation issues. The existing approaches provide only inadequate expressiveness to describe background knowledge about events and other resources in background knowledge which are related to event processing. They do not provide adequate description methods for 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 real-world domain problems on a higher level of abstraction. The identification of critical complex events requires processing vast amounts of data and metadata within and outside the event processing systems. For some of application scenarios, an intelligent event processing engine is required which can understand, what happens in terms of events and can (process) state and know what reactions and processes it can invoke, and which new events it can signal.

The fusion of event processing approaches and knowledge representation methods can lead to more knowledge-able event processing systems. However, the existing knowledge representations and inferencing techniques are not suitable for event processing applications because of the different requirements of event processing applications such as real-time processing of events.

Knowledge representation for events goes beyond event types and their hierarchies, the relationships of events to other non-event concepts in the application domain can specify highly complex events. Background knowledge for event processing can be integrated from the domain and application specific ontologies for events, processes, states, actions, and other concepts. Specific domain, task and application ontologies need to be dynamically connected and integrated into the respective event processing applications, which also leads to a modular integration approach for these ontologies. Capturing domain-specific complex events and generating complex reactions based on them is a fundamental challenge.

The main research problem that we address in this dissertation is the utilization of background knowledge about events and other non-event concepts and objects for a more knowledgeable complex event processing. This integration enhances the expressiveness of event processing semantics and makes the event processing systems more flexible. More precisely, we consider one or more event streams which include highly frequent events of different types and one or more knowledge bases, which include background knowledge about the events and other related resources, in the target application domain. Because background knowledge can be huge, we consider that the metadata have to be stored in external knowledge bases and cannot be handled in the main memory of a single event processing node.

To express the main aim of this dissertation briefly, we consider the following research statement:

> *In the light of existing ontological background knowledge about events, to extend the state of the art Complex Event Processing approaches in terms of expressiveness, automation and adaptation, so that event processing systems can exhibit from higher-level complex event specification based on the background knowledge, be more agile and higher flexible.*

The research questions that we address in this dissertation are twofold:

1. **Lack of Knowledge Representation Methods:** Event processing needs a knowledge representation methodology. The current event processing systems do not provide any knowledge representation methods for events, and there is no precise logical semantics about other related concepts and objects in the target domain.

   A formal specification builds a stable foundation which is needed for description and reasoning. Formal specification is also needed for comparing different systems without the danger of misunderstandings. Event processing needs as its basis a formalization and specification which can describe simple events, complex event patterns, situations, pre- and post-conditions, (re-) actions and other related concepts. Definition of events by logic is not addressed in state-of-the-art event processing solutions (see Chapter 3). Events have special characteristics and attributes which differentiate them form other objects. Events can happen. They can be considered as entities which unfold over time and exist only over a specific time window.

The following detail research questions related to this problem are addressed in this dissertation:

- How should raw events and complex events (event patterns) be represented based on relations of resources in background knowledge? How should knowledge about events and event patterns be represented?
- Which extensions are required so that a query language for detecting of complex events based on knowledge graph patterns and event operation algebra can be specified?
- What can be an adequate representation of events, actions, states, situations and other related concepts for event processing?
- How should ontologies about events, actions, states, situations and other related concepts be created and managed from the ontology engineering perspective, when we have to take into account the restrictions of event processing?

We specify complex event patterns based on the relations exist in background knowledge to improve the expressiveness and flexibility of the complex event processing patterns.

2. **Limitation of Processing Methods for Fusion of Events and Background Knowledge:** The existing event processing methods (reviewed in Chapter 3) can only detect and process events based on their syntax and incoming sequence . The expressiveness of event processing decreases when the system uses ontological knowledge about the events and their environment and combines it with detection and reaction rules.

   One of the obvious questions raised in the course of working on event processing systems is the complexity to which event processing engines can detect and process complex events. This question is difficult to answer because the capability of processing complex events when compared to each other depends on complex parameters. These parameters can only be defined based on the specific use cases of event processing. Because of this, the existing evaluations of event processing engines have not been able to answer this question properly. It is known that current event processing engines can only detect and process events based on their syntax and incoming sequence. The expressiveness of event processing decreases when the system uses ontological knowledge about the events and their environment and combines it with reaction rules.

   Event processing approaches should be extended so that they can include ontological semantics of events, processes, states, actions, and other concepts into event processing without affecting the scalability and real-time processing. The detailed research questions regarding this topic are:

   - If description logic expressiveness of event detection queries is a limitation for real-time processing or scalability of the system?
   - To what extent can we optimize the trade-off between expressive reasoning on background knowledge and real-time event processing? To what extent is the fusion of background knowledge with the event data stream a bottleneck for real-time event processing? Is it possible to avoid this bottleneck by using query optimization approaches and distribution on networks of processing nodes?
   - What are the assets and drawbacks of event enrichment approaches when they should handle huge amounts of background data and run computational intensive reasoning on external knowledge bases?

- Is it possible to preprocess detection queries to optimize their event detection query plans and select the most optimal one based on execution costs?

1.3 Our Approach

1.3.1 Methodology of the Thesis

We follow a constructive design science research methodology as described in [Hev04, Vai07, Bas09, Ven10]. In our research methodology we apply in particular the seven guidelines proposed by Hevner et al. in [Hev04]. Our methodology in this research can be described in the following points.

- **Awareness of the Research Problem:** With the study of state-of-the-art complex event processing approaches and the event processing use cases in business environment, we have identified a main heretofore unsolved problem (Guideline 2: Problem Relevance [Hev04]) regarding the usage of background knowledge, expressiveness of existing event processing approaches and the requirement for knowledge-based event processing systems.

  The problem that we address in this dissertation is defined as a difference between current state-of-the-art event processing approaches and the goal state for knowledge-based event processing. The current CEP systems can process events based on event operation algebras, they normally process events in real-time and can scale up to some levels of high throughput of event stream. Our goal system is a knowledge-based event processing system that can process events not only based on event operation algebra but more based on reasoning on related concepts in background knowledge.

  Our problem has its roots in two different research areas of computer science, event stream processing (also data stream processing) and knowledge representation. A good basis for our research is formed by the study of stat-of-the-art in complex event processing and related subjects from knowledge representation and logic programming. The awareness of the advantages and disadvantages of different existing approaches makes it possible to be able to develop solutions for our research problem.

  We have identified business use cases which include the applications with this problem and can profit from the research results of this work. From the requirements of CEP use cases, we analyzed the requirements for the knowledge-based event processing so that we can design artifacts for solving the research problem based on specific requirements.

- **Suggestion:** We have designed new artifacts (Guideline 1: Design as an artifact [Hev04]) which can provide solutions for the unsolved problem identified. We have developed new artifacts (Guideline 1: Design as an artifact [Hev04]) for knowledge-based complex event processing to improve the expressiveness and flexibility of event processing systems while meeting the requirements.

One of the basic requirements of event processing systems is the processing of events in real-time or close to real-time. We analyzed the trade-off between highly complex expressive reasoning on external knowledge bases and real-time event processing. We provide new artifacts which can be used to improve the latency of knowledge-based event processing. Our research provides clear and verifiable contributions (Guideline 4: Research Contributions in [Hev04]) in the area of complex event processing. We provide methodologies for detection of complex events based the relations of concepts and individuals in background knowledge. Different approaches are investigated to improve the throughput/latency of knowledge-based event processing.

- **Development:** In our search for an effective artifact (Guideline 6: Design as a Search Process [Hev04]) we have utilized different approaches to satisfy the requirements of event processing applications.

  We have looked into possibilities for the enrichment of event streams in the upstream of main complex event processing system and analyzed important service parameters like event stream throughput, latency and scalability. We searched for potential approaches to optimize the costs of event enrichment while achieving use case expectations on latency and throughput.

  The preprocessing of complex event queries is one of the other approaches that we have looked into for optimizing the latency of the system by disjunction of complex query to multiple queries which then can be processed onto a network of event processing engines by normal syntactic processing without further need of querying external knowledge bases.

  Furthermore, we have looked into approximation approaches for event processing, because in some of the business use cases false positives in event detection are acceptable as long as the system can meet latency requirements. We provide approaches for sampling event stream based on the given complex query and previously learned occurrence probability of different event types.

  Our research relies upon rigorous formal methods (Guideline 5: Research Rigor [Hev04]). We propose approaches of formalisms for the modeling of the connection between streaming event data and background knowledge. We have provided approaches for the specification of complex queries based on relations in background knowledge and combination with event operation algebra.

- **Evaluation:** We rigorously demonstrated the utility and efficacy of the artifact designed using executed experimental evaluation methods (Guideline 3: Design Evaluation [Hev04]).

  We have specified appropriate metrics (e.g., expressiveness, processing costs, latency, throughput, scalability) for our evaluation and we gathered and analyzed of appropriate data from existing, publicly available data. Furthermore, we evaluated our approach under controlled environment using synthetically generated data sets so as to be able to change the properties of event data stream like type and throughput in order to test the behaviors of the event processing system.

We set up different experiments for different event processing methods and evaluated the quality and quantity of our event processing methods. We have evaluated our work based on a selected use case and adequacy criteria of knowledge representation. We evaluated the efficiency of the knowledge-based approach for real-world business applications and demonstrated different required expressiveness level of knowledge representation in different use cases. Through iterations and analysis of our experiment results we improved our event processing enrichment and preprocessing algorithms. We used the experimental results to search for alternative approaches for fusion of background knowledge and event stream so that we could satisfy the requirement of event processing.

The result of our research are published at technology-oriented conferences [Tey09b, Tey09a, Tey09c, Tey10b, Tey10a, Tey11b, Tey11a, Tey12c, Tey12b, Tey12a, Tey14] as well as presented at management-oriented (business-oriented) workshops [6], which allowed us to communicated the results of our research to both audience groups (Guideline 7: Communication of Research [Hev04]).

1.3.2 Contributions of the Thesis

This dissertation presents a framework for knowledge-based complex event processing that includes the following contributions:

- **Survey and Comparison of Event Processing Semantics:** A number of complex event processing systems have been developed in the past, but so far there are no comprehensive surveys and comparisons of their processing semantics. All of the existing survey studies compare their event processing performance or their scalability but they do not consider their different event processing semantics (Chapter 3).
- **Use Cases for Knowledge-Based CEP:** We provide descriptions of potential use cases of knowledge-based event processing and identified the benefits of using external knowledge bases in these applications scenarios (Chapter 4).
- **Models and Architecture for Knowledge-Based CEP:** We provide a model for knowledge-based event processing and describe an architecture for the fusion of event stream and external knowledge bases (Chapter 5).
- **Knowledge Representation for CEP:** We contribute an extensible and modular ontology representation method which can be reused, customized or extended based on the requirement of different applications and tasks (Chapter 6).
- **Representation of Complex Event Patterns based on Background Knowledge:** We use a hybrid query rule language for the specification of complex event patterns based on fusion of knowledge from event stream and external knowledge bases. We propose a categorization of such event query rules based on the different factors (Chapter 7).

[6]Semantic Complex Event Processing – Corporate Semantic Web Workshop at Xinnovations September 2010, Berlin. `http://2010.xinnovations.de/corporate-semantic-web.html` (in German)

- **An approach for Plan-Based Semantic Enrichment of Event Streams:** We propose an approach for multi-step event enrichment and detection that processes events in multi-passing through multiple event processing steps. We provide an algorithm for finding an optimized plan for event enrichment and detection in order to optimize the caused load on the external knowledge bases for knowledge acquisition (Chapter 8).
- **An approach for Complex Queries Rewriting:** We propose an approach for the enrichment of complex event patterns by pre-processing and rewriting the detection queries from abstract (high-level) and knowledge-base-dependent form to simple event detection patterns that are independent of knowledge base (Chapter 9).
- **An approach for Event Processing on Sampled Event Stream:** Knowledge-based event processing can cause high processing costs on the event processing engines and external knowledge bases. One possible solution can be to process only a sub-stream of the original stream, when the use case allows a partial detection of the existing complex events. We provide an approach for optimized sampling of event stream based on the stream properties and the event detection pattern (Chapter 10).

1.4 Related Research Areas

Our research area is at the intersection of research areas, which we will introduce briefly in the following:

- **Active Databases:** One of the research communities who initiated the research on event processing was the communities of active database systems [Pat99, Cha94a] in the 1990s. Active databases are a specific type of databases which support mechanisms for automatic responses to events that take place inside or outside a database. Researchers' efforts in this research field have been directed at improving the understanding of composite events from single events. Event Condition Action (ECA) rules are defined as what can trigger actions after some events have happened and certain conditions are given. These approaches are reviewed in this dissertation and our solution approach is built based on the formal semantics for event detection which have its roots in active database research.
- **Event-Driven Architecture (EDA):** The research area around the software architectural style event-driven architecture is about the decoupling of the event producer from the event consumer and components are executed in response to event notifications [Tay09, Bru10, Etz10b]. This area is also called event-based programming.
- **Publish/Subscribe Technologies:** The research on publish subscribe technologies [Bal05, Hin10, Müh06] are targeted at finding solutions for timely delivery of event notifications from publishers to subscribers. Publish/subscribe systems (Pub/Sub) can be seen as a middleware between publishers and subscribers, it is made up of a network of brokers which match the event notifications and route them to the next brokers so that they can find their way through the network to the target subscriber.

- **Complex Event Processing (CEP):** Complex Event Processing has its roots in different research communities and fields like: software engineering, database systems, distributed systems. CEP [Luc01, Etz10b] is an emerging enabling technology used to achieve actionable, situational knowledge from huge amount of events in real-time or almost close to the real-time. CEP is one of the main subjects of this dissertation project.

- **Data Stream Processing:** Data stream processing is similar to complex event detection, data items arrive continuously to the system as a sequence of items, and the data stream processing system has to evaluate a user defined query (similar to relational database queries) repeatedly against new data from the data stream.

  The main difference between complex event processing and data stream processing is the type and formal semantics of the data query [He14]. In the CEP area the aim is to detect complex events based on the algebra operators and to detect complex events, e.g., based on their incoming sequence and properties. And in data stream processing there are queries similar to database SQL queries to filter data from the stream. Data stream processing is also discussed in detail in Chapter 3.

- **Logic Programming and Rules:** Detection of complex events can be realized based on event detection rules and the triggering of reactions and signaling of new events (e.g., by using reaction rules). Examples of these rules are Event-Condition-Action Rules and Reaction Rules. The relevant part of this area is described in Chapter 2.

- **Knowledge Representation and Semantic Technology:** Technologies related to knowledge representation, Ontology engineering, rules and reasoning on background knowledge are also relevant to our work, because we use these technologies to represent knowledge about events and their application domain. By doing reasoning on the background knowledge we aim to generate derived knowledge and integrate this knowledge with the knowledge coming from the event stream. The area of Knowledge Representation and Semantic Technology are briefly reviewed in Chapter 2.

- **Sensor Networks and Semantic Sensor Networks:** The semantic sensor network research area[7] has the aim of solving some of the existing problems in sensor networks by profiting from semantic technologies (background knowledge as ontologies, rules). This research area aims to benefit from the combination of semantic technologies and sensor networks.

1.5 Related Research Projects

Some of the important related research projects to our work are listed in this section.

- **The Large Knowledge Collider (LarKC) Project**[8] The aim of the EU project (Seventh Framework Programmes) Large-Scale Integrating Project (LarKC) (2008 - 2011) was to develop a platform for massive distributed incomplete reasoning that would remove scalability barriers of existing reasoning systems for the Semantic Web.

[7] See. W3C Semantic Sensor Network Incubator Group `http://www.w3.org/2005/Incubator/ssn/` or International Workshop on Semantic Sensor Networks `http://knoesis.org/ssn2012/`

[8] `http://www.larkc.eu/` retrieved April, 2014

- **Event recognition for intelligent resource management (PRONTO) Project**[9] emphasizes the role of event recognition in intelligent resource management. The project proposed (2009-2012) a methodology for fusing data from various sources, analyzing it to extract useful information in the form of events. The resulting knowledge can be delivered for decision making through a user-friendly intelligent resource management (IRM) service including a digital map interface (MAP). During the PRONTO project, the PRONTO technology was tested in two case studies: 1. Emergency Rescue Operations (ERO) 2. City Transport Management (CTM).
- **BiCEP Project**[10] a project from the Systems and Software Engineering Group at the University of Coimbra[11], had (2007-2010) the goal of studying and improving performance and produce benchmarks for Event Processing systems.
- **PLAY project** The research goals of the PLAY project [12] was to develop an elastic and reliable architecture for dynamic and complex, event-driven interaction in large highly distributed and heterogeneous service systems. The results of the project focus on describing the events in RDF data model, large scale distributed storage of RDF data and publish/subscribe system with RDF data models.

1.6 Structure of this Dissertation

The dissertation is organized in 4 parts and 11 chapters as illustrated in Figure 1.1. After the description of the main research problems, the next chapter will discuss the state of the art of relevant technologies, complex event processing and knowledge management.

The chapters of this dissertation are organized as follows:

- Chapter 2 Background on Semantic Technologies: At the beginning of this work, we provide in this chapter a brief introduction on related subject from knowledge representation and logic programming.
- Chapter 3 Related Work on Event Processing: In this chapter, we introduce the main concepts of event processing, so that the reader can follow the upcoming chapters. This chapter reviews the-state-of-the-art methods for complex event processing and introduces the main terms and expressions. We analyze the event processing approaches and compare them based on their event processing language. Furthermore, we describe the state of the art development on semantic event processing approaches and clarify the difference between these existing approaches and our work.
- Chapter 4 Use Cases of Knowledge-Based CEP: In this chapter we describe some of the use cases of complex event processing and extend them to the usage of background knowledge. We describe the benefits of knowledge-based complex event processing for the different application scenarios and analyze their requirements parameters, like expressiveness level of event detection queries, level of reasoning on background knowledge and the requirement for real-time event processing.

[9] `http://www.ict-pronto.org/` retrieved April, 2014
[10] `http://bicep.dei.uc.pt/` retrieved April, 2014
[11] `http://www.uc.pt/`
[12] `http://www.play-project.eu/` retrieved April, 2014

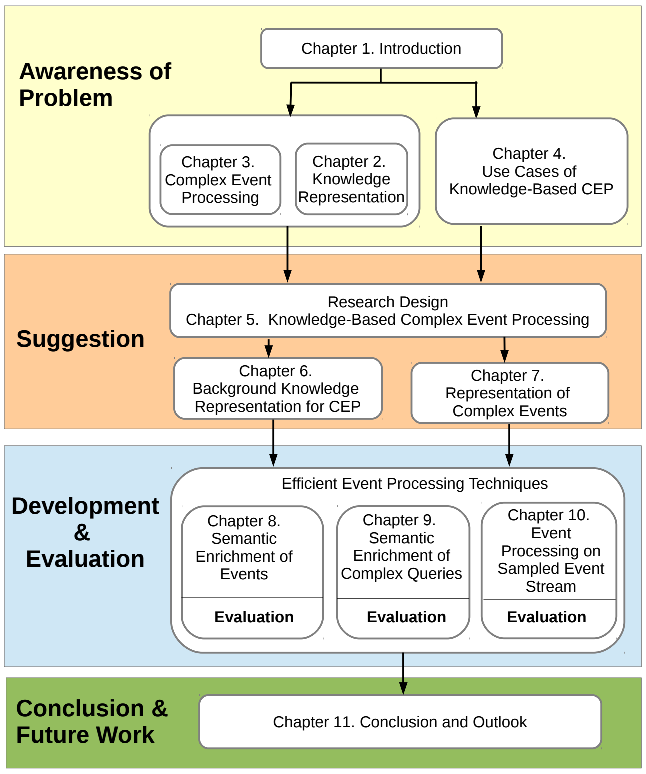

Awareness of Problem
Chapter 1. Introduction
Chapter 3. Complex Event Processing
Chapter 2. Knowledge Representation
Chapter 4. Use Cases of Knowledge-Based CEP
Suggestion
Research Design
Chapter 5. Knowledge-Based Complex Event Processing
Chapter 6. Background Knowledge Representation for CEP
Chapter 7. Representation of Complex Events
Development & Evaluation
Efficient Event Processing Techniques
Chapter 8. Semantic Enrichment of Events
Evaluation
Chapter 9. Semantic Enrichment of Complex Queries
Evaluation
Chapter 10. Event Processing on Sampled Event Stream
Evaluation
Conclusion & Future Work
Chapter 11. Conclusion and Outlook

Figure 1.1: Structure of this Dissertation

- Chapter 5 Knowledge-Based Complex Event Processing: In this chapter, we propose our basic concepts for knowledge based complex event processing. We describe how the fusion of domain background knowledge with the event data stream should be handled. We propose our main concepts for the fusion of event stream and background knowledge. We introduce three main ideas for processing methods which are then developed into subsequent chapters.
- Chapter 6 Knowledge Representation for Semantic CEP: We investigate in this chapter how ontologies about events, actions, actors, processes, situations, and other main concepts should be handled. We propose keeping these kinds of ontologies as upper-level ontologies, which then can be extended depending on the different requirements of use cases.
- Chapter 7 Representation of Complex Event Patterns: The representation of complex events based on the background knowledge is described in this chapter. We provide a detail description of the representation of events based on the combination of event operation algebra with the knowledge graph patterns.
- Chapter 8 Semantic Enrichment of Event Stream: One of the event processing approaches that we propose for knowledge-based event processing is the semantic enrichment of event streams. We provide an algorithm for finding an optimized plan for event enrichment and detection in order to optimize the caused load on the external knowledge bases for knowledge acquisition. Our aim is to find low-cost event detection plans while meeting user-specified latency expectations.

- Chapter 9 Semantic Enrichment of Complex Queries: Enrichment of event detection queries is another processing approaches for fusion of background knowledge with the event stream, so that the event processing agents can process the detection queries without accessing the knowledge base. We describe how event queries can be preprocessed and rewritten into simple event detection patterns that can be processed independently, without accessing external knowledge bases.
- Chapter 10 Event Processing on Sampled Event Stream: Sampling of the event stream is one other processing approach in which we want to reduce the throughput of raw event stream to improve real-time processing (latency and throughput of event processing). In some of the CEP use cases, it is applicable that we have false positives in event detection, but the complex events are detected at the right time. In Chapter 10 we introduce an approach for optimized sampling of event stream based on stream properties and the event detection pattern. Our approach can calibrate the raw stream at a specific rate while maximizing the number of detected complex events or calibrate the number of detected complex events while minimizing the throughput of raw events. We describe the assets and drawbacks of these approaches, and show for which kind of use cases these approaches can be suitable.
- Chapter 11 Conclusion and Outlook: Our research can go off in different research directions. In this chapter, we present our ideas for further extending of our research work in different directions. For example, we describe its combination with data mining, and uncertain event processing. In this final chapter we conclude the dissertation project and summarize its results.

1.7 Excluded Research Topics

In this dissertation we excluded some of the related topics. In this section, we list some of the research topics which we excluded in this dissertation to be able to focus on the main research questions. We are aware of the fact that the following subjects are highly relevant and real research problems.

1. **Uncertain Events and Noisy Event Stream:** We do not consider that input event stream as a noisy stream. This is the case when an event processing system has to work on real data, e.g., the event stream is procured by sensor devices or transported through an unreliable wireless network. In this research, we assume that our event stream is stable and events are captured as they happen. We assume that events are specific events and all of the communication channels are reliable and we do not process events over uncertain data [Was08]. Nevertheless, the combination of this topic with the knowledge-based complex event processing is described as one of the future research directions (see Chapter 11.5).
2. **Out-of-Order Event Data Arrival:** One of the problems an event stream might have is that event notification messages arrive at the processing system in an order that is out of their occurrence or capturing order. In this research, we assume that the events in an event stream are in a sequence of completely ordered event messages, and the order is the same as when it was captured by sensing devices while they are happening in the real world.

3. **Event Pattern Mining:** We do not address any kind of pattern mining on the event stream, we assume that complex event queries should be defined by the users of event processing system, and can be defined based on their event detection business interests. We do not address the problem of event pattern mining (this topic is also mentioned as a future work in Chapter 11.5).

1.8 Research Outcomes and Connecting Publications

Parts of this research have already been published at the following conferences and workshops [Tey09b, Tey09a, Tey09c, Tey10b, Tey10a, Tey11b, Tey11a, Tey12c, Tey12b, Tey12a, Tey14].

- Chapter 5 is partially published in [Tey09b, Tey09a, Tey09c, Tey10b].
- Chapter 6 is partially published in [Tey10a].
- The high level stock market monitoring from Chapter 4 is published as a demonstration paper in [Tey12c]
- Chapter 7 is partially published in [Tey12b, Tey12a]
- Chapter 8 is partially published in [Tey14].

Our plan-based event processing approach presented in Chapter 8 extends the research results from the previous work on plan-based event processing [Akd08, SM09]. These systems deal with planning the event acquisition to reduce network transmission costs. In our approach, we have to optimize the load and transmission costs of knowledge acquisition costs.

Part I

Background

Chapter 2

Background on Semantic Technologies

As specified in the problem description, our research topic is at the intersection of two research fields (also two separate research communities, distributed event-based systems (distributed systems, database technologies) and semantic web technologies (knowledge representation, logics, semantic web, pragmatic web, rule languages and rule-based systems)), event processing and semantic technologies.

In this chapter, we review the most relevant parts, methods and technologies of classical logic, knowledge representation, and semantic web technologies.

Contents

2.1 Introduction

The roots of knowledge representation can be found in classical logic. Classical logic was developed in ancient times[1] and has its roots in propositional logic and set theory. George Boole [Boo53] developed Boolean algebra[2] whose design was based on the value of variables as truth values true/false. Boolean algebra operators are conjunction, disjunction, and negation that can operate on variables with truth values. Based on the truth values are then the propositional calculus that enables a formal system of inference rules and axioms that allows the derivation of certain formulas. The language of propositional calculus consists of a set of primitive symbols (variables, like P, Q, R) and a set of operator symbols (operators like $\land, \lor, \rightarrow$) [Men87].

[1]http://plato.stanford.edu/entries/logic-ancient/
[2] http://plato.stanford.edu/entries/algebra-logic-tradition/

First-order logic (FOL, aka predicate logic) is built on top of propositional calculus and introduces the concept of quantified variables. It allows the expression of logical sentences with ”*for every*” $\forall$, the universal quantifier or the existential quantifier *”there exists”* $\exists$. For example it allows sentences like ”For every S, if S is a stock market, then S has a price”, $\forall s \in Stocks \exists p \in Prices : hasPrice(s, p)$.

Second-order logic and higher-order logic allows quantifiers over relations, variables and functions that allows the expression of more complex sentences than the first-order logic.

The following sections describe briefly the basic technologies of the semantic web that we used in our knowledge-based approach for the representation of external knowledge bases. We present an overview on the knowledge representation methods, description logic and rule languages.

2.2 Semantic Web Technologies

Tim Berners-Lee expresses his vision [BL01] for the Semantic Web as follows:

> *”The Semantic Web is not a separate Web but an extension of the current one, in which information is given well-defined meaning, better enabling computers and people to work in cooperation. ... To date, the Web has developed most rapidly as a medium of documents for people rather than for data and information that can be processed automatically. The Semantic Web aims to make up for this. Like the Internet, the Semantic Web will be as decentralized as possible. Such Web-like systems generate a lot of excitement at every level, from major corporation to individual user, and provide benefits that are hard or impossible to predict in advance.”*

Researcher have many different views [Pas04] about the nature of the Semantic Web. Two example of these views are:

- The machine-readable-data view :

  > *”The Semantic Web is a vision: the idea of having data on the Web defined and linked in a way that it can be used by machines not just for display purposes, but for automation, integration and reuse of data across various applications.” [Pas04]*

- The servant-of-humanity view:

  > *”The Semantic Web is a vision of the next-generation web, which enables web applications to automatically collect web documents from diverse sources, integrate and process information, and interoperate with other applications in order to execute sophisticated tasks for humans.” [Pas04]*

A stack of the Semantic Web technologies is provided by the so-called Semantic Web Layer Cake[3] shown in Figure 2.1. The Semantic Web Layer Cake[4] displays the various technologies and protocols which are known as Semantic Web Technologies. Each layer in the Semantic Web Layer Cake, uses the semantics and syntax of the layer below.

The Semantic Web Layer Cake has the following layers:

[3] Also known as Semantic Web Stack [Hor05].

[4] Steve Bratt. Emerging Web Technologies to Watch `http://www.w3.org/2006/Talks/1023-sb-W3CTechSemWeb/` retrieved February, 2014

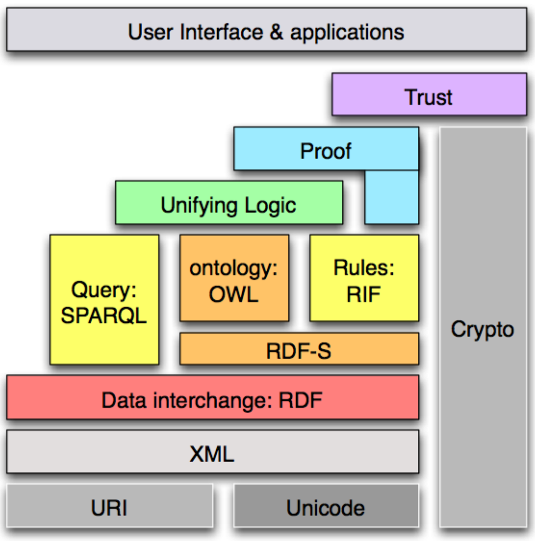


Figure 2.1: The Semantic Web Layer Cake

- **URI:** The Uniform Resource Identifier (URI)[5] is used to identify resources on the Internet.
- **XML, Namespace and XML Schema:** Extensible Markup Language (XML)[6] is a simple and flexible text format derived from SGML[7]. An XML namespace[8] is identified by a URI reference [RFC3986]. XML Schemas[9] express shared vocabularies.
- **RDF and RDF Schema:** The Resource Description Framework (RDF) and RDF Schema are described briefly more in Sections 2.3 and 2.3.1.
- **Ontology vocabulary:** Ontology vocabularies are described briefly in Section 2.5.
- **Logic:** The basis of the Semantic Web is in logic [BL98].
- **Proof and Trust:** The proof layer is intended to support deductive processing and proof validation. The usage of digital signatures in semantic Web can lead to trust [Ant04].
- **Rules:** Rules are described briefly in Section 2.8.

2.3 The Resource Description Framework

The Resource Description Framework (RDF)[10] is a recommendation of the W3C and was developed to add machine-readable metadata to Web-based resources. RDF is an established standard to describe and represent information about resources on the Web.

The W3C Semantic Web Activity Statement[11] specifies RDF as:

[5] http://www.w3.org/Addressing/ retrieved February, 2014
[6] http://www.w3.org/XML/ retrieved February, 2014
[7] http://www.w3.org/MarkUp/SGML/ retrieved February, 2014
[8] http://www.w3.org/TR/REC-xml-names/ retrieved February, 2014
[9] http://www.w3.org/XML/Schema retrieved February, 2014
[10] http://www.w3.org/RDF retrieved February, 2014
[11] http://www.w3.org/2001/sw/Activity retrieved February, 2014

"The Resource Description Framework (RDF) is a language designed to support the Semantic Web, in much the same way that HTML is the language that helped initiate the original Web. RDF is a framework for supporting resource description, or metadata (data about data), for the Web. RDF provides common structures that can be used for interoperable XML data exchange."

2.3.1 RDF Schema

RDF Schema (RDFS) – RDF's vocabulary description language[12] – is a recommendation from the W3C, and supplies mechanisms to depict groups of related resources and the relationships between them. RDFS vocabulary descriptions are written in RDF, which applies the terms defined in the W3C recommendation document. RDFS provides knowledge on the interpretation of RDF model statements [Fen07].

2.4 SPARQL Query Language

Simple Protocol And RDF Query Language (SPARQL)[13] is a W3C recommendation and a query language for RDF. According to the W3C recommendation for SPARQL:

"SPARQL can be used to express queries across diverse data sources, whether the data is stored natively as RDF or viewed as RDF via middleware. SPARQL contains capabilities for querying required and optional graph patterns along with their conjunctions and disjunctions. SPARQL also supports extensible value testing and constraining queries by source RDF graph. The results of SPARQL queries can be results sets or RDF graphs."

The SPARQL query is made up of a set of graph patterns called a basic graph pattern (BGP). The graph pattern is a fundamental building block of SPARQL query. It is like an RDF triple except each of the three positions, subject, predicate and object, can be a variable. SPARQL defines a network protocol for the exchange of queries [Wal07a].

2.5 Ontology

Ontology is a machine-interpretable conceptual model of an application domain and can be applied using applications for the reasoning of domain knowledge. Ontologies play a key role in the Semantic Web. There are various definitions of the term ontology in different application contexts and communities [Stu07], from philosophy to computer science.

The most common ontology definition in computer science literature [GP04] is given by Gruber [Gru93]:

"An ontology is an explicit specification of a conceptualization" [Gru93].

Studer et al. [Stu98] provides a more detailed ontology definition:

[12] http://www.w3.org/TR/rdf-schema/ retrieved April, 2014
[13] http://www.w3.org/TR/rdf-sparql-query/ retrieved April, 2014

"An ontology is a formal, explicit specification of a shared conceptualization. Conceptualization refers to an abstract model of some phenomenon in the world by having identified the relevant concepts of that phenomenon. Explicit means that the type of concepts used, and the constraints on their use are explicitly defined. Formal refers to the fact that the ontology should be machine-readable. Shared reflects the notion that an ontology captures consensual knowledge, that is, it is not private of some individual, but accepted by a group" [Stu98].

The main components of an ontology are concepts, relations, instances and axioms [Stu07]: *Concepts* describe and represent important categories in the domain of interest. *Relations* represent connections between the concepts of the domain. *Instances* are used to represent concrete objects or individuals in an ontology. *Axioms* represent sentences that are always true. They are useful for checking the consistency of the ontology itself and inferring new knowledge.

2.6 Description Logics

Knowledge representation has its root in the early development of artificial intelligent [vH07]. Description Logics (DLs) [Baa03, Rud11] is a knowledge representation formalism developed based on first order logic. It came out of the semantic networks [Min69] and frame-based systems [Min74]. DL focuses on the decidable fragment of the first-order-logic. DLs are a family of formalizations for the knowledge representation so that they are referred to as logics (plural form of logic).

The syntax of the description logic includes three main building blocks [Rud11]:

- A-Box (Assertions) like *"John Doe is a healthy human"*
  in DL syntax: $HealthyHuman(JohnDoe)$
- T-Box (Terminologies) like *"Every human is dead or live"*
  in DL syntax: $Human \sqsubseteq Dead \sqcup Live$
  or *"Healthy humans are not dead"* in DL syntax: $HealthyHuman \sqsubseteq \neg Dead$
- and RBox (roles) like *"If somebody buy something, he has to pay it"*
  in DL Syntax: $buy \sqsubseteq pay$.

The terminology of DL is different than the FOL. In DL concepts are similar to classes in FOL, roles are like properties/predicates in FOL and individuals are objects in FOL.

The different variation of description logics allows different expressivity. The $\mathcal{SROIQ}$ attribute language allows atomic negation, concept intersection, universal restrictions and limited existential quantification. A $\mathcal{SROIQ}$ knowledge base includes a A-Box, T-Box and R-Box.

The semantic of description logics is specifically based on the model theory and uses as its central notion interpretations (in DL annotated as $\mathcal{I}$). An interpretation can be seen as a potential *"reality"*. An interpretation $\mathcal{I}$ provides a nonempty set called domain or universe, and a function called interpretation function that is able to connect the elements (individuals, concept and roles).

2.7 The Web Ontology Language

The Web Ontology Language (OWL)[14] has been standardized by the W3C consortium. OWL 1 language is based on the description logic $\mathcal{SROIQ}^{(\mathcal{D})}$. OWL has three sub-languages **OWL Lite**, **OWL DL** and **OWL Full**. Each sub-language realizes different aspects of these requirements [W3C04]:

The complete language is OWL Full and the other two are restrictions of OWL Full. OWL Lite is OWL DL with more restrictions. Considering these restrictions, the OWL sub-languages are in the following inclusion-relationship:

$$\text{RDF(S)} \subset \text{OWL Lite} \subset \text{OWL DL} \subset \text{OWL Full}$$

where $\subset$ stands for both syntactic and semantic inclusion [Stu05].

OWL 2[15] is another Web ontology language and is based on the description logic $\mathcal{SHROIQ}^{(\mathcal{D})}$. OWL 2 adds new functionalities compare to the OWL1 which are: keys, property chains, richer datatypes, data ranges, qualified cardinality restrictions, asymmetric, reflexive, disjoint properties, and enhanced annotation capabilities. OWL 2 provides 3 sub-languages[16] which are OWL 2 RL, OWL 2 QL and OWL 2 EL.

2.8 Rules and Rule Languages

Over the last decades rule-based systems have been investigated in the scope of declarative programming languages and expert systems. The basic idea is that the user should concentrate on describing what they want to see as results and let the system decide how to do the task [Pas11].

Several types of web rule languages have been developed for the representation rules like semantic web rules or business rules. Web rules can be categorized into different rule types such as, derivation rules, integrity rules, transaction rules and reaction rules [Pas11]. The most relevant rule languages are the W3C Rule Interchange Format (RIF)[17] and the Rule Markup Language (RuleML)[18].

Two main types of inferencing on the rule-based systems are possible. The forward chaining rule system that starts inferencing based on the rules given by a data set and extracts new data until it achieve the optimal goal (the true value of the if clause). The backward chaining rule system starts inferencing from a goal and operates backward until it can generate data that support the initial goal.

2.9 Temporal Logic

Temporal logic[19] is a variant of modal logic for the expression of temporal information. Modal logic extends propositional and predicate logic to include operators expressing modality. Modals words are expressions of modalities - qualifiers of a statement. In temporal logic the truth value of propositions depends on the respective world, e.g., operators like “necessarily” and “possibly” denote a proposition is true.

[14] `http://www.w3.org/standards/techs/owl` retrieved April, 2014
[15] `http://www.w3.org/TR/owl2-overview` retrieved July, 2014
[16] `http://www.w3.org/TR/owl2-profiles/` retrieved July, 2014
[17] `http://www.w3.org/standards/techs/rif#w3c_all` retrieved April, 2014
[18] `http://ruleml.org` retrieved April, 2014
[19] `http://plato.stanford.edu/entries/logic-temporal/` retrieved April, 2014

The temporal logic of actions (TLA) [Lam94] is a logic for specifying and reasoning about concurrent systems. Temporal logic is used in computer science for formal specification [Lam94, Man92], formal verification (proof calculus) [Lam94, Man92], description of behavioral system requirements and specification checks (model checking).

Temporal Description Logic:
The extension of description logic to describe the temporal aspects is the temporal description logic [Art01]. Artale et al. [Art98] present an interval-based temporal languages for representing and reasoning about actions and plans. In their temporal language the actions are represented by describing what is true while the action itself is occurring, and plans are temporal relations based on actions and states.

2.10 Situation & Event Calculus

Situation calculus [McC69, Rei91] is a logic formalism for representation and reasoning about dynamical systems. Situation calculus is built based on the basic concepts of situations, actions and fluents [vH07]. Reiter [20] specifies a situation is a finite sequence of actions and it is a history of actions.

Briefly described, in situation calculus actions are things that can be performed by agents to change a dynamic world which moves from one situation to another situation. Fluents are functions that describe the effect of actions. For example $putdown(x)$ is an action that can put down a physical item x on a table and as the effect of this action the fluent $ontable(x)$ is true when the item x is on the table.

Event Calculus [Kow92, Kow86] is another temporal reasoning formalisms. In event calculus it is possible to specify the fluent values at a given time point when the actions and their effects take place. For example $HoldsAt(on(x, table), t)$ specifies that at time point t the item x is on the table. Predicate $HoldsAt$ says that the fluent holds at time t. Further event calculus predicates are $Initiates, Terminates, Happens, Clipped$ for describing initiating, terminating, happening and clipping.

Event calculus and situation calculus are found in event recognition systems [Pas06a, Pas06b, Pas08, Art09]. Several formalizations are proposed for complex event patterns based on a logical knowledge representation (KR) interval-based event/action algebra, namely the interval-based Event Calculus [Pas06a, Pas06b, Pas08].

2.11 Summary

In this chapter, we have briefly reviewed the relevant topics on knowledge representation and semantic web technologies. One further relevant topic is SLD-Resolution (**L**inear resolution for **D**efinite clauses with **S**election function) [Kow74] which is the basic rule for inferencing in logic programming and is the resolution method used in Prolog (the general purpose logic programming language).

The following chapter provides a brief overview over state-of-the-art approaches for event processing and other relevant topics, e.g., data stream processing.

[20] `http://www.ida.liu.se/ext/etai/rac/notes/1997/09/note.html` retrieved February, 2014

Chapter 3

Related Work on Event Processing

The chapter sets out the basic concepts of event processing. We go into the related work on event processing and describe different event processing approaches based on their data model and event processing models. We compare the models used in existing event processing approaches and describe existing optimization methods for reducing different event processing costs. Furthermore we discuss the related approach of data stream processing and describe their different data processing approaches. The different types of stream windowing and slicing forms are also discussed. Additionally, we review existing approaches for sampling and randomization of data streams and discuss the potential usage of these approaches in event processing.

Contents

3.1 Introduction

The history of event processing is more than two decades old. The event processing technologies were developed over the course of the history of computer science in different fields and research communities. Chakravarthy and Jiang [Cha09] identify three period of technology development for event processing, *“mid to late-eighties in the research area of active databases”*, *“nineties in area of active object-oriented databases”* and *“beyond 2000 in complex event processing.”*. The main work on formal semantics for event operations was developed in active databases and are extended in the following efforts.

Cugola et al. provide in [Mar11] a survey on the research area of complex event processing and data stream processing. They named this field *"Information Flow Processing" (IFP)* and survey IFP systems in their different software architecture, data model, rule model, and detection rule language. In another survey and requirements analysis by Schmidt et al. existing event processing methods are presented and analyzed. They categorize the state of the art approaches in event processing based on their event processing method into two man categories; *rule-based approaches* and *non-rule-based approaches* [Sch08]. Some of the event processing systems use non-deterministic finite state automata like Cayuga [Bre07] and SASE+[Dia07] or Esper [Ber07].

This chapter starts with the basics of event processing technology. We set out the main assumptions, definitions and terms used in this research area and survey briefly the main approaches for event processing. We describe some of the high impact research efforts. Furthermore we describe some of the relevant approaches for Data Stream Processing.

3.2 Event Processing

- **Definition of Event:** The Merriam-Webster dictionary[1] describes an event as an occurrence, *"something that happens or a noteworthy happening"*.

  In computer science literature different definitions for an event can be found. Gehani et al. define an event in [Geh92a] as *"a happening of interest. Events happen instantaneously at specific points in time"*. Chakravarthy et al. define event his work on SNOOP in [Cha93] as *"an instantaneous, atomic (happens completely or not at all) occurrence."*. Luckham [Luc01] defines an event as *"an object that is a record of an activity in a system." or "occurrences of significance in a system."* Mühl et al. [GM06] define event as *"anything that happens."* Real-world occurrences can be defined as events that happen over space and time [GM06, Man09b]. Etzion et al. [Etz10b] define the term event as *"an occurrence within a particular system or domain."*

  The Event Processing Technical Society (Event Processing Technical Society (http://www.ep-ts.com) (EPTS)) defines the term event in the glossary [DL11] for event processing as *"anything that happens, or is contemplated as happening."* We follow the same definition as specified in EPTS glossary [DL11] and define an event as a *concrete semantic object* containing data describing things that happens or is contemplated as happening. We especially stress that events are concrete semantic object which have different meanings based on the context of the happening. In the following chapters we set out the details of events as semantic objects.

- **Event Notification:** In computer science the notification about the happening of an event has more usage than the event itself. A notification notifies about the occurrence of an event. An event notification is also known as *event object, event message or event tuple*. The EPTS [DL11] defines event objects as an *"an object that represents, encodes, or records an event, generally for the purpose of computer processing."* An example of an event object is a stock tick message that reports a stock trade event. Or a tuple which represents the occurred event.

- **Event Attribute:** A component of the structure of an event. (Event property)

[1] http://www.merriam-webster.com/dictionary/event retrieved May, 2012

- **Event Type:** also known as event class, event definition, or event schema [DL11] *"A class of event objects."* An event type is a class of event objects (event class, event definition, event schema).
- **Raw Event:** Raw or "low-level" events can be seen as 'low-level' information chucks which flow directly from event producers and have to be processed by an event processing system. Raw event: An event object that records a real-world event: detection time, timestamp when the event is detected. (creation time)
- **Complex Event:** The EPTS Glossary [DL11] defines a complex event as follows: *"An event that summarizes, represents, or denotes a set of other events."*
- **Composite Event:** A derived, complex event that is created by combining base events using a specific set of event constructors like disjunctions, conjunctions, sequences. (aka compound event)
- **Event Stream:** The EPTS Glossary [DL11] defines event stream as *"a linearly ordered sequence of events."*
- **Event History:** Event history can be considered as a partially ordered set of event instances, in which the order of events are the time of occurrences.
- **Event Pattern:** An event pattern is a template that matches certain sets of events [Luc01]. Event filters are constructive views on event sources such as an event cloud or event stream, which might be materialized in an event database. A filter F is a stateless truth-valued function that is applied to an event instance sequence. An event instance n matches a filter F if and only if it satisfies all attribute filters of F [GM06]. A common model for an event instance is, e.g., a structure consisting of attribute/value pairs like {*(type, StockQuote), (name, "Lehman Brothers"), (price, 45)*}.

  An event is anything that happens, or is contemplated as happening [DL11]. Real-world occurrences can be defined as events that happen over space and time [Müh06, Man09b]. An event instance is a concrete semantic object containing data describing the event. Events can be a compound of other events that build the complex events. An event pattern is a template that matches certain sets of events [Luc01]. CEP is about the detection of complex events from a cloud of events which is partially temporal ordered by matching complex event patterns against event instance sequences. Streams of events should be processed in one or more event processors to detect the complex events based on their syntax, sequence and their senses and semantics. EPTS glossary [DL11] specifies that *"a complex event can be composed or derived from other events called its members."* Detection of complex events is required to be able to define and trigger reactions to the complex events.
- **Event Processing Agent:** Luckham [Luc01]defines an event processing agent as *"An event processing agent is an object that monitors an event execution to detect certain pattern of events."*

- **Complex Event Processing:** Events can be a compound of other events that build the complex events. Complex Event Processing (CEP) is about the detection of complex events from a cloud of events, which is partially temporal ordered by matching complex event patterns against event instance sequences. Streams of events should be processed in one or more event processors to detect complex events based on their syntax, sequence and their senses and semantics. Complex events can be composed or derived from other events called members. Detection of complex events is required to be able to define and trigger reactions to complex events.
- **Complex Situation:** A specific sequence of events. Are the situations the same as complex events?
- **Event channel:** A conduit in which events are transmitted from event sources (emitters) to event sinks (consumers). (event connection, event pathway, event topic)
- **Transaction time:** The timestamp when the event is processed. (arrival time)
- **Event Processing Engine:** A set of event processing agents and a set of event channels connecting them. (Event Processing Network (EPN), Event Processing Platform (EPP))

Throughout this dissertation, we will use a variety of terms like "Event", "Event Stream", "Event Cloud", "Event Processing Agent" and "Event Processing Network". In this thesis, we use the terminology provided by Event Processing Glossary of Event Processing Technical Society [DL11] for the research area of event processing. Further terms about knowledge representation are also used which are mostly known in research communities around knowledge representation and semantic web research area.

We describe here some of the most important terms, because they might have other meaning and definitions in other computer science research, e.g., the term *"agent"* which is not the same as the agent found in the multi-agent systems. Other terms are introduced in the respective chapters and where they are first used. Further acronyms used in this dissertation are listed in Appendix A.

3.3 History of Event Processing Research

The technologies of event processing have been developed in different research communities over the past two decade. The three main periods of event processing development as identified by Chakravarthy et al. [Cha09], are the late-eighties in active databases, nineties in object-oriented database, and beyond 2000 under the term *"complex event processing"*.

Table 3.1 shows a summary of the most influential research effort and commercial systems for complex event processing.

3.4 Event Specification in Active Databases

In the late-eighties there was great interest in active databases [Day88, McC89, Bee91, Loh91, Sto91]. Several specification mechanisms for event triggering and event expression have been proposed which built the fundamentals of today's event processing systems, and are used in various applications.

Date	**Description of Research or Commercial Activities**
1986	An Event/Trigger Mechanism in Design Databases [Dit86]
1988	HiPAC Project in Active Databases [Day88]
1988	The POSTGRES Rule Manager [Sto88]
Nineties: Active Object-Oriented Databases	
1992	ODE [Geh92a]
1992-93	COMPOSE [Geh92b, Geh92a, Geh93]
19994	Snoop [Cha94a, Cha94b]
1995	REACH [Buc95]
1997	Sentinel [Cha97, Cha95]
Post 1998, 2000	
1998	David Luckham and Brian Frasca from Standford publishes the Book ”Complex Event Processing in Distributed Systems”
2002	First version of ruleCore CEP Server created. `http://www.rulecore.com/`
2002	First DEBS Workshop `http://www.debs.org/DEBSConferences`
2003	Coral8 founded `http://www.sybase.com/` (Now superseded by Sybase CEP)
2003	StreamBase [2] company founded.
2004	SENACTIVE founded. `http://www.uc4.com/`
2004, Oct	First alpha release of Esper [Ber07] released.
2005, Apr	Progress Software acquires Apama `http://www.progress.com/`
2006, May	iSpheres backs out, eventually bought by Avaya. `http://www.avaya.com/`
2007	1st International DEBS Conference, June 20-22, Toronto, Canada
2007, July	BEA acquires a ESPER license and creates own fork known as BEA EventServer
2008, Jan	IBM acquires AptSoft
2008, March	ruleCore CEP Server 2008 released. `http://www.rulecore.com/`
2008, June	Event Processing Technical Society founded. [ept08]
2008, March	Sybase gets Coral8, to be used in Sybase RAP. `http://www.sybase.com/`
2009, March	Aleri acquires Coral8. (Sybase inc acquired Aleri inc.)
2010	Opher Etzion and Peter Niblett publish the book “Event Processing in Action”

Table 3.1: Short History of Event Processing Research. (partially adapted from EPTS)

The main idea behind active databases was to make databases capable of actively developing an expressive event specification language with well-defined semantics, algorithms for the detection of composite events, and an architecture for an event detector along with its implementation. The fundamental model underlying the work on active databases (also object databases) is the Event-Condition-Action(ECA) model first introduced in [McC89].

In the following we describe the composite event specification in active databases. We describe in more details the three main approaches ODE [Geh92a], Compose [Geh92b, Geh92a, Geh93] and Snoop [Cha94a, Cha94b]. At the end of this section, we list and describe briefly further related approaches.

One of the initial efforts on active databases is event specification in **ODE** [Geh92a]. Gehani et al. propose in [Geh92a] a language for specifying composite events by using event expressions, formed using event operators and events (primitive or composite). In ODE an event is a happening of interest and events can happen instantaneously at various time points. Events are in object-oriented databases related to actions that happen to objects and their states. Events are mostly local to a particular object or have specific scopes, like a database. An object type can be, for example a scope for an event, i.e., the schema modification of an object type. An event expression is the mapping of an event history to another event history which includes events satisfied by the given event expression. They present how a complex event can be specified by using event expressions.

The basic events in ODE [Geh92a] are *object state events* (e.g., immediately after an object is created), *method execution events* (e.g., immediately before a function is applied to an object), *time events* a local system time that can trigger timer events, and *transaction events* (immediately after a transaction begins).

Composite events in ODE [Geh92a] can be combined from basic events using logical operators and special event specification operators. Examples of event specification operators are *relative, prior, sequence, choose.* These operators are reused and extended in further specifications of composite events in active databases, like Snoop [Cha94a, Cha94b].

COMPOSE [Geh92b, Geh92a, Geh93] provides a formal semantics for event detection. COMPOSE's composite event processing language has its roots in the ODE System [Agr89]. The event history *(h)* in Ode is a finite set of event occurrences. Event occurrences are defined as a tuple of *(primitive event, event identifier)*. Event identifiers are used for ordering the event stream, e.g., a timestamp can be used as an identifier to order the events.

Event expression E is specified as a primitive or composite event which is a mapping from (domain) histories to (range) histories:

$$E : histories \rightarrow histories$$

$E[h]$ denotes the application of event expression E on the history of events h, so that we get as result a history in which E takes place. An event expression in Compose can be NULL, any primitive event a, or an expression formed using the operators. The operators are $\wedge$, *! (not), relative and relative* $+$.

The semantics of event expressions are defined as follows (E and F are two event expressions):

1. $E[null] = null, \forall E$, where $null$ is empty history.
2. $NULL[h] = null$
3. $a[h]$, where a is a primitive event and h is a history composed of event occurrences of the form (a, eid).
4. $(E \wedge F)[h] = h_1 \cap h_2$ where $h_1 = E[h]$ and $h_2 = F[h]$. This operation detects the simultaneous occurrences of E and F with the same identifies. The two composite events can have the same identifiers in contrary to primitive events.
5. $(\neg E)[h] = (h - E[h])$. NOT expression detects primitive event instances which are not of type E. This expression looks only for events which have occurred but are NOT of type E. This is not equivalent to the detection for the absence of E.

6. $relative(E,F)[h]$ The *"relative"* operation captures all event occurrences in h to which F is satisfied, and the history starts immediately with the event occurrence of type E. Formally $relative(E,F)[h]$ is specified as follows:
   Let $E^i[h]$ be the i^{th} event occurrence in $E[h]$ and let h_i be obtained from h by deleting all event occurrences whose *eids* (event ids) are less than or equal to the *eid* of $E^i[h]$.

$$relative(E,F)[h] = \bigcup_i F[h_i]$$

   where i ranges from 1 to the number of events in $E[h]$.

7. $relative+(E)[h] = \bigcup_{i=1}^{\infty}(E)[h]$ where $relative^1(E) = E$ and $relative^i(E) = relative(relative^{i-1}(E), E)$.

   Expression $relative+(E)[h]$ is the detection of a pattern with unlimited length over an event history.

Gehani et a. [Geh92b] claim that this event expression language with operators $\wedge$, *!, relative, and relative+* have the same expressive power as regular expressions [3], and reducing it will make the expressive power less than that of regular expressions. They provide further useful operators for the specification of composite events like $\vee$, *prior, sequence, first, — pipe,* $(< n > E)$ n^{th} *occurrence of E, every, firstAfter, before(E), happend(E), prefix(E).*

Chakravarthy et al. provide an operation semantic for *Snoop* [Cha94a, Cha94b]. This operational semantic is built based on the event specification operators defined in **ODE** [Geh92a]. Snoop[4] provides an event specification language along with the semantics of composite events over a global event-history In Snoop event E (primitive or composite) is a function of the time domain onto the Boolean values, True and False.

$$E : T \rightarrow \{True, False\}$$

If an event of type E happens at time point t, then the function is True, otherwise it is false.

The precise semantics of composite event detection are specified by Snoop event operators as follows:

1. **OR ($\bigtriangledown$) Operator:** $(E_1 \bigtriangledown E_2)(t) = E_1(t) \vee E_2(t)$

   The OR ($\bigtriangledown$) operation matches the events if at least one of E_1 or E_2 can be matched.

2. **AND ($\triangle$) Operator:** $(E_1 \triangle E_2)(t) = (\exists t^1)(E_1(t^1) \wedge E_2(t)) \vee (E_2(t^1) \wedge E_1(t)) \quad t^1 \leq t)$ The Snoop $\triangle$ (similar to AND) operation matches the event when an instance of E_1 occurs and an instance of E_2 has already occurred at an earlier or the same time point, or vice versa (E_1 occurred before E_2).

3. **ANY Operator:** The ANY operator matches, if exactly m match of events happens out of n events in time, ignoring the relative order of their occurrence.

$$ANY(m, E_1, E_2, \ldots, E_n)(t) = \exists t^1 \exists t^2 \ldots \exists t^{m-1}$$
$$(E_i(t^1) \wedge E_j(t^2) \wedge \ldots \wedge E_k(t^{m-1})) \wedge (t^1 \leq t^2 \ldots \leq t^{m-1} \leq t) \ \wedge$$
$$(1 \leq i \ldots k \leq p) \wedge (i \neq j \neq \ldots \neq k \neq p) \wedge m \leq n$$

[3]Regular expressions are widely used for specifying sequences.

[4]We describe Snoop in more details because Snoop is one of the high impact research efforts in event processing field.

4. **SEQUENCE Operator:** $(E_1; E_2)(t) = ((\exists t^1)(E_1(t^1) \wedge E_2(t)) \wedge t^1 \leq t)$
The sequence operator matches when E_2 occurs and E_1 has already occurred at a time before E_2.

5. **Aperiodic Operators** (A, A^*)**:** The aperiodic operation of Snoop allows the expression of an aperiodic event in a time interval marked by two events. Snoop provides two different variation of aperiodic operator, the non-cumulative and cumulative operation.

   The A operator (aperiodic non-cumulative) is matched each time E_2 occurs between E_1 and E_3. The $\sim$ symbol means on all occurrences of E_3 (aka every occurrence of E_3).

$$\begin{aligned} A(E_1, E_2, E_3)(t) = (E_1(t^1) \wedge \sim E_3(t^2) \wedge E_2(t)) \\ \wedge \ (t^1 < t^2 \leq t) \quad \vee \quad (t^1 \leq t^2 < t) \end{aligned}$$

   The A^* operator is the aperiodic cumulative operator. A^* signals only once within the given interval of two marker events (E_1 *and* E_3).

$$A^*(E_1, E_2, E_3)(t) = (E_1(t^1) \wedge E_3(t)) \quad \wedge \quad (t^1 < t)$$

   The operation accumulates the zero or more occurrences of E_2 between the E_1 and E_2. The operation is done and closes with the occurrence of E_3 and not with the happening of E_2.

6. **Periodic Operators (**P**,** P^***):** The period operator $P(E_1, [T], E_3)(t)$, E_1 and E_3 are two events, T is a constant amount of time. The operation detects all occurrences of E_1 to E_3 in constant time T. Formally defined as:

$$\begin{aligned} P(E_1, [T], E_3)(t) \ = \ (E_1(t^1) \wedge \sim E_3(t^2)) \ \wedge \\ (t^1 < t^2 \leq t) \ \wedge \\ t^1 + i * T = t \ \ for \ \ 0 < i < t \end{aligned}$$

   The cumulative variation of periodic operator accumulates times of occurrences of periodic events, formally:

$$\begin{aligned} P^*(E_1, [T], E_3)(t) \ = \ (E_1(t^1) \wedge \sim E_3(t)) \ \wedge \\ \wedge \ t^1 + T \leq t \end{aligned}$$

7. **Not (**$\neg$**) Operator:** The not operator detects the non-occurrence of an event. Not operation $\neg(E_2)[E_1, E_3](t)$ denotes the non-occurrence of event E_2 in the closed interval formed by E_1 and E_3. Formally defined in Snoop:

$$\begin{aligned} \neg(E_2)[E_1, E_3](t) = (E_1(t^1) \wedge \sim E_2(t^2) \wedge E_3(t)) \\ \wedge t^2 \leq t^2 \leq t \end{aligned}$$

Snoop also introduces the concept of parameter contexts which influence the detection behavior of snoop operators. For the detection of a complex event multiple matches might be available. Based on the semantic context of operators different matches of primitive events are available, e.g., for the event history (a b b) during the matching of (A;B) pattern, the complex event might be matched once or twice depending on the semantics of event detection system.

Snoop defines different contexts for their operators, *unrestricted, recent, chronicle, continuous, and cumulative* which are specified in [Cha94b]. These context can change the behavior of event processing. Unrestricted context is the normal case of event detection operations and might produce a lot of complex events occurrences which might not all be useful for the applications.

The details of event processing context are specified in [Cha94b]. We briefly review these contexts, because they are introduced first in Snoop and have effects on event detection behaviors.

In *recent context* only the most recent occurrence of the initiator of the composite event is considered and will be used for event detection, while all other non-initiator event instances will be deleted. The recent context is useful for applications in which high throughput of raw events should be used and multiple occurrences of the same event type do not affect the event detection pattern.

In *chronicle context* an occurrence of an initiator is paired with a terminator event instance and they build unique couples, e.g., the oldest initiator with the oldest terminator. This is useful in applications where different occurrences of types of events and their correspondence needs to be matched, for example, detection of events between aborts, rollbacks, and other transaction operations in a database.

In *continuous context* each initiator of composite event starts the detection of the event, the incoming terminators cause the detection of one or more composite event of the same type. This kind of context is interesting for trend analysis and forecasting application in which a moving window specifies the data for event detection.

In *cumulative context* all occurrences of the incoming event of the same type are accumulated until the composite event is detected. This context has useful applications in which multiple occurrences of the same event type needs to be grouped and used in a meaningful way when the event occurs.

SnoopIB [Ada06] is an extension of Snoop to interval-based event specification and detection for active databases.

Further Approaches in Active Databases

There are further research efforts done in the area of active databases. We list and describe briefly some of the related approaches:

ADAM is a rule management in object-oriented databases developed by Diaz et al. [Dia91, Pat91]. The responses of the active database system are declaratively expressed using ECA rules. ADAM provides an insight into rules in an object-oriented context while stressing Uniformity. Uniformity means in ADAM the consideration of rules as “first-class” objects that are described using their attributes and methods. Rule management operations are interpreted and implemented as methods which apply all of the advantages of the object-oriented paradigm.

HiPAC(High Performance ACtive database system) [Day88](also the Project name) is an object-oriented DBMS which tries to combine active database management with time constrains as to provide contingency plans. Contingency plans are alternate actions (reactions) executed, when the system detects that it cannot complete a task in time.

HiPac is based on two main models: knowledge model and execution model. The knowledge model builds the declarative semantics of rule. These rules can express integrity constraints, access constraints and alerts. HiPAC as an active database system provides five main functional components: object manager, transaction manager, event detectors, rule manager, and condition evaluator.

ACOOD (ACtive Object Oriented Database system) [Ber92] is an active database system based on the object-oriented database whose objective is to support efficient interaction between the active DBMS and applications. Like in HiPac the rules in ACOOD are in the form of ECA rules.

SAMOS (Swiss Active Macheanism-Based Object-Oriented Database System) [Gat92, Gat94]. In SAMOS events can be divided into two main categories: primitive events corresponding to elementary occurrences, and composite events. SAMOS provides operators like sequence operator to specify complex events.

REACH [Buc95] is an object-oriented database management system (OODBMS). It includes a ECA rule manager which is accessed during event detection. The event composition semantics are defined relative to the database transaction boundaries. The composite event specification is the same as Snoop. REACH is one of the first demonstrators of an OODBMS.

Sentinel [Cha97, Cha95] is a reactive DBMS that provides an event-based rule which can be used for some database functionalities like integrity enforcement, view materialization, management of index structures and applicability of compiled query plane when access methods change. Rules are ECA rules and consists of: an event expression, one or more conditions, an action, and a set of attributes. Sentinel uses event detection graphs (EDG) to represent an event expression. Complex events are conjunctions, disjunctions or sequences of primitive events.

3.5 Event Processing Methods

Event processing methods are approaches used to detect complex events, specified by different complex event expressions, from raw and primitive simple events. Different approaches are developed for event detection. A survey and requirements analysis about the event processing methods is provided in [Sch08].

Existing methods for event processing can be categorized into two main categories, rule-based approaches and non-rule-based approaches (aka logic-based or non-logic-based approaches). Some of the non-rule-based event processing approaches are based on formalizations, such as Finite State Automata [Geh92b], Event-Graph[Cha94b, Pat99] and Petri Nets [Gat94].

These event processing methods are implemented in different research as research prototypes or in a company's commercial products. In the following we briefly review the event processing methods and the related implementations.

3.5.1 Finite-State Machine

Finite-State machine (FSM) (aka finite state automata (singular: automaton) or just finite automata) provide a simple computational model. A state machine is a mathematical model of computation designed to model software programs and sequential logic programs. A state machine can be in one of the finite number of states and at a specific time point it is in only one of these states. The state can be changed by triggering an event or when other conditions are fulfilled.

A Turing Machine [Tur48] is defined as a finite-state controller with a movable read/write head on an unbounded storage tape. When the head movement is restricted to move only in one direction, we get the general case of a finite-state machine. The reads of sequence of symbols is the input of the state machine and the writes of symbols are the output. Since the movement of the head over the sequence of symbols (tape) is strictly one-way, it can be referred to the input sequence read and the output sequence functionalities. A machine of this kind is called a transducer, or Mealy machine, after George H. Mealy (1965) [Mea55], because it maps input sequences to output sequences.

In event processing state machines can be used for event detection, because the raw event stream can be considered as input sequence and output complex events can be the writes of symbols. State machines are used in event processing in approaches from active data bases like COMPOSE [Geh92b], ODE [Geh92a], SAMOS [Gat92, Gat94], ADAM [Dia91, Pat91], ACOOD [Ber92]. In the following we briefly described recent event processing approaches (or CEP systems) that use the finite state machines:

Other event processing systems are *SASE* [Gyl06] and *SASE*+ [Dia07] that are on non-deterministic finite automaton (Non-Deterministic Finite Automatons (NFAs)). SASE+ provides some of the typical event processing operators like *SEQUENCE, NEGATION* and sliding window operators. The event processing language of SASE+ provides Kleene closure (aka Kleene star or Kleene operator) over event streams that find multiple applications in RFID data streams.

One further automaton-based system is *Cayuga* [Bre07] which is a research project at Cornell University[5]. Cayuga provides a query language for the expression of complex event patterns called Cayuga Algebra. It also supports some special performance optimization like indexing and garbage collector.

Esper[6] [Ber08] is an event processing engine that uses finite state automata for event detection. Esper provides an event query language which has operators similar to CQL (Continuous Query Language) `SELECT`, `FROM` and `WHERE`, and supports correlations and SQL-like queries over event streams (the Esper event processing language is a scripting language and supports some operators from CQL).

Esper also provides operators for the specification of event detection patterns and special operators to define the event stream consumption policy like "Every" operators that specify precisely how the event stream should be matched to the pattern. Event detection conditions can be specified over *sliding time windows* and be used to trigger a reaction based on them. These actions are implemented in the Java programming language as Esper itself is written in `Java`.

[5]Cayuga `http://www.cs.cornell.edu/bigreddata/cayuga` retrieved May, 2012

[6]EsperTech `http://www.espertech.com/products/esper.php` Esper provides an open source version of the CEP engine. Retrieved May, 2012

3.5.2 Graph-Based Approaches

Chakravarthy et al. propose the use of the Event Detection Graph (EDG) [Cha94b, Pat99] for the detection of composite events. In this approach the event detection expression (complex event query) is build up as a graph pattern like a tree structure, so that different event types are defined as the leaves of a tree and nodes are the event operation algebras like AND, OR, SEQ, NOT. The events are then streamed into this structure and in each node one of the rules are executed on the event stream.

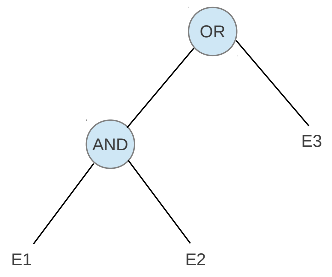


Figure 3.1: Example of Sentinel Event Detection Graph

In **Sentinel** [Cha97, Cha95] Chakravarthy et al. propose event graphs for the expression of composite events. Sentinel [Cha97, Cha95] uses an event graph or event detection graph (EDG) to represent an event expression in contrast to other approaches such as extended finite state automata used by Compose [Geh92b, Geh92a, Geh93]. Figure 3.1 shows an example of an event detection graph in Sentinel.

Gatziu and Dittrich [Gat94, Gat92] propose the use of Coloured Petri Nets [Jen96] in the SAMOS system for the detection of composite events.

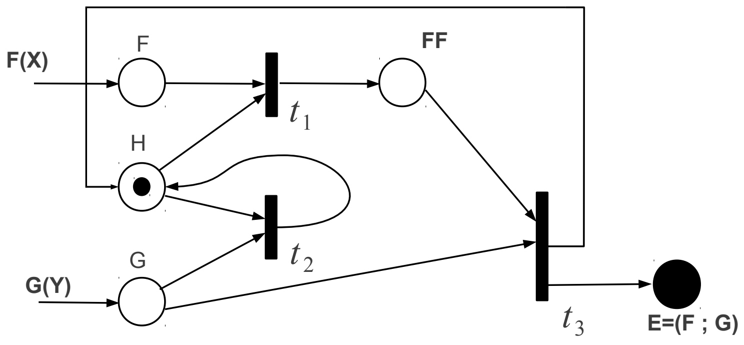


Figure 3.2: Example of a Petri Net Pattern for Sequence Operation in SAMOS (figure adopted from [Mot95])

3.5.3 Rule-Based Approaches

The most common approaches for event processing (like industrial event processing products) use a rule engine for the processing of events without permanent storage of events, the storage of historical event data is only optional for other purposes. The event data stream can be easily moved throughout the system without any necessary storage. Rule-based event processing engines can process events in real time, because they can keep the complete rule set (including facts) in the main processing memory. But these approaches cannot achieve high scalability or high performance, when they have to process huge amount of domain background knowledge (similar to static reference data) and use the available knowledge for event detection. The main problem here is that they have to keep the whole knowledge base in the main processing memory, and this will be impossible when the background knowledge is huge, e.g., all of the background knowledge about the companies traded on the stock exchange market worldwide.

One of the logic-based approaches is introduced in [Pas06a] which proposes a homogeneous reaction rule language for complex event processing. It is a combinatorial approach of event and action processing, formalization of reaction rules in combination with other rule types, such as derivation rules, integrity constraints and transactional knowledge.

Several complex event processing systems have been proposed and developed [Mar11]. Existing methods for event processing can be categorized into two main categories, rule-based approaches and non-rule-based approaches [Sch08]. One of the processing models for CEP are non-deterministic finite state automata, which are used in systems such as Cayuga [Bre07, Dem06] and SASE+ [Dia07], or ESPER [7]. One of the rule-based approaches is introduced in [Pas10] which proposes a homogeneous reaction rule language for complex event processing. It is a combinatorial approach of event and action processing, formalization of reaction rules in combination with other rule types such as derivation rules, integrity constraints, and transactional knowledge. Also several event processing languages have been proposed such as Snoop [Cha94a, Agr08], Cayuga Event Language [Bre07, Dem06] , SASE [Gyl06], XChangeEQ [Bry07].

Some of the commercial CEP products are TIBCO BusinessEvents [8], Microsoft StreamInsight[9], Oracle CEP[10] and Sybase CEP[11].

Some of these CEP systems can integrate and access external static or reference data sources. But these systems do not provide any inferencing on external knowledge bases and do not consider reasoning on relationships of events to other non-event concepts.

Prova[12] [Koz06] is a rule language and a rule engine. Prova (Prolog + Java) provides an open source rule language whose design is based on reactive messaging, combination of imperative, declarative and functional programming. Prova implements SLD-Algorithm for backward reasoning in java virtual machine. Prova is implemented in Java programming language.

[7] http://esper.codehaus.org retrieved May, 2012
[8] http://www.tibco.com/ retrieved May, 2012
[9] http://www.microsoft.com/en-us/server-cloud/solutions/business-intelligence/ retrieved May, 2012
[10] http://www.oracle.com retrieved May, 2012
[11] http://www.sybase.de retrieved May, 2012
[12] Prova http://www.prova.ws/

One of the important design principles in Prova is reactive messaging that allows the organization of several Prova rule processing engines into a network of communicating agents. A Prova agent is a rulebase that is able to send messages to other Prova agents by using primitive message passing primitives. The reactive messaging functionality in Prova makes it possible to build workflows based on communicating Prova agents. In both sending and receiving message primitives, 5 different parameters are sent between the agents. The parameters are: a conversation ID of the message (XID); name of message passing protocol (Protocol); Destination (on sending) or Sender (on receiving); the message type broadly characterizing the meaning of the message (Performative); and the message itself, a list containing the actual content of the message (Payload).

The event processing functionality in Prova uses reactive messaging and rule-based workflows. An event processing graph can be mapped to a reactive message passing workflow into Prova. Prova uses Metadata attributes like (AND, OR) to build the workflows and thus event algebra operations.

One of the recent rule-based systems is ETALIS [Ani11b]. ETALIS is a rule-based stream reasoning and complex event processing (CEP). ETALIS is implemented in Prolog and uses the Prolog-inference engine for event processing. ETALIS provides two event processing languages: ETALIS Language for Events *(ELE)* and *EP-SPARQL*. ELE provides features like classic event operators and count-based sliding windows. EP-SPARQL [Ani11a] is a language for complex events and stream reasoning. The formal semantics of EP-SPARQL is along the same lines as SPARQL [Pru08]. EP-SPARQL can be used in ETALIS for reasoning on RDF triple stream (event stream can be mapped to RDF stream).

A major distinction between ETALIS and Prova is that ETALIS is a meta-program implemented on top of a Prolog system with only one global KB in which every piece of knowledge, such as incoming events is globally applied, whereas Prova allows for local modularization of the KB and local event processing states within complex event computations and event message based conversations. This leads to a branching logic with local state transitions as it is common, e.g., in workflow systems and distributed parallel processing.

3.5.4 RETE Algorithm in Event Processing

The RETE Algorithm provided by Forgy [For79, For90] is a logical matching algorithm for matching data tuples (or facts) to the rules in a pattern-matching system. The RETE algorithm is based on the forward changing and inferencing of facts and rules.

RETE algorithm builds directed acyclic graphs as a high-level representation of the given rule sets, which are generated in run-time and includes objects such as nodes of the network. All of the operations on data tuples like relational query processor, performing projections, selections and joins are executed on the network of objects.

In RETE, facts are loaded into the memory, and the rule engine creates Working Memory Elements (WMEs) for each of the facts. Each WME includes a set of n-tuples. The Rete network is divided into two sub-networks, *Alpha and Beta* networks. The left-side of the network is called Alpha network and includes the network part that is responsible for selecting individual WMEs based on conditional matching to WME attributes. Conditional matching may include several tests within the network for the testing of several attributes. The right side of the network is called Beta network and performs the join operations on WMEs. The join operations are down on the interim results from two other nodes and join result is stored in a beta memory node. The results from the join nodes are processed in the final stages by terminal nodes, so that they produce the final agenda of the rule set given to the rule engine.

The RETE algorithm have been used as an event processing engine [Wal07b], because the inserted facts can be considered as event messages that arrive into the system as data stream, and the event detection pattern is the given rule set. RETE has also been used in commercial event processing products like TIBCO Business Events [13] and Drools Fusion[14].

Miranker [Mir87] describes the advantages and disadvantages of the RETE algorithm. The main advantage of RETE is that the large amount of interim results and states stored in the memory nodes minimizes the number of comparisons of two WMEs and the stored similar results are reused and shared with other WME tests. In the use of sharing the structural similarity in rules, RETE can speed up the rule matching process.

The drawbacks of the RETE algorithm are twofold: the primary disadvantage of RETE is WMEs updates (e.g., removal of one of the WMEs) causes a restart of the whole calculation and repetition of the entire sequence process upon its addition to the network. The second disadvantage is the storage of states in memory nodes which is highly memory intensive and may be combinatorial explosive. Thus sharing network structure might not be realizable in a parallel environment due to communication costs. The performance of the original RETE algorithm is improved by further optimization approaches like RETE II[15] and RETE-NT[16].

3.5.5 Storage-based Event Processing

The basic and more naive approach might be storing incoming event data on a database and steadily querying and pulling the database. Events can be processed first after their storage on a database. The main disadvantage of this approach is that processing is possible only after storage and the database is pulled with each new incoming event. This approach can work for use cases which do not have high event throughput and huge amount of background knowledge to process. The advantage of this approach is that a complete reasoning on the whole knowledge inventory is possible. Scalability and real time processing are the problems of this approach which makes it impossible to use for time-sensitive use cases like algorithmic trading, or fraud-detection systems. The usage of distributed databases can improve scalability but it can affect real-time processing latency.

[13] http://www.tibco.com/products/event-processing/complex-event-processing/businessevents/default.jsp

[14] http://www.jboss.org/drools/drools-fusion.html

[15] http://www.pst.com/reteII.html retrieved August, 2014

[16] http://www.infoworld.com/t/business-rule-management-systems/worlds-fastest-rules-engine-822 retrieved August, 2014

Glombiewski et al. [Glo13] show that a standard database can be used for event processing by using data access technologies like JDBC[17]. Their experiments show that such event processing systems can handle small and medium-size event processing workloads.

In-Memory Databases

In recent years, emerging technologies in computer hardware technology make it possible for main memory hardware to be cheaper than ever before [Sto07]. Chapranow et al. [Sch13] describe blade servers as providing roughly 500 Gbytes of main memory.

The idea behind in-memory databases is to keep the complete database in the main memory so that DBMS can gain better performance than conventional disk-based database techniques [Sch13, Sto07, Kal08, Wil09].

Many in-memory databases prototypes or commercial systems have been developed like Hyper [Kem12], MonetDB [Man09a], H-Store [Sto07], HANA [Wil09].

3.6 Data Stream Processing

Also several data stream processing systems have been proposed so far like Telegraph [Cha03] and Stream [Gro03]. Data stream processing systems aim at handling continuous database queries over high throughput data streams. These systems are similar to event processing systems and have similar properties [Cha09].

3.6.1 Data Stream Entropy

Entropy is a useful measurement in data stream processing, it shows the diversity and randomness of the data objects coming over the data stream [Bhu06, Lal06]. Let us assume a data stream of n different data items. We donate the frequency of an item i of n items by m_i and the total number of items in the stream by: $m = \sum_{i=1}^{n} m_i$

The number of distinct items actually present in the stream is annotated by n_0, since it is possible that not all n items actually appear in the stream. Let's consider the following example, a stream drawn from a set of n=4 different possible objects A, B, C, D (or event types in this thesis). A stream of this object set is S=(A, A, B, B, C, A, B, A, C). For this stream, the total number of items is $m = 4 + 3 + 2 = 9$, with the number of distinct items $n_0 = 3$. The entropy, frequency distribution of the stream is expressed by:

$H \equiv -\sum_{i=1}^{n} \frac{m_i}{m} log(\frac{m_i}{m})$

For the above example stream S, the entropy is

$H_S = -(4/9)log(4/9) - (3/9)log(3/9) - (2/9)log(2/9) = 1.53$

The entropy of a stream attains its minimum value of zero when all the stream items are the same, and it achieves its maximum value of $logm$ when all of the stream items are distinct. [Bhu06]. An entropy value close to $logm$ indicates a randomly distributed stream while a low entropy value indicates the existence of patterns in the data stream. The monitoring of data stream entropy have been used to detect anomalies [Gu05, Wag05, Xu05].

[17] `http://www.oracle.com/technetwork/java/javase/jdbc/index.html` JDBC is a Java-based data access technology (Java Standard Edition platform)

3.6.2 Data Stream Processing Systems

In the following, we review some of the most relevant data stream processing system. *The Stanford Stream Data Manager (STREAM)* [Gro03] is a data stream management system (Data Stream Management System (DSMS)) that uses a declarative rule language called continuous query language (Continuous Query Language (CQL)). CQL is an extension of Simple Query Language (SQL) for querying streams over sliding windows which are either time- or tuple-based data windows.

The semantics of CQL is composed of three main parts:

1. A relational query language that includes relation-to-relation operators.
2. A window specification language that can specify data windows using stream-to-relation operators.
3. A set of relation-to-stream operators for the specification of interactions from the relations to the stream.

Kramer et al. [Kra04] propose an infrastructure called *PIPES* for providing flexible and extensible DSMS building blocks. *PIPES* covers the operational semantics and functionality of the CQL. The language provides options for query optimization which is based on approaches adopted from the temporal database community. Query optimizations enables applications of temporal transformation rules within the context of streams [Kra09].

Chandrasekaran et al. [Cha03] propose another DSMS called *TelegraphCQ* which uses a continuous query language based on stream-only approach. *TelegraphCQ* includes entities named *Eddies* that are used for routing tuples through a network of query modules. Continuous queries can be specified over sliding time-based windows. Telegraph is implemented based on the PostgreSQL[18] database system and modifies it for the processing of streaming data.

Gigascope is another DataBase Management System provided by Cranor et al. [Cra03]. *Gigascope* is specially developed for network applications like traffic analysis, intrusion detection, router configuration analysis, network research and network monitoring. *Gigascope* uses a query language called GSQL which is a pure stream query language following the SQL syntax and it supports `join, selection, aggregation` and an additional `stream merge` operator.

One further DataBase Management System is *Aurora* [Aba03] whose aim is to process streams of data coming from various sources. Aurora provides a stream query algebra called *SQuAl* that includes seven primitive operations, e.g., `select, filter` and `aggregate`.

3.7 Publish/Subscribe Model

Publish-Subscribe is a communication paradigm in which users can express their interest in specific messages in the form of patterns (called subscriptions) and some other systems (agents) can publish their messages so that they can be routed to interested users. Communication parties are called, publishers and subscribers. Messages are mediated by a network of brokers who can filter the messages and route them to other brokers, and thereby to final subscribers. The messages in the Publish/Subscribe model can be considered the same as event notification messages.

[18] `http://www.postgresql.org/` retrieved Aprilt, 2010

Publish/Subscribe systems are categorized into two common forms: topic-based and content-based systems. In topic-based publish/subscribe systems, messages are published and filtered based on ”topics”. Communication channels are tagged with topic names (keywords). Publishers of the messages have to specify one to several topic names for their messages. Subscribers can subscribe to the topics given and will receive all of the messages related to this topic. The topics are broadcasted through the network. In content-based publish/subscribe systems, the subscriber is able to define constraints on the content of the messages, and inside the broker network published content is filtered by brokers and delivered to subscribers.

The Publish/Subscribe model is implemented in several research and commercial systems like Padres [Fid05], Rebeca [Par10], Hermes [Pie03a] Gryphon [Str98], Scribe [Row01], Bayeux [Zhu01], Siena [Car01].

A survey of publish subscribe systems is provided by Liu et al. [Liu03], and the different variations of these systems are analyzed by Eugster et al. in [Eug03].

3.8 Comparison of Languages and Event Processing Methods

Voisard et.al. [Voi11] introduce a framework to classify event-based systems (ESBs) called ARCHITECT. The ARCHITECT framework provides concepts to investigate ESB systems based on their individual architectural layers and uses abstraction hierarchies without relying on technical details. These layers are:

1. *Language layer:* For the specification of event queries.
2. *Execution layer:* For the execution of event queries.
3. *Communication layer:* For exchanging events between system components.
4. *Capturing layer:* For capturing events from event sources.

Each layer has the same set of concepts including: *data*, *operations*, and *key actions*:

- *Data* describes entities in a layer. It shows event objects in the specific layer.
- *Operations* identifies the specific operators used by the layer.
- *Key actions* describe actions in layers w.r.t. the operational functionality of the layer. For example the key action in a language layer is the definition of a declarative language as event detection queries.

Most existing systems for complex event processing provide the language and execution layer, while some of them also provide a communication and capturing layer.

A large number of different languages with different styles have been proposed so far. Etizon et al. in [Bry09][19] categorize the styles of existing event processing languages in the following main categorizes: *inference Rules, ECA Rules, agent oriented style, SQL extensions, state oriented and imperative/script-based languages*. Some examples of these event description languages are listed in Table 3.2. Each of these event processing languages provide a somewhat different operational semantics in comparison to others.

Most event processing systems use rule engines for event processing (running backward or forward chaining). Different event processing approaches are listed in Table 3.3 with examples of systems that apply these approaches.

[19]Research results from the EPTS languages analysis working group

Language Style	**Examples**
Inference Rules	Prova, *XChange*^{EQ}, ETALIS [Ani11b], TIBCO
ECA Rules	Prova, *XChange*^{EQ}, AMIT [Adi04], WBE, RuleCore
Agent Oriented	Prova, Starview, EventZero, Spade [Hir09], Agent Logic,
SQL extentions	Aleri, Coral8, StreamBase, Oracle
SQL extentions	Esper, Aleri, Coral8, StreamBase, Oracle
State Oriented	Oracle
Imperative/Script-based	Esper, Apama, Netcool Implact

Table 3.2: Categories of Event Processing Language Styles and Example Languages

Event Processing Approach	**Examples**
Finite-State Machine	Esper [Ber08], *Cayuga* [Bre07]
Rule-Based	Prova, ETALIS, TIBCO StreamBase, AMIT [Adi04]
Graph-Based	Sentinel [Cha97, Cha95], SAMOS [Gat94]
Agent-Oriented/Workflow-Based	Prova

Table 3.3: Categories of Event Processing Approaches and Example Systems

Further relevant event processing systems are AMIT [Adi04] , Spade [Hir09] TIBCO StreamBase[20], Apama[21]. A survey of approaches for RDF stream reasoning is provided by Margara et al. [Mar14b].

3.9 Knowledge-based Event Processing

The research area of complex event processing is an emerging research area and has been increasingly gaining attention in the past years. Some of the recent efforts in the area of event processing is focused on the usage of ontological background knowledge to improve event processing expressiveness.

In the following, we briefly describe these approaches including approaches for stream reasoning [Val09] and show the differences between these approaches and our approach for knowledge-based complex event processing.

One of the initial efforts on semantic event processing is *SToPSS*, the *Semantic Toronto Publish/Subscribe System* presented by Petrovic et al. in [Pet03]. It extends the conventional event processing in publish/subscribe system with some semantic capabilities. Taxonomies are created that provide a conceptualization of hierarchical relationships between concepts and the synonyms of the concepts. These taxonomies are then used in the matching processes of events to user-specified event patterns. By using the taxonomies the event patterns are expanded to further matches of streaming events. The fundamentals of the semantic enrichment of event streams [Pet03] are proposed, which we have extended by using external knowledge bases to include complex ontologies with different level of expressiveness.

[20] `http://www.tibco.com/products/event-processing/complex-event-processing` retrieved April, 2014

[21] `http://www.softwareag.com/corporate/products/bigdata/apama_analytics/overview/` retrieved, April 2014

Zhou et al. [Qun12] provide another system called *SCEPter* which can enrich event object with additional knowledge extracted by inferencing on ontologies. Annotations are used to filter or group different event objects by using their event types. *SCEPter* is able to extract single event instances based on their background knowledge. The approach provided does not target the detection of groups of events based on their relation chain in background knowledge.

3.9.1 Stream Reasoning Systems

Della Valle et al. [Val09] proposed *Stream Reasoning* a novel approach for the integration of data stream processing, the Semantic Web and logical reasoning systems. Stream reasoning technologies should provide foundations, methods and tools for the integration of streaming world with the more static Web of Data.

Sequeda et al. [Seq09] propose the concepts and visions of Linked Stream Data (LSD) which applies the Linked Data[22] principles to streaming data. LSD should allow publication of data stream in combination with Linked Data Web.

Several stream reasoning systems and approaches have been developed, like CQELS [LP11], Streaming SPARQL [Bol08], SPARQLStream [Cal10]. However the research area of stream reasoning is still young. In most existing stream reasoning approaches the streaming data is in the Resource Description Framework (RDF) data format, and the stream reasoner has the task of providing reasoning on a sliding window of RDF data.

Some stream reasoning languages and processing approaches have also been proposed. Barbieri et al. propose Continuous SPARQL (C-SPARQL) [Bar10] and EP-SPARQL [Ani11a] as a language for continuous query processing and Stream Reasoning.

Anicic et al. [Ani11a] provide an extension of SPARQL query language for processing stream of RDF data called *Event Processing-SPARQL* (EP-SPARQL). It provides additional operators like `SEQ, EQUALS, OPTIONALSEQ` and `EQUALSOPTIONAL` which can be used to define complex pattern to detect complex events in the stream of RDF data. EP-SPARQL can be used with the ETALIS event processing engine [Ani11b].

Le-Phuoc et al. [LP11] present CQELS (Continuous Query Evaluation over Linked Streams), a native and adaptive query processor for unified query processing over RDF stream and linked RDF data. CQELS processing engine can optimize query processing by continuously reordering of operators according to some heuristics about the streaming data. Furthermore, the external disk access is improved by data encoding and caching of intermediate query results. The CEQEL query language[23] provides window operators that allow specification of sliding window over RDF Streams.

Linked Stream Benchmark (LSBench) [Dan12] benchmarks CQELS against C-SPARQL and JTALIS (the java wrapper for ETALIS). SRBench (SRBench) [Zha12] is a general purpose benchmark for streaming RDF engines. The benchmark uses weather sensor data, since they are inherently stream based and time bound, to model a realistic use case.

[22]Linked Open Data `http://linkeddata.org/`
[23] `https://code.google.com/p/cqels/wiki/CQELS_language` retrieved April, 2014

3.9.2 Differences to This Work

Although some of the existing approaches either directly work on RDF streams or allow querying external data sources, they do not address the detection of semantic enriched events on the basis of huge amount of existing background knowledge. Most of the systems are designed for main memory event processing with pure syntactic event pattern matching or simple RDF pattern matching, without any further expressive semantic support.

In this work, we have addressed the problem of a hybrid approach - expressive reasoning on background knowledge to be used in high-performance real-time event processing. To the best of our knowledge, none of the existing systems is optimized for such hybrid event processing over huge amounts of semantic background knowledge (low frequently changing) and high-throughput event stream. We address the problem of a trade-off between the high expressiveness of the background knowledge used that leads to higher levels of computational complexity, and the efficiency and scalability needed in real-time event processing.

3.10 Windowing Specification over Stream

The processing of data in stream format cannot be realized as a blocking operations. In stream processing systems the data stream is divided into small portions so that processing of data can be realized in a non-blocking manner. Chakravarthy et al. [Cha09] specify a window over the data stream as follows:

"A historical snapshot of a finite portion of a stream at any time point."

Chakravarthy et al. [Cha09] identify two basic types of windows: time-based (or a physical window) and tuple-based (or a logical window). A time-based window is specified as `[Range N time units, advance M time units]` while a tuple-based window is `[Row N tuples, advance M tuples]`. The range specifies the size of the first window and an advance component defines how the window moves forward to a new window of data. In a time-based window the time of event data is mostly the arrival time of an event at the processing engine. Absence of an advance component specifies a disjoint window.

A data window can be specified by using the start and end boundaries of the window which are indicated by the variables WS and WE. A window is mostly specified as a sliding window that can include the most recent data from the data stream. A sliding window can be an overlapping/rolling window or tumbling/disjoint window.

Sliding Data Window

In a sliding window both WS and WE are changed so that the window is a small slice of the stream. The frequency of WS and WE changes are the same, therefore the size of the data in window is not continuously growing.

- **Overlapping/Rolling Window:** An overlapping window shares a portion of the current window with the next upcoming window, so that the data processor can also access a portion of old data. In forward movement on the stream is $WE_{new} \leq WS_{old}$.
- **Disjoint/Tumbling Window** In tumbling window the data in the current window is disjointed from the data of an upcoming window. The data processor can only access the passing data once. In forward movement is $WS_{new} > WE_{old}$.

Landmark Window

In a landmark window the start WS of the window is fixed while the end WE changes. The landmark window includes data from the start of the stream, the size of data in window continuously grows and is not bounded to an end.

- **Partitioned Window:** Arasu et al. [Ara06] define another type of window specification. They specify partitioned sliding window as an operation that can be run over on a stream S and takes a positive integer N and a subset $A1, ..., Ak$ of S's attributes as parameters. Partitioned window is specified by the reference to S in the query with `[Partition By A1 ,...,Ak Rows N]`. The windowing operation partitions the stream S into different substreams based on equality of attributes $A_1, \ldots, A_k$ (similar to SQL Group By). The window size is defined by using tuples with size N independently of the stream.

  For example, in a use case that has a network of road sensors sending data streams of vehicle speeds in different streets in a city, we may want to partition the data based on different vehicle IDs. We may want to have a stream for the time of each speed number so that we receive data from each vehicle.

- **Predicate-Based Window:** One other type of partitioned window [Gha06] is the predicate-based window. The predicate-based window can be specified based on query predicate. A query predicate is a query which can qualify tuples into the query answer.

  For example, in the case of a query like *"continuously report the readings with temperature greater than* 90° *considering only the last reading for each sensor"*, an EXPIRE operator is applied before the query filter, so that only the last readings for each sensors can be evaluated.

- **Semantic Window:** The above window types (sliding, partitioned and predicate-based) are useful for a variety of applications, however their functionality are limited to expressing highly complex windows required by some of the applications. The functionality of a stream operator is fixed and does not change according to specific application demands.

  Jiang et al. [Jia04, Jia05] specify the concept of semantic windows over data stream. They call the functionality of an operator its global property and the window property of an operator as its local property. They argue that the window can be based on a computation, which is used to determine a meaningful portion of stream so that computation on data can be realized correctly and efficiently.

  The semantic window has the task of determining which tuples in the current window should be deleted after it adds a new tuple into the current window. The semantic window specification uses a CW (current window) reference and a NT (new tuple reference) reference. A semantic window specification defines a finite portion of tuples from the data stream by continuously evaluating semantic window conditions (SWC) on new tuples. In the case that new tuples can be successfully evaluated, the new tuples are added to the current window. Jiang et al. [Jia04, Jia05] defined the window conditions based on the formal semantics of SQL query language and use SQL statements for defining the SWC.

3.11 Cost Optimization in Event Processing

Distributed event processing includes different types of costs, like event acquisition costs (network transmission costs), computation costs for the evaluation of detection rules (e.g., cost of filter and join operators). Different approaches have been proposed to optimize the costs of distributed CEP. Two relevant approaches to our work are [Akd08, SM09].

Akdere et al. in [Akd08] propose an approach for minimizing event transmission costs while meeting user latency expectations. The approach is a plan-based approach for multi-step event acquisition and processing that are based on temporal event relations and event occurrence statistics. An optimal plan is presented which is based on an exponential-time dynamic programming algorithm and two polynomial-time heuristic algorithms which can generate near optimal plans for detecting multiple complex events with common sub-expressions. Their work has its focus on reducing network communication costs by using the temporal properties of event sources.

Schultz-Møller et al. [SM09] provide an approach for the `Next`CEP system that is designed for distributed complex event processing with query rewriting. In their approach, the user-queries given are rewritten in a more efficient form for processing by a cluster of automaton-based event processing engines to achieve better scalability and high performance processing. Their simulations and experiments show a significant improvement in scalability as a result of query rewriting.

3.12 Real-Time Processing

One of the important aspects in event processing is real-time processing and is seen as an important constraint in event processing. Several definition are given in the literature for real-time systems [Lap92] like:

> *A real time system is a system that must satisfy explicit (bounded) response-time constraints or risk severe consequences, including failure.*

As defined, the explicit response-time constraints are as important as the failures and errors in the functionality of the system. A failed system is specified as a system that cannot satisfy one or more functionality requirements specified in the system specifications. Further definitions for real-time systems are:

> *A real time system is one whose logical correctness is based on both the correctness of the outputs and their timeliness.*

> *A real time system is any information processing activity or system which has to respond to externally generated input stimuli within a finite and specified period.*

Kopetz [Kop97] defines a real-time system as:

> *A real-time computer system is a computer system where the correctness of the system behavior depends not only on the logical results of the computations but also on the physical time when these results are produced. By system behavior we mean the sequence of outputs in time of a system.*

The POSIX standard 1003.1 [24] defines ”real-time” for operating systems as: ”the ability of the operating system to provide a required level of service in a bounded response time.” One of the POSIX standard definition key elements is the definition of sufficient performance constraints and performance-related functions to allow real-time application to achieve a deterministic response from the system.

From the definitions given in [Lap92, Kop97], the following two important facts about real-time systems can be concluded:

- The correctness of the system does not only depend on the logical result but more on the time the results are delivered.
- The failure to respond in the given time is as bad as the wrong results.

In event processing systems events should be detected within a specified time latency. Response time in event detection is the time that the system requires to detect a complex event after all constituent events have arrived. The expected latency should be defined by the specific use cases of event processing systems, e.g. some of which are described in Chapter 4.

Classification of Real-Time Systems

Real-time systems are classified in the literature according to different aspects and perspectives [Kop97]. One of the major classifications is based on the deadline time at which the task should be done and the correct results delivered to the user. The hard, soft and firm real-time systems are specified in the literature [Lap92].

- A *hard real-time system* is defined as:

  > *A system in which failure to meet a single deadline may lead to complete and catastrophic system failure.*

  Missing a deadline is defined as a total failure of the system and has a catastrophe as consequence, such systems are like systems for aircraft control, nuclear plant control, the detection of critical conditions.

- A *soft real-time system* is defined as:

  > *A system in which performance is degraded but not destroyed by failure to meet response-time constraints.*

  In soft real-time systems missing a deadline is undesirable for performance reasons. Examples of these applications are multimedia applications, booking systems, the displaying status information. By this definition of soft real-time systems, all practical information systems can be considered as soft real-time systems.

- A *firm real-time system* is defined as:

  > *A system in which a few missed deadlines will not lead to total failure, but missing more than a few may lead to complete and catastrophic system failure.*

[24]POSIX (Portable Operating System Interface) is an operating system interface standardized by ISO/IEC, IEEE and The Open Group `http://pubs.opengroup.org/onlinepubs/7908799/xsh/realtime.html` retrieved April, 2014

In firm real-time systems some small numbers of missed deadlines can be tolerated but other deadlines are hard deadlines. Examples of such firm real-time systems are embedded navigation controller for autonomous robots in which missing some critical navigation deadlines causes the robot to go out of control.

- **A** weakly hard real-time system [Ber01] is defined as:

  A system in which the distribution of its met and missed deadlines during a window of time w is precisely bounded.

  Weakly hard real-time systems are similar to firm real-time systems, but the number and degree of missed deadlines that the system can tolerate are clearly specified.

3.13 Summary

We have reviewed the related work to knowledge-based event processing. We have described the different event processing methods and event expressions and provided examples for each of them.

In the following chapter, we will put forward some of the use cases for knowledge-based complex event processing. We describe what the event streams are and what kind of information can be considered as background knowledge in each use case scenario.

Chapter 4
Use Cases of Knowledge-Based CEP

In this chapter we describe some of the selected CEP use cases to highlight the benefits of using background knowledge in CEP. We get into the details of event processing in specific use cases and describe their raw event streams as well as the query patterns for the detection of complex events. We outline the different properties of the use cases and describe them briefly. Examples of background knowledge relations and event detection patterns are provided to describe the complex patterns based background knowledge. Furthermore, we compare the requirements of different use cases.

Contents

4.1 Introduction

Complex Event Processing has many different use cases which profit from the real-time event detection and reactions to detected complex events. Some of these use cases are Business Activity Monitoring (BAM), Fraud Detection, Smart Offices/Cities, Logistics and Cargo, Information Dissemination, Event-driven Adaptive Systems, Supply Chain Management and Intelligent Systems in Health Care.

The extension of CEP to knowledge-based event processing enables CEP systems to profit from domain background knowledge and enable a more knowledge intensive event processing. In all of the use cases mentioned, knowledge-based CEP can improve the quality of event processing and profit from using event semantics in combination with existing knowledge about other concepts in the target application domain. The benefits of adding ontological background knowledge to the area of event processing is the same as data processing (described more in section 5.3).

In the following sections, we describe some use cases knowledge-based CEP and illustrate how they can profit from the usage of ontological background knowledge about the application domain. We analyze their different requirements such as real-time high performance processing, scalability, or the required level of reasoning on events and their related background knowledge. Furthermore we compare the requirements.

The main properties of these use cases can be illustrated by answering the following questions:

- What are the raw events, event types and event streams?
- What are the complex events?
- How critical is real-time processing for the use case? Are processing deadlines, hard deadlines?
- What kind of knowledge can be considered as background knowledge to be used for event detection?
- What kind of complex events can be detected based on background knowledge?
- What new knowledge can be derived based on the inferencing on background knowledge? What is the complexity of the inferencing on background knowledge for the specific use case?

4.2 Requirement Analysis of SCEP Use Cases

The use cases of knowledge-based CEP have different requirements for the processing of events and integration of knowledge from external knowledge bases. We categorize these requirements in the following points:

- *Real-Time processing of events:* Different use cases might be different w.r.t. meeting detection deadlines. Some of them might have absolute hard deadlines for the detection of events, while some others can tolerate the missing of deadlines, or only a predictable number of deadlines.
- *Scalability of CEP system:* The ability to process high throughput event streams in addition to the ability to fuse large-size external knowledge bases.
- *Reasoning on Background Knowledge:* The level of required reasoning on background knowledge can be different from use case to use cases.
- *Expressiveness of Event Detection Queries:* The complexity of event detection rules can be different.
- *Elasticity of SCEP system:* The requirement to process data with highly variable sizes/dimensions on demand and be able to optimize processing costs.
- *Requirement for guarantee Detection:* In some of the use cases it is required that the system detects all complex events and provides a granite for the detection of complex events that has happened. In some other use cases, it is accepted if the system provides an approximation of the complex events and does not detect all of them, i.e., detect some of the complex events, but be able to detect these events in real-time.

4.3 SCEP Use Cases

In this section we get into some of the relevant CEP use cases to illustrate the benefits of knowledge-based CEP. We describe briefly the scenario of these use cases and describe the raw events, back ground knowledge and the complex events which can be detected. At the end of each of the use case we provide an overview about the data sources used for that use case.

4.3.1 High Level Market Monitoring (HLMM)

In today's economy, companies are highly interconnected and dependent on each other. For example, companies require raw material, suppliers and financial credit. Business are also dependent on laws legislated by politicians. We can see some kind of domino effects between companies which impact on their businesses and the price of their stock market shares. In the case that the business of one of the companies changes, the business of other companies might be affected as well.

The question in this use case is if it is possible to detect complex events according to companies' relations so that decision makers can react to them in a timely manner. The target is not to predict the start of the domino effect, but to notify as soon as the first stones fall.

A complex event detection pattern is shown in Figure 4.1 which is defined based on the background knowledge about companies and company dependencies. It shows stock market events and insight into the related background knowledge about the companies. We can see that companies have some business dependencies on each other, company Y1 produces raw material M1, the business of another company Y2 depends on this raw material for its production and might have big trouble if they cannot supply the material. A third company X3 finances company Y2 and might have some financial problems if company Y2 gets material shortage trouble. A stock market broker might be interested in this dependency chain and can define a complex event detection pattern for this particular complex event without even knowing what these companies are. He might be interested to know when the stock prices of these three companies start falling. The aim of this event detection is not to predict stock market prices, but to be informed when the prices of these three companies falls.

```
{ .., {(Name, ‘‘GM’’)(Price , 20.24)(Volume, 8,835)},
      {(Name, ‘‘SAP’’)(Price , 48.71)(Volume, 8,703)},
      {(Name, ‘‘MSFT’’)(Price , 24.88)(Volume, 46,829)},
 ... }
```

Listing 4.1: Stock Exchange Event Stream

Suppose Mr. Smith is a stock broker and has access to a stock exchange event stream, like those listed in Listing 4.1. He is interested in particular kinds of stocks and would like to be informed when there are some interesting stocks available for sale. His particular interest or his particular stock handling strategy can be described in a high-level language which describes the interest using background knowledge about companies.

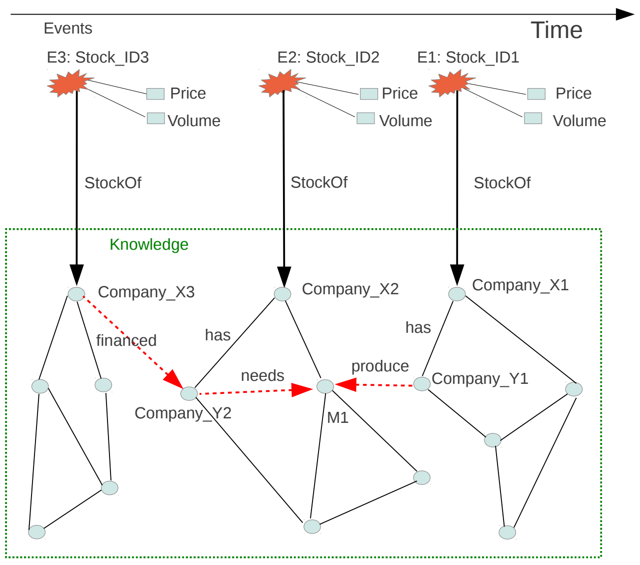


Figure 4.1: Example of Relations on Stock Market Events and Companies Background Knowledge

Mr Smith would like detect complex events using the abstract query listed in Listing 4.2:

```
Select stocks of companies which
                 have production facilities in Europe,
                 produce products from Iron,
                 have more than 10,000 employees,
                 are at the moment in restructuring phase and
                 their price/volume have been increasing in the past 5 minutes.
```

Listing 4.2: An Example of a High-Level Complex Events Query in Natural Language

As we can see, the above query cannot be processed without having background knowledge to define the concepts in this query. Mr. Smith needs an intelligent system which can use background knowledge about companies like those listed in Listing 4.3. This background knowledge 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, build, Cars), (Cars, areFrom, Iron),
  (OPEL, hatProductionFacilitiesIn, Germany),
  (Germany, isIn, Europe),
  (OPEL, isA, MajorCorporation),
  (MajorCorporation, have, over10,000employees),
  (OPEL, isIn, reconstructionPhase), ... }
```

Listing 4.3: An Excerpt of a Knowledge Base about Companies.

We can also assume that Mr. Smith works for a company and may need to share this knowledge base with other brokers. Each of these brokers may be able to gather new information about companies and update this knowledge base, e.g., the Opel Company is not in restructuring phase, or a company has a new chief executive officer.

High Level Stock Market Monitoring	
Property	**Description**
Raw event source	Live stream of stock market rates from different stock markets
Background knowledge resource	Background data about stock market companies, e.g., extracted from Linked Open Data and integrated into corporations internal knowledge
Complex event type	Events defined based on company relations and the relations of other resources to other companies
Derived knowledge	Extraction of further company relations
Real-Time factor	In the case that some of the detection deadlines have been missed, financial disadvantages are expected. This can be considered as a soft real-time system in which the number of times that the system misses the deadline are predictable (described in Section 3.12).

Table 4.1: Overview of Data Sources for High Level Stock Market Monitoring

4.3.2 Extended Urban Computing (EUC)

The idea of applications for urban computing was one of the use cases of the EU LarKC project [Del09, Del10] for reasoning on background knowledge. The use case describes the situation of a tourist in a foreign city who wants to travel to several places in the city, e.g., to visit historical monuments, museums, concerts or visit friends. In the case that the destinations of the user route is unknown, the user can express his requests or aims so that the optimal routes to the destinations are selected by the system based on user-defined preferences.

The task for a reasoning system is to find optimal routes for this person based on his position in the city, his preferences, position of places to visit, and the traffic situation of the target city. In the LarKC project the task was to reason only on the stored background knowledge as the more static knowledge, and not on the stream of events.

We extended this scenario for knowledge-based event processing. Consider the following situation, suppose we have a set of users in a city who are equipped with smart mobile devices and are moving through a city. In this city, we have several associate partners like museums, concert organizers and shops which provide real-time special offers, for example a discount of 50% off on concert tickets[1]. Users should be informed about these special offers in real-time. Notifications should be realized based on user profiles/interests, real-time requests, their positions and the traffic situation in the city. After that, the users were informed they can accept these offers and be served on a first-come-first-served basis.

In this use case scenario, we have the following five event streams:

1. The GPS positions of each user
2. Live traffic of the city
3. Special offer events
4. User accepts or denies events

And the following background knowledge is required:

1. Knowledge about the city map

[1]Similar to online services like `http://www.eventful.com`, and `http://www.groupon.com/`

Extended Urban Computing	
Property	**Description**
Raw event source	The five event streams about user position, live traffic, special offers, user requests, offer accepts
Background knowledge resource	Background knowledge about city map, offers, user preferences
Complex event type	Complex machetes on events based on offers, user positions, traffic and the relations of other resources
Derived knowledge	Extraction of further relations, e.g., about offers and user preferences
Real-Time factor	In the case that some of the detection deadlines are missed, only one offer is not matched to a user. This can be considered as a soft real-time system.

Table 4.2: Overview of Information Sources for Extended Urban Computing Use Case

2. Knowledge about potential special offers, types of special offer events and their other properties
3. Knowledge about user preferences and interests (user subscription queries)

The aim is to detect events by using the background knowledge stored in the knowledge base and by doing reasoning on this knowledge. Detection might be done in parallel or in a sequence of filtering steps. The detection of complex events includes reasoning on the background knowledge which can be a computation intensive task.

An example of a complex query is if two special offers are available that are from two specific deals offered by shops that are near to each other so that it is worthwhile for the user to walk to that position to get both offers.

Table 4.2 provides an overview over event stream sources and background knowledge used in extended urban computing use case.

4.3.3 Ambient Assisted Living (AAL)

Ambient Assisted Living [Bra12] is about all the concepts, products and services which can be developed with the aim of improving the quality of life, especially in accordance with the requirements of elderly people. AAL can be understood as age-appropriate assistance systems for healthy and independent lives.

Let's take for example Mr. Smith, he is a senior citizen and he lives in a smart home well-equipped with different sensors signaling event streams from different areas of the house, e.g., movement in different areas of the house, or from different household appliances, e.g., coffee machine, refrigerator, washing machine. The refrigerator can register all its content and each time when one of the items is taken by Mr. Smith it can signal the ID number of the food item. There are also medical sensors in the house which send blood pressure, blood sugar and heart frequent measurement of Mr. Smith in regular time intervals.

Ambient Assisted Living	
Property	**Description**
Raw event source	Event streams about activities of elderly people
Background knowledge resource	Background knowledge about foods, drugs, allergies and diseases, doctor preferences
Complex event type	Detection of critical situations in elderly people
Derived knowledge	Extraction of further relations, e.g., about foods, drugs, diseases, user/doctor preferences and the relations to other resources
Real-Time factor	In some cases absolutely hard real-time, because of the danger of death. In some other cases, it can be considered soft real-time because the system should only warn the user about possible upcoming critical situation.

Table 4.3: Overview of Information Sources for Ambient Assisted Living Use Case

Mr. Smith's family doctor advises that some food ingredients are not generally good for Mr. Smith' health condition because of his chronic diseases like hypertension (high blood pressure) and diabetes. In the case that Mr. Smith has abnormal blood pressure over a period of time, he should not eat certain type of foods, or in the case that Mr. Smith has already eaten one particular type of food, he should not take another type within a certain time period.

An intelligent system should monitor for drug and food interactions and combine this with all of Mr. Smith's activities. The system should send warnings to him and in critical situations send alarms to his family doctor. The system will receive a stream of raw event signals from different household sensors and medical sensors, and have access to a knowledge base which can store all the background knowledge about food types, Mr. Smith's diseases, and their relations.

The task of a monitoring system is to process the event stream close to real-time and integrate background knowledge from external knowledge bases. Background knowledge bases also includes a logic reasoner for inferencing on the existing knowledge for examples about food types, diseases and their relations. All the drug interactions, alcohol/food interactions with drugs, disease interactions with drugs can be monitored by a monitoring system if huge amounts of knowledge about drug, food and disease interactions are available in a knowledge base. Different warnings and alarms with different classifications like *"Major"*, *"Moderate"* or *"Minor"* as highly/moderately/minimally clinically significant can be triggered in the case that one of the event detection patterns are evaluated to be true.

Table 4.3 provides an overview over event stream sources and background knowledge used in ambient assisted living use case.

4.3.4 Further Application Scenarios

Many other use cases, which profits from the integration of external knowledge bases, can be considered for event processing. Some of these use cases are Knowledge-Based Fraud Detection (FD), Fleet Management (FM), Energy Management (EM), Active Office Environment (AOE), Public Health Monitoring (PHM) and Hospital Activity Monitoring (HAM). The description of all these use cases are beyond the scope of this work, therefore we will describe only some of the relevant use cases.

Public Health Monitoring (PHM)

One of the high priorities of health-related organizations is to monitor public health and detect risk events like epidemics and outbreaks of diseases. Some of the recent examples of such infections are the 2009 flu pandemic, a global outbreak of a new strain of H1N1 influenza virus, most commonly referred to as "*swine flu*" and influenza A virus subtype H5N1, known as "*bird flu*". The early detection of such pandemic events requires the capability of expressive and near real-time processing of massive event data coming from heterogeneous and distributed event stream channels. For such use cases, an event processing engine must be able to identify, interpret, map and integrate different atomic event data from different sources and process them.

In the public health scenario different event notifications to the public health monitoring system will have to be submitted through different notification channels. For example, physicians from their medical practice can notify the system about their patients of their symptoms, initial diagnosis and confirmed final diagnosis. Other event data streams might come directly from patients at home using different sensoring devices. An intelligent processor can then fuse all these information pieces.

Some of the research efforts for public health monitoring have been done on the monitoring of Twitter[2] data stream. Stewart et al. [Ste12] propose approaches for health-related Twitter monitoring. They discuss an early warning system for notification about upcoming disease outbreaks and introduce several biosurveillance algorithms and techniques (Hutwagner et al. [Hut03]; Basseville et al. [Bas93]) and use them to analyze crowd behavior during the 2011 *enterohemorrhagic Escherichia coli* (EHEC) outbreak in Germany. Lampos et al. [Lam10] studied the twitter data stream during the 2009 *Swine Flu* outbreak.

Hospital Activity Monitoring (HAM)

Several sensors can be installed on patients' body and deliver different event streams about the health status of patients like body temperature, blood pressure, heart beat, special muscle movements. In healthcare scenarios, medical event categorization engine gathers events from different patient body sensors. These events are then processed and complex events can be detected that can be used to control the business processes in hospitals. For example, a complex event can be a patient in an emergency situation, a patient who needs monitoring during the course of post-surgery.

The event stream is integrated from different resources like events about the usage of drugs. The knowledge about the regulation of off-label drug warrants and specific patient drug allergies can be stored in a knowledge base. An event processing engine can process these event streams and use background knowledge about the patient, diseases and drugs to identify complex medical problems. By stating and identifying such complex events doctors and nurses can use such automatically-analyzed information and make medical decisions faster. In a hospital unit where several patients are monitored, such an intelligent event-driven system can help medical staff to identify critically-ill patients.

[2]Twitter `http://twitter.com` is a microblogging service on the Web.

Usage of CEP in Diagnostics Process: Another potential usage of knowledge-based CEP in E-health is to use it as a supplement tool to support doctors in the diagnostic process. In several cases, doctors have to monitor event data streams and process this information in combination with their medical knowledge about diseases. These processes are in some cases time consuming. For example, doctors need to give commands and monitor body reactions, or they should administer drugs and monitor body reactions to those specific drugs at different times. An intelligent event-processing system can collect all these events and analyze them to detect complex events and prepare medical information for doctors so that they can make diagnosis more precisely and faster.

Hospital Hygiene Monitoring: Currently a health care monitoring system, HyReminder [Wan11] is deployed at the University of Massachusetts Memorial Hospital. The HyReminder system can monitor the hand hygiene of their healthcare workers. In the hospital each doctor equipped with an RFID badge so that the system can track their activities throughout the hospital. RFID readers can generate event data which are then transmitted to a central event processing system. As an example, query for hygiene monitoring, according to the US Center for Disease Control[3], all doctors who enter and leave patients' rooms (captured by RFID readers at the doorway) should clean their hands (captured by RFID readers at hand sanitizers or lavatories) within a period of time and before they enter another patient's room.

This scenario can be extended when the event processing system includes background knowledge about different patients, their diseases and knowledge about the healthcare workers. A complex monitoring query can be specified based on different diseases, their relations and the knowledge about workers, so that different critical categories can be specified to monitor hospital hygiene based on different dynamic factors. For example, cleaning time and cleaning intensity can be different for different patients.

4.4 Comparison of Use Case Requirements

As we have discussed, each use case has its unique properties and requirements. Table 4.4 compares the different requirements of the identified use cases for knowledge-based event processing. We compare the following use case requirements and properties:

- *Stream Throughput:* If the use case has a high, middle or low rate raw event stream.
- *Real-Time Event Processing:* If the use case require hard, soft real-time, firm real-time, weakly hard real-time processing (described in Section 3.12).
- *The Complexity of Detection Rules:* If the event detection rules use multiple event detection operators that are chained to detect complex events or the event detection rules are simple rules with one or two event detection operators. We consider the cases that users in these use cases have to write such complex event detection rules; for this, we have defined 3 main categories of high, middle, low complex event detection operators. In all the uses cases that we have introduced in this chapter we have cases in which users have to write rules with multiple event detection operators.
- *Reasoning on Background KB:* If the use case requires high, middle, low expressive reasoning on background KB.
- *Scalability:* If the use case requires working on large data sets (background and streaming data) and requires high, middle or low scalability.

[3] http://www.cdc.gov/

Use Case	Stream	Real-Time	Reasoning[4]	Complexity of Detection Rules[5]	Scalability	Approximation
HLMM[6]	high	weak hard	high	high	high	no
EUC[7]	high	middle	high	high	high	yes
PHM[8]	high	soft	high	high	high	no/false positive
HAM[9]	high	weak hard	high	high	high	no/false positive
AAL[10]	high	weak hard	high	high	high	no
AOE[11]	high	firm	high	high	high	no
SNF[12]	high	soft	high	high	high	yes

Table 4.4: Comparison of Use Case Requirements and Properties

- *Approximation:* If the use case can accept false positives or false negatives for the detected complex events (Yes or No).

In this research, we propose different event processing approaches (Chapters 8, 9, 10). The knowledge representation method (Chapter 6) and the event processing approach should be carefully selected, extended, customized and integrated based on case requirements and properties presented in Table 4.4 (examples are shown for some the use cases).

In the conclusion, Chapter 11, we return to this table to show that the proposed event processing approaches should be used depending on use case requirements and properties.

4.5 Summary

In this chapter, we have pointed out some of the relevant use cases in event processing which can benefit from the fusion of domain background knowledge and event streams. We have illustrated the benefits of knowledge-based CEP in some of the use case scenarios, and briefly analyzed the different requirements like reasoning, expressiveness, scalability and real-time processing.

In the next chapter, we describe our main concepts for knowledge-based complex event processing. Furthermore in the next chapters, we provide different event processing approaches for semantic-enabled CEP and show how the different event processing approaches might be applicable in different use cases depending on requirement factors.

[4]This means reasoning on external KBs

[5]If the event detection rules using event detection operators are simple rules or complex chaining rules that use multiple operators.

[6]High Level Market Monitoring

[7]Extended Urban Computing

[8]Public Health Monitoring

[9]Hospital Activity Monitoring

[10]Ambient Assisted Living

[11]Active Office Environment

[12]Semantic News Filtering

Part II

Models, Problem Analysis and Suggestions

Chapter 5

Knowledge-Based Complex Event Processing

This chapter provides an overview on the idea of knowledge-based complex event processing. We describe the concept of fusion of event stream and background knowledge, its components and the abstract view of semantic-enabled CEP. Subsequently, we provide the data model of the knowledge-based event processing. Finally, we discuss the optimization parameters of the knowledge-based event processing which are the subject of the subsequent chapters.

Contents

5.1 Introduction

The identification of critical events and situations requires processing vast amounts of data and metadata within and outside the systems. Modern declarative rule languages based on logical representation of events in combination with ontologies or (business) vocabularies, can enable the evolution from pure syntactic event processing systems into semantically enrich and more *"intelligent"* reactive systems. The combination of event processing and knowledge representation leads to novel event processing systems, which can understand more about the events and their application domain.

As introduced in Chapter 1, we propose a framework for knowledge-based (Semantic) Complex Event Processing (Semantic-Enabled Complex Event Processing (SCEP)) which extends CEP with integration and usage of domain background knowledge. Knowledge about event types and their hierarchies, i.e., specialization, generalization of events are useful for aiding the event processing systems to understand more about the events. Semantic models of events improve the quality and quantity 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 are linked to domain knowledge such as domain vocabularies/ontologies and knowledge data instances. Semantic inferencing is used to infer relations between events, such as transitivity or equality between event types and their properties.

In the SCEP approach, event processing technology works, in addition to ontologies, in tandem with rules technologies. It offers efficient detection of relevant situations of interest from huge amounts of events, while providing the required expressiveness to describe higher-level logic such as business rules policies and reactive work-flows used, e.g., for agile Semantic Business Process Management.

In the following sections, we provide a description of knowledge-based event processing. We describe the main concept for fusion of event stream with background knowledge, point out its benefits of using background knowledge and illustrate the abstract view on semantic-enabled CEP. Furthermore, we build a data model for events, event streams and background knowledge. Finally, we describe the most important parameters of semantic-enabled event processing.

5.2 Fusion of Event Stream and Knowledge

For the realization of SCEP, we propose using external Knowledge Bases (KBs) which can manage background knowledge (conceptual and assertional, T-Box and A-Box of an ontology) about events and other non-event resources. The deployment of external KBs for event processing means events can be detected based on reasoning on type hierarchy, temporal/spatial relations or relations to other concepts/objects in the application domain.

An example of the connections of events to relevant concepts in background knowledge can be the relationship of a stock market event (e.g., a significant price change) to the products or services of that company (see Section 4.3.1 for description of stock market monitoring use case) .

A complex event query can be defined based on the terminology of ontologies and schema of event data streams. This allows users to define queries which are specified abstracts based on relationships in background knowledge, and not only based on syntactic matching of the values of events. A specified event query should be continuously evaluated against data coming from both sides, the event stream and the external knowledge base.

As described in Chapter 3, most existing approaches for event processing use rule engines or finite-state automaton for the processing of events without permanent storage or indexing of event data. The event stream can flow through the system without any necessary storage. The storage of historical event data is only optional for purposes other than event detection.

Rule-based event processing engines can process events in real-time, because they can handle whole facts and rules in the main processing memory. However, these approaches cannot achieve high scalability or high performance, in the case that they have to process huge amounts of domain background knowledge for event detection. The challenging problem is that rule engines have to keep the whole KB in the main processing memory space, which is a limited space. It is impossible to use huge amounts of background knowledge for event processing, e.g., using background knowledge about the companies traded on the stock exchange markets worldwide.

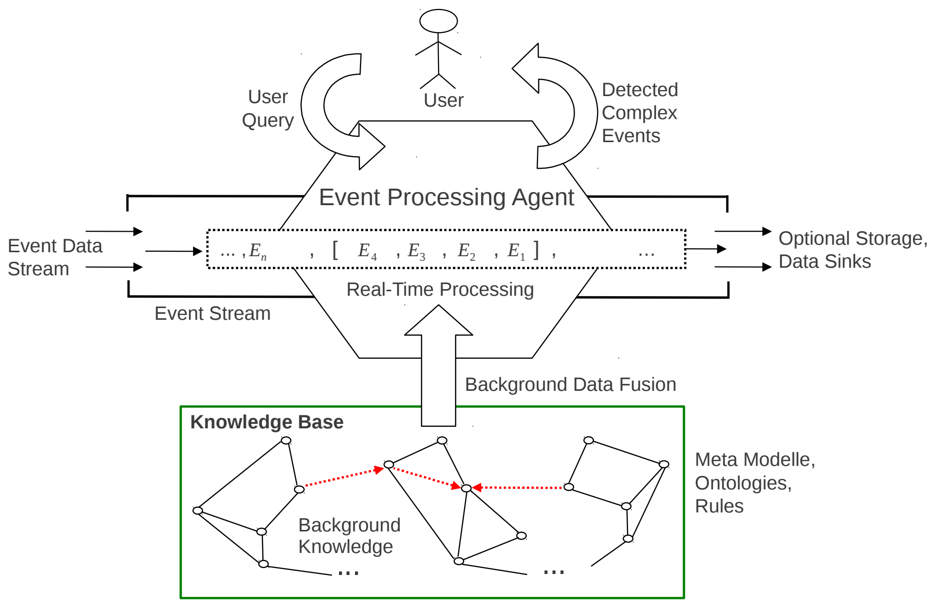


Figure 5.1: Abstract Overview of the Knowledge-Based Event Processing using External KB

To integrate and aggregate domain background knowledge with incoming event stream and to timely process of integrated knowledge a high scalable and high performance processing approach is required. Particularly, inferencing on huge amounts of background knowledge can be a time and computation-intensive process. An implementation of SCEP should provide several acceptable quality of service metrics like low-latency, high-throughput and high scalability. On the other hand, it should also provide expressive reasoning on background knowledge and derive events based on the inference on background knowledge. We can see that there is a trade-off between all of these metrics which should be optimized for the target use case.

The abstract view of the proposed SCEP approach is shown in Figure 5.1. Our approach illustrates usage of an external knowledge base which includes ontologies and rules, and an incoming event stream which comes from event producers, e.g. sensors or event adapters. The SCEP engine should process the event stream based on semantic relations of event stored in the external KB. The knowledge base includes TBox (assertions on concepts) and the primitive event objects as ABox (assertions on individuals) which are used for semantic reasoning on events. As mentioned, in SCEP a part of the knowledge about events might be the relatively static knowledge about pre-defined event classes, i.e., the event types in an event ontology, while the other part are real-time data streams. The system has to combine these knowledge references and infer new knowledge. The result of complex event processing are the detected complex events that are notified to the users. The CEP engine can also identify critical complex actions which might be triggered after these events happen and are detected.

Most existing approaches for event processing use rule engines to process events. The input raw event stream can flow through the system without any necessary storage or indexing of data. The storage of historical event data is only optional for other purposes. The rule-based event processing engines can process events in real-time, because they can handle the given facts and rules in the main processing memory. However, these approaches cannot achieve high scalability and high performance, in the case they have to process huge amounts of domain background knowledge and use this knowledge for event detection. The main problem is they have to keep the whole KB in the main processing memory, and this is impossible if the background knowledge is huge, e.g., the background knowledge about companies traded on the stock exchange markets worldwide.

A naïve approach for SCEP might be a storage-based approach, e.g., one can store all of the background knowledge on an external KB and start polling the knowledge base on every incoming single event. The results from the KB can then be processed with an event data stream. This approach may have scalability and performance problems when the throughput of the event stream is high or the size of the background knowledge is huge, or even when expressive reasoning should be done on the knowledge base. The main disadvantage of this approach is that processing is only possible after storing/indexing the data and the databases are polled with each new incoming event. This implementation might be applicable for use cases which do not have high event throughput or huge amount of background knowledge to process. The advantage of this approach is that a complete reasoning on the whole KB inventory is possible. Scalability and real-time processing are the problems of this approach which makes it impossible to use for time-sensitive use cases.

5.2.1 Input Event Stream

An event is defined in the EPTS Glossary [DL11] as “an object that represents, encodes, or records an event, generally for the purpose of computer processing” (also described in Chapter 3). More importantly, are notifications that are sent as data streams which notify about events which have already happened. These notification data stream through the system as data items. Event notification (aka event objects) can be seen as a multiset of attribute/value pairs which are annotated with a set of timestamps. Timestamps mark down the different occurance time of events, e.g., start and end timestamps.

The EPTS Glossary [DL11] defines an event stream as “a linearly ordered sequence of event.”. In this research, we consider an infinite sequence of event notifications with the same schema as the event stream (more precisely specified in section 5.2.3).

In SCEP an event instance can be specified using a set of RDF triples (RDF graph). A mapping of an event notification in the form of attribute/value pairs to a set of RDF triples (RDF graph) can be used to create an event instance in RDF. This knowledge can be used in the next steps for the processing of events, e.g., in the condition part of an Event-Condition-Action (Event-Condition-Action (ECA)) reaction rule. Listing 5.1 shows a set of RDF triples which defines an instance of this event.

In an SCEP system it is important that the system can build a connecting map from one or two attribute/values to resources in the background knowledge so that it can retrieve knowledge about the events. This connecting map enables existing knowledge (non-event concepts) to link up to the event instances. For example, the name of the stock which the event is about is not identified by a simple string name but by a Uniform Resource Identifier (URI) which links to further knowledge about the stock type.

```
    { (event:e1002 , rdf:type , event:stockPriceChange),
      (event:e1002 , stock:Name , stock:LehmanBrothers),
      (event:e1002 , stock:hasPrice , "45")  }
```

Listing 5.1: Example of an Event Notification in RDF

A mapping of event instances in other formats to RDF data format is not always necessary. The two part of data, event stream and background KB can be handled separately and the results can be integrated/merged at the event processing step. In our research, we keep these two data sources separate to enable the possibility to include huge amounts of background knowledge and achieve high scalability (difference of our work on the RDF stream reasoning approaches 3.9.1).

5.2.2 External Knowledge Base

One or more external KBs store existing background knowledge (conceptual and assertional, T-Box and A-Box of ontologies) on events and other relevant concepts/objects in use cases. In many event processing use cases, the amount of background knowledge can be huge so that they cannot be included as rule sets in the main memory of event processing agents for reasoning. Therefore, we propose using external KBs for storing and reasoning on background knowledge.

Background knowledge can be stored in RDF data format in external triple stores (specific kind of database for storage and management of RDF data). This external KB can be queried from event processing agents to retrieve knowledge about events and their relations to other resources. Communication between the event processing engine and external knowledge bases are in the same architecture, which is specified in OWL-QL [Fik04]. We can see client-server communication between the event processing agent and external knowledge bases to retrieve background knowledge about events. The KB can be queried by using a SPARQL Protocol And RDF Query Language (SPARQL) [Pru08] query and the results are then send back to the event processing engine.

The external KB also includes a description logic reasoner to reason on the relations between events and other relevant non-event objects in the application domain. The reasoner should be seen as an internal component of the external KB that works on top and can access the stored data for reasoning. In this way, we have two reasoning on the whole knowledge; the first one, based on event algebra operations reasons on the event stream and the second one, based on description logic, reasons on background knowledge. Reasoning on background knowledge can be specified in different expressiveness levels like, RDFS, OWL-Lite, OWL-DL, OWL-Full (see. Chapter 2). The result of the SPARQL queries sent to the KB also includes results gained from reasoning on KB (based on inferred triples generated after reasoning on data based on specified expressiveness levels).

One or more connecting links can be specified from attributes/values of events to the relevant background knowledge resources. We assume this kind of links is also made available at the time of event type specification. The connecting links are also part of background knowledge stored in KBs.

5.2.3 Data Model of Knowledge-Based Event Processing

In this section, we specify a data model for our research problem. First, we introduce our adopted models for events and event streams, second we describe our model for background knowledge about events. Our event model is adopted from the event models in state-of-the-art event processing approaches like Cayuga [Dem06, Dem07, Bre07], DistCED [Pie03b], and in [SM09].

In our model we establish a few definitions to be able to describe other concepts.

Definition 5.1. ***(Event)*** *An Event/Event Object is a tuple of $\langle \bar{a}, \bar{t} \rangle$ where $\bar{a}$ is a multiset of fields $\bar{a} = (a_1, ..., a_n)$, and is defined by schema $\mathbb{S}$. The $\bar{t} = (t_s, ..., t_e)$ is a sequence of timestamps representing the different occurrence times of the event, the start t_s and end timestamps t_e of the event.*

For example, an event in stock market applications has fields *(name, price, volume, timestamps)*, like *(IBM, 80, 2400, 10:15, 10:15)* the start and end time of an event, because it is an instantaneous event. An *event* can also be considered as a set of attribute values $\langle \bar{av}, t_s, t_e \rangle$ where $\bar{av}$ is a multiset of attribute value tuples $\bar{av} = ((a_1, v_1), \ldots, (a_n, v_n))$, for the above example, we will have $(((name, IBM), (price, 80), (volume, 2400)), 10:15, 10:15)$.

Definition 5.2. ***(Event Type)*** *An Event Type (aka event class, event definition or event schema) is a class of event objects. [DL11] (We refer to it also as "syntactic event type")*

Definition 5.3. ***(Event Stream)*** *An Event Stream is an infinite sequence of timely-ordered events with the same or different schema $\mathbb{S}$.*

EPTS glossary [DL11] defines the event stream as "a linearly ordered sequence of events." An event stream can be generated by merging several different event streams.

An event instance is a single event. Composite events can be detected based on the temporal relationship of events, e.g., two events happening in sequence to each other in stream or happening at the same time. An event processing engine can detect events based on their temporal relationships or based on the syntactic matching of their attributes. Detection is defined based on formal semantics which are defined event operation algebra, like operations defined in Snoop [Cha94a]. Different event processing systems have already extended these operations and specified different event detection operations.

On the other side, for the modeling of background knowledge about events, we have adopted a knowledge model from description logic [Baa08] and especially from OWL-QL[Fik04].

Definition 5.4. ***(Knowledge Base)*** *A knowledge base (KB) is a pair $(\mathcal{T}, \mathcal{A})$, where $\mathcal{T}$ is a TBox and $\mathcal{A}$ is an ABox. An interpretation $\mathcal{I}$ is a model of a KB $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ if $\mathcal{I}$ is a model of $\mathcal{T}$ and $\mathcal{I}$ is a model of $\mathcal{A}$.*

A knowledge base KB can also be considered as a set of logical axioms K_S representing a logical theory in which a set of entailed logical axioms K_E are true so that $K_S \subset K_E$.

A new set of knowledge is driven from existing logical axioms K_S. Entailment is specified based on the entailment rules, e.g., in *"description logic"* different expressiveness levels are defined, which can be mapped to different entailment rule sets.

After the specification of two models for the two worlds, event processing, and knowledge representation, we can assume some of the fields of events (e.g., attribute/values) are in relation to some of the resources in KB (such as individuals, concepts, roles and sentences). It is possible to ask the knowledge base and retrieve background knowledge about the attributes of the events. We hold that an event processing agent can be seen as a client who initiates knowledge queries to an external knowledge base server and retrieves background knowledge about an event instance. A knowledge query contains a query pattern that matches a collection of sentences in KB and is built based on attributes of events.

Definition 5.5. ***Event2KB Link –*** *An Event2KB Link is a logical axiom which is embodied by one of the attribute/value pairs of a raw event instance and is linked it to an individual in the knowledge base.*

The problems that we address in this dissertation can now be substantiated in the following two main problems:

1. **Formal Semantic of Event Detection:** What is formal semantics when we define event detection rules (queries) based on the fusion of these two worlds, event algebra operations and description logic entailment rules. Detection of events can be specified based on knowledge query patterns (graph patterns) and its combination with event algebra operation (interval-based/point-based event detection algebra).

   Our goal is to specify the semantics of event detection queries based on background knowledge and event operation algebra, as defined by the users

2. **Optimized Event Processing:** To what extend is it possible to optimize the event processing approaches, when event processing agents have to retrieve knowledge from external knowledge bases. Event processing should be done in an optimized manner to avoid bottlenecking of polling external knowledge base servers and computation intensive reasoning on knowledge. Initiating queries on external KBs has different costs and the total cost of an event detection is the sum of the costs for detecting single raw events.

   Our goal is to optimize computation costs on external KBs for knowledge inferencing and acquisition, while the system is capable of meeting latency expectations without missing any complex events, as defined by the users.

Definition 5.6. ***(Semantic Event Type)*** *The semantic event type of an event instance e can be identified when connected to one or more resources in a background knowledge KB using an Event2KB link. Relations in the background knowledge base KB specify the semantic event type of the event instance.*

5.2.4 Updates on Background Knowledge

The background knowledge about the application domain might in different time intervals have some knowledge updates which add new sentences to the knowledge base or delete some of the existing sentences. The updates of background knowledge can be distinguished in two main categories, *T-Box updates* and *A-Box updates*.

1. *A-Box updates* induce the effect that the reasoner must recompute the inferred triples, i.e., the answers to a specific query to the knowledge based is changed after the update. This means all materialized inferred triples in the cache system might be recalculated. This depends on how reasoning on the external reasoner (a component on top of the KB) works and which algorithms it uses for the reasoning (e.g., tableau reasoning, SLD-resolution or rete-based algorithms, see. Chapter 2). The update operation is a blocking operation on the external knowledge base (similar to relational databases).

   In this research, we assume that we have *A-Box updates* on the external knowledge base, but the frequency of these updates are not as high as the frequency of the event stream.

2. *T-Box updates* have much more effect on the system and can make user queries invalid, because the terminology of the application domain is changed. User queries are specified based on the T-Box of the knowledge bases and some of the user queries which are related to the *T-Box update*, might become invalid after the update.

   In this research, we assume the *T-Box* of the background knowledge is stable and we do not have any updates on this part, or we can assume the whole system can be revalidated (checking if the queries are still valid with the new T-Box) and event processing should be restarted after such an update.

5.3 Benefits of Background Knowledge for Event Processing

We see using background knowledge with complex event processing as major a advantage over the state of the art CEP approaches:

- **High Expressiveness**: Expressiveness means the user can express his interest on complex events not only based on syntactic matching of event attributes and interval sequence of events (event operation algebra), but also based on the relations of concepts/resources in background knowledge and description logic reasoning. An event processing system can process precisely expressed complex event detection patterns and reactions to events can be directly translated into business operations.

- **System Flexibility**: Flexibility means a CEP system is able to integrate new business changes into the systems in a fraction of the time rather than changing the complete rule set for event detection. Complex event patterns are independent of current businesses and are defined in a higher level of abstraction based on business strategies. If the business environment changes, it can be considered simply as an update to the background knowledge, while the complex event detection patterns, which are defined based on the business plans, should not change.

- **Extracting High-Level Information from Raw-Data:** One further benefit of this method is the user can define event queries in a high-level language and does not need to be aware of details, e.g., in stock market monitoring use case, users can specify detection queries like, *"companies which produce products from iron"* and does not need to know all the products of companies which are not easy for humans to remember. This is useful to simplify the specification of event detection rules so that users can more easily define rules based on abstract high level background knowledge.

- **Information Integration:** Furthermore, complex event processing can benefit from 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.

5.4 Time, Clocks and Ordering of Events

Lamport [Lam78] provides fundamental concepts about the occurrence of events and the order of event occurrences in a distributed system, and define a partial ordering of the events in a distributed setting. Lamport proposes the concept of logical clocks in comparison to physical system clocks. He provides a distributed algorithm for synchronization for the system of logical clocks that can be used for the total ordering of events. The total ordering of events can be used for solving synchronization problems. Lamport's algorithm is specialized for synchronizing physical clocks, and deriving a bound value on how far clocks can get out of synchrony.

Our research is based on the concepts of Lamport's logical clocks and the total ordering of events in a stream. Our event model in Section 5.2.3 is built based on logical clocks and the total ordering of events.

Regarding the order of events we hold the following assumptions in this research:

- **Totally Ordered Events:** We assume the raw event stream is totally ordered. In this research, we do not address the problems of out of synchrony or out of order events.
- **No Out Of Order Event Stream:** We assume during different processing steps/phases the order of events can be kept so that the event stream has the same order before and after different processing steps (the same order upstream can be found downstream).
- **Holding Event Intervals:** We assume time intervals between event instances can be kept in different processing steps.

5.5 Event Processing Parameters

Knowledge-based event processing depends on several factors and parameters that affect event processing properties and system behaviors.

5.5.1 Processing Cost Parameters

In processing events based on knowledge acquired from external KBs we can observe different cost parameters which sum up the total cost for event processing. The following costs are the main parameters which have an effect on throughput and latency of the event processing system. Our aim in analyzing these costs is to find solutions for optimization of these costs and their effects on latency and throughput of CEP system.

- **Cost of Event Acquisition:** Event are normally produced by external event producer systems/devices and are forwarded to event processing agents. Each event forwarded will incur the network transmission cost for transporting the event between event producer and event processing. In some of the CEP systems all the raw events produced by the event producer are transported to the event processing agent. However, in some other systems, only raw events which are required for matching the complex events are transmitted to the event producer [Akd08].

- **Computation Cost of Matching Complex Events:** The cost of computation for matching the complex events depends on the algorithm used for event detection, like rule-based event detection systems, e.g., backward reasoning (SLD-Resolution), forward reasoning (Rete algorithm) and CEP systems based on state automaton. Depending on the number of facts and rules participating in the event detection and the type of stream windowing (number of events in a computation window), the event processing system can have different computation costs.

- **Cost of Background Knowledge Acquisition:** Event processing has to send knowledge queries for acquiring knowledge from external KBs. Acquisition of background knowledge about events produce two types of costs, 1. transmission cost of knowledge to the EPA and 2. computation cost on the KB to produce query results from the external KB (materialization and reasoning).

- **Computation Cost on external KB:** The computation cost on the external KB is twofold, the first one is computation cost similar to database systems like the cost for materialization and computation of joins and filters. The second cost is for description logic reasoning on existing knowledge to infer new knowledge based on them. Different levels of DL reasoning, e.g., RDFS, OWL-Lite, OWL-DL, OWL-Full have different computation costs based on their different complexity and the different number of reasoning rules.

5.5.2 Scalability Parameters

Scalability is one of the important properties of a CEP system which indicates the ability to handle growing amounts of processing work. We consider the following dimensions for the scalability of SCEP system:

- **High Frequent Event Stream:** The input event stream can be of high frequency like, event streams of click stream on an online-shop, event streams of vehicles passing toll booths on a freeway[1] or event streams of all transactions in a bank. In combination with an event stream the sliding window specification in a user query different amounts of event instances can be selected for event processing which highly affects the response time of the CEP system. The combination of sliding window size and event stream frequency can be seen as one of the combined dimensions of CEP systems. For example, a time sliding window of one hour and an event stream with throughput of 1000 events/sec. will cause the EPA to have to handle 3.6 million events in the main memory. The memory required to store them can be calculated according to the size and number of events.

- **Number of Different Event Streams:** In some use cases, the EPA has to process events from different event streams. For example, in extended urban computing (see Chapter 4), the EPA has to process event streams from special product offers, GPS positions of mobile devices in a city, and offer acceptance transactions.

- **Large Number of Event Detection Rules:** For detection of complex events large number of detection rules might be used, and based on them the complexity of event processing will be different.

[1] Linear Road: A Stream Data Management Benchmark `http://www.cs.brandeis.edu/~linearroad/` retrieved April, 2014

- **Large Amounts of Historical Reference Data:** In some cases the event processing system has to compare events with similar events in the past to be able to detect complex patterns.
- **Huge Amounts of Background Knowledge:** In some use cases, huge amounts of background knowledge have to be processed to detect complex events. For example, in eHealth use cases, knowledge bases have to store huge background knowledge about diseases, drugs, health risks and possible drug side effects.

5.5.3 Optimization of Trade-off

An implementation of (SCEP) should provide several acceptable quality of service metrics like low-latency, high-throughput and high scalability. On the other hand, it should also provide expressive reasoning on background knowledge and derive events based on the inference on background knowledge. We can see that there is a trade-off between all these metrics which should be optimized for target use case.

The expected event processing latency and throughput can be defined based on the requirements of each specific use case. Based on this expected latency the other costs should be optimized so that the whole system can provide the services expected.

5.6 Summary

In this chapter, we have provided a description of the basic concepts of knowledge-based event processing. We have described the main concepts for fusion of event stream with background knowledge and the benefits of using background knowledge. Furthermore, we have illustrated an abstract view on semantic-enabled CEP and built a model for events, event stream and the fusion of background knowledge with event stream which will be used in the next chapters. Finally, we have described the most important event processing cost factors and scalability factors.

In the next chapter, we will present our approach for the engineering of required ontologies for semantic-enabled event processing like events, processes, states, actions, and other concepts. Such ontologies are required to conceptualize the relevant concepts for CEP. In our approach to the usage of external knowledge bases and fusion of background knowledge with the stream of events such ontologies are required to represent domain background knowledge.

Chapter 6

Knowledge Representation for Semantic CEP

We describe in this chapter our concept for the representation of ontological background knowledge required for enabling knowledge-based complex event processing. We focus on a methodology for integration domain and application specific ontologies for events, processes, states, actions, and other concepts that relate to change over time. We propose using an extensible and modular ontology representation method which can be reused, customized or extended based on the requirements of different applications and tasks. In our approach, the event processing domain should be described by a modular and layered ontology model which can be reused in different application areas. We describe different upper-level ontologies and illustrate how they should be reused.

Contents

6.1 Introduction

The event processing area requires a special knowledge representation approach. In this chapter, we propose a knowledge representation approach for CEP which integrates domain and application specific ontologies for events, processes, states, actions, and other concepts that relate to change over time. Specific domain, task and application ontologies need to be dynamically connected and integrated into the respective event processing applications, which also leads to a modular integration approach to be used for these ontologies. Capturing domain-specific complex events and generating complex reactions based on them is a fundamental challenge.

Event Models	Description of Application Area / URL
CIDOC CRM [Doe03]	Events in museums and libraries http://cidoc.ics.forth.gr/OWL/cidoc_v4.2.owl
ABC Ontology [Lag01]	Conceptualization of events in digital libraries http://metadata.net/harmony/ABC/ABC.owl
Event Ontology [Rai07]	Description of events in digital music and in general http://purl.org/NET/c4dm/event.owl
DOLCE+DnS Ultralite [Gan03]	Upper ontology for modeling event aspects in social reality http://www.loa-cnr.it/ontologies/DUL.owl
Event-Model-F [Sch09]	Enabling interoperability in distributed event-based systems http://events.semantic-multimedia.org/ontology/2008/12/15/model.owl
OpenCyC Ontology	Description of human consensus reality, upper ontology with lots of terms and assertions http://www.opencyc.org/
VUevent Model	An extension of DOLCE and other event conceptualizations http://semanticweb.cs.vu.nl/2009/04/event/
IPTC. EventML	Collecting and distributing structured event information http://iptc.org/
GEM [Wor04]	The Geospatial Event Model
Event MultiMedia[Ege04]	Event model for multimedia
LODE [Sha09]	Modeling for publishing records of events as Linked Data
CultureSampo [Hyv09, Ruo07]	National Publication System of Cultural Heritage

Table 6.1: Some of Existing Event Ontologies

Event knowledge bases can represent complex event data models which are linkable 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 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.

Knowledge about event types and their hierarchies, e.g., specialization, generalization or other forms of relations between events can be used in event processing. Semantic (meta) models of events can improve the quality of event processing by using event data in combination with ontologies and rules (knowledge bases).

6.2 Related Work on Event Ontologies

Ontologies are the conceptualization of the application domain which allow reasoning on events and other related resources. A number of different event ontologies for modeling events and their relationships have been proposed. Shaw R. et al. [Sha09] provide a comparison of existing event ontologies with an analysis based on their main constituent properties like type, time, space, participation, causality and composition. Each of the event ontologies have been developed to describe events in different application domains, like historical events in museums, digital libraries, musical events, agent-based system. Existing event ontologies describe some of the common concepts of events and vary in other aspects depending on the target application area. For example, *"Event ontology"* [Rai07] has been developed to describe events in conjunction with other music-related ontologies. Table 6.1 lists some of the existing ontologies for representation of events. To the best of our knowledge none of the existing ontologies are specially developed or used for complex event processing.

The existing ontologies are described in different ontology languages like resource description framework schema (RDFS) or web ontology language (OWL) with different levels of expressivity. Different ontology languages with different levels of expressiveness and limitations might be proposed for the description of event processing ontologies. The level of expressiveness of the ontology language and reasoning on events should be defined depending on the use case of the semantic event processing.

6.3 Reusing Existing Ontologies

In ontology engineering literature several methodologies have been proposed, e.g., Methontology [Fer97], On-To-Knowledge [Sur03] or NeOn [SF08]. In such ontology development methodologies ontology reuse is recommended, because it is thought to reduce engineering costs by avoiding having to re-build already existing conceptual models. Apart from reducing costs, reusing existing ontologies increases interoperability from the viewpoint of the Semantic Web, where ontologies are considered shared knowledge [Sim09].

6.4 Upper-Level Ontology Modules for CEP

The idea of upper-level ontologies, such as the Suggested Upper Merged Ontology (SUMO)[1] is to provide a common basis for ontology modules and to allow a bus-like base to connect different modules. The reuse of upper-level ontologies is important to ensure interoperability between different ontology modules. It provides the description of general concepts needed in all ontology modules. For the sake of clarity, an upper-level ontology does not say anything about the relations between different modules. It is a self-contained ontology that provides the semantic model of the foundational notions. The other ontology modules import upper-level ontologies as a whole and can be seen as refinements regarding a specific task, application or domain

We propose that event processing domains be described by a modular and layered ontology model which can be reused in different application areas. Figure 6.1 shows upper-level ontology modules for event processing domain which can capture general top concepts such as event, situation, process, temporal, spatial, action, agent concepts. Event ontology includes core concepts and attributes of events and connection between modules can be realized with, e.g., ε-connection [Haa01], Distributed Description Logics [Bor03]. A temporal ontology captures all of the concepts related to the time of when the event occurs and can be seen as time ontology module. Spatio ontology includes concepts related to the geological information of the events.

We hold that besides a general top event ontology, also specific domain-ontologies and task/application ontologies are needed. For example, to describe an event about *"stock quote price change"* different attributes are needed from those describing an event about *"a state change of a Web service"*. It would never be a complete approach which covers all potential application domains. Events can have different relationships to each other in different domains but a simple hierarchical relationship between events is not satisfying.

Other specific domain, task and application ontologies are modular sub-ontologies of the common top ontologies. They represent more concrete specialized concepts for the application domain. Domain-specific events are modeled as domain classes of the domain ontology and have relationships to other classes of other domain ontologies.

[1] Suggested Upper Merged Ontology (SUMO) `http://www.ontologyportal.org/`

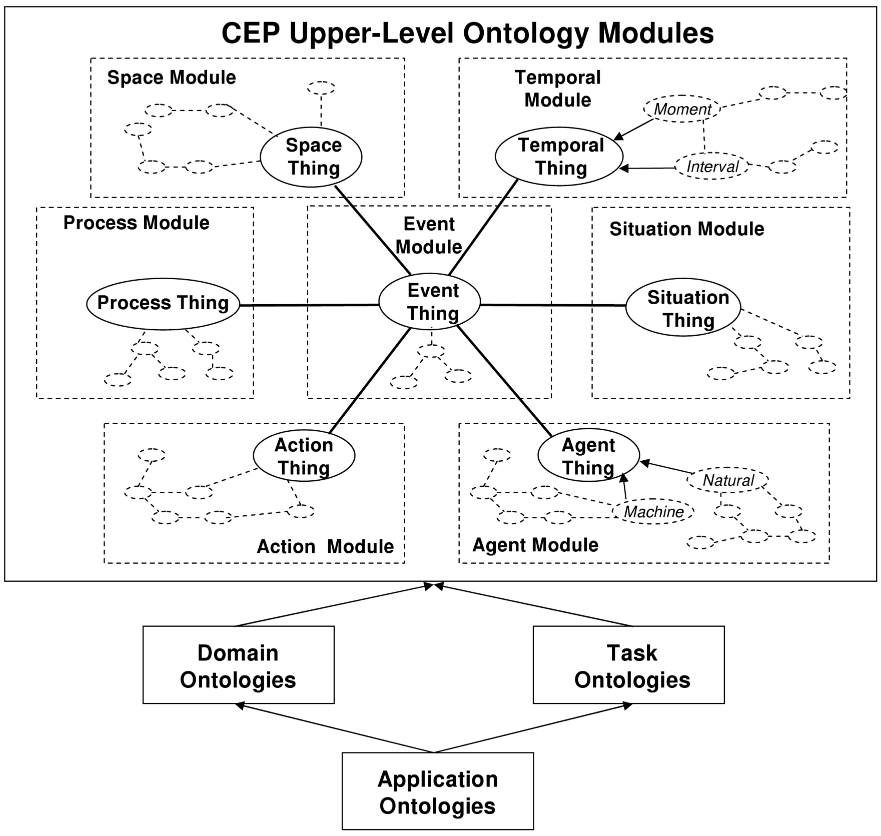


Figure 6.1: Modular Top Level Ontologies

The reasons for modular top ontologies for CEP are the same for ontology and database modularization like, scalability, complexity management, understandability, context-awareness, personalization and reuse [Stu09]. By means of modular ontologies, the event processing system can achieve better scalability for data querying, reasoning on ontologies, evolution and maintenance. Different ontology languages with different levels of expressiveness and limitations might be proposed for the description of CEP ontologies.

6.5 Module Discovery and Development

We used a semantic-driven strategy [Stu09] for our work on event ontology modularization. We tried to extract relevant fragments of ontologies with special meanings and found modules which make sense for final users [Gra06] in different use cases of SCEP. For module discovery and development, we followed the following steps:

- **Step One - Investigation of existing ontologies:** We have investigated existing event ontologies which have conceptualized events in the other research fields. We observed that each of these ontologies have been developed for different use cases and depending on the target use case different concepts are defined. Comparing the existing ontologies, we found a common set of concepts and attributes which are defined in several existing ontologies. It is clear that a common set of concept cannot fit the needs of different applications for event attributes. It seems impossible to develop a single event ontology which can fit all the requirements of different applications. However, it is possible to identify the common core aspects of events like: type, place references, time (period or moment), etc. It can also be observed that the same concepts are represented in different ontologies with different attributes.

- **Step Two - Ask questions about the factual aspects of events:** Our second step was to ask questions about the domain of events. We found the following journalistic questions about events useful to extracting the most important factual aspects of events. The questions are simple questions about occurrences like: *What happened? When do it happen? Where did it happen? Who was involved? Who reported it? Why did it happen? How did it happen? What else could be said about the event?*

  The answers for each of these questions can define one of the important aspects of the surrounding event. These answers can be represented in different ontologies, because they are general aspects which can be defined in standalone ontology modules, like time, place, space, agent, action, process, etc.

- **Step Three - Define competency of module:** After asking competency questions about the application domain of event processing, the sphere of competence for the area for each of the ontology modules could be identified. The competency of each of the ontology module can be specific, e.g., actor ontology module has the task of representing actors, but not any other aspects of surrounding events, like time.

- **Step Four - Collection of the most common concepts of each module:** We looked for the most relevant terms and assertions in different ontologies so that they can conceptualize the domain of event processing in upper-level ontologies. In each module the most relevant concepts and assertions for that module based on the competency of the module were collected. Our aim was to keep the modules simple and lightweight so as to reduce their complexity and increase their reusability. Form relevant existing ontologies the most important and relevant concepts and attributes were extracted for each ontology module. For example, for the agent ontology module existing actor or agent ontologies were used to extract the concepts and assertions.

- **Step Five - Design of module linking and importing:** We tried to define the future potential interfaces of ontology modules so that they have minimal connectivity to each other, i.e., they can be connected from only one or two concepts to each other. A connecting ontology should use these minimal connection points to import the modules for building an ontology and extend it depending on use case requirements. This can increase the understandability and the reusability of ontology modules.

We have identified the following ontology modules which can be defined in the form of upper-level modules for the area of semantic event processing:

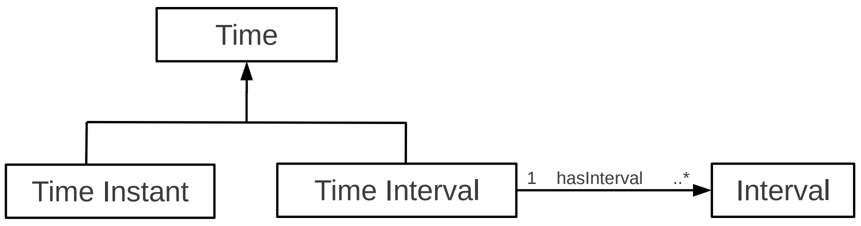


Figure 6.2: A Lightweight Time Ontology

6.5.1 Ontology Modules

Time Ontology Module

Time is the most important attribute of events. In some of the application domains only the timestamp of the event capturing is considered, in others the start and end time of the event, a time interval (time window), or in some cases the duration of event. Different time ontologies [Hob04, Fik02, Zha11] are proposed for the conceptualization of time. A time upper-level ontology module should contain only the main concepts and common parts of these ontologies which can be meaningful and reusable.

Figure 6.2 shows a simple ontology module. It shows two different types of time, time instance and time interval. Based on time instances event detection operators can be specific like time-point-based event or interval-based event detection semantics.

Spatial Ontology Module

Also different spatial ontologies like place ontologies [Hyv07], geo information ontology[2] and place ontologies for the Semantic Web [Abd07, Jon01] are proposed. Some of the existing geo ontoligies are proposed to be used in Geographic Information Systems (GIS) which can improve the quality of these system by adding geo or geospatial semantic information, e.g., for increasing the interoperability between different geo systems.

The Geo ontologies describe the two dimension view of a place, in a space ontology this conceptualization can be extended to a three dimension view. Space ontologies are also proposed for the description of a place in 3D format.

Event Ontology Module

An event ontology module should contain the highest concepts about events like event type. The event type can be utilized to connect the knowledge about the event type to the happened event instance. The subclasses of event type can specialize the event and represent the answer to the question *"What happened?"* The event can have several subclasses in an *is_a* hierarchy.

Figure 6.3 shows a simple ontology module. It includes other concepts like time, space, situation, action, actor and process. These concepts can be further described in their ontology modules. An event can have an event property with one value. An event is divide into two main types, a simple event and a complex event. A complex event is derived or composed of one to several simple events.

[2] http://www.geospatialmeaning.eu/category/geo-ontologies

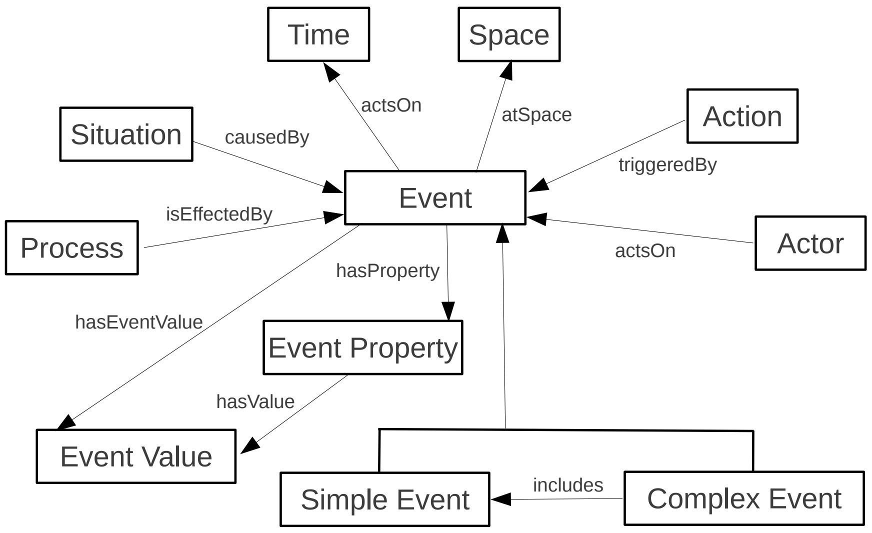


Figure 6.3: A Lightweight and Extendable Event Ontology

Situation Ontology Module

Also several situation ontologies[3] have been proposed. In CEP related literature situations have been considered to be the same as complex events. However, we see that there is a difference between situations and complex events. Situations are the effects of events. A situation is initiated or terminated by one or more (complex) events, i.e., complex event plus context, and holds at a validity interval.

In our ontology, situations are discrete situation-individuals. They have a fixed context like time allocation (date), location and participant. They are not repeatable and are temporally connected. Situation-descriptions assign content and temporal properties to situation-individuals. They describe the dating of situation individuals by mapping into the time. Situation descriptions connect situation types with validity intervals. Situation-types are defined as relation with time argument. They might have several time allocations (validity intervals). They can be repeated and their validity intervals might not be successional. Types of situation-descriptions are based on differences between situation-individuals (sorting of individuals) and are based on the differences between situation with respect to their interaction with the time structure (monotony, homogeneity).

Figure 6.4 shows a simple situation ontology module. We distinguish between two main situations types, heterogeneous situation and homogeneous situations.

Examples of heterogeneous situations are:

- Dynamic Change Situation, e.g., *dynamic changes of traffic lights*
- Time Frame Situation, e.g., *a situation exists only within 5 minutes*
- Frequency Situation, e.g., a situation which happens with regularity like *a telephone rings 3 times.*

Examples of homogeneous situations are:

- State Situation, e.g., *a traffic light is on green.*
- Process Situation, e.g., *a person prepares an email.*

[3]Situation ontologies like http://www.loa-cnr.it/ontologies/ExtendedDnS.owl

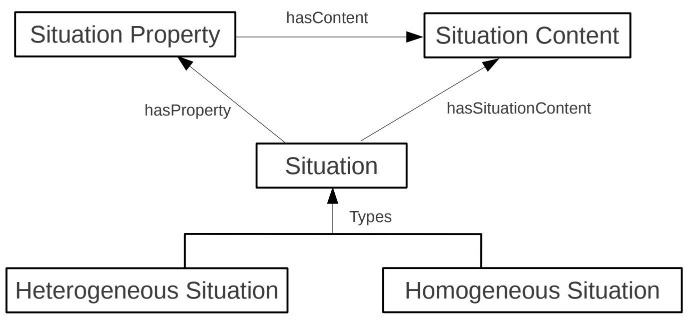


Figure 6.4: A Lightweight and Extendable Situation Ontology

- Iterative Situation, e.g., *a situation in which a person coughs and sneezes.*
- Habitual Situation, e.g., *a person who smokes.*

Any further conceptualizations of situations can be specified and extended to domain and application specific ontologies. The situation ontology can be connected and interlinked to other ontology modules, like event and time ontologies.

Action, Agent and Process Modules

- **Action:** Also several action ontologies[4] have been proposed to represent knowledge about actions. These ontologies can be generalized as an upper-level action ontology module.
- **Agent:** Several agent ontologies like [Lac05] or in the area of multi-agent systems like [Xia04] have been proposed to conceptualize the behaviors of agents. An agent can be considered an actor, actor ontologies like [Gib07] also exist in parallel. An actor can be a human actor or a machine.
- **Process:** Process ontologies like COBRA [Ped08], which include other time or event ontologies, in the area of business process management have been proposed. Other process ontologies can be found in the area of Semantic Web Services and business process execution like [Nit07]. An upper ontology for process description can be built based on these existing ontologies.

6.6 Reusing CEP Ontology Modules

Reusability is one of the main advantages of decomposing complex systems into modules and therefore a key argument for modular ontologies. Instead of creating specific ontologies for each use case, the ability to reuse modules in a flexible manner is important. This presumes knowledge about the modules and how they can be connected and used concertedly. Research on ontology merging, matching and alignment have pointed out that reusing ontologies is a challenging task. Especially, if there are heterogeneous modules that are conceptualized using different ontology languages. In fact, it is necessary to know how to connect modules when they are needed. For this reason, we advocate the necessity of an ontology capable of describing how modules can be linked to each other.

[4]Action ontologies like `http://ontology.ihmc.us/C-Map/Action.html`

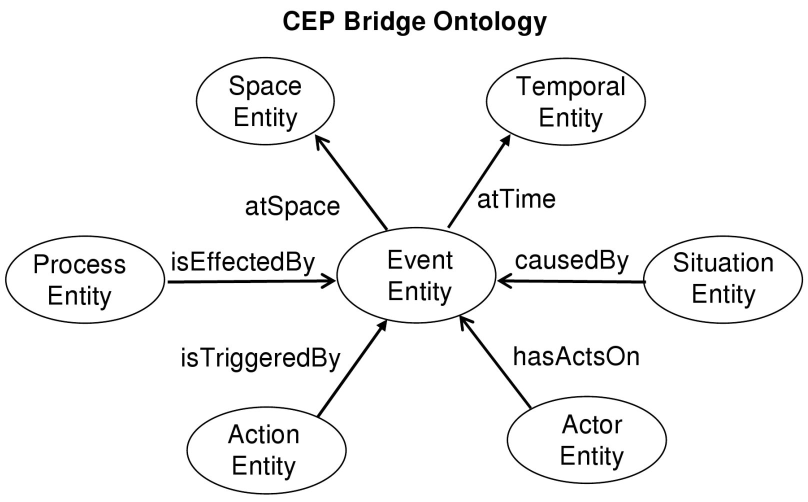


Figure 6.5: Event Processing Bridge Ontology

To reuse the upper-level CEP ontology modules above, we have developed a general connector ontology, as shown in Figure 6.5. This ontology is a bridge ontology for linking developed modules. Based on use case requirements, several bridge ontologies might be developed for different combinations of modules. For example, in Business Activity Monitoring (BAM)[5] use case only event, time and process modules might be interesting while other modules like space modules should not be imported.

6.7 Usage of Ontologies in Semantic CEP

The ontologies proposed in this chapter can be used to conceptualize relevant concepts for event processing. Knowledge data instances that can be generated by using the ontologies proposed has multiple usage potentials in different event processing approaches. In the following, we mention some of these usage scenarios and how these ontologies can be used in our event processing approaches (described in the upcoming chapters).

- By using the proposed ontologies, the required knowledge about events and other related concepts can be generated and stored in a knowledge base. An inference engine can use this knowledge base for inferencing on the basis of the specified expressive level. The inference engine requires polling and querying the knowledge base in order to extract facts from it. Steady polling of the knowledge base might cause performance problems when the size of the data (instances) enlarges over time (e.g., due to storage of the event stream history). This approach is discussed in the upcoming chapters, which describes the benefits and disadvantages of steady polling the knowledge databases.
- Our basic model for event streams in subsequent chapters (5, 8, 9 and 10) is based on the assumption that we have event objects described using attribute value pairs, and we can build a link from the attribute values to the resources in the knowledge base. The proposed ontologies in this chapter help to model, understand and integrate the attribute/value tuples so that we can link them to external KBs and use them for event processing.

[5] `http://www.gartner.com/it-glossary/bam-business-activity-monitoring/` retrieved April, 2014

- The proposed ontologies can be used for describing the event stream history and history of the occurred situations, actions and other related concepts. The generated data can be stored and integrated into the existing KB so that it can be used for CEP. This kind of knowledge about the history of data streams can be integrated with other domain knowledge data. In the final chapter of this thesis in Section 11.5.1, we describe an approach which extends our basic model and uses the knowledge about the history of complex events for event detection.

6.8 Summary

In this chapter, we have presented our work on developing upper-level ontology modules for complex event processing which can be reused for different use cases. Several existing ontologies have been investigated to discover their commonality for building lightweight upper ontology modules. We have described our approach of using journalistic questions to find the factual aspects of events to discover ontology modules surrounding the concept of events. We have discussed the main concepts of each ontology modules and described them briefly.

The main contribution of this chapter is the consideration of ontologies for CEP as upper-level ontologies which can be adopted and extended depending on target domains and applications.

Chapter 7

Representation of Complex Event Patterns

Representation of complex event pattern based on event correlations and relations in background knowledge is described in this chapter. We get into the details of expression of complex events based on event algebra operations and relations in background knowledge. We propose a hybrid query language to define patterns based on the fusion of knowledge from event stream and external knowledge bases. We show the possibility of expressing event detection queries based on relations in background knowledge without getting into the details of language design for semantic-enabled CEP. Furthermore, we propose using declarative rules to express complex event patterns and show the most relevant categories for such hybrid query rules.

Contents

7.1 Introduction

As we have described, existing event processing languages (see Section 3.8) fail to provide the required expressiveness for describing complex event detection pattern based on fusion of background knowledge and event correlations. In this chapter, we get into the details of requirements and concepts for an event detection query language for knowledge-based event processing.

A query language for knowledge-based event processing should have two main expressiveness attributes. It has to provide main event processing expressiveness based on event algebra operations and matching event attributes. On the other side, knowledge-based event processing requires the expressiveness of event detection based on relations of resources in background knowledge. The formal semantic and syntax of event processing language should allow the extraction of knowledge from external knowledge bases, and event detection based on description logic inferencing on knowledge stored in external KBs. The event query has a hybrid semantic because it blends event algebra operators with query parts to include background knowledge.

As described in Chapter 5 the knowledge on KBs are stored in graph-based data format like RDF. This data format can enable the querying of knowledge and improving data integration capabilities. The usage of SPARQL [spa08] as a graph pattern matching language (RDF query language) can be exploited for event filtering by writing event patterns in terms of SPARQL queries, so that an event pattern can be defined using a graph pattern.

Existing extensions of SPARQL query language are designed to address RDF stream data and are not suitable for the fusion and integration of background knowledge and streams of events. The background knowledge is RDF formatted data and is the static part (more static because of low rate updates) of the information, while the stream of events might not always be RDF formatted data. Furthermore, background knowledge about the events might be collected from different sources of background knowledge.

We introduce in this chapter declarative rules which can be used as query pattern for the detection of events based on the fusion of event stream with background knowledge. This chapter introduces our concepts for a hybrid language for event processing and is organized as follows: We start the chapter with the description of formal semantics for event algebra operations based on time intervals in Section 7.2. Subsequently in section 7.3, we describe the detection patterns for extraction of knowledge from external KBs. Section 7.4 presents hybrid query rules. In section 7.5 we describe the most relevant categories of event query rules and their differences. We discuss the problems of query language design in section 7.6, and conclude the chapter.

7.2 Event Detection Operators

In this section, we describe the event detection operators that we use for SCEP, and in the following sections we describe how we combine these operations with query for the knowledge extraction of external KBs.

As we have defined in Section 5.2.3, an *Event* is a tuple of $\langle \bar{a}, \bar{t} \rangle$ where $\bar{a}$ is a multiset of fields $\bar{a} = (a_1, ..., a_n)$, and is defined by the schema $\mathbb{S}$. $\bar{t} = (t_s, ..., t_e)$'s a time-ordered sequence of timestamps representing the different occurrence times of the event, the start t_s and end timestamps t_e of the event. $\bar{t} = (t_1, ..., t_n)$ is a sequence of times in which t_1 is t_s, and t_n is t_e

Our event detection patterns are defined based on interval-based semantic for event detection. Each event is represented by the sequence of times $\bar{t} = (t_1, ..., t_n)$, one to many timestamps. Let's consider a time window w which is the time period between timestamps t_{ws} and t_{we}.

We define the operational semantics for the four main event detection operations, SEQ, AND, OR and NOT from window w (a time or count window) as follows:

- SEQ**:** The $\mathbb{SEQ}$ operation detects the event e_1 with $\bar{t^1} = (t_s^1, \ldots, t_e^1)$ and e_2 with $\bar{t^2} = (t_s^2, \ldots, t_e^2)$, if and only if timestamps t_s^2 is equal or later than time t_s^1, and are inside the window w.

  $$\mathsf{SEQ}(e_1, e_2)[w] = \forall(t_s^1, t_s^2)(e_1(\bar{t^1}) \wedge e_2(\bar{t^2}) \wedge t_s^1 \leq t_s^2 \wedge (e_1, e_2) \in w)$$

- AND**:** In the case that the two events happens inside window w.

  $$\mathsf{AND}(e_1, e_2)[w] = \forall(t_s^1, t_s^2)(e_1(\bar{t^1}) \wedge e_2(\bar{t^2}) \wedge (e_1, e_2) \in w)$$

- OR **:** One of the e_1 or e_2 happens in window w.

  $$\mathsf{OR}(e_1, e_2)[w] = \forall(t_s^1, t_s^2)((e_1(\bar{t^1}) \vee e_2(\bar{t^2})) \wedge (e_1, e_2) \in w)$$

- $(e1, \mathsf{NOT}$ **:** In the case that e_1 happens in window w and e_2 does not happen.

  $$(e_1, \mathsf{NOT}(e_2))[w] = e_1(\bar{t^1}) \wedge (e_1) \in w \wedge e_2(\bar{t^2}) \notin w$$

7.2.1 Further Operations

Further event detection operations based on time intervals can be defined. Let us consider two event instances e_1 and e_2,

$$\forall(t_s^1, t_s^2)(e_1(\bar{t^1}) \wedge e_2(\bar{t^2}) \wedge (e_1, e_2) \in w \;\; \wedge$$

- $\mathsf{BEFORE}(e_1, e_2)[w] \rightarrow (t_e^1 < t_s^2) \wedge (t_s^2 > t_e^1)$
- $\mathsf{MEETS}(e_1, e_2)[w] \rightarrow (t_s^1 \leq t_s^2) \wedge (t_s^2 > t_e^1)$
- $\mathsf{OVERLAPS}(e_1, e_2)[w] \rightarrow (t_s^2 \in \bar{t^1}) \wedge (t_e^2 > t_e^1)$
- $\mathsf{FINISHES}(e_1, e_2)[w] \rightarrow (t_e^1 = t_e^1)$
- $\mathsf{INCLUDES}(e_1, e_2)[w] \rightarrow \bar{t^2} \in \bar{t^1}$
- $\mathsf{STARTS}(e_1, e_2)[w] \rightarrow (t_s^1 = t_s^2)$
- $\mathsf{COINCIDES}(e_1, e_2)[w] \rightarrow (t_s^1 = t_s^2) \wedge (t_e^1 = t_e^2)$

Similar to Snoop [Cha94a, Cha94b] (described in Chapter 3, Section **??**) event detection operators, further operators like aperiodic and periodic operations can be considered.

7.3 Querying Background Knowledge

We use attribute-value tuples of an event instance to query external KBs about relevant background knowledge of that event instance. The attribute/values of an event instance are linked to resources in background knowledge, so that we can extract knowledge about the events. We have defined the *Event2KB Link* (Chapter 5, Definition 5.5) which is used to identify corresponding resource within the RDF graph. In this way it is possible to use SPARQL queries to extract knowledge about linked resources in KBs.

The concept of basic graph patterns (RDF Basic Triple Patterns (BGPs)) is introduced as sets of triple patterns in the context of SPARQL queries. An event pattern can be defined using a graph pattern. We use BGPs to express triple patterns, and include within it an event detection pattern. The formal semantics of BGPs are along the same lines as defined in SPARQL [spa08, Wal07a].

Figure 7.1 shows an example of a tree-shaped query which can be used to extract knowledge about an event. Triple *(?e2 :p1 ?s1)* is an Event2KB link, which can connect an event instance to a resource in background knowledge. In this pattern ?s1 is an unbounded variable in the triple pattern and :s4 and :s5 are resources which are identified by URIs. Most of the predicates in the triple patterns are bounded (are not variables) in tree-shaped queries.

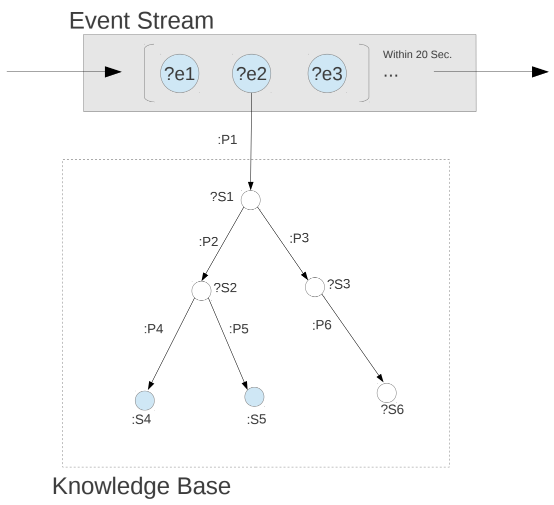


Figure 7.1: A Tree-Shaped Knowledge Query

7.4 Declarative Rules as Hybrid Event Detection Pattern

In this section, we introduce declarative rules as hybrid patterns to express the event detection pattern for the detection of events based on integrated knowledge. Event query rules are declarative rules which can be used to express patterns of complex events. The aggregated knowledge from event streams and background KBs can be queried by different types of event queries. These event queries have hybrid semantic, because they use event operation algebra to detect events. We use BGPS to define patterns on background knowledge and combine these patterns with the event detection operations introduced.

Figure 7.2 shows an abstract example of an event query rule (event detection pattern) which combines event algebra operations with SPARQL basic graph pattern query blocks. It defines a pattern for three events in a time window which are defined with *SEQ* and *AND* event detection operators. Each event instance (e1, e2, e3) is linked to resource patterns from the KB, so that the BGPs define a pattern for query background knowledge about these events. The graph pattern blocks can be separately considered as SPARQL query blocks that can be sent to KB to extract knowledge. Such complex queries like in Figure 7.2, can be used to discover background knowledge relations about events because events are not only detected based on event operations (SEQ, AND) but more based on their relations in the KB.

In the example presented in Figure 7.2, all the predicates express the connection of Subject to Object by a single predicate.

One further event detection pattern for relation detection is shown in Figure 7.1. Event e1 is connected to resource s1 in background knowledge by predicate c1. In the same way are other events e2 and e3 connected to resources in KB. However, predicate placeholder $p*$ presents the connection of *"Subject"* to *"Object"* through any arbitrary number of predicates which can connect the subject to the object. In this way, we can represent the connection of events through any kind of background knowledge relations. This kind of graph patterns is defined in SPARQL 1.1 Query Language[1] and is called property paths.

[1] http://www.w3.org/TR/sparql11-query/#propertypaths

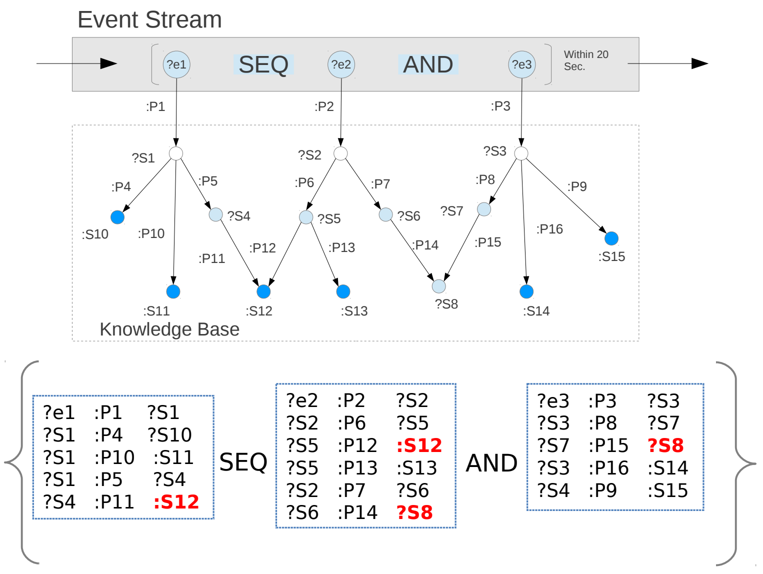


Figure 7.2: An Example of a Hybrid Event Query based on Relations in Background Knowledge

```
{ {(?e1, c1, ?s1) (?s1, p*, ?s)}

 {[?e1 SEQ ?e2]}

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

 {[?e2 SEQ e3] }

 {(?e3, c3, ?s3)  (?s3, p*, ?s)}  }  [Within 5 min.]
```

Listing 7.1: Pseudocode of Event Detection Pattern

7.5 Categories of Event Query Rules

Event query rules can be categorized into several categories based on the usage of knowledge queries (SPARQL queries, BGPs blocks) inside the event query rule. Before we start with the description of event query rules, we need to define some terms.

Definition 7.1. ***Background Knowledge Query (KQuery)*** *refers to the SPARQL BGP query block within the event detection pattern as KQuery.*

Definition 7.2. ***Event sQuery Rule*** *A sQuery Rule is a declarative rule which defines a detection pattern for user-defined complex events and can include one or more set of KQueries to query external KBs. With the term event sQuery rule we refer to the whole event detection rule which includes set of triple patterns and are combined with event algebra operations (e.g. AND, SEQ).*

Definition 7.3. ***Event Pattern Rule (ePattern)*** *is an event detection rule without the use of KQuery predicates. With the term ePattern, we identify only the event detection part which includes event algebra operations.*

Our criteria for these categorizations are based on the following factors:

1. The number of KQueries in the defined detection rule *(whether it is a single block or several KQueries)*
2. Whether the rule defines any kind of relation discovery between events, so that Kquery includes variables depending on incoming event data. Whether query is generated based on event attributes or it is independent (Query describes only the semantic type of the event). *(Generated or Not Generated)*
3. With generated KQueries, it is the number of event attributes used for generating KQueries. *(Single attribute or several attributes)*
4. With generated KQueries 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)*

We describe in the following 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 combinations which can be processed using different event processing approaches.

Event query rules can be categorized into several categories based on the usage of knowledge queries (SPARQL queries) inside the query rule. Before, we begin with the description of different rules, we will provide two definitions for *Event sQuery Rule* and *Event Pattern Rule*.

In the following, 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 rather to emphasize interesting rule combinations which can be processed using different event processing approaches. We provide for each category an example in the form of a pseudocode example. The terms used in these examples are described in Table 7.1.

We provide examples for each of the event query types which are in a pseudocode of a declarative rule language. The rule language is similar to the Prova rule language[2] that can be used for event detection.

Category A - Single KQuery

In this category, the event sQuery rule includes only one single knowledge query and uses its results in one or more variables within the event detection rule. A KQuery is used to import knowledge about event instances or types. One or more tuples of events are used to build the basic triple pattern inside KQuery. 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.

The event processing engine can use external information resources to import knowledge about an event instance so that it is able to detect complex events based on their relations to resources in background knowledge and its streaming sequence. Category-A event sQuery rules can be categorized into three subcategories:

[2]http://prova.ws/ describe in Chapter 3

Terms	**Description**
EStream	Raw event stream
EQuery	Event pattern query
SQuery	SPARQL query part of event sQuery rule
KB_id	ID of the target KB
SResult	Result set of the SPARQL query
UDef	Event type tuples defined by users
ETuple	Event instance tuples defined by UDef
sparq_select	Rule predicate used to querying the external KB

Table 7.1: Summary of the Symbols Used in Rule Examples

Category A1 - Raw KQuery

This category of sQuery rule is the simplest form of event query rules. The KQuery included is only about the resources in background knowledge. Background knowledge query is independent from event stream, however complex event detection is defined on the results of this query in combination with the event stream. In some cases, on each event KQuery should be resent to the KB to query the latest data version.

One example of this case is when a stock broker wants to detect a complex market event and defines a high level event query like this: *“Notify when the shares of automotive companies in Germany, which have laid off more than 10% of their employees in the past two days, increase by 2%.”* The broker can have a handling strategy to monitor such stock market shares without even knowing the exact names of these shares. He assumes that such shares will always increase when companies lay off more than 10% of their employees.

In Listing 7.2 a pseudocode example of an event sQuery rule in category A1 is shown. The variables used are described in Table 7.1.

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

Listing 7.2: Example of a Category A1 Event Query Rule.

Category A2 - Generated KQuery

In this category of sQuery rules each incoming event generates a different KQuery which is then sent to the target knowledge base. One example of this case is: *“Notify when the shares of software companies, which in the last minute have published news about their new software products and have increase by 2%.”* In this case fresh background knowledge about software companies (last 5 min.) is required to know which stock market shares might be interesting. With each 2 percent increase in price, it should be checked if it is the share of software companies which have published good news about their software products. The attribute/values of an event instance are used to generate basic triple patterns of a KQuery. Based on user definitions (UDef) some of the tuples (ETuple) (attribute, value) of an event instance are selected and used to generate a single SPARQL Query. Listing 7.3 shows a pseudocode example for such an event sQuery rule.

```
stream(CEvents):-
  ETuple = getSingelEvent(EStream,UDef),
  SQuery = generateSPARQL(UQuery, ETuple),
  SResults = sparql_select(KB_id, SPQuery),
  eProcessing(SResults, EStream, EQuery).
```

Listing 7.3: Example of Category A2 Event Query Rule.

Category A3 - Generated SPARQL from Multiple Events

This category of sQuery rules is similar to category A2, but the KQuery is generated from multiple events. Within a sliding window (e.g. a time window) of two or more events a single KQuery is generated. One example of this case is: *"Notify when the shares of state banks have increased by 2%, which have financed metal companies in Germany whose shares have increased by 4% and published good news about their products in the last 5 min."* The detection of this event also requires updated knowledge from the KB in order to know which banks financed which companies and which one published good news.

Multiple events are used to generate a single KQuery, the CEP engine can wait for new event messages and then generate a SPARQL query based on the emitted events, and query for background knowledge about them. In most cases, this category of sQuery rules can be rewritten to two or more joint A2 or A1 rules.

Category B - Several KQueries

Queries of this category include several KQueries and combine them with event detection rules. This means that several category A rules are combined to build category B. Category B rules is able to combine results from KBs with events using event operation algebra. This category of rules makes it possible to build a map between background knowledge and the relations of events to each other. An example of this category is an event query rule for the detection of the complex events (from use case high-level stock market monitoring) shown in Figure 4.1.

A stock broker is interested being notified when the stock price of companies, which depend on each other for material supply and financial support, have fallen. The CEP engine needs to know which products this company is currently producing, which other companies might need this material and which companies are currently financing these companies.

In Listing 7.4 the pseudocode for an event query rule of category B is shown, two KQueries are generated based on two incoming events and the results are used for event detection. KQueries can be generated and executed in sequence based on the sequence of events, or they can be processed in parallel.

```
stream(cEvents):-

  ETuples1 = getEvents(EStream, UDef1),
  ETuples2 = getEvents(EStream, UDef2),

  SQuery1 = generateSPARQL(UQuery, ETuples1),
  SQuery2 = generateSPARQL(UQuery, ETuples2),

  SResults1 = sparql_select(KB_id, SQuery1),
  SResults2 = sparql_select(KB_id, SQuery2),
eProcessing(SResults1, SResults2, EStream, EQuery).
```

Listing 7.4: Example of a Category B1 Event Query Rule.

Category B1 - Several KQueries Combined with Event Operations

Category B1 is based on category B, but the results from the KQuerys predicates are combined with AND, OR, SEQ or similar event algebra operations. The whole query is evaluated on sliding windows of event streams. KQuery predicates are not dependent on each other, i.e., the results from one is not used in another KQueries, so that they are not dependent of the results of the other KQueries.

Because of the fact that this category of rules requires multi-step knowledge acquisition from external KBs, they can be executed based on some pre-calculated processing plans. For this category of rules, it is possible to find a cost-optimal execution plan. This processing approach is discussed in the following chapters about processing approaches.

```
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).
```

Listing 7.5: Example of a Category B2 Event Query Rule.

Category B2 - Chaining KQueries

In category B2 several KQueries 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 event instances included in a slide of event stream (by means of a sliding window, counting or timing window). This means different sliding windows can be defined to wait until some event happens and then a KQuery is executed. SPARQL queries might be defined in a sequence chain. The results are directly used for event processing or used in another follow-up KQuery.

Category B2 can include several category-A rules, because it contains several KQueries which can be raw or generated KQueries. Listing 7.5 presents a pseudocode of an event query rule of category B1 in which two KQueries are generated.

Category B3 - Chained and Combined KQueries

In this category KQueries are used in combination with all possible event algebra operations. Event operations are used to combine the results from several KQueries or several KQueries 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 attributes/values from incoming events are used for the generation of complex KQueries.

7.6 Discussion on SCEP Query Language Design

A SQL-like query language for event processing has a better life-cycle economy than the same query in complex declarative rules. Non-expert users can better formulate their subscriptions in visualized tools which then create and manage such SQL-like queries. Different SQL-like query languages have been proposed for event processing, which only provide the expressiveness to detect events based on attributes of events or event correlations.

7.7 Summary

Designing an event processing query language is a challenging task. It should satisfy the requirements of event processing use cases and be justified with involved parties from industry and academia, and can only be done in a standardization process. In this work we have proposed a query language which allows us to express event detection patterns based on event operation algebra and relations in background knowledge.

In the following chapters we introduce methods to process these queries so that we can optimize the trade-offs between real-time processing and complex reasoning on external KBs and processing costs.

Part III

Knowledge-Based Event Processing Approaches

Chapter 8
Semantic Enrichment of Event Stream

One of the approaches for the fusion of events and background knowledge is the enrichment of the event stream prior to event processing so that event processing agents can process events based on background knowledge relations. This chapter provides a description of our concepts for the semantic enrichment of event streams. We describe the different components of the event enrichment process and illustrate possible architectures to achieve the required scalability and throughput. We analyze the different costs of event enrichment and describe the most important impact factors. We propose an approach for multi-step event enrichment and detection which process events in multi-passing through several event processing steps. Furthermore, we provide an algorithm for finding an optimized plan for event enrichment and detection in order to optimize the caused load on external knowledge bases for knowledge acquisition. Our aim is to find a low-cost event detection plan while meeting user-specified latency expectations. Finally, we discuss the advantages and disadvantages of the event enrichment approach and conclude the chapter.

Contents

8.1 Introduction

Our approach for knowledge-based CEP uses several information resources which can be aggregated to build an overall knowledge base for event processing. The event processing agent has the task of using data from different resources for the detection of complex events. The knowledge available for the semantic-enabled event processing (SCEP) is $KB_{SCEP} = KB_b \cup KB_s \cup KB_q$ in which KB_s is knowledge from the incoming event stream, KB_q is knowledge extracted from the user-given event detection query and KB_b is background knowledge about the events, situation, actions, processes and other related concepts.

For the integration and aggregation of background domain knowledge of the incoming event stream and the timely processing of the whole knowledge a highly scalable and real time processing approach is required. Particularly, inference on the huge amount of background knowledge can increase processing time based on inference semantics and amount of knowledge.

One basic approach for processing events is storage-based which is the processing of events after their storage on a database system and continuously polling the database on each arriving event (described in Section 3.5.5). For the realization of knowledge-based event processing the required domain knowledge can be stored on external KBs so that CEP engine can poll the KB on each arriving event. The results from the KB can then be processed with the event data stream. This approach may have scalability and performance problems when the throughput of the event stream is high, the size of background knowledge is large or expressive reasoning should be realized. The main disadvantage of this approach is that processing is only possible after storage/indexing of data and databases are polled with each new incoming event. This implementation might be applicable for use cases that do not have high event throughput or huge amounts of background knowledge to process. Scalability and real-time processing are the main shortcomings of this approach which makes it impossible for use in time-sensitive use cases.

For improvement of scalability and throughput performance of knowledge-based event processing, we propose the Semantic Enrichment of Event Streams (SEES) approach, which is an approach for the enrichment of events with ontological background knowledge prior to a syntactic event processing step. The event stream is enriched with newly derived events or newly added attribute/values to each event instance. The events derived are generated from raw events and are only generated for internal usage.

An initial effort for semantic enrichment of events is realized on publish subscribe systems [Pet03] using a taxonomy of related concepts and events mapped to super types of other events for semantic matching to user subscriptions. However the main subject of this approach is to match similar events to users' subscriptions, and there is no description logic reasoning on background knowledge about events.

Zhou et al. [Qun12] developed a system for semantic processing of events called *SCEPter*[1]. The *SCEPter* approach can annotate/enrich events with metadata and is able to enrich events for each incoming event. Annotations are utilized to classify or filter different categories of event instances, and they provide concepts for the combination with *historical event data* and propose concepts of *query rewriting*. However, *SCEPter* does not provide concepts for the detection of events based on the relations of events and other related concepts in background knowledge bases.

We extend existing approaches for event enrichment with further concepts for the detection of events based on their complex background knowledge and with providing concepts for planning event enrichment process. The planning of event enrichment and detection steps can be used to optimize the costs of event enrichment.

The rest of this chapter is organized as follows: First, we provide a description of our approach for Semantic Enrichment of Event Stream (SEES) and discuss the distributed setting of event enrichment on an event processing network (Section 8.2). Different costs for event enrichment and detection are discussed to find usable metrics for measuring costs (Section 8.2.5). Then we describe how we extract subgraphs from the user queries to use them in multi-step processing approach and based on them we provide concepts for planning event enrichment to optimize costs while the system satisfies user-required latency expectations (Section 8.3). Finally, we discuss the advantages and disadvantages of semantic enrichment of event processing, and conclude the chapter.

[1]SCEPter is also described in Chapter 3, due to its relevance to the subject of this chapter, we repeated it here.

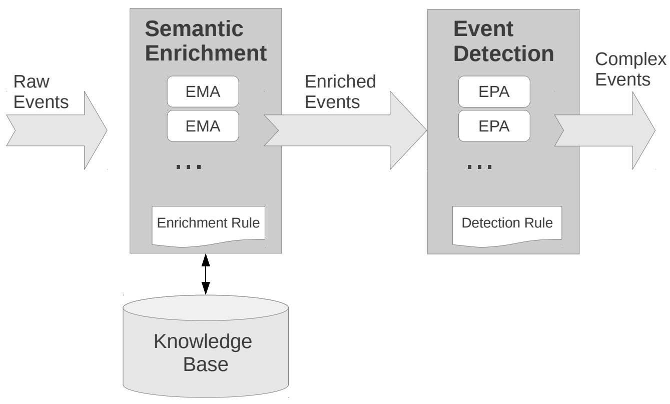


Figure 8.1: Semantic Enrichment of Event Stream (SEES)

8.2 Semantic Enrichment

SEES is about the enrichment of events with background knowledge prior to complex event detection with new derived event attributes. In an Event Processing Network (Event Processing Network (EPN)), several Event Mapping Agents (Event Mapping Agents (EMAs)) have the task of generating derived events by querying external KBs and executing reasoning on existing knowledge. Mapping agents can be replicated to achieve better scalability. In the following processing step, the enriched event stream can be monitored by multiple event processing agents. EMAs can be replicated and deployed in parallel to achieve efficient scalability with respect to throughput.

Definition 8.1. ***(Event Mapping Agents (EMA))*** *An Event Mapping Agent (EMA) is a software system that have the task of receiving events and generating newly derived events by querying external KBs and mapping incoming events to new events. Derived events are completely new events or are received raw events with updated attributes (adding new fields or removing old ones).*

In order to enrich the event stream, new events can be derived from raw event instances. Such derived events can contain attributes that are *inferred* from external knowledge bases. Figure 8.1 illustrates the process of semantic enrichment of an event stream. As discussed before, the raw event stream is enriched by one or many EMAs resulting in an enriched outbound event stream processed by a set of EPAs in order to detect complex events.

Figure 8.2 shows the parallel setup of event enrichment in an EPN. The main event stream is split into several sub-streams. The task of event enrichment can be distributed and shuffled into several (EMAs). Each EMA uses a shared or dedicated KB. Task scheduling can be realized by using simple round-robin scheduling or load distribution based on the processing capacity and load of each EMAs. The parallel setup of the event enrichment approach is similar to the concept of map reduce [Dea08], incoming events are mapped to new events and then a CEP engine reduces the events to detected complex events.

Figure 8.3 illustrates a sequential setting of multi-step event enrichment and detection. In each step the input event stream is enriched and the event detection process is executed on the enriched event stream. The stepwise processing of events is useful for avoiding the full enrichment of the complete event stream; in each step a module of background knowledge can be enriched to event stream and used for event detection. In this way, the system can optimize the cost for event enrichment. This kind of optimization is the subject of Section 8.3.

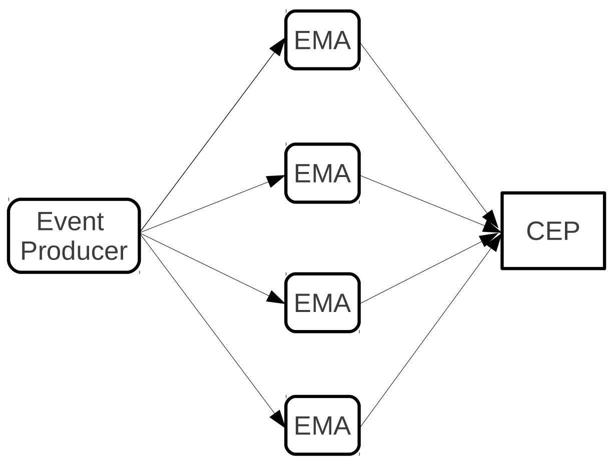


Figure 8.2: Parallel Setup of Event Enrichment

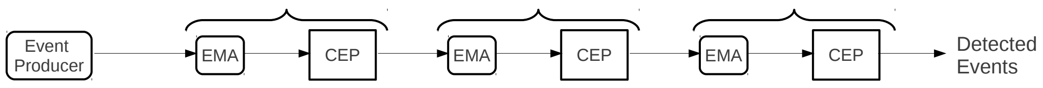


Figure 8.3: Serial Multi-Step Event Enrichment and Detection

A combination of both parallel and serial approaches is also possible so that in each parallel enrichment process a sequential enrichment process is organized to optimize enrichment costs.

8.2.1 Assumptions

Regarding our SEES approach we make the following assumptions:

- **Reliable Communication Channels:** We assume that all network communication between components, i.e., KB, EMAs, Event Processing Agents (EPAs) are reliable communication channels and the communication network has fault tolerance properties. Communication between each EMA and the KB is a client-server communication, in which the EMAs are in the role of the client and the KB the server.

- **External Knowledge Bases Provide ACID Properties:** External knowledge bases are a special kind of databases optimized for the storage of Data in RDF triple format. These knowledge bases have ACID properties (Atomicity, Consistency, Isolation and Durability), as specified by Jim Gray [Gra81] for Database transactions. Additionally, we consider a timeout for knowledge retrieval from external KBs, in the case that query evaluation on the knowledge base takes longer than a specific timeout, the query is returned with null results.

- **SCEP in SEES Setting Provides BASE Properties:** The usage of semantic enrichment for the realization of SCEP disposes the system to not being able to provide ACID Properties (Atomicity, Consistency, Isolation, Durability) (ACID) properties, because of reasons for high scalability and performance consideration. However we consider the SCEP system capable of having BASE (Basic Availability, Soft-state, Eventual consistency) properties, as described in [Vog08, Gil02].

- **Entropy of Event Stream:** We assume that event stream entropy (described in Section 3.6.1) varies only within a certain value ϕ. If we monitor the event stream in large time intervals (larger than the monitoring window of the queries) then the entropy of the event stream only varies in $\pm\phi$. Entropy is a factor that defines how the event stream is generated from different event types.

8.2.2 User Queries for Semantic Enrichment

As described in Chapter 7, the given user query $sQuery$ is a hybrid query and a combination of two parts:

1. Event detection pattern (using event operation algebra)
2. Query pattern for background knowledge (Basic Graph Patterns BGPs, defined in SPARQL).

For the semantic enrichment of events, the following two cases for defining user queries are possible:

1. The user can define two separate queries, one for semantic enrichment and the other for the detection of events from the enriched event stream.
2. The CEP system extracts two different queries from the user query given and extracts two separate queries for enrichment and event detection.

In our work both of the cases above are considered and applied. Nevertheless for the sake of simplicity, in the next sections we suppose the user defines two separate queries for enrichment and event detection.

8.2.3 Types of Event Enrichment

Event stream enrichment can be categorized into two types of event enrichment. In the first type the system generates new events and adds them to the output event stream in addition to raw instances. The second type is the fat event generation, i.e., generation of events with more attributes/values.

Definition 10.3 defines Event as a tuple of $\langle \bar{a}, \bar{t} \rangle$, where $\bar{a}$ is a multiset of fields $\bar{a} = (a_1, ..., a_n)$, and is defined by schema $\mathbb{S}$. In the process of event enrichment we will add new attributes to the multiset of fields of event instances or generate new event instances.

Enrichment of event streams is possible in the following two cases:

- **New Event Generation:** In the case of new event generation, (EMAs) can derive new event instances from each singular event instance and add them to the event stream. From a single event notification tuple they produce several event instances and send them downstream. The generated events are from the same type of incoming stream or are generated into new event types based on the enrichment of events.

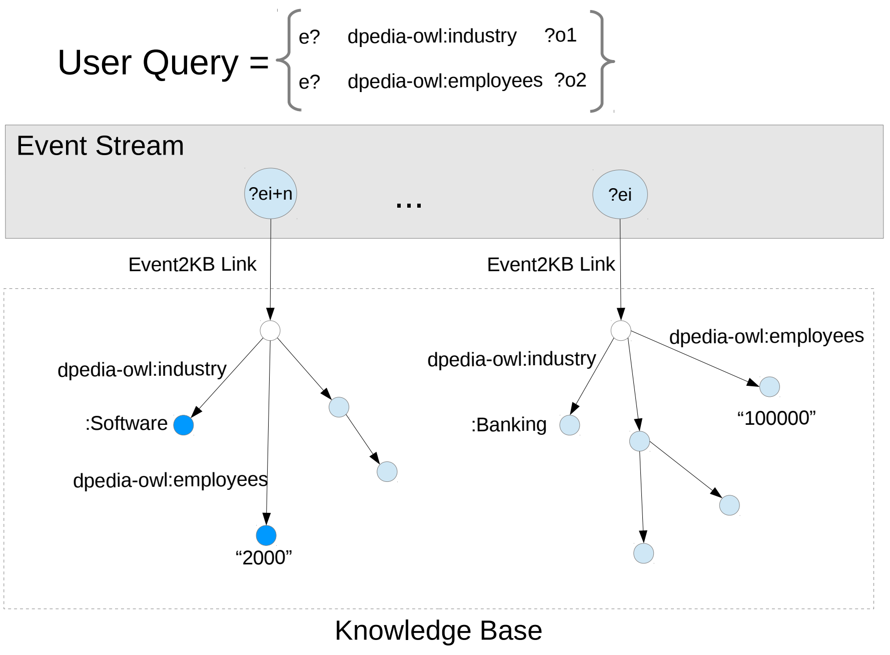


Figure 8.4: An Example of Simple Enrichment of Events

- **Fat Event Generation:** In the fat event generation approach, EMAs derive new attribute fields for each event instances and add them to existing attributes of events. For each event $\langle \bar{a}, t_s, t_e \rangle$ the multiset of fields $\bar{a} = (a_1, ..., a_n)$ are extended to new fields and a is extend to $\bar{a} = (a_1, ..., a_(n+m))$ and m is the number of added fields.

In this research, we use the fat event generation approach and add new attributes to event instances.

8.2.4 Forms of Event Enrichment and Detection

Based on the event detection query different forms of event enrichment for event detection is possible. We distinguish between two forms of event enrichment.

```
PREFIX dbpprop: <http://dbpedia.org/property/>
PREFIX dbpedia: <http://dbpedia.org/resource/>
PREFIX dbpedia-owl: <http://dbpedia.org/ontology/>

select * from pattern[every tick=StockTick where timer:within(10 sec)]
   SPARQLPATTERN [define tick.url as ?url] {
  ?url dbpedia-owl:industry dbpedia:Computer_software .
  ?url dbpedia-owl:location dbpedia:United_States .
  ?url dbpprop:products dbpedia:Computer_network . }
```

Listing 8.1: An Example of a sQuery with 3 Subgraphs

Star-Shaped Event Detection Pattern

For the star-shaped pattern, event instances are enriched by attributes extracted from the background knowledge. Background knowledge is built on the basis of the RDF graphs and underlying ontology.

We have defined the *Event2KB Link* (Chapter 5, Definition 5.5) used to identify corresponding resources or nodes within the RDF graph to a given raw event instance from the event stream. Recursively traversing the edges within the graph starting from the *Event2KB Link* root of the subgraph containing available information related to the event instance and user query. The graph is similar to a *star topology*.

In Figure 8.4 a specific enrichment is visualized according to a given user query. Two predicates of interest namely `dbpedia-owl:industry` and `dbpedia-owl:employees` are evaluated for the corresponding resource identified by the *Event2KB Link*.

The inferred background knowledge is integrated along with the raw event stream to generate a new event stream which is forwarded for event detection. Specific enrichment instructions can be defined because the ontology (T-Box) is known in its entirety and predicates/relationships of interest w.r.t. specific event instances can be named precisely in order to retrieve related background knowledge.

Example 8.1. ***(An Example of a Star-Shaped Event Query)*** *Suppose we have a stream of stock market events and are interested in detecting companies located in the US, are in the computer software industry, and produce products for computer networks. An implementation of this event query in an event query language is shown in Listing 8.1.*

Event Enrichment for Detection of Event Relationships

SCEP queries can specify detection patterns based on a sequence of events which can be extended to the sequence of resources from the background knowledge. The enrichment of events should enable the detection of complex events based on enriched knowledge and sequence of events. Depending on user query various kinds of relationships can be retrieved from the event stream and background knowledge.

An example of such a relationship is visualized in Figure 8.5. In this example a complex event pattern spanning over three event instances is specified. The query given at the top defines a connecting path between the nodes associated with the event instances e_1, e_2, and e_3 (separated by event operators). One can observe the subjects are partially bounded as well as unbounded, e.g. `?s3`. The order of occurrences of the three arbitrary event instances e_1, e_2, and e_3 is defined using the event algebra operator `SEQ`. Thus, the sequence of e_1 followed by e_2, and subsequently followed by e_3 is matched in the case of resources referenced by e_1, e_2, and e_3 that can be connected by a path corresponding to the triple statements from the query.

Example 8.2. ***(A Simple Event Query with 3 Subgraphs)*** *Suppose we have a stream of stock market events and are interested in detecting companies whose stock quotes have decreased by a certain amount and have supplier-resource-demand relationship with each other. An implementation of this event query in a simple event query language is shown in Listing 8.2.*

```
PREFIX dbpprop: <http://dbpedia.org/property/>

select * from pattern[every tick1=StockTick and every tick2=StockTick where timer:
    within(2 min)]

  SPARQLPATTERN [define tick1.url as ?url1]
  {  ?url1 dbpprop:products ?resource . }

  SPARQLPATTERN [define tick1.url as ?url2]
  { ?url2 dbpprop:hasResourceDemand   ?resource . }
```

Listing 8.2: An Example of a sQuery with an Event Pattern Relation in Background Knowledge

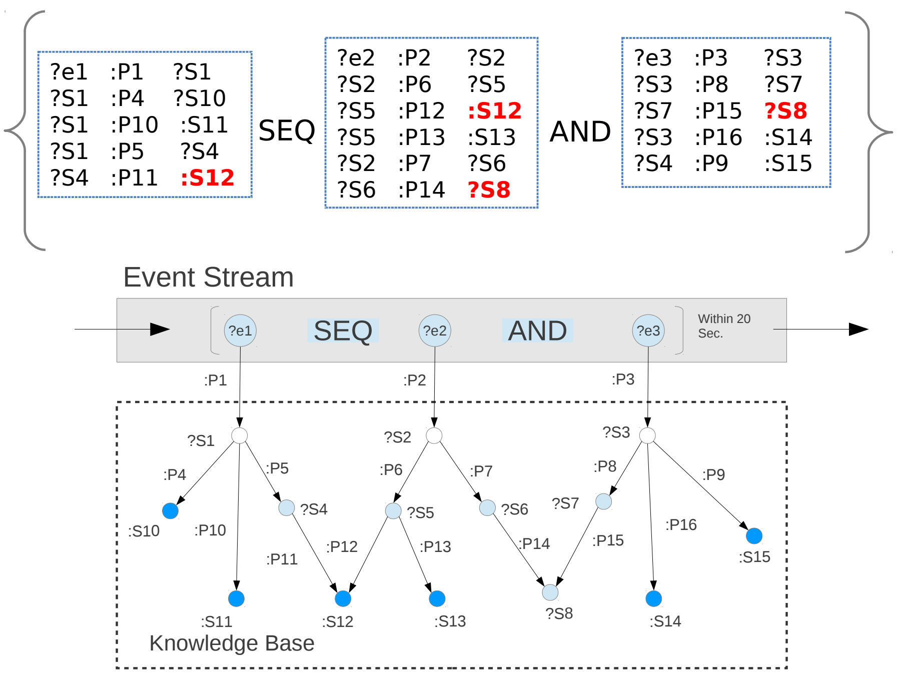


Figure 8.5: Relations between Events

8.2.5 Costs of Semantic Enrichment

In Section 5.5.1, we described the different cost parameters for knowledge-based event processing. For the process of event enrichment, the main cost is for querying an external knowledge base and knowledge acquisition. It consists of costs for several subtasks that are brought together and form the total cost of knowledge acquisition.

The costs of the subtasks are:

- **Query Selectivity:** This cost depends on the type of background data, its homogeneity and the user query given which includes joins and asks for different data types.
- **Cost of Projections and Joins:** This cost depends on the type of background data, its homogeneity and the user query given (if the query includes joins and asks for different data types).
- **Reasoning Costs:** This cost is incurred by the reasoner, which is included as a component of external knowledge base. The estimation of the reasoning cost is complex and depends on several factors, like ontology language, level of reasoning and data instances (A-Box) [Kol12, Haa08, Sir06].
- **Data Transmission Costs:** This cost is the cost of transmitting query results to the event processing engine. This cost is based on the size of the query results and network properties.

  In this work, we assume the network is reliable and dedicated to the transmission of query results. The main factor which affects data transmission cost is the size of the query results.

One of our assumptions is the event processing system can have access to external KBs only over available interfaces and the EPA cannot know much more about the internal details of the query processing mechanism and algorithms like, indexing structure or data distribution techniques. Because of this assumption, we approximate the costs of querying external KBs for semantic enrichment of events depending on subsequent cost factors from the view point of the event processing agent.

We use the following statistical data to approximate the cost of queries sent to external KBs:

- **Number of Queries to External KB:** The number of queries sent to KBs for retrieval of background knowledge about events is one of the cost factors. The number of queries sent to KBs for a set of event instances can vary depending on enrichment and detection plans. Optimized event enrichment and detection can organize enrichment and detection steps to avoid unnecessary queries.
- **Number of Results for each Query:** Each query to external KBs returns a number of results to the main EPA. The transmission cost for returning results from external KBs to the EPA is another cost factor. We consider the number of results, and the number of resources returned for each of the SPARQL queries.
- **Reasoning Factor**: Each query incurs a different reasoning cost on the external knowledge base. Reasoning cost depends on different factors like the size of data on which reasoning is to be realized, and the complexity of reasoning logic. The calculation of reasoning cost is highly complex. We can only estimate and compare the cost of reasoning for different graph patterns (BGPs) based on given predicates, resources, and statistics about the data stored on KB.

8.3 Plan-Based Semantic Enrichment

The process of semantic enrichment can be optimized to reduce the cost of event enrichment and increase throughput of event processing by reducing the amount of raw event enrichment tasks. We propose an approach for the optimization of knowledge-based event detection by using a technique for multi-step and greedy knowledge acquisition and event detection. In our approach we use sequential setting and several steps for event stream enrichment and detection of complex events from the enriched stream. In each step a part of the knowledge is enriched to the events. The following event detection engine can filter out some of the raw events based on enriched knowledge so that only relevant raw events are forwarded to the next step. By using this approach we can avoid the unnecessary full enrichment of all raw event instances.

Figure 8.6 shows an example of our approach for multi-step knowledge acquisition and event detection. Each SCEP node includes an Event Mapping Agent (EMA) and an Event Processing Agent (EPA) engine so that it can enrich and detect events. At the initial processing step (the starting point of event processing) the raw event stream flows through the system. In the first step EMA node enriches the event stream with knowledge acquired by using subgraph $g1$, and the EPA can filter out event instances $e4, e5$ depending on the knowledge used. We observed that a partial enrichment of events can be used to filter out some of the events to avoid unnecessary enrichment of all raw events.

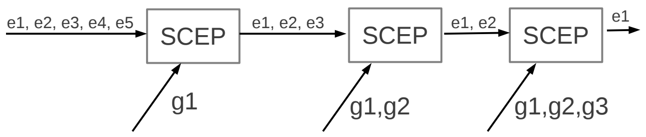


Figure 8.6: Multi-Step Greedy Knowledge Acquisition for SCEP

The trade-off between knowledge acquisition costs (computation load on external KB and result transmission) and event processing latency are important factors for planning the execution of event enrichment and detection. Our aim is to discover a low-cost event detection plan while meeting user-specified latency expectations so that we can reduce the polling load on the external knowledge base. One of the important constrains for generating plans is user-specified latency expectation. We search for a plan which can meet this expectation and generate an acceptable load on the triplestore side. In the final step, we have to check the whole *sQuery* on the event stream, we look only for good filters to pre-filter the event stream so that we can reduce costs.

The user query given can be preprocessed and separated into several subqueries. In our approach we generate a plan for stepwise processing of the generated subqueries so that we can pre-filter the raw event stream to reduce the cost of event enrichment. In each step, we check only a part of the user query. If any of the subqueries cannot be matched, the whole query is not matched and the EPA sends the event to the event sink.

The givens for the optimization problem are a user query *sQuery*, raw event stream (including event types) and heuristics about the external KB. We require for an optimized execution plan a user query with acceptable latency and costs (computation, materialization and network transmission costs).

We assume that the events can be kept in the same order as they arrived at the base point of the system and the time distances between event instances are not significantly changed, or they are reordered to the same time intervals as the original event stream (We can set up a system that can reorder the event stream by using a buffer memory).

One further assumption is that background knowledge is not significantly changed while an event instance goes through the multiple processing steps. The enrichment and detection of events depend on the KB version in use as the event happens/is captured.

In the following, we describe first how we generate subgraphs from the user query and then we get into the details of our approach for planning the event enrichment by using extracted subgraphs.

8.3.1 Extraction of Subgraphs from User Query

The user-given query *sQuery* includes a graph G_{user} (the given triple pattern combined with event algebra operations) which can be pre-processed and rewritten to several subgraphs G_{SUBs} so that we have $G_{user} = \{G_{SUB1} \cup \ldots \cup G_{SUBn}\}$. The G_{user} is matched if and only if all of its subgraphs G_{SUBs} are matched.

The first step of our approach is to split the G_{user} into several main subgraphs G_{Events} based on nodes which represent the raw events and are separated by the event algebra operations (e.g., `AND`, `SEQ`). For each event we have a tree structured graph (a cycle-free directed graph) which has one of the raw events as its root. We also mark the nodes which are in the intersection of G_{Events}.

In the second step, we divide the G_{Events} based on its tree structure. By starting from its roots (the events) we traverse the tree to its leaves and divide the tree by its branches so that we generate several subgraphs G_{SUBs}. A subgraph is generated for each path branch from the root to one of the leaves. For each raw event type we also generate all the joint possibilities of the subgraphs (based on the event operator). At the end of this step we have a multi-set of graphs MS_{GSUBs}.

For example, for event pattern e_1, as shown in Figure 8.5, we will generate the subgraphs in Listing 8.3.

```
{
 { (?e1 :p1 ?s1) (?s1 :p4 :s10) },
 { (?e1 :p1 ?s1) (?s1 :p10 :s11) },
 { (?e1 :p5 ?s4) (?s4 :p11 :s12) },
 { (?e2 :p2 ?s2) (?s2 :p6 ?s5) (?s5 :p12 :s12) },
 { (?e2 :p2 ?s2) (?s2 :p7 ?s6) (?s6 :p14 ?s8) },
... }
```

Listing 8.3: Extracted Subgraphs from an Event Pattern

We expand the subgraphs to their semantically similar patterns. For example, if the leaf nodes of the subgraphs are bounded resources, we check them against the knowledge in KB to find out if they are connected to other resources through the *sameAS* predicate. (The *sameAS* predicate gives two resources exactly the same semantic meaning.) If we can find new resources through the *sameAS* predicate, we will generate a new subgraph G_{SUB} with the new resource and add it to the multi-source. We also follow up on the type hierarchy of resources (e.g., rdfs:subClassOf) and add new upper resources to the type hierarchy. We also check the properties of the graph to find subproperties in them and add new graphs to the multiset. In this way, it is possible to take the semantics of the resources (and their relations) into account in the calculation of subgraphs.

Algorithm 1: Algorithm for Generating List of Subgraphs

```
Data: $sQuery$, KB
Result: A multiset of subgraphs $MS_{GSUBs}$
$MS_{GSUBs} = \emptyset$ ;
Generate $G_{user}$ by separating sQuery from event operators;
foreach graph in $G_{user}$ do
    Find all $G_{Events}$;
    foreach graph in $G_{Events}$ do
        $G_{SUB}$= traverse all paths from event root to the leaves and extract paths;
        foreach graph $g$ in $G_{SUB}$ do
            $G_{similar}$ = Find all semantically similar graph to the graph $g$;
            $MS_{GSUBs} = MS_{GSUBs} \cup G_{SUB} \cup G_{similar}$ ;
        end
    end
end
return $MS_{GSUBs}$;
```

Algorithm 1 shows our algorithm for generating lists of subgraphs from user queries. The givens in this algorithm are, user query $sQuery$, and access to the KB statistics.

On Algorithm 1 the terms are as follows:

- $sQuery$ is a user-specified event detection query

- G_{user} is a set of subgraph gained by decomposing the user query graph from the event operators
- G_{Events} is a set of event variables, for example ?e1, ?e2 on Figure 8.5
- G_{SUB} is a set of subgraphs of G_{user}
- $G_{similar}$ is a semantically similar graph to the graph
- MS_{GSUBs} is a multiset of subgraphs extracted from $sQuery$

Algorithm 1 starts from the user query sQuery. It decomposes the query graph from the event algebra operators to several subgraphs that include event variables (e.g. $?e_1, \ldots, ?e_n$). Then each of the subgraphs traverses from the root node to the leaves, so that we can generate all the possible paths from the root node (event variables) to the leaves. Finally, we find all possible semantically similar graphs to each of the graph and add them all to the multiset of subgraphs.

We generate a multiset because we want to count how many times a subgraph is repeated in the complete query graph, so that we can plan the event enrichment based on the repitation factor of the subgraphs in the query graph.

8.3.2 Planning Multi-Step Event Enrichment and Detection

Our approach for multi-step event enrichment and detection (shown in Figure 8.3) consists of several steps of event enrichment followed by event detection steps. In each of these steps a part of the event enrichment is executed, so that event detection can be realized based on the added knowledge of the enrichment step.

We describe our planning approach initially by using simple forms of event detection queries and finally by using more complex event detection patterns that include event detection operators. Figure 8.7 shows the processing of a star-shaped event pattern which includes several attributes to be matched (a,b,c,...). In each processing step the required knowledge for one of these attributes is enriched to the raw event stream ($K_A, K_B, K_C, \ldots$). If an event instance cannot be matched to one of the attributes it is forwarded to the event sink, otherwise the processing is continued in the following step, until all the attributes are matched. For the execution of this process an optimal processing is required.

The two main tasks for multi-step processing are to be defined for each processing step:

- A knowledge query that can extract the required knowledge from the external KB for event detection without missing any of the event components
- An event detection pattern based on the enriched knowledge for detection.

For the optimization of the processing steps, we need to setup subgraphs at the initial steps which are less likely to be matched, so that the downstream discharge of the processing steps can be minimized and costs for the event enrichment optimized.

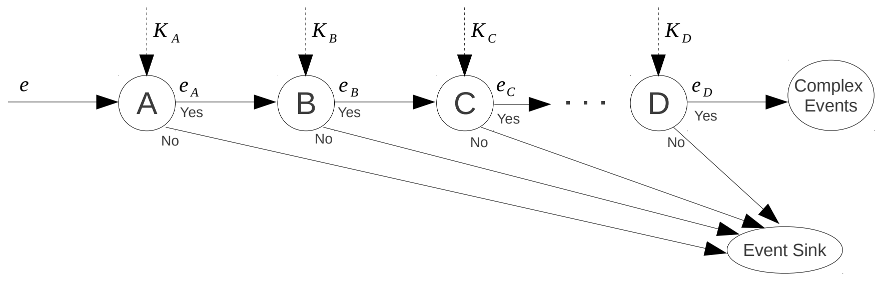


Figure 8.7: Processing of Star-Shaped Pattern

Forms of Operators in Multi-Step SCEP:

Based on the event algebra operators specified in the user query, multi-step processing might have different forms and processing network topology. The enrichment and detection of event instances might be continued in several steps or be stopped, depending on the matching/unmatching of patterns. For example, for the matching of a query with an *OR Event Operator* that can be decomposed into several subqueries. For the matching of the whole query it is sufficient if only one of the subqueries can be matched, and it is unnecessary to match the other subqueries.

Event operators divide the user query *sQuery* into several query parts which we name hereafter *event component parts*. As we have previously described, user query is a highly abstract event detection pattern based on relations in background knowledge and is a combined query from event algebra operators and knowledge graph patterns (RDF Basic Triple Patterns).

Let's consider the following user query which includes query parts combined with event operators:

$sQuery = Q1 \quad OP \; Q2 \quad OP \quad Q3 \; \ldots \; OP \quad Qn$

We assume *sQuery* is dividable into event component parts (Q1, . . . Qn), because the specified complex event includes n events. Each of the subgraphs can be separated into several sets of subgraphs.

$\{Set_{Q1}, \;\; Set_{Q2}, \;\; \ldots, \;\; Set_{Qn}\}$

OR Operator:

The OR operator in the user query makes it interesting for event detection, because if each of the event components is matched, the complete query is matched. The set of subgraphs which can be used for multi-step event processing is the union of all the subgraph sets:

$sQuery_{subgraphs} \equiv Set_{Q1} \cup Set_{Q2} \ldots \cup Set_{Qn}$

Figure 8.8 shows the effect of the OR operator in multi-step event processing. Our system can enrich and detect events in multi-steps in A, B, C, . . . steps; in each step a knowledge query (K_{Query}) is sent to the external KB and the result of that in the form of knowledge ($K_A, K_B, K_C, \ldots$) is enriched to the event stream and used for event detection. If the events can be matched (the case of YES on Figure 8.8), the complex event is detected and if not the event is forwarded to the next processing step until the event pass through all the event processing steps.

For *OR operator* the following two approaches are possible:

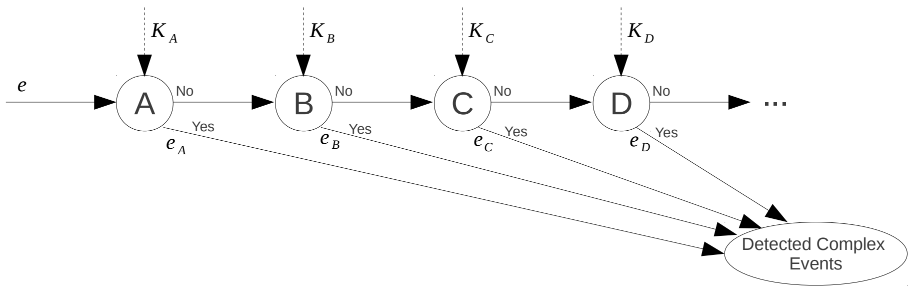


Figure 8.8: Multi-Step Event Processing with OR Operation

- **Full Enrichment of a Single Event Component:** In each step, we enrich the required knowledge to detect one of the event components. This query can be defined based on one of the query subgraph sets ($Set_{Q1}, Set_{Q2}, Set_{Q3}, \ldots$). The event detection pattern does not need to include the OR event operator and can be specified to detect one of the event components. The union of all of subgraph sets used in all the steps is equal to the $sQuery_{subgraphs}$.
- **Partial Enrichment (with at least one element of each subgraph set):** In each processing step we enrich the event stream with a knowledge query which includes at least one element of each subgraph set, and detect events based on this subgraph. The event detection pattern in each step has to include the OR event operator in order not to miss any of the complex events. The union of all of the subgraph sets used in all the steps is equal to the $sQuery_{subgraphs}$.

AND Operator:

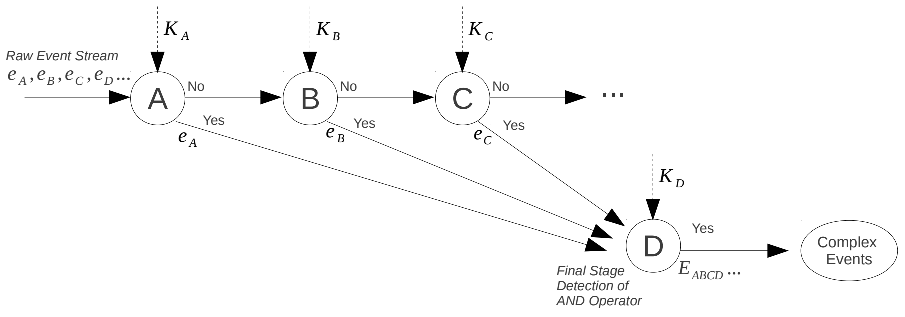
Let $sQuery = Q1\ AND\ Q2\ AND\ Q3\ \ldots$ be the user query, in each of the processing step, we have to check if the incoming raw event instance e can be specified by $Q1$ or $Q2$, or $Q3$ so that it can be forwarded to the next step, or be dropped to the event sink.

For the *AND operator*, the following processing approaches might be applicable depending on the user query given:

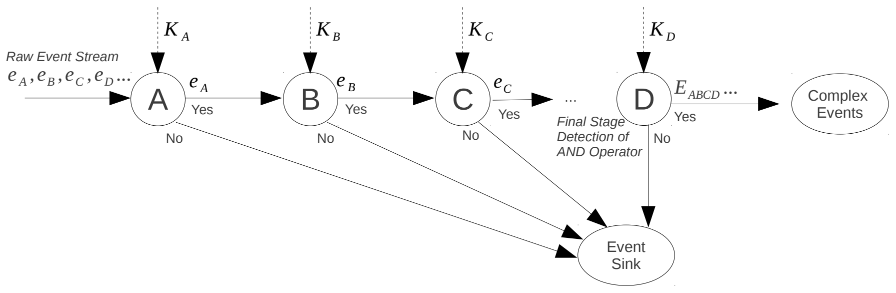
- **Full Enrichment for one of the Event Components:** In each step the required knowledge for the detection of one event component is enriched. As shown in Figure 8.9a the event stream has to be processed by several processing steps. A final processing step is considered w.r.t. to joining and applying the AND operator to the results.
- **Partial Enrichment of Intersection Set of Subgraph:** To optimize event processing, we might be able to find an intersection set of all subgraph sets, so that we can use the elements of the intersection set for pre-filtering the raw event stream.

  $initial_{subgraph} = Set_{Q1} \cap Set_{Q2} \ldots \cap Set_{Qn}$

  If an event instance can be matched by using the intersection set of subgraphs, it can be forwarded to the next processing step. This case is similar to the star-shaped processing approach and includes a final processing step, as shown in Figure 8.9b.



(a) Full Event Enrichment



(b) Partial Event Enrichment

Figure 8.9: Multi-Step Event Processing with AND/SEQ Operators

- **Partial Enrichment (with at least one element of each subgraph set):** In this approach, we enrich the event stream with a set of subgraphs which includes at least one graph pattern of each subgraphs sets, so that we can check all of the event components. This approach is illustrated in Figure 8.9b.
- **Intermediate AND Processing:** It is possible to have intermediate joining processing steps to join the results in order to check the AND/SEQ operators and not transport the intermediate results. This is similar to the approaches in distributed databases used for joining the results from distributed data sources.

SEQUENCE Operator:

The SEQUENCE operation is similar to the AND operation. Also for the SEQ operation all of the subgraphs should be matched so that the complete *sQuery* is matched. However, the matches should be done in a sequence, i.e., if a first subgraph is matched then it needs to match the others.

The sequence operation for our multi-step operation can be handled as an AND operation, because the matching of the sequence of subgraphs should be done within the data window, i.e., if SEQ matches then AND can also be matched. Furthermore, we assume that we can only access the raw event stream at the base point of the processing step (the first processing step).

NOT Operator:

The NOT operation means the query cannot be matched to the existing events inside the window given. Let's consider that we have user query $sQuery = NOT(Q1)$, so that Q1 consists of subgraphs $Q1 = Q_1^1, \ldots, Q_n^1$. This can be handled as an inverse case of the star-shaped pattern if the events can be matched then the complex event is not matched, and if not the complex event is detected.

Figure 8.7 shows the processing approach for the star-shape, in which the inverse case is the NOT operator.

Combination of Operations:

In the case user queries are combined using the four main operations, for example the following user query with a combination of OR and AND operation.

$sQuery = Q1\ OR\ \ Q2\ \ AND\ \ Q3$.

We can simply divide these query into subqueries from the OR Operators, and then process them based on the approaches described above.

Or we can distribute the OR operation within the AND operation, which then can be handled as AND operator with two subqueries including OR operators.

$sQuery = (Q1\ OR\ Q2) AND (Q1\ OR\ Q3)$.

8.3.3 Estimated Matching Probability Factor

For the planning of subqueries, we need to estimate the matching probability of subgraphs. We expect that queries with a high number of results are more likely to be matched. We assume we can have some statistics about the external knowledge bases (described in Section 8.2.5), so that we can estimate the enrichment cost based on the collected heuristic data about external KBs.

We consider the following statistics about the KB:

- The estimated total number of existing triples in the KB for each of the predicates (N_p)
- The estimated total number of triples stored in the KB $(N_{Triples})$.

Furthermore, we make the following assumptions:

- All the subgraphs with only a single triple pattern have a bounded predicate.
- All the other subgraphs have at least one triple pattern with a bounded predicate (not a variable predicate in the triple pattern). Any updates on the knowledge base can only minimally change the above statistics about the KB.
- The statistics can be recalculated within time-intervals.

We expect that queries with a high number of results are more selective (materialization costs) and incur a high load. For the subqueries with two or more triple patterns, the computation of joins will cause more computation load than subgraphs with a single triple pattern.

The calculation of the reasoning costs on the KB is highly complex and depends on the complexity of the reasoning algorithm and the reasoning rules. As we need only an estimation of the costs for each subgraph, we define an estimation factor for each of the predicates. Based on the reasoning rules, each of the predicates can activate a different set of rules which will cause different computation costs. For example, based on the reasoning level and complexity of the rules used the predicates like *"rdfs:subClassOf"* or *"owl:sameAs"* can activate other reasoning rules with other predicates like *"owl:intersectionOf"*. We call this factor the reasoning factor $F^{p}_{reasoning}$ and assume we can define for each of the predicates a reasoning factor, a number between 0 and 1. For example, predicates like *"rdf:type"* and *"owl:sameAs can"* have the highest factor of one. We assume we can define this factor manually, by looking at the chains in the reasoning rules. We consider the reasoning factor for object properties that are not explicitly defined in the reasoning rules as zero and data predicates also have a reasoning factor of zero.

The estimated matching probability factor of predicates F^{EM}_{p} is calculated by the following equation:

$$F^{EM}_{p} = 2/(F^{p}_{reasoning} * N_p/(N_{Triples} - N_P) + N_p/N_{Triples}) \tag{8.1}$$

where

$F^{p}_{reasoning}$ is the reasoning factor of the predicate p between 0 and 1
N_p is the total number of triples stored in the KB with predicate p
$N_{Triples}$ is the total number of triples stored in the KB.
F^{EM}_{p} is the estimated matching probability factor of predicate p

8.3.4 Filter Functionality Estimation of Subgraphs

In the multi-step SCEP the throughput of an event stream in each step is highly dependent on the rate of events matched in the previous step. Subgraphs with fewer results are good filters for event detection, because they are less likely to be matched. In the case that they are used at the beginning of multi-step event enrichment and detection, the discharge rate of events in the following step (downstream steps) can be highly reduced.

A *filter functionality factor* is introduced for each of the subgraphs. This factor is calculated based on the *estimated matching probability factor of the predicate* (F^{EM}_{p} introduced above) and the graph structure properties of the user query. The subgraph marks a specific part of the graph pattern of a user query which can have different properties, e.g., if the subgraph is positioned in the leaf of a tree structure, or if it is in the intersection of subgraphs (see Figure 8.5). The intersection nodes are nodes on the graph where event operations divide the graph.

We define two factors for the graph properties of the subgraph, F_{Leaf} is 2 if final leaf of G_{SUB} is bounded and 1 if not.

$$F_{Leaf} = \begin{cases} 2, & \text{if final leaf of } graph\ pattern\ G_{SUB} \text{ is bounded.} \\ 1, & \text{if not.} \end{cases} \tag{8.2}$$

Intersection factor F_{Inter} is 2 if G_{SUB} includes intersection nodes and 1 if not.

$$F_{Inter} = \begin{cases} 2, & \text{if } graph\ pattern\ G_{SUB} \text{ includes intersection nodes.} \\ 1, & \text{if not} \end{cases} \tag{8.3}$$

Sometimes in a user query a subgraph is repeated in several places on the graph pattern. In this case this subgraph might have a better filtering functionality for event detection than the other subgraphs. We consider this effect with the factor for repetition of subgraphs: $F_{Repetition} = N_{Repetition} / N_{total}$

$N_{repetition}$ is the repetition count number of the G_{SUB} in the multi-set MS_{GSUBs} (how many times a subgraphs appears in the whole graph pattern).

We estimate the cost of different subgraphs included in the query on the basis of some heuristics of the stored data on KB (as shown example data in Table 8.1) so that we can compare the subgraphs and organize the sequence order of processing in multi-step SCEP. Based on the estimated cost we can calculate the savings by changing the order of subgraphs.

One of the heuristics about the predicates used in BGPs is how many answer triples on average have the triple pattern ($AvgNumberOfResults$), e.g., how many triple results on average has a single triple pattern with *dbpedia-owl:location* property.

The *filter functionality factor* of each subgraphs is calculated by the following equation:

$$F_{Filter} = (F_{Leaf} * F_{Inter} * F_{Repetition} * max(F_p^{EM})) / AvgNumberOfResults \tag{8.4}$$

where the terms are:

F_{Leaf} is the leaf factor of subgraph G_{SUB}.
F_{Inter} is the intersection factor of subgraph G_{SUB}.
$F_{Repetition}$ is the repetition factor of subgraph G_{SUB}.
F_p^{EM} is the estimated matching probability factor of the predicates of graph G_{SUB}.
$AvgNumberOfResults$ is the average number of KB results of graph G_{SUB} with predicate p.
F_{Filter} is the filter functionality of subgraph G_{SUB}.

8.3.5 Estimation of Processing Cost

As previously indicated, the calculation of the processing costs of knowledge-based CEP is highly complex and depends on several factors. However, we aim to estimate the cost of different subgraphs included in the query based on some heuristics of the data stored on KB, so that we can compare the subgraphs and organize the sequence order of processing in multi-step SCEP. Based on the estimated cost we can calculate the savings by changing the order of subgraphs.

In the multi-step processing of events, in each step of the process we send a knowledge query to the KB and enrich the result to the event stream. These queries are basic graph patterns that the subject and their predicates are bounded and we look for objects of the triple pattern. Let's suppose we have a database table with N entries of triples with the same predicate, and we search for answers matching to our triple pattern (bounded subject and predicate).

Transmission Cost:

If the query to the external KB has several results, the EMA can retrieve them and transmit them to the enrichment base node. In the case that the user query includes BGPs with RDF data properties, the result of such triple pattern has only one single literal as a result. If the BGP includes an object property it can have several result resources as results (URIs). For our cost estimation we can count the number of result items (resources or literals) returned from knowledge base. The literals are categorized into typed (XSD[2] data types) and their types are identified as float, double, integer, long, etc.

For our cost estimation we can count the number of result items (resources or literals). Let's assume that we have E number of events in a large time window, in one of the processing steps, enrichment agent sends E number of SPARQL queries to KB, i.e., the average estimated cost for the transmission of $R * E$ number of results in which R is the average number of results for knowledge queries.

8.3.6 Generation of Execution Plan

Our aim is to find a low-cost event detection plan which can meet user-specified latency expectation and result in reduced transmission and processing loads. We have shown in Algorithm 1 our approach for the extraction of subgraphs from the user query. Based on the order of subgraphs in an execution plan and number of processing steps, different setups of multi-step event processing with different latencies and load costs can be generated. The total cost of a plan is estimated with the total number of queries sent to the external KB, and the total number of transmitted results within a time window (for a user query).

Our approach extends the concepts by Akdere et al. [Akd08] for plan-based complex event detection across distributed sources and Schultz-Möller [SM09] for distributed complex event processing with query rewriting. In their approach, they were only looking for one optimal plan which can reduce event transmission costs to the event detection base, whereas in our approach we are looking for an execution plan for event enrichment with acceptable latency and reduced enrichment cost. The main difference is our focus is on the enrichment of events with background knowledge.

Our algorithm does not look for an absolute optimal execution plan, but instead for an acceptable plan which can satisfy user requirements and generate an acceptable load on external KBs (an acceptable plan can be defined by use case). Due to the huge number of possible permutations of the subgraph set, the total number of possible plans is exponential to the size of subgraph set N. The number of possible permutation with N subgraphs and N processing steps is 2^N.

To be able to know the exact latency and load of each execution plan, we can only run the plan on the stream of events, use background knowledge bases, and monitor the latency and loads of the event enrichment and detection. The optimal plan depends on the data stored in the KB and the entropy of the event stream (data stream entropy described in Section 3.6.1), so that we will only find plans based on the estimated heuristics. Event detection plans in our approach can be first optimized based on existing concepts [Akd08, SM09] to optimize event transmission costs and afterwards be planned for semantic-enabled event processing.

[2]XML Schema Part 2: Datatypes Second Edition `http://www.w3.org/TR/xmlschema-2/` retrieved April 2014

Our approach for searching for an acceptable plan is presented in Algorithm 2. We ordered the list of subgraphs based on their estimated filter functionality factor F_{Filter}. To generate an execution plan, we use this list as the initial execution plan. Our algorithm is a greedy algorithm which starts with an initial plan. To avoid the exponential search effort for testing the costs of all possible plans, we start with an estimated plan (a plan that might have an acceptable cost and latency) and then run several iterations with other plans which might improve latency and load until we find an acceptable plan for the user query.

Our algorithm starts by using a two-step processing plan. If the average latency is acceptable and the subgraph set has more elements, then a new processing step is added. This process is continued until the latency of the overall system is greater than the user latency expected.

We monitor the latency and the total result transmission for a time period, but if the requirements can not be satisfied, then we change the execution plan until we have an acceptable plan. If the latency is under the threshold of the user's expectation, we change the plan to check if we can reduce the caused load on the external KB. In the case that the load is acceptable for the external KB, we can accept the current plan as our execution plan.

Algorithm 2: Algorithm for Selecting an Execution Plan for Subgraphs

```
Data: $MS_{GSUBs}$ a multiset of subgraphs
Data: $latency_{expected}$ user specified latency expectation
Result: $plan$: an execution Plan for enrichment and detection
$FirstStepGraph = getFirstElement(sort(MS_{GSUBs}, F_{Filter}))$;
$SelectedGraph = \emptyset$;
while hasNextPlan do
    $NextStepGraphs = MS_{GSUBs} \setminus (FirstStepGraph \cup SelectedGraph)$;
    $plan = (FirstStepGraph, NextStepGraphs)$;
    $(latency_{current}, load_{current}) = execute(plan, t)$;
    if $(latency_{current} \leq latency_{expected} \bigwedge load_{current} \geq load_{previous})$ then
        $SelectedGraph = getFirstElement(sort(NextStepGraph, F_{Filter}))$;
        $MS_{GSUBs} = MS_{GSUBs} \setminus (SelectedGraph \cup NextStepGraph)$;
        Add a new processing step, after all plans for the number of steps have been searched ;
        $load_{previous} = load_{current}$;
    else
        return $plan$;
    end
end
```

On Algorithm 2 the terms are as follows:

- MS_{GSUBs} is a multiset of subgraphs extracted from $sQuery$
- $latency_{expected}$ is the user expected latency
- $plan$ is a execution plan
- $FirstStepGraph$ is a selected subgraph for event processing in initial step

- $NextStepGraphs$ is a set of selected subgraph for next event processing step or steps
- $execute(plan, t)$ executes the plan for time t
- $load$ is a transition load (current or previous depending on algorithm state)

8.4 Evaluation

In this section, we evaluate the proposed concepts for planning multi-step semantic event enrichment. We start with experiments on star-shaped event enrichment and continue with experiments on the effects of our multi-step approach on performance and latencies in comparison with the single-step event processing.

Methodology and Experiment Setups:

We have implemented a prototype of our multi-step approach and its algorithms in Java. We use the OpenRDF framework[3] to process the triple patterns and send them as SPARQL queries to the endpoint of an instance of an external triplestore[4]. For the event detection steps, we used the Esper[5] event engine. In our experiments, we forwarded the output stream of event enrichment to the event detection step so that the two steps built our approach for multi-step processing.

We have two implementations of the system, one is based on event processing on the Storm network[6] that builds an event processing network, and the other is based on simulating the event processing network by using java multi-threading for different EPAs and EMAs. We present here our experiment with EMAs and EPAs as multiple threads in Java.

To cleanly separate the impact of our approach from those of the underlying implementation and configuration choices, we compared the evaluation metrics with each other on the same implementation setup and used abstract performance metrics. We compared different transmission costs of different event processing approaches (with different plans) on the same implementation using the same data sets. To separate the impact of specific data on our experimental results, we executed the experiments on different event streams, knowledge bases and queries.

For our experiments we needed different test datasets as an event data stream (dynamic data part) and a background knowledge base. We used in our experiments both synthetic and real-world data sets.

[3]OpenRDF http://www.openrdf.org/
[4]Special database for the storage of RDF data
[5] `http://esper.codehaus.org`, version 4.6.0
[6]http://storm-project.net/

Event Stream Dataset:

For our experiments, we used an event stream which simulated the event stream of a stock market exchange. We used a list of 500 stock market companies (S&P500)[7], each event was the price change of a company in the stock market. The event stream was generated by randomly selecting one of the companies in the list and sending the event object to the stream. Each event instance included a string for the stock symbol, a string for the stock name, an integer for the latest stock prices, an integer for the last stock volume and a string for the URL link.

In our experiments, the event producer was able to produce a high throughput so that we could run stress tests on our SCEP implementation. We used real-world stock market quotes extracted from the Yahoo finance web service [8]. We mapped the stock market symbol to DBpedia URLs to be able to extract background knowledge about them.

Background Knowledge Dataset:

As background knowledge we used a complete mirror of DBpedia[9] (version 3.4). Each of the event instances includes a URL which mapped to a DBpedia resource. By using this link, the SCEP system can extract background knowledge. To each event instance a payload was added which was a key-value set of the enriched attributes and the extracted value for the attribute.

Experiment Setup:

As our evaluation metrics did not depend on run-time environment and hardware setup, configurations used in our experimentation did not impact our evaluation results, due to the fact that we compared the different approaches to each other. However, we mention some of the results of event processing in some of the experiments, like the performance of event processing and the latency of detection complex events.

In our experiments we used a single instance of an external knowledge base which was an instance of the Virtuoso triplestore[10] and was installed on a host (Intel Xeon CPU E31245 @ 3.30GHz) with 8 GB RAM and Ubuntu Linux 12.04. We used the default cache configuration of virtuoso[11], so that we had little memory and could have an external knowledge with limited cache capacity.

The EMAs (event enrichment system) and EPAs were installed on a separated host (Intel Core i7-2600 CPU @ 3.40GHz) with 16 GB RAM. Each of the processing agents were different java threads on the same host. The knowledge base host and event processing host were connected through a dedicated 100 Mbit LAN.

[7] Standard and Poor's 500 index (S&P500) `http://www.standardandpoors.com/`
List of Companies `http://www.sandp500stocks.com/`

[8] http://finance.yahoo.com API, extracted data as CVS

[9] `http://dbpedia.org/` Extract structured Data from Wikipedia

[10] Virtuoso triple store, version 06.01.3127, open source version

[11] In virtuoso.ini setup file we have, NumberOfBuffers = 10000, MaxDirtyBuffers = 6000. With 8 GB system memory we set NumberOfBuffers = 680000 and MaxDirtyBuffers = 500000

No.	Predicate	Numbers	% of KB	Results
1	*dbpedia-owl:location*	219880	0.076%	2
2	*dbpedia-owl:industry*	31047	0.011%	2
3	*dbpedia-owl:numberOfEmployees*	12425	0.004%	1
4	*dbpprop:products*	11899	0.004%	2
5	*dbpedia-owl:subsidiary*	2663	0.001%	1
6	*rdf:type*	11085199	3.849%	3
7	*dcterms:subject*	13606126	4.724%	4
Other Predicates		263044482		
No. Triples in the KB		288013721		

Table 8.1: Distribution of RDF Predicates Used in our Queries in DBpedia Dataset

8.4.1 Evaluation of Multi-Step Processing

We conducted several experiments with different types of SCEP queries to investigate the effect of multi-step event enrichment and detection. The main factors that we investigated are overall performance, detection delay time (total transit time of events identified as complex event) and overall load on the external knowledge base (number of eQueries to the KB, number of transmitted results).

Experiments with Star-Shaped Event Patterns:

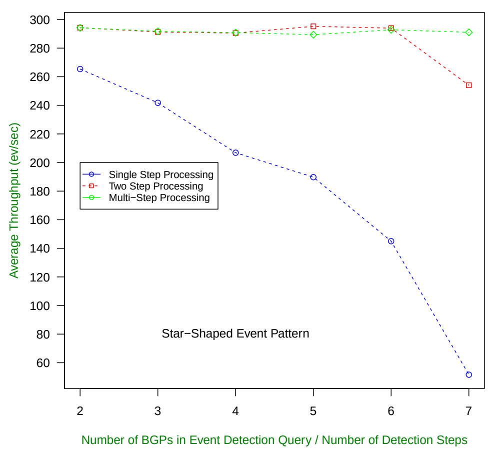
One of the pattern types used for event enrichment and detection is the simple star-shaped event patterns. The aim of this experiment was to compare the different properties of single-step, two-step and multi-step (more than two step) processing approaches. We conducted several experiments and changed the number of basic graph patterns (BGPs) in the event enrichment queries and measured the average processing performance, the latencies of detection of complex events and the transmission costs. Our experiments were done on queries that included from 2 to 7 BGPs. The number of BGPs specified also the number of processing steps, e.g., a query with 3 BGPs was processed in a maximum 3 steps.

In Appendix B, our experimental queries for event enrichment and detection are listed. In Section B.1 the SPARQL queries and in Section C.1 the Esper queries are shown, respectively. We use in our experiment with star-shaped event patterns, the SPARQL queries B.9 to B.14 for event enrichment (including 2 to 7 BGPs) and Esper queries B.28 to B.33 for event detection.

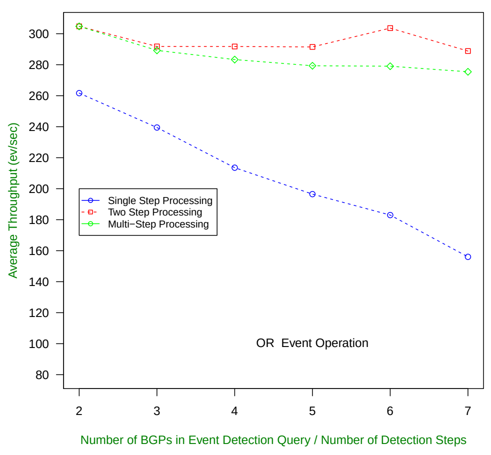
Listing 8.4 shows the star-shaped query that we used for event enrichment. This query shows all of the 7 predicates used in our queries. The combination of these predicates for the generation of different queries with 1 to 7 different BGPs are listed in Section B.1.

```
SELECT ?stockEvent WHERE {
   ?stockEvent   stock:hasCompany   ?company .
   ?company dbpedia-owl:location ?location .
   ?company dbpedia-owl:industry ?industry  .
   ?company dbpedia-owl:numberOfEmployees ?employees .
   ?company dbpprop:products ?products .
   ?company dbpedia-owl:subsidiary ?subsidiary  .
   ?company rdf:type ?rdftype .
   ?company dcterms:subject ?dcterms .  }
```

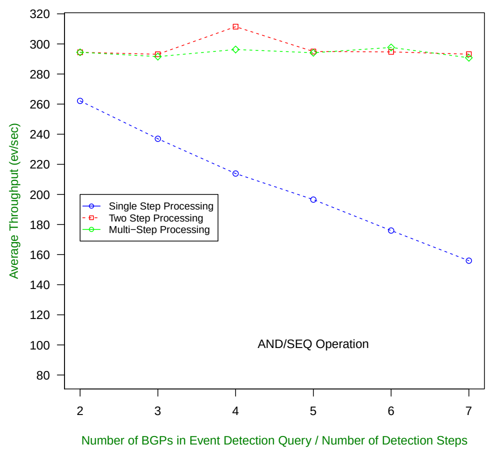
Listing 8.4: Star-Shaped Query for Event Detection



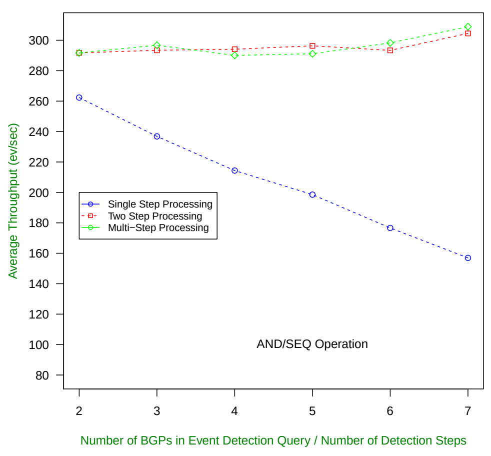
(a) Star-Shaped Pattern



(b) OR Operator



(c) AND Operator



(d) AND+NOT Operators

Figure 8.10: Performance Comparison of Multi-Step Processing

Figure 8.10a shows the comparison of processing performance of single-step processing with two-step and multi-step approaches for different star-shaped patterns. In single-step processing we enriched each event instance with the results of the complete query, i.e., sending the query as a whole to the KB and enriching the results to the event stream. In the two-step process we used the first BGP (with predicate *dbpedia-owl:location*) for the first processing step and the rest of the query in the second step, e.g., for a query with 7 BGPs, 1 BGP in first step and 6 in the following step. In experiments on multi-step processing, we extended the processing steps to the number of existing BGPs in the user query, i.e., for a query with 7 BGPs we would have 7 processing steps.

The performance of the single-step processing approach is continuously reduced w.r.t. the number of BGPs shown in Figure 8.10a. We observed that the performance of two-step processing and multi-step processing are similar and they significantly differ from the performance of single step processing.

In each of the processing steps based on the filter functionality factor of queries, a large amount of events are filtered out and only a subset is forwarded. As we expected, the forwarding of events to the next step causes a delay for the detection of complex events. Figure 8.12 shows the comparison of latencies of different approaches (single, two and multi-step), it only shows the latency of detected complex events (not the latency of the dropped events), i.e., the time difference between event generation and detection of complex event in the final stage. Single step processing has the lowest latency due to the fact the events are processed in a single step.

The comparison of transmission costs are shown in Figure 8.11a. We sent 50000 raw events (50k events) through the system and counted the total number of transmitted results (No. of transmitted RDF resource or literals) returned from the knowledge base to the event detection point. In Figure 8.11a we show only the transmission cost saved in comparison to the case where we do a single-step processing. As it is shown, we can significantly save costs if we do two-step or multi-step processing. For a query with two BGPs we can save up to 60% of the transmission costs and for a query with 6 to 7 BGPs up to 90% of costs.

One interesting observation is the difference in transmission cost saving between two-step processing and multi-step processing (in our case up to 7-steps) are similar, i.e., if we add more steps beyond the second step we will not save much more in processing costs. However, the existing small difference in cost reduction between two-step and multi-step depends on the event detection query used in the first processing step.

With the comparison of the two Figures 8.12 and 8.11a we can argue that two-step processing strikes a good balance between performance, latency and generated transmission costs. Single step processing might not be suitable for use cases with a high throughput event stream and incurs a high load on the KB, but has an acceptable latency when complex events are detected. Two-step processing has high performance and saves transmission costs, but event detection latency should be checked against expected latency of the target specific use case.

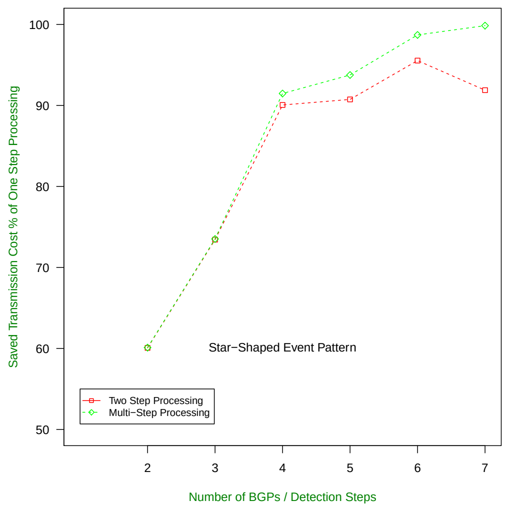
8.4.2 Different Effects of Event Operators

In this section we get into the experiments with event detection queries with different event operators to analyze the effects of event algebra operators on multi-step event processing. We conducted different experiments for OR, AND, SEQ, and NOT operators.

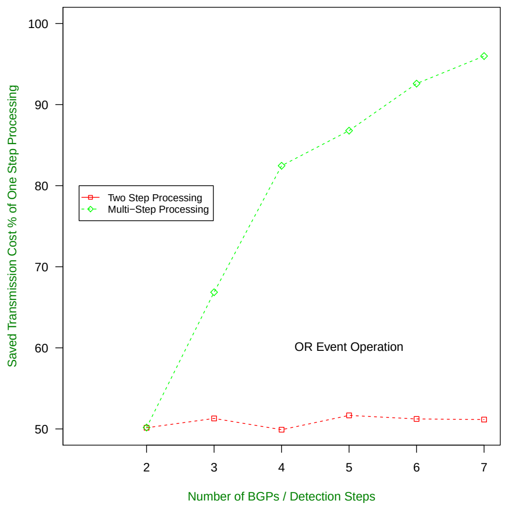
We used for event enrichment the SPARQL queries sQ2 to sQ7 and sQorm2 to sQorm7 listed in Appendix B. For event detection based on different operations we used eQopm1 to eQopm7 and eQorm1 to eQorm7 (change based on event operators).

OR Event Algebra Operation:

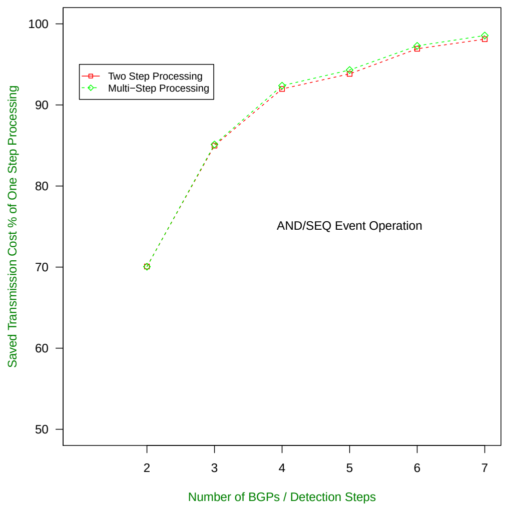
OR operation has an impact on the topology of multi-step event processing, due to the nature of the OR operator. The performance of multi-step processing with the OR operator is shown in Figure 8.10b. As we can see the performance is significantly higher than single-step processing and closer to multi-step processing. The transmission cost reduction is shown in Figure 8.11b which shows the cost reduction for two-step processing is mostly around 50% in comparison to single-step processing and it significantly increases with the multi-step approach. However, average latency for the detection of complex events is increased with the usage of the multi-step approach.



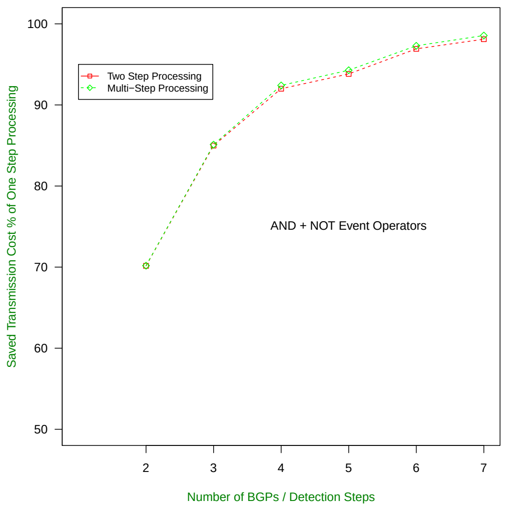
(a) Star-Shaped Pattern



(b) OR Operator



(c) AND Operator



(d) AND+NOT Operator

Figure 8.11: Data Transmission Reduction - Comparison of Two-Step and Multi-Step with Single-Step

AND and SEQ Operators:

As previously described, the processing of AND and SEQ can be handled in a sequential process, and every single event instance can be checked for possible matching in subgraphs/sub-events. Performance and cost reduction is similar to star-shaped event patters. The results of our experiments are shown in Figure 8.10c and 8.11c. The cost reduction here differs from the cost reduction effect of the OR operator. The cost reduction can only be compared with single step processing and not with the OR operator.

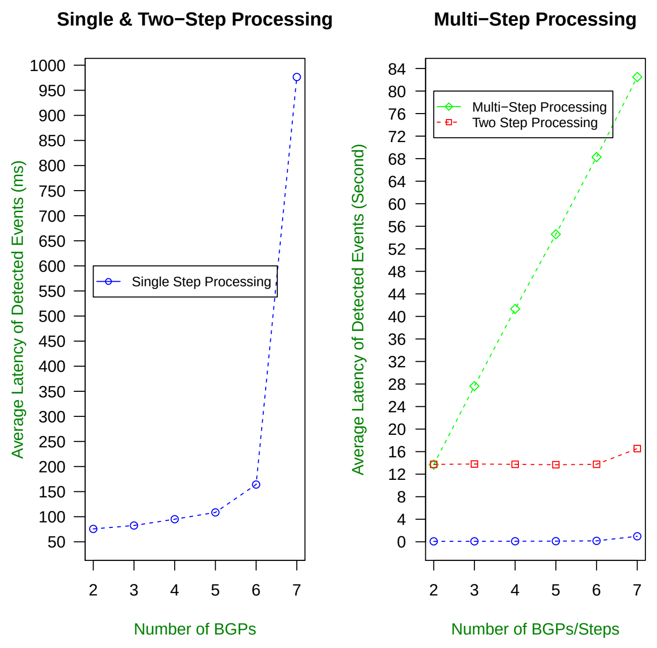


Figure 8.12: Average Latency of Detected Complex Events in Multi-Step Processing (Star-Shaped Query)

NOT Operator:

We used the NOT operator together with an AND operator due to the fact that only a single NOT operator can change the detection topography in multi-step processing. The result of our experiments shows that the performance and cost reduction of the (AND+NOT) operator is similar to the AND operator. The performance and cost reduction is shown in Figures 8.10d and 8.11d.

8.4.3 Planning of SEES:

We compare performance, enrichment and transmission costs of different plans provided by our algorithm (marked with * in Figures 8.13a and 8.13b, and Tables 8.3 and 8.7) with some of the randomly selected plans. We evaluate the proposed planning algorithm for different SCEP query types, star-shaped and combination with different event operators.

Star-Shaped Event Patterns:

For the evaluation of execution plans for the star-shaped event pattern, we used the same event enrichment and detection queries as used for experiments in Figure 8.10a and 8.11a. In our experiment we consider multi-step event enrichment and detection for a query with 7 different subgraphs (7 BGPs) which are processed in 7 processing steps. Table 8.2 lists the calculated filter functionality factor for the extracted subgraphs (higher factors are good raw event filters). Our planning algorithm uses this list as an initial execution plan. The given query with 7 subgraphs has 5040 different execution plans.

Nr. from Table 8.1	Predicate	Filter Factor
5	dbpedia-owl:subsidiary	432615
3	dbpedia-owl:numberOfEmployees	185441
4	dbpprop:products	96819
2	dbpedia-owl:industry	37106
1	dbpedia-owl:location	5239
6	rdf:type	45
7	dcterms:subject	42

Table 8.2: Filter Functionality Factor for Extracted Subgraphs

Plan Nr.	Execution Plan	1	2	3	4	5	6	7
1	5,4,3,2,1,6,7	16	34	51	69	87	105	125
2*	5,3,4,2,1,6,7	16	33	50	68	86	104	123
3	7,6,1,2,4,3,5	285	572	859	1147	1435	1724	2013
4	1,2,3,4,5,6,7	70	141	212	284	356	428	501

Table 8.3: Cumulative Transmission of Results with Different Plans for a Query with 7 BGPs Star-shaped (1000 Results)

Figure 8.13a shows the cumulative KB result transmission for four different execution plans. Our experiments show Plan-3 (7,6,1,2,4,3,5) has the maximum KB results transmitted and Plan-2* (5,3,4,2,1,6,7) has the minimum result transmission (selected by our algorithm). Table 8.3 lists the cumulative transmission of the KB results for the four query plans. Every other possible query plan has a cumulative result transmission between these two plans (Plan-2* and Plan-3).

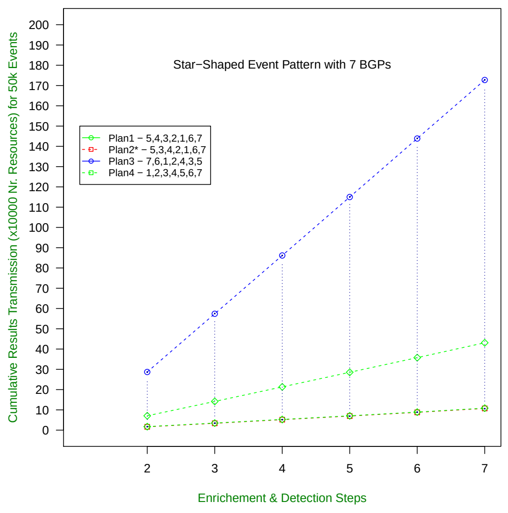
The optimization of execution plans also has its effects on the performance of the overall system. The Throughput performance for Plan-2 is about 270 events/sec. and for Plan-3 (the worst plan) 310 events/sec.

In conclusion, one good plan can be a two-step processing plan in which the first step filters a high rate of raw events and in the following step the rest of the extracted query subgraphs are matched. In this way, we can improve the throughput and latency of event processing and reduce processing and transmission costs.

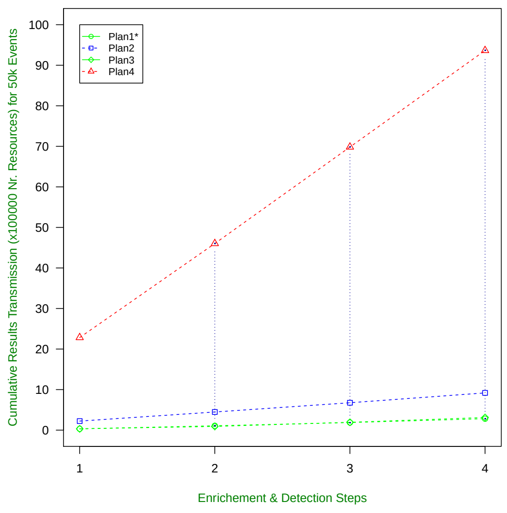
Effect of Operators on Planning:

We consider a query with two event component parts which are combined with an event operator. The predicates of two query parts are listed in Tables 8.4 and 8.5. In our first experiment we used the AND operator and set up 4 different processing steps. Table 8.6 shows the different predicates used for event enrichment and detection in the 4 processing steps. In 4th step, the AND operator is applied to complete the event detection semantics specified by the original query.

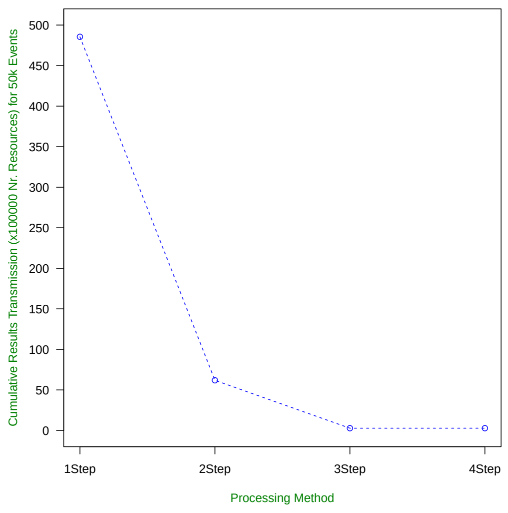
Table 8.7 shows Plan-1* (generated by our algorithm) and 3 other randomly selected plans. Figure 8.13b shows the cumulative transmission of results with different plans, as listed also on Table 8.7. Plan-4 shows one of the maximum cost plans and plan-1* one of the optimal plans, any other plans may incur a cost in this range.



(a) Plans for Star-Shaped Pattern



(b) Plans for AND Operator



(c) Different Steps for AND OP

Figure 8.13: Cumulative Transmission in Multiple Processing Steps

Nr. from Table 8.1	**Predicate**	F_{Filter}
5	dbpedia-owl:subsidiary	432615
4	dbpprop:products	96819
1	dbpedia-owl:location	5239
6	rdf:type	91

Table 8.4: Properties of Subquery 1

Nr. from Table 8.1	**Predicate**	F_{Filter}
3	dbpedia-owl:numberOfEmployees	185441
2	dbpedia-owl:industry	37106
6	rdf:type	91
7	dcterms:subject	42

Table 8.5: Properties of Subquery 2

Steps	Enrichment Predicates	Matching Predicates
Step-1	5,3	(5 OR 3)
Step-2	4,2	(4 OR 2)
Step-3	1,6	(1 OR 6)
Step-4	7	((5,4,1,6) AND (3,2,6,7))

Table 8.6: Execution Plan-1

Plans	Step-1	Step-2	Step-3	Step-4
Plan-1*	5,4	3,2	1,6	7
Plan-2	5,3	4,2	1,6	7
Plan-3	7,6	1,2	4,3	5
Plan-4	1,2	3,4	5,6	7

Table 8.7: Four Different Event Enrichment Plans

Figure 8.13c shows the comparison of cumulative transmission costs in the case that we apply a different number of processing steps. We observed a high cost reduction from single-step processing to 4 step processing.

The effect of the *SEQ Operator* is similar to the AND operator with the small difference that the first event component of the complex event should happen before the other event components. And the effect of the OR operator is similar to the star-shaped pattern shown.

8.5 Summary and Discussion

In this chapter, we have described our concepts for the semantic enrichment of event stream. We described the different types and forms of event enrichment and discussed the main advantages and disadvantages of event enrichment. Furthermore, we provided a planning approach for the optimization of enrichment costs so that the system can detect complex events within user-defined latency expectations and, on the other side, optimized event enrichment costs. In order to generate different enrichment plans the given user query is pre-processed and several subgraphs enrichment of events are generated.

The application of SEES in event processing use cases should be realized considering the advantages and also disadvantages of this approach.

Advantages of SEES:

- **Distribution of Processing Task (Higher Scalability):** The main advantage of SEES is the capability of distributing the task of event enrichment to multiple EMAs, which allows better *scalability* and overall performance of the SCEP system.

- **Possibility for Knowledge Modularization:** One further benefit of task distribution is it allows for modularization of domain background knowledge to multiple knowledge modules. Each EMAs can use some of the KB modules.

- **Including Update Knowledge:** As we have described in Chapter 5, the external KB might have some knowledge updates (A-Box updates) with clearly lower update frequency than the main event stream. One advantage of semantic enrichment, compared to other approaches, is extracted knowledge from the KB, which is the latest version of the available knowledge so that the latest updates can be used for the enrichment of events. In the case that the knowledge changes the event processing engine can include these updates in the event detection process.

Disadvantages and Problems of SEES:

- **Large Number of Derived Events:** The main disadvantage of the semantic enrichment of events can be the management of huge amounts of generated new events which are produced by EMAs. These events should be processed by the final CEP engine to match the complex query. From a single event instances of several new events are generated, or several new attribute fields are added to the event attributes (fat event enrichment). Only a part of these derived events are used at the end to match the complex events and the rest are moved to the event sink (because they are only generated for internal usage, i.e., unnecessarily derived events are deleted without any usage). The SEES approach can scale while lots of derived events are unnecessarily produced. According to our planning approach, an optimal plan for event enrichment and detection reduces the costs of event processing.

- **Enrichment Latency:** The enrichment of the event stream prior to the event detection process increases the latency of the CEP system, because in the first step the system has a latency for the enrichment of events, the results of which are transmitted to the second step for event detection.

Considering all the assets and drawbacks of the SEES approach, we can conclude that this approach is applicable for use cases of knowledge-based event processing where the scalability of event processing is an important factor; latency expectations allows a prior processing step for the enrichment of events thereby reducing enrichment costs.

In the next chapters we will describe two further approaches for the realization of SCEP, fusion and processing of background knowledge with the raw event stream.

Chapter 9
Semantic Enrichment of Complex Queries

One of the approaches for the fusion of events and background knowledge is the enrichment of complex event queries with background knowledge so that complex events can be detected without accessing the external knowledge bases. In this chapter, we propose an approach for the enrichment of event queries by preprocessing and rewriting detection queries from abstract (high-level) and knowledge-base-dependent format to simple event detection patterns that are independent of knowledge bases and can be processed by EPAs. We discuss the advantages and disadvantages of semantic enrichment of complex queries and evaluate our concepts by presenting the results of our experiments.

Contents

9.1 Introduction

One possible approach for the optimization of knowledge-based event processing is to enrich the event pattern rather than enriching the event stream (presented in the previous chapter). Extracted knowledge from external KBs can be retrieved and integrated into the event detection queries so that they can be rewritten to new and simpler forms of event detection patterns (simpler to process).

The queries for knowledge-based complex event processing (presented in Chapter 7) consist of two main parts, the graph pattern for querying external KBs and event detection operators for the detection of correlations between events. As described in Chapters 5 and 7, the processing of such queries requires access to background knowledge so as to be able to understand the complex events. The enrichment of the complex event queries means preprocessing and rewriting complex queries, so that complex events can be detected without accessing the external knowledge bases.

In the following, we provide an example of knowledge-based event processing and describe the idea of query preprocessing with the use of this example.

Example Use Case: Detection of Stock Market Events Based on Company Relations: One example for the potential usage of query preprocessing can be found in high-level stock market monitoring (described in Chapter 4). A stock broker is interested in detecting stock market events based on business relations of companies. He can access background knowledge about the different trading stocks and their companies. He wants to use this knowledge to detect interesting stocks from the raw event stream so that he can start timely reactions to buy or sell shares.

In order to enable such complex queries, a knowledge base is need, that includes the required background knowledge, i.e., the underlying ontology containing predicates like `hasResourceDemand` (a company has specific demands) and `isSupplierOf` (a company supplies a specific material to another company). Suppose the user knows the ontology, he can write a knowledge-based event detection query like those listed in Listing 9.1 (knowledge-based queries are described in Chapter 7).

The event detection query listed in Listing 9.1 detects two stock market events (e1 and e2). The companies of these stocks have material relationship to each other, one company has demand for the material produced by the other company. If the producing company has problems producing the material the other company might also get problems. For the broker this relationship enables more in-depth knowledge and reasoning about companies that can be used in the stock market exchange.

```
{
{(?e1, :e2kb1, ?company1) .  (?company1, :hasResourceDemand, :Material1). }
 (e1.price < 2%  SEQ  e2.price < 2% )
{ (?e2, :e2kb2, ?company2).  (?company2, :isSupplierOf, :Material1) . }
} [Within 5 min.]
```

Listing 9.1: Pseudocode of Event Detection Query for the Relation of Two Stock Market Events

In this example, the preprocessing and enrichment of the complex query is to separate it into two parts by dividing the query from the event algebra operator (in this example the SEQ operator). Each of these query parts is made up of knowledge graph patterns that can be sent to the external KBs. The results of these queries are a long list of event instances (stock market names). The complex query is ***semantically equivalent*** (specified in Definition 9.3) to a set of simple queries that can be built using the event algebra operator and the result lists. One example of such a simple query is $(Renesas < 2\% \ SEQ \ Apple < 2\%)$, because the Apple company and Renesas company have some business relationship (We assume the required knowledge about their business is stored in a KB.).

This chapter is organized as follows: First, we set up the main problem of query preprocessing and rewriting, build our problem model and define the terms (Section 9.2). We get into the details of our approach for query enrichment (Section 9.3). Furthermore, we propose an extension approach for solving the problem of false positive event detection by dividing the event processing into two separate processing phases (Section 9.3). The evaluation of the approaches proposed is done by comparing the performance of the event processing system with and without preprocessing and rewriting (Section 9.4).

9.2 Complex Event Query Preprocessing and Rewriting

In this section, we describe our approach for the realization of the SCEP engine by preprocessing complex event queries. We propose an approach for Event Query Preprocessing and Rewriting (Event Query Pre-processing and Rewriting (EQPR)) complex queries.

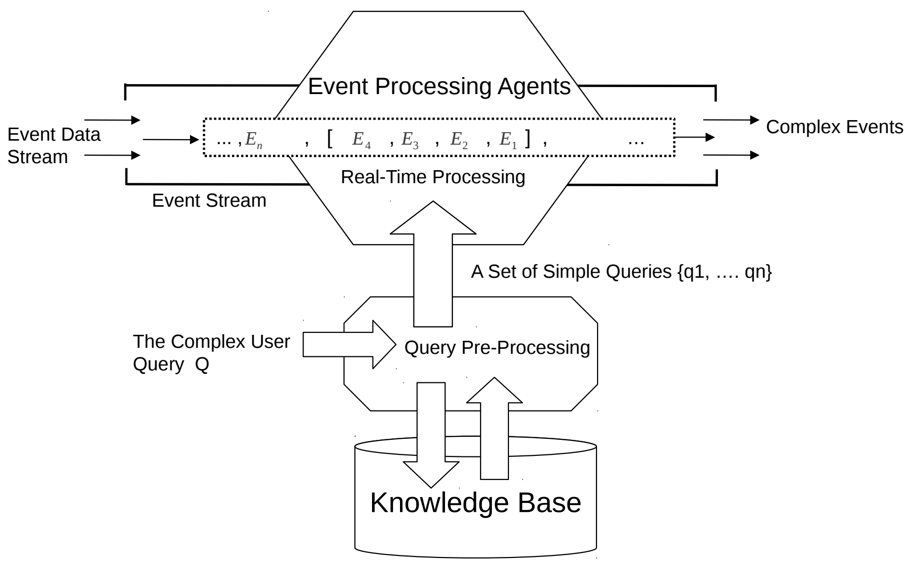


Figure 9.1: Knowledge-Based Event Processing with Query Preprocessing and Rewriting

Figure 9.1 shows the abstract overview of our approach for knowledge-based event detection query preprocessing and rewriting. Prior to the event processing step highly-abstract event detection queries are preprocessed and enriched with the required background knowledge. The query is then rewritten to new simple queries. A comparison of Figure 9.1 with Figure 5.1 (from in Chapter 5) illustrates the additional preprocessing step.

We define *Knowledge-Dependent Query* and *Simple Event Detection Query* as follows:

Definition 9.1. ***Knowledge-Dependent Query (sQuery)*** *A knowledge-based complex event detection pattern (described in Chapter 7) is called a knowledge-dependent query.*

Definition 9.2. ***Simple Event Detection Query (eQuery)*** *With the term simple event detection query, we refer to simple event detection queries (syntactic event detection patterns) that can be processed without usage of external knowledge-bases.*

The original complex event query *sQuery* is preprocessed using a knowledge base and is divided into a set of simple event queries $eQueries = \{q_1, \ldots, q_n\}$. Simple queries can be processed only with information from the event stream without using background knowledge.

The complex query *sQuery* can be considered a propositional formula that can be converted into disjunctive normal form (DNF) $eQ_1 \vee ... \vee eQ_n$. If any of the simple queries has results, then the complex event query is satisfied. Preprocessing is done by a processing agent that can access the knowledge base and divide the complex query into several simple queries, e.g., the query example in Listing 9.1 can be divided into several simple queries, which includes the names of companies in the syntactic event detection pattern.

Our query rewriting approach is based on results extracted from the KBs for the graph patterns inside the sQuery. The sQuery is separated into graph patterns. The results for each of the graph patterns (subqueries) are similar to the materialized view [Yan97, Gup99, Agr00] in databases. Graph patterns specify views on the knowledge stored in the knowledge base. Materialized views are similar to the caching technique that is used to accelerate query processing in databases.

Castillo et al. [Cas10] use the materialized view technique in the RDF data store to accelerate query processing. They propose an algorithm to automatically suggest a set of indexes as materialized views based on a workload of SPARQL queries. And they use the selected set of indexes to decrease the cost of the SPARQL query processing workload.

The preprocessing and rewriting of queries are not similar to query expansion approaches, because in query expansion other new queries are derived that are not given in the main query. In EQPR each of the query generated is a partial query of the complete query, a match to any of the simple queries matches the whole query. The sQuery has more results than a single eQuery. The result set of rewritten eQueries is equivalent to the result of an sQuery.

9.2.1 Problem Setting for Event Pattern Rewriting

The problem that we are addressing is the preprocessing and rewriting of knowledge-dependent queries ($sQuery$) (as we have also referred to $sQuery$ in the previous chapters) to simple event detection queries ($eQuery$).

Givens to the problem are:

- A knowledge base (KB) that stores all required knowledge including ontological background knowledge (A and T-Boxes).
- A knowledge-dependent query composed of SPARQL based graph patterns (BGPs) and event algebra operators.
- A set of event streams $S = \{S_1, \ldots, S_m\}$ that includes event instances from a set of event types $E = \{E_1, \ldots, E_n\}$.

What is required is a method to rewrite the $sQuery$ into a set of simple queries (Set_{eQuery}) so that:

- A set of simple queries detects the same complex events as the original knowledge-depended query. $sQuery \equiv Set_{eQuery}$
- Simple queries can be distributed and processed by more than one EPA.

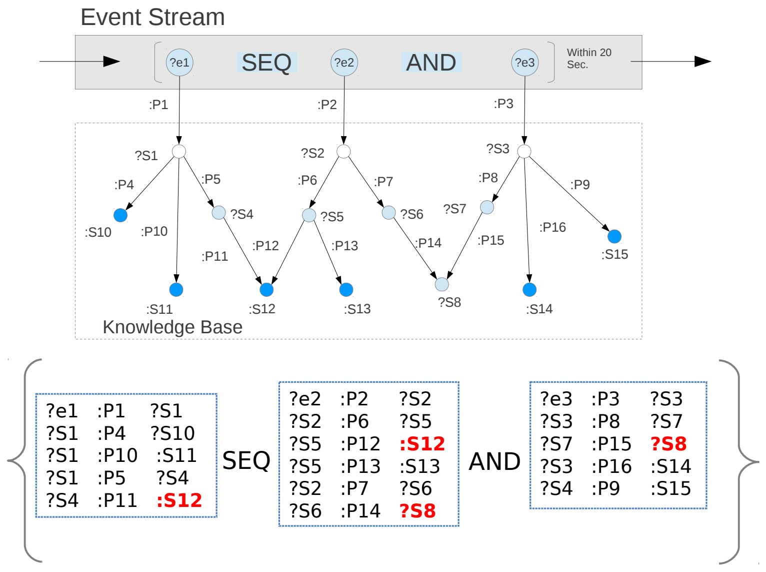
Assumptions:

- The system has a limited size of the main memory space for the processing of events so that the total number of event detection rules is limited to the main memory space.
- The user-specified event detection query has the structure described in Chapter 7. We focus firstly on the main four event detection operators SEQ, AND, OR and NOT.
- Updates on the stored knowledge on the external knowledge-base are infrequent. In some of the use cases (see Chapter 4) the knowledge base is not updated frequently, e.g., high level stock market monitoring. For example, the directory of a company is not changed every minute, or a company will not normally lay off lots of employees every hour.

  In this approach, we do not address the problem of shifting knowledge updates to event detection queries. Any change in the knowledge base requires the reprocessing of all the complex event queries related to the change. It requires an update on some of the rewritten simple queries.

9.2.2 Complex Query Rewriting

As we have described in Chapter 7, the sQuery can be divided into several subqueries by separating the query from the event algebra operators, e.g., the example provided in Listing 9.1 can be divided into two parts. Figure 9.2 shows an example of a knowledge-based event detection query.



```
?e1  :P1   ?S1          ?e2  :P2   ?S2          ?e3  :P3   ?S3
?S1  :P4   ?S10         ?S2  :P6   ?S5          ?S3  :P8   ?S7
?S1  :P10  :S11   SEQ   ?S5  :P12  :S12   AND   ?S7  :P15  ?S8
?S1  :P5   ?S4          ?S5  :P13  :S13         ?S3  :P16  :S14
?S4  :P11  :S12         ?S2  :P7   ?S6          ?S4  :P9   :S15
                        ?S6  :P14  ?S8
```

Figure 9.2: An Example of a Knowledge-Based Event Query Rule

The knowledge-based query *sQuery* is separated by the event operations into a set of graph patterns $G = \{G_1, \ldots, G_n\}$ (here also called subqueries). Each of the event operators in the *sQuery* separates an event type (BGPs specify a semantic event type by using the knowledge graph pattern). A SPARQL query can be generated by removing the event operators from the *sQuery* and by using the basic graph patterns included inside the *sQuery*.

The set of graph patterns G have a set of results R, when they are sent to external KBs. The results include event instances that are potential answers to the *sQuery*, because they have specified relations in the KB (specified in sQuery).

If the variables in the graph patterns are distinct and the graph patterns have no variables in the intersection nodes (graph pattern variables), they can be sent individually as stand-alone SPARQL queries to the KB, otherwise they have to be joined pairwise to build the BGPs of a SPARQL query.

Suppose we have a result set $R = R_1 \ldots R_n$ for the set of graph patterns G extracted from the sQuery. By combining the result sets inside the given event operators in sQuery, the sQuery can be rewritten into a set of simple queries $S = eQ_1, \ldots, eQ_n$. We say that the set of simple queries S for the event detection has *equivalent operational semantics* as sQuery. This means the final results of event detection using the original sQuery is the same as using several simple queries from set S.

Definition 9.3. ***Equivalent Operational Semantics*** $(\equiv)$ **:** *A complex knowledge-based query sQuery and the set of simple queries* Set_{eQuery} *on schema* R_{eQuery} *are said to have equivalent operational semantics, if they can detect the identical output of complex events.*

$$sQuery \equiv Set_{eQuery}$$

Simple queries can be generated by replacing event types (BGPs) with event results, integrate and combine them with the event operators used. Based on the subquery results and type of event operators different kind of simple queries can be generated. The set of simple queries Set_{eQuery} generated might be large depending on the subquery results. Suppose we have n number of results sets and each of the result sets has at maximum m number of results. The cartesian product is an ordered pair that is non-commutative and non-associative. The maximal size of Set_{eQuery} is n^m.

In the following we illustrate the generation of simple event queries by using an example.
Example - Query Rewriting: The sQuery shown in the Listing 9.1 includes two event instance detection e_1, e_2. The SEQ event operator connects the two events e_1 and e_2 so that we have two subqueries $G_1\ SEQ\ G_2$. The $sQuery$ is equivalent to subqueries and the simple queries generated.

$sQuery \equiv G_1\ \ SEQ\ \ G_2$

Suppose we have two result sets for these subqueries $R_1 = \{a, b, c\}$, and $R_2 = \{x, y, z\}$.
$sQuery \equiv e_1 \in R_1\ \ SEQ\ \ e_2 \in R_2$
We can rewrite the $sQuery$ using the following matrix form:

$$\begin{bmatrix} a & b & c \end{bmatrix} SEQ \begin{bmatrix} x \\ y \\ z \end{bmatrix} \longleftrightarrow \begin{bmatrix} (a\ SEQ\ x) \\ (a\ SEQ\ y) \\ (a\ SEQ\ z) \\ (b\ SEQ\ x) \\ (b\ SEQ\ y) \\ (b\ SEQ\ z) \\ (c\ SEQ\ x) \\ (c\ SEQ\ y) \\ (c\ SEQ\ z) \end{bmatrix}$$

We have the disjunctions of the event query rules:
$sQuery \equiv \{(a\ SEQ\ x) \vee (a\ SEQ\ y) \vee (a\ SEQ\ z) \vee (b\ SEQ\ x) \vee (b\ SEQ\ y) \vee \ldots\}$
The matching of each of the simple queries determines the sQuery to be true. In general, the generated simple queries are in a disjunctive normal form (DNF). The sQuery can be evaluated to be true if any of the event detection rule clauses is determined to be true. The result sets of the graph patterns might have intersections or be disjointed which affect the query rewriting process.

9.2.3 Effect of Event Operators

Because each of the event algebra operators has its operational semantics, the rewriting of queries should be done based on the event detection operators used. In this section, we investigate the effects of the four main event operators (SEQ, AND, OR and NOPT) on query rewriting.

SEQ Operator

If the sequence operator is used in the sQuery then two event instances that happens in the sequence and have the specified relations in background knowledge have to be matched. We have two graph patterns that might be connected to each other and are operationally equal $sQuery\ \equiv\ G_1\ SEQ\ G_2$. This means that by using the graph patterns we can get the result sets from the KB so that $G_1\ SEQ\ G_2\ \equiv\ R_1\ SEQ\ R_2$.

Based on the definition of the SEQ operator, the results have to be used for query rewriting.
$\mathsf{SEQ}(e_1, e_2)[w] = \forall(t_s^1, t_s^2)(e_1(\bar{t^1}) \wedge e_2(\bar{t^2}) \wedge t_s^1 \leq t_s^2 \wedge (e_1, e_2) \in w)$
The set of simple queries can be generated by building a cross product of the two result sets.
$Set_{eQuery} \equiv R_1 \times R_2$
In the case that the result sets have any intersections, the sQuery can be rewritten based on the intersection set.
$R_{intersection} = R_1 \cap R_2$
$R_{intersection} \neq \emptyset) \implies Set_{eQuery} \equiv R_{intersection} \vee (R_1 \times (R_2 \backslash R_{intersection}))$
If an event e_x arrives into the CEP system, it can be matched as a complex event, if it can be found in the set of $R_{intersection}$ or it can be matched to the simple queries generated from the cross product by using the result sets.

AND Operator

The AND operator can be seen as a sub-operator of the SEQ operator, because every complex event that can be detected by the SEQ operator can also be detected by the AND operator. The operational semantics is similar to the SEQ but without the sequence constraints.
$\mathsf{AND}(e_1, e_2)[w] = \forall(t_s^1, t_s^2)(e_1(\bar{t^1}) \wedge e_2(\bar{t^2}) \wedge (e_1, e_2) \in w)$
A complex query with the AND operator is equivalent to its graphs and event results.
$sQuery \equiv G_1\ AND\ G_2 \equiv R_1\ AND\ R_2$ Like the sequence operator the set of simple patterns can be built by the cross product of the result sets.

OR Operator

The OR operator is an interesting operator for query rewriting and is defined as follows:
$\mathsf{OR}(e_1, e_2)[w] = \forall(t_s^1, t_s^2)((e_1(\bar{t^1}) \vee e_2(\bar{t^2})) \wedge (e_1, e_2) \in w)$
$G_1\ OR\ G_2 \equiv R_1\ OR\ R_2$
The arriving event e_x has to be matched to one of the graphs so that the complex event is matched. The graphs can be replaced with the result sets. The two result sets can be joined to form:
$Set_{eQuery} \equiv\ if\ e_x \in (R_1 \cup R_1)$
We showed that the knowledge-based query can be mapped to a search problem in the result sets. The results can be kept in the main memory if it can be fitted to the main memory space. It is possible to use database technologies to build indexes on them for efficient searches and use further optimization approaches. We can assume the search of an event in a result set can be done with the asymptotic constant cost of $\mathcal{O}(1)$.

NOT Operator

The NOT operator matches the absence of an event within a specified window. In the case that e_1 does not happen in the window we have:
$\mathsf{NOT}(e_1)[w] = e_2(\bar{t^1}) \notin w$
In the knowledge-based query, we have the inverse of a simple matching operator.
$sQuery \equiv NOT\ G_1 \equiv\ if\ e_x \notin R_1$
If an event e_x arrives the system has to determine if the event is in the result set or not. If not the operation is matched. Similar to the OR operator, the matching of NOT operator is a search problem in the data items.

9.2.4 Tasks of Query Preprocessing and Rewriting

The preprocessing and rewriting of a sQuery consist of several subtasks. The givens in these tasks are the knowledge stored in the knowledge base and the user-specified knowledge-based query (sQuery). Based on the query structure, the graph structure and the event operators used, the sQuery can be divide into a set of simple queries (eQuery).

The Tasks for the preprocessing and rewriting are:

1. Extract a set of graph patterns from sQuery based on event Operators. $sQuery = \{G_1, \ldots, G_n\}$
2. Remove the event operator and generate a graph query (a SPARQL query). $Q_{sparql} = createSPARQL(G_1 \cup \ldots \cup G_n)$
3. Sent the Q_{sparql} and retrieve the results from the external KBs for each event variable.
4. Find the intersections between result sets $R_{intersection}$ and merge all the result sets $R_{complete} \equiv R_1 \cup \ldots \cup R_n$
5. Rewrite the sQuery based on the type of event operators described in Section 9.2.3.

Result sets are simple queries that can be applied to EPAs to process the raw event stream. As described, each of the matches of simple queries activates the original sQuery. The results of the tasks can also be different sets of query partitioning, the best optimized partition can then be chosen based on other related attributes and models, e.g., cost models.

9.2.5 Advantages and Disadvantages of EQPR

Advantages of the EQPR approach are:

- **Improvement of event processing throughput:** The performance of the EPA is highly improved, because there is no communication with external KBs. In the case of the processing of events by a network of several EPA the system can achieve maximum system performance, because of the distribution of simple queries on multiple EPAs. Each EPA has only to process a limited number of simple queries based on their processing capacity (while monitoring the complete stream).
- **Improvement of system scalability:** The main advantages of this approach are its highest scalability level because of the distribution of EPAs, and its high performance due to simplicities of the final queries.

Disadvantages are:

- **Out of Date Knowledge:** The background knowledge used for event processing might be different than the version of knowledge used for the preprocessing of queries.
- **Generation of Large Number of Queries:** The EQPR approach generates a large number of simple queries that have to be loaded into event processing agents. A central EPA has limited main memory and can load the queries with this limitation.
- **Delay on Knowledge Base Updates:** One disadvantage, in the case of knowledge base updates, is the delay on the regeneration (refreshing) of simple queries and their updates.

9.3 Two-Phase Event Processing Approach

As we have described in the previous section, the main disadvantage of EQPR is that the extracted knowledge from the knowledge base might be out of date at the time of event processing. Query graphs are materialized and event detection queries are rewritten using extracted results from the knowledge base. Any changes on KBs are not included in the simple queries generated. Knowledge from external knowledge bases is used in the EQPR approach for query rewriting at the time before the EPA processes the event stream.

In this section, we propose an approach for event processing that is based on two-phase knowledge which in turn is based on communication with external KBs in two separate event processing phases. We call this approach *Two-Phase Knowledge-Base Query (2KBQ)* which is illustrated in Figure 9.3. In this approach the event processing agent can include knowledge base updates. We use the first processing phase to filter out a subset of arriving raw events to reduce the processing load and improve total performance. If events are not matched to the subgraph they cannot be matched to the complete graph. Our approach consist of two main phases in which in the first phase the raw events are pre-filtered by machining them to a large/size potential event candidate. Potential event candidates are extracted from the KB by using a subquery of the complete query. In the downstream the events matched are then matched to the complete event detection pattern.

The first processing phase is used to select event instances that are possible candidates for matching to the detection pattern. The system selects a subquery (a subgraph) of the given query. Using this subgraph the system extracts a module of the external knowledge that is queried for event detection. We assume we have some heuristic data about the knowledge-graph stored in the KB that some parts (modules) of the knowledge is not as frequently updated as other parts of the knowledge. The sQuery is then rewritten into a simple event detection query (based on the subgraph) that can detect complex events. In the first phase we process the events based on this simple event query. Because in the first phase complex events are detected based on a subgraph of the sQuery; the set of detected complex events is a subset of existing complex events based on the complete user query.

In the second processing phase the complete query is used to query external KBs and detect complex events. In this way, the latest updates on external KBs can be used for event processing.

The 2KBQ approach can reduce the processing load on external KBs and improve the throughput of the system, because there is no communication with external KBs in the first processing phase used to filter part of the stream. Only if the EPA can detect event candidates in the first phase, then in the following phase can they be proved by matching to the latest knowledge using the complete query. The initial processing phase selects some of the instances most suspicious to the target complex events.
The two phase of this approach:

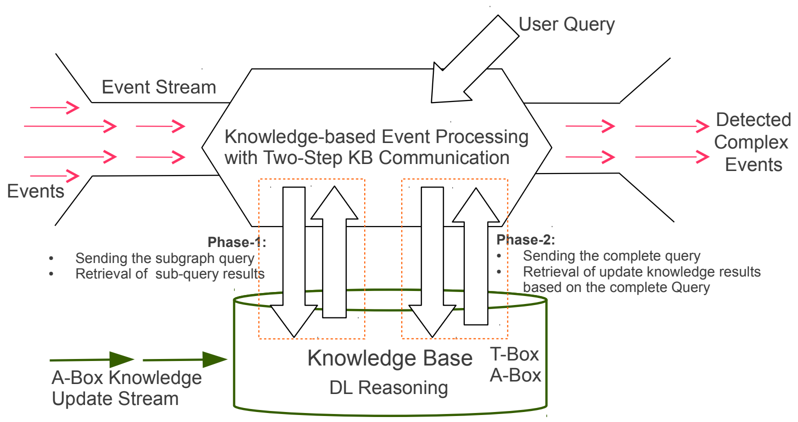

User Query
Event Stream
Events
Knowledge-based Event Processing with Two-Step KB Communication
Detected Complex Events
Phase-1:
Sending the subgraph query
Retrieval of sub-query results
Phase-2:
Sending the complete query
Retrieval of update knowledge results based on the complete Query
A-Box Knowledge Update Stream
Knowledge Base
DL Reasoning
T-Box
A-Box

Figure 9.3: Knowledge-based Event Processing with Two-Step KB Communication

- **Phase 1 - (Preprocessing Task) Selection of a Subquery:** Select a subquery from the given user query that includes a subgraph for each instance. The user query, on the event algebra operators, is divided into several modules. From each module a subgraph (one or more BGPs) is selected that includes event2KB links[1]. The subgraph is selected based on some heuristic data about background knowledge. The heuristic is about the probability of updates on different ontology modules of the knowledge base, e.g., if we know predicate p_x is updated less infrequently than other predicates of the given user query then p_x will be selected. The user query includes a set of basic graph patterns G_{sQuery}. Some of the BGPs include events as variable in the pattern (like ?e1, ?e2, ?e3, in the given Example 9.2), we refer to this set as E_{BGPs}. The E_{BGPs} set includes all the triple patterns that include the event of the event variables $?e_1, \ldots, ?e_n)$. In the case that the BGPs in E_{BGPs} include the same predicate and the same objects, then they can be merged and identified as one BGP.

  We generate a set of subgraphs $G = g_1, \ldots, g_n$ from the BGPs in the E_{BGPs} set. Based on the heuristics about updates on the KB and frequency of updates of different predicates in BPPs, we include some of the other BGPs from G_{sQuery} with the lowest update frequency. (This step is an optional step to improve filtering functionality in this phase.)

```
{ { ?e1 :p1 ?s1 .  ?s1 :p2 :o1. }
    [?e1 SEQ ?e2]
 { ?e2 :p3 ?s2 .   ?s2 :p4 ?o2 . ?s2 :p5 :o3 . }
   [?e2 SEQ e3]
 { ?e3 :p5  ?s3 .  ?s3 :p6 ?o4.   }
 } [Within 5 sec.]
```

Listing 9.2: Example of a Knowledge-Based Event Detection Query

Listing 9.2 shows an example of a knowledge-based query. From the given example query in Listing 9.2 the basic graph patterns shown in Listing 9.3 is extracted and used for event detection in the first event processing phase.

```
g1 = { ?e1 :p1 ?s1 .  }
g2 = { ?e2 :p3 ?s2 .  }
g3 = { ?e3 :p5 ?s3 .  }
```

Listing 9.3: Example of a Selected Subgraph

[1]The links that connect an event instance to the resources in the KB, (described in Chapter 5).

- **Phase 1 - Event Processing based on the Selected Subquery (Event Processing Subtask):** The set of subgraphs G is used to query the KB so that the results for each event instances can be returned. The results are then merged so that we have at the end a set of results R_G. In the first phase the raw event stream is processed using the result set R_G.

  Each arriving event instance has to be found in R_G so that it can be used as a candidate for matching as a complex event in the next phase. Event processing in this phase can achieve a higher throughput because no communication with external KBs is needed for event processing. Events are only matched based on the sets of results R_G.

  In the case that the result set can be indexed and cached in the main memory space, we can match the events to the result set with a constant complexity $\mathcal{O}(1)$. If the volume of results is higher than the space of the main memory, they can be stored on external distributed databases in which H number of hosts participate (i.e., using state-of-the-art approaches in distributed databases which employ distributed hash tables [Han11]), and we can access the data with the routing cost of $\mathcal{O}(H)$. The matching process of events to the set of results might be realized in a parallel and distributed setup, so that the matching process can scale and maintain the high throughput rate.

  The output-stream of the first phase is sent to the second processing phase. The events in the up-stream and down-stream have the same event sequence order.

- **Phase 2 - Final Processing of the Detected Complex Events:** In this processing phase, the events detected from the first phase are processed using the complete user query (sQuery). We send the whole user querying for the detection of complex events to the external KB and check the detection against the latest version of the knowledge. This is realized by polling the KB on each arriving event instance from the first processing phase. Consequently, the frequency of querying the KB is not as often, due to filtering, as the event stream in the first phase.

By using the 2KBQ approach, the system may not detect all of the existing complex events, because in the first phase it filters events using *out of date knowledge*, which is extracted before the event processing time point. For this reason the 2KBQ approach is applicable only in some special use cases in which complete detection of existing complex events is not required.

9.3.1 Correctness and Completeness of Detected Complex Events

Because events are processed in a second event processing phase, in the 2KBQ approach complex events are detected based on a complete query sQuery and the latest version of the background knowledge. The second processing phase provides the proof of correct detection of complex events.

However, the 2KBQ approach cannot detect all existing complex events if the knowledge stored in the external KBs is changed in the brief interval between simple query generation in the first phase and event processing in the second phase. In this case, processing in the first phase may drop some of the event instances, which are components of the queried complex events by sQuery.

In such situations, the system cannot detect all the existing complex events. This means the 2KBQ approach can provide 100% precision in the detection of complex events, but not 100% recall of existing complex events from the raw event stream. In other words, we have no false positives (none of the complex events are wrongly detected), but we have false negatives (some of the complex events are not detected) relevant to the NOT event detection operator. If the system detects no complex events, it does not mean there are no complex event in the event stream.

The 2KBQ approach is only applicable for use case scenarios that have no problems if some of the complex events are not detected, but they attempt to detect them based on abstract knowledge-based queries as well as detect the complex events close to real-time.

9.4 Evaluation

In this section, we evaluate the proposed concepts for query rewriting (described in Sec. 9.2) and two-phase KB communication (described in Sec. 9.3).

We have conducted several experiments with different queries so that we can test performance and latency improvements using the concepts provided to compare the processing of events by polling the external knowledge base.

For the evaluation we used the same methodology and experiment setup[2] used in Chapter 8.4. We used the same event stream dataset and a mirror of DBpedia as the background knowledge dataset.

To cleanly separate the impact of our approach from those of the underlying implementation and configuration choices, we compared the evaluation metrics with each other on the same implementation setup and abstract performance metrics used. We compared the event performance of different approaches on the same implementation and data setups. To separate the impact of specific data on our experimental results, we executed the experiments on different event streams, knowledge bases and queries.

In our experiments, we used the same query sets (Listed in Appendix B) that we used to evaluate the event enrichment approach (described in Chapter 8).

For the one-step event processing approach we use the event enrichment approach from Chapter 8 in a single processing step. We used for event enrichment the SPARQL queries similar to sQ2 to sQ7 and sQorm2 to sQorm7 listed in Appendix B. For event detection based on different operations we used eQopm1 to eQopm7 and eQorm1 to eQorm7 (changed depending on event operators), but we adapted the objects to the triple patterns specified in Table 9.1.

Table 9.1 shows the different triple patterns we used in our experiments and lists the number of results for each of the patterns.

[2]In our experiments we used a single instance of an external knowledge base which is an instance of the Virtuoso triple store[3] and is installed on a host (Intel Xeon CPU E31245 @ 3.30GHz) with 8 GB RAM and Ubuntu Linux 12.04. We use the default cache configuration of virtuoso[4], so that we had little memory and can have external knowledge with limited cache capacity and poor performance results. We had an instance of Esper event processing engine installed on a host (Intel(R) Core(TM) i7-2600 CPU @ 3.40GHz) with 16 GB RAM and Ubuntu Linux 12.04. We ran the Java threads with a maximum heap memory of 14GB. The event processing agent and the triple store were installed on a separate host. The knowledge base host and event processing host were connected by a dedicated 100 Mbit LAN.

No.	Predicate, Object	Number of Results
1	*(?s dbpedia-owl:location dbpedia:United_States)*	10760
2	*(?s dbpedia-owl:industry dbpedia:Software)*	1068
3	*(?s dbpedia-owl:numberOfEmployees ?o)*	12425
4	*(?s dbpprop:products dbpedia:Laptop)*	9
5	*(?s dbpedia-owl:service dbpedia:IT_service_management)*	16
6	*(?s rdf:type dbpedia-owl:Company)*	88054
7	*(?s dcterms:subject Category:American_brands)*	132

Table 9.1: Number of Results for Different Triple Patterns

9.4.1 Improvements Using Query Rewriting

To evaluate our concept for the rewriting of complex queries to simple queries, we conducted different experiments with the same complex query sets. We processed the stream of events once with the one-step event processing without query rewriting (KB Polling) and once with the rewritten queries (matching with simple queries).

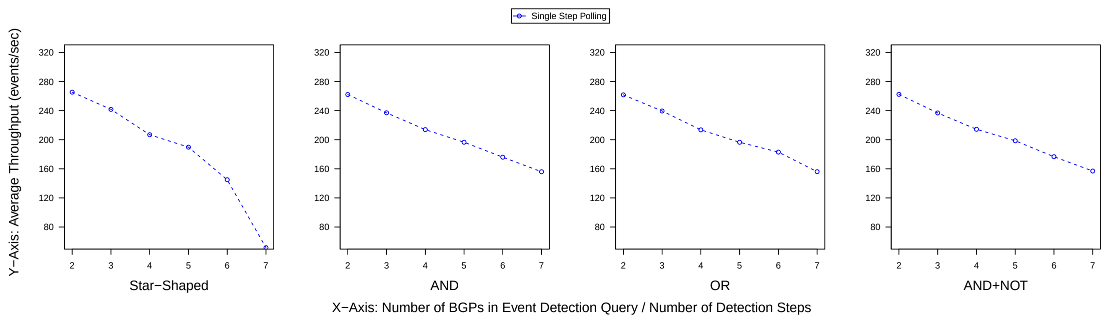


Figure 9.4: Knowledge-Based Event Processing with Single-Step KB Polling Approach

Figure 9.4 shows the performance of event processing using the single-step KB polling approach. For star-shaped queries and queries using AND/SEQ, OR or NOT operator performance decreased with the number of basic triple patterns used in event detection queries, which also indicated its increase with the number of materialized results from the external KB.

In the following we first experiment with the performance of the system by rewriting the star-shaped queries and queries with *OR* operators. Afterwards, we get into the details of the experiments with the AND and SEQ operators.

Star-Shaped Queries and Queries with OR Operator: These types of queries can be rewritten to queries that match an incoming event instance to a set of events. We conducted several experiments with this type of queries using triple patterns from Table 9.1. Our experiments show it was possible to process such kinds of queries with up to 28,000 events per second.

The performance of event processing with such kinds of queries is consistent w.r.t. large-size event result set (up to one million cached event instances), due to the fact the system can match the arriving events to a set of events. The event processing system can be kept constant up to the heap memory size of the system. In comparison to the KB polling approach (shown in Figure 9.4) we can see a significant improvement in event processing performance.

Queries with SEQUENCE or AND Operators: These types of queries can be rewritten to queries that include SEQ or AND operators. We rewrote this type of queries to the event detection queries like those listed in the 9.4. The results retrieved from querying the external KB can be added to the event detection query so that we can have several queries or one single query with large number of objects to be matched like *(in ('url1-1', 'url1-2', . . . , 'url1-n') those shown in Listing 9.4)*.

```
Select * from pattern [every e1=StockEvent AND e2=StockEvent] where

e1.url in ('url1-1', 'url1-2',  . . . , 'url1-n')

e2.url in ('url2-1', 'url2-2',  . . . , 'url2-n')
```

Listing 9.4: Example of an Esper Event Detection Query with Sequence Operator

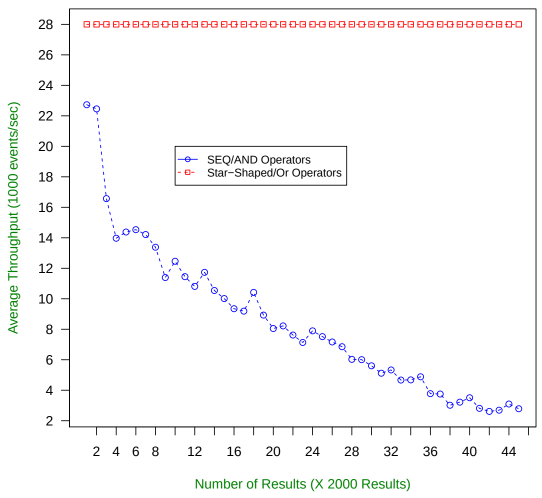


Figure 9.5: Performance of Rewritten Queries with Different Cached Results

Figure 9.5 shows the result of our experiments with rewritten queries with AND operators. We sent queries to the external KB that could return from 4000 to 88000 KB results (each point the experiment is with a query with x2000 result numbers). We ran the event processing with the rewritten queries like those shown in Listing 9.4. We sent the 500k events through the event processing system and calculated the median performance of the last 20 second of the system. Our result showed that the performance of the system for Star-Shaped and OR operators are constant, while the performance of the queries with AND/SEQ operators decreased continuously with the rise of the number of KB results.

This experiment illustrates that a single system has its cache capacity and queries with large number of results should be rewritten to several queries that can be distributed across an event processing network.

9.4.2 Improvements using Two-Phase Processing

Our experiments with the two-phase processing approach shows this approach depends greatly on the filter functionality of the first processing phase and the query used in the final phase. The performance of the first step is similar to the above experiments with star-shaped queries or queries with OR operators, due to the nature of matching events to a KB result set. However, the performance of the second phase is similar to the experiment results shown in Figure 9.4, due to the usage of the KB polling approach in the final checking phase. For the set of basic triple patterns shown in the Table 9.1 we observe an event processing performance between 25k to 28k events per second.

9.4.3 Knowledge Update Problem

In the case of any updates on external KBs, some of the rewritten queries might be invalid because of changes in KB result sets. These kinds of updates in the KBs and the subsequent result updates can be propagated to the rewritten queries. We realized this by repeating the rewritten task and comparing the resulting queries with the queries deployed. In our experiments this kind of query updates results in a break in the event processing performance (a break is about a microsecond) before returning the system to the normal performance level.

The main challenge of KB updates is that the event processing system cannot provide guarantees for correct complex event detection. The system can only provide high precision and recall for complex event detection, because it can only process KB results received from external KBs.

In the case of deploying rewritten simple queries to an EPN, e.g., several event processing engines deployed on Storm network[5], Apache Kafka[6], Akka[7] or Finagle[8], the queries can be updated without a performance break.

9.5 Summary

In this chapter, we have described our approach for the preprocessing of queries and their rewriting to simple queries that can be processed without KB communications. The main disadvantage is background knowledge used for event processing might be different than the version of knowledge used for the preprocessing of queries and the EPAs might detect incorrect complex events. To solve this problem, we proposed another approach called 2KBQ that provides accuracy guarantee for the detection of complex events. In the 2KBQ approach raw events are processed in two processing phases by different event processing engines, first filtering based on simple set matching and then polling the KB on a lower rate of event frequency.

In the following chapter, we will propose an approach for optimized sampling of event stream based on stream properties and event detection pattern. The sampling of raw event stream is an approach to reduce the processing costs of knowledge-based event processing.

[5] `http://storm.incubator.apache.org/` retrieved April 2014
[6] `http://kafka.apache.org/` retrieved April 2014
[7] `http://akka.io/` retrieved April 2014
[8] `https://github.com/twitter/finagle` retrieved April 2014

Chapter 10

Event Processing on Sampled Event Stream

In some of the event processing use cases the computation on high throughput event stream is so intensive that real-time event processing systems can hardly process the event stream. One possible solution can be to process only a sub-stream of the original stream, when the use case allows a partial detection of existing complex events. The event processing engine monitors only a sampled stream instead of the original event stream. Existing approaches for data stream sampling are mostly designed to apply an aggregation function to the data stream. In this chapter, we present an approach for optimized sampling of event stream based on stream properties and event detection pattern. Our approach can calibrate the raw stream at a specific rate while maximizing the number of complex events detection, or calibrate the number of detected complex events while minimizing the throughput of raw events.

Contents

10.1 Introduction

Real-time processing of events is hard to achieve when the event stream throughput is high or processing of events requires intensive computation. The enabling of system scalability by using additional computational resources has its high costs that might be higher than user expectation for the realization of use case application. On the other side, in some of the application scenarios it is not necessary to detect all existing complex events from the event stream. For some of the use case, it is acceptable if the CEP system detects only a portion of the existing complex events from the event stream in real-time, rather than detecting all of them with a processing delay.

One solution to this problem is to reduce the rate of events so that the event processing agent can monitor only a small fraction of the original event stream, e.g., in a use case that has an event stream with around one million events per second while the applied event processing engine can only process 500 thousand events per second. The raw event stream should be reduced so that the event processing system can process arriving events in real-time.

In some of the use cases it is acceptable to detect a specific portion of the complex events because they need not be detected. The complete detection of complex events causes unnecessary higher processing costs.

For example, in an urban computing use case (described more in Section 4.3.2), special product offers are matched to people with a smartphone[1] who are on the go in a city. The users specify their interests to the certain types of special offers. The smart phone owner subscribes to the event stream by sending his preferences. The CEP system is also notified about the updated GPS coordinates of users to match the offers to the nearest users. The matching is done based on detection pattern (user preferences), user coordinates and properties of the offers, e.g., types, GPS coordinates of shops. In this use case, it is enough to match only 10 offers per user per hour, rather than detecting all existing offers.

For such use cases, we are interested in sampling the event stream so that we can detect a specific percentage of complex events within a time window in order to reduce the sampled stream throughput as much as possible, thereby reducing the event processing costs. For example, in an event stream we have approximately 100 complex events per hour, but the use case requires only 10 complex events to be detected per hour (the detection time of the complex events is distributed over the course of an hour). The problem we are addressing is the reduction of the raw event stream so that we can detect 10 complex events per hour and reduce the event stream as much as possible.

The main challenge we are addressing is how to sample the event stream so that we can detect the highest possible number of complex events, or detect the required number of complex events. We assume that the entropy (described in Section 3.6.1) of the event stream is known and does not rapidly change over a short period of time (shorter than the time period of sliding window).

In our approach for knowledge-based event processing (presented in the previous chapters), the enrichment of events with background knowledge requires intensive computation and communication with external KBs. The available processing throughput capacity of a knowledge-based CEP system (implemented with the approaches proposed in the previous chapters) might be less than the throughput of the raw event stream. Knowledge-based CEP can process the sampled stream and not the original raw event stream.

In data stream processing a usual approach for applying aggregation functions on the data stream is to generate a sampled stream. There has been a large body of work in data stream algorithms dealing with the generation of a sampled stream using randomization approaches. Therefore we first review the existing algorithm for sampling data stream (Section 10.2). Subsequently, we describe the differences between sampling in data stream and sampling in event detection (Section 10.3), and then get into the details of the sampling research problem and propose our approach for sampling event streams based on specific event detection patterns and event types (Section 10.4). We discuss the correctness of detected complex events after sampling the event stream (Section 10.5). Finally, we present the experiment evaluation of our approach for the sampling of event streams for CEP (Section 10.6).

[1] A cell phone with Internet connection and data processing ability.

10.1.1 Relation to Knowledge-Based Event Processing

As we have shown in the previous chapters, knowledge-based event processing is very much computation and communication intensive. The event processing agent has to communicate with external knowledge bases, query external KBs and do reasoning on the extracted knowledge that can have different complexities and computation costs. The sampling of raw event streams can be a way to reduce computational costs. Computation can be done on the sampled stream and not on the original stream, so that we can have a cost reduction linear to the sampling rate of raw events.

Our approach for sampling knowledge-based event processing is based on the identification of *semantic event types* (event types are defined based on background knowledge). Figure 10.1 shows an event detection pattern based on the external knowledge graph. As shown in Figure 10.1, each of the event component (each of the subgraphs) can be identified as one event type (e.g., the shown event types A, B and C).

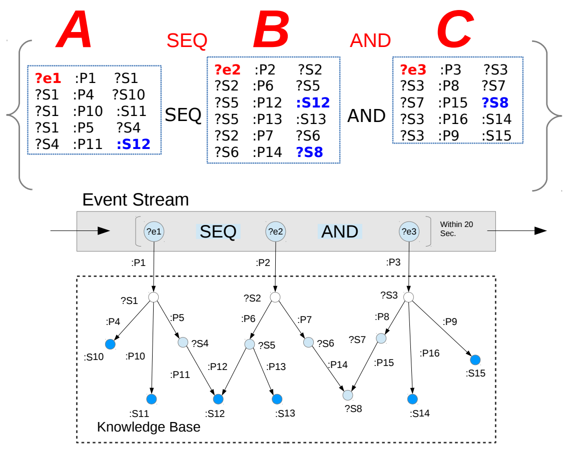


Figure 10.1: Knowledge-based Event Detection Pattern.(Identification of Event Types)

The processing of events includes two processing steps, a sampling step and an event detection step. For the identification of the event types a pre-processing step is required, in which the events are only controlled with the required partial knowledge, i.e., checking one of the event attributes is enough to eliminate other event instances and identify approximately the event types for the sampling process. An approximated event type identification is sufficient to decide if an event instance is to be dropped or included in the sampled event stream. In the following step knowledge-based event detection as described in the previous chapters can be applied to the sampled event stream.

10.2 Review of Sampling Algorithms

Sampling methods are simple methods for generating synopsis construction from the data stream [Agg07]. There has been a significant amounts of research on data stream sampling [Mut03, Bab02, Gem06]. In this section, we review briefly the most signification and closely related algorithms on data stream sampling: Bernoulli sampling, Reservoir sampling and Min-Wise sampling.

Almost in data stream processing different functions are applied to the sampled stream, and different approaches are proposed to optimize the sampling so that functions on the sampled stream can produce results close to the original stream. Most sampling algorithms are designed to be applied to aggregation function on the data stream [Agg07] (e.g., mean or sum calculation of the numbers as data items).

The following are the most important approaches for sampling methods:

- **Bernoulli Sampling:** In Bernoulli sampling each data item included in the sampled stream is included with an equal probability p independent of the other items. Sampling probability is independent of the data types and is equal for each data item. The size of the sampled stream is random and for a dataset S (data stream) it is a binomial distribution of $(|S|, p)$. The advantages of Bernoulli sampling are simplicity and simplicity of parallelization.

- **Reservoir Sampling:** Reservoir sampling [Vit85, McL83] is a basic and simple algorithm for sampling data streams. Reservoir sampling can select a random sample of k items from a database S of n items, in which $|n|$ is the size of database S and is a large or unknown number. The sampled k items is called reservoir.

  The process of sampling data items is as follows: the algorithm includes the first k items of S in the sampled reservoir. For each upcoming i^{th} element of S, it generates a random number j between 1 and i. If j is less than k, the j^{th} element of the reservoir is replaced with the i^{th} element of S. For each item of S, the sampling probability is k/i. The Reservoir sampling can sample the data stream in a single pass using constant space and in $O(k(1 + log(n/k)))$.

- **Min-Wise Sampling:** Min-Wise Sampling [Nat04] picks a random tag (a number) between 0 and 1 for each arriving data item. It stores items with the smallest random tag so that each item has the same chance of least tag and uniform. Every subset of the original stream to be included in the sampled stream is uniform. The disadvantage of Min-Wise sampling is the additional memory space required to store the tags and has higher CPU processing costs compare with reservoir sampling. The advantages are that total number of items need not be known in advance and the sampling process can be run on multiple streams separately and merged at the end.

Furthermore, algorithms are proposed based on the algorithms above, and some of them as extensions. Gemulla et al. [Gem06] proposed a reservoir-based algorithm for the maintenance of sample synopses of data stream datasets. Chakrabarti et al. [Cha10] proposed a algorithm to estimate stream entropy.

The existing data stream sampling are not directly applicable in the area of event processing because event detection queries are complex due to their usage of different event detection operators, while event detection needs to sample the event stream so that a rate of complex events can be detected from the stream. To the best of our knowledge, there is no research effort on sampling event stream for usage, which could contribute to the optimization of the sampling process for event detection.

The problem of reducing the event processing load in CEP is similar to the problem of load shedding in data stream processing. Load shedding in CEP is similar to load shedding of items in data streams [Moz10] that have different keys with different rates for aggregate queries.

He et. al [He14] has investigated the problem of load shedding in the area of complex event processing. They studied CEP load shedding under various resource constraints and formalized classes of CEP load-shedding scenarios as different optimization problems. They constructed shedding algorithms with performance guarantees and showed the complexity results that reveal the difficulty of the problem. Their results illustrate the difficulty of developing load-shedding algorithms that maximize utility of the CEP system.

10.3 Event Stream Sampling for CEP

The process of event stream sampling for complex event processing is shown in Figure 10.2 (a, b and c are event instances of event Type A, B and C). Some of the event instances have been selected to be included in the sampled stream while others were dropped. The order and timestamps of the events have not changed with the transference from the original stream to the sampled stream.

In Figure 10.2 the rectangle (rectangle frame that includes some of the events) marks a window of events. Let's assume the EPA has to match the detection pattern *(B SEQ A)*, depending on the formal semantics of the sequence event algebra operator (the event operator algebra are specified in 7.2). It detects 5 different complex events because b and a exist 5 times in the window. If the EPA has to detect the pattern *(B SEQ C)*, it generates 2 complex events. The detection of the operators on the sampled stream, in comparison with the original stream, might generate different numbers of complex events, and in some cases different complex events (incorrect event detection).

In this work we describe a sampling algorithm which can reduce the original stream of events to the sampled stream so that the EPA can detect the maximum possible complex events from the event stream.

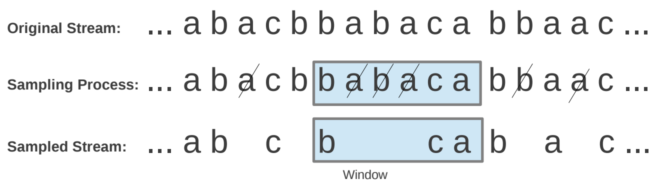

Original Stream:
... a b a c b b a b a c a b b a a c ...
Sampling Process:
... a b a c b b a b a c a b b a a c ...
Sampled Stream:
... a b c b c a b a c ...
Window

Figure 10.2: Sampling of Event Stream for Different Event Types

10.3.1 Problem Setting for CEP Stream Sampling

An event sub-stream L of the original event stream S is built based on a parameter p, $0 < p \leq 1$, for $1 \leq i \leq n$, the event instance e_i is included in L with a probability of p. The EPA is only allowed to monitor sub-stream L and cannot see the original stream S. Hereafter, we represent the term *"sampled event stream"* as L and *"original event stream"* as S. In the case that we have the same setting of *Bernoulli Sampling Setting*, a sampling process with repeatedly perform independent but identical Bernoulli trials. All events of the stream have the same probability of being included in the sample event stream.

The following two optimization problems for stream sampling can be considered:

- *Sampling for a Specific Rate of Complex Events:* We are interested in detecting a specific volume of existing complex events from the event stream (specific number of complex events in a time unit). The EPA should detect $n \pm \tau$ number of complex events, where τ is the tolerance rate while the sampling process should be optimized so that the throughput of the sampled stream is minimized as much as possible.
- *Sampling for Reduction of the Throughput of Raw Events to a Specific Rate:* The use case application requires reduction of the sampled stream throughput at a specific rate because the EPA has a limited processing capacity. We look for a sampling method that can limit the throughput to a specific rate while maximizing the number of complex events detected.

The givens to the optimization problem are:

- The original event stream and set of event types that the stream is generated from.
- The event detection pattern, Query.

The following two categories of sampling approaches are possible based on event type identification:

- *Identifying Event Types:* Sampling probability rates are calculated based on the event types of each event instance.
- *Sampling based on Event Order:* Sampling of events is only realized based on the order of events in the event stream. The sampling probability is calculated based on sequence and temporal relations of event instances. The sampling selector jumps in the stream and picks up event instances without knowing their event types.
- *Sampling based on both Event Order and Type:* Sampling of events is not only realized based on the identification of types but also on the order of events.

10.3.2 Assumptions for Stream Sampling

Before we describe our sampling approach for complex event detection, we point out our basic assumptions for the sampling problem. The sampling of event streams requires several conditions so that it can be used in the event processing applications. In the following we will describe the kinds of conditions that we assume will present itself in our application scenario.

- **Stream Entropy is known:** We assume the entropy (described in Section 3.6.1) of the event stream is known. We know which event types are in the event stream and what the rates of the different event types in the stream entropy are. The approximate rate of different event types in the event stream is known and the different event types are distributed uniformly across the whole stream. Let's suppose an event stream of n different event types $E_1, \ldots, E_n$. A chuck (a long time data window) of this stream for a period of time t, includes approximately w_1 of event type E_1 and w_n of event E_n. The ws are a number between 0 and 1 and the sum of all the values is 1. We call this value *weightings of event types* in the event stream.

  Our assumption is that the stream entropy is known or it is possible to be estimated, and the estimated stream entropy is not significantly different to entropy of the real upcoming event stream.

- **Entropy is not rapidly changing:** We assume the entropy of the event stream does not significantly change during event processing or its changes happen in long intervals (i.e., longer than the time window of event pattern detection).
- **Homogeneous Stream:** Event stream is relatively homogeneous and in each randomly selected chunk of the stream we can find almost s sub-stream with the same entropy.
- **Event Processing Capacity:** The EPA has the capacity to process a complete original stream over a relatively short period of time. However, we sample the event stream to avoid overloading the CEP system over a long period of processing time.
- **Delay for Event Sampling:** We assume the time delay caused by event stream sampling in a separate processing step is acceptable for target use cases, when it has to reduce the event stream to the limited event processing capacity of the EPA.

Our assumption shows that the event sampling approach is unsuitable for event detection use cases in which complex events rarely happens and event processing is used to detect anomalies.

10.4 Stream Sampling for Complex Event Detection

10.4.1 Uniform Sampling

In uniform sampling each event instance has the same probability to be included in the event stream and we have the same setting as with *Bernoulli Sampling*. The event stream is sampled independently of the event pattern. Uniform sampling only guarantees reduction of the raw event stream to a certain rate, but it does not provide any guarantees on the rate of detected complex events from the sampled stream.

Uniform sampling of event stream is independent of the user-specified detection pattern. For different kinds of event patterns and different event algebra operators uniform sampling provides the same sampling probabilities.

10.4.2 Filtering Rates for Each Event Types

Contrary to uniform sampling, in this approach we have different filtering rates for different event types. Each event type can have an independent filter probability, so that based on these probabilities the event instances can be included in the sampled stream. The problem is to find optimize filter probabilities to optimize stream sampling. The givens to this optimization problem are stream entropy and event detection patterns.

Event Pattern: The event detection pattern is defined by the user. The EPA can know the event types used in the event detection pattern. We focus on the four main operators *AND, SEQ, OR, NOT*; their operational semantics are the same as most CEP systems [Ber08, Bry07, Ani11b, Bre07]. We assume we know the types of events used in the event pattern, and the usage frequency of each event type in the event pattern (the number of each event type in the event pattern, e.g., in the pattern *(A SEQ A SEQ A SEQ A SEQ B)* 3 occurrences of A event type and one occurrence of B event type). The frequency of different event types in this example is 2.

10.4.3 Calculation of Sampling Probability Rates

Our approach for sampling the event stream is based on sampling different event types with different probabilities so that each event instance based on its event type has a different sampling probability. Sampling probabilities are calculated based on event stream entropy (the weighting of each event type in the stream entropy) and the given event detection pattern (the event types used in the event detection pattern).

In this section we describe our approach to calculating sampling probability computation for different event types. We propose an algorithm which uses event stream entropy and user-given event detection pattern to compute appropriate sampling probability for each event type. Our algorithm profits from a multi-trail technique that starts with a set of initial sampling probabilities. Multi-trail technique means the algorithm has to process the stream with different sampling probabilities in multiple trails to calculate sampling probabilities for each event type.

The algorithm calculates sampling probabilities for each event type that can produce the requested result value on the event detection side, i.e., the amount of detected complex events or the throughput of sampled stream. We call such result value *the user requested result value (URV).*

Definition 10.1. ***(User Requested Result Value (URV))*** *The URV value is the value requested by the user as the result of event processing. It can be the amount of detected complex events or the throughput of sampled stream.*

The user specifies an input value to the sampling calculation algorithm. For example, the URV can be the rate of the detected complex events that should be 50% of the existing complex events. In this case the URV is 50% of the complex events. If the rate of processed raw events by the EPA should be 70% of the original event stream, the URV is 70%.

Our sampling algorithm can be used for the optimization of the following two goals:

- **Sampling to optimize the throughput of sampled stream forwarded to the EPA:** The algorithm calculates the sampling probability rates so that the sampling system can reduce the raw event stream to a specific rate and maximize the number of detected complex events.
- **Sampling to optimize the rate of detected complex event:** The number of required complex events in a time period is given. The algorithm calculates the sampling rates to find the required number of complex events and minimize the throughput of raw events.

Hereafter, we call the above optimization values *desired optimization result values (OPV).*

Definition 10.2. ***Desired Optimization Result Values (OPV)*** *The OPV value of the sampling system has to optimize depending on the URV. It can be the amount of detected complex events or the throughput of sampled stream.*

Considering the URV and OPV described above, the following two cases are possible:

- If the URV is the rate of complex events, then the OPV is the rate of raw events.
- If the URV is the rate of raw events, then the OPV is the rate of complex events.

Algorithm 3: Algorithm for Sampling Probability Calculation

```
Data: Pattern Query, Stream S, TimeInterval T, URV, MaxTrailNumber
Result: $P$ : A set of sampling probabilities for event types
$E = extractEventTypeSet(Query)$ ;
$P = estimateSamplingProbablities(Query, StreamEntropy)$ ;
for $i \leftarrow 1$ to $E.size$ do
    $E_i$ = Select an event type with maximum weighting from E;
    NumberOfTrails=0 ;
    repeat
        $Result^t = sampleAndDetect(Query, S, T)$ ;
        if $(URV - \eta \geq Result^t_{URV} \leq URV + \eta)$ then
            if $(Result^t_{URV} \geq URV + \eta)$ then
                $P^E_i = P^E_i - Calibration$ ;
                else
                    $P^E_i = P^E_i + Calibration$ ;
                end
            end
            else
                if $(Result^t_{OPV} \leq Result^{t-1}_{OPV})$ then
                    $P^E_i = P^E_i + FineCalibration$ ;
                    else
                        $P^E_i = P^E_i - FineCalibration$ ;
                    end
                end
            end
        end
        NumberOfTrails++ ;
    until
    $((URV - \eta \geq Result^t_{URV} \leq URV + \eta) \wedge NumberOfTrails > MaxTrailNumber)$;
    $P.update(P^E_i)$ ;
end
```

Algorithm 3 shows our calculation of sampling probability rates for each event type. The algorithm uses a multi-trail technique to continuously calculate the sampling rates for each event type to generate the URV. Our algorithm includes 3 loops that run the multi-trail technique; we describe them in the following 5 processing steps:

- **Step 1:** Calculate the initial sampling probabilities $P = \{P^E_1, \ldots P^E_N\}$ for each of the event types $E = \{E_1, \ldots E_N\}$ based on stream entropy, the number of event type repetition in the pattern and the event operators in the detection pattern. A sampling rate is assigned for each of the event types in the event stream. The sampling rates of the event types that are not in the detection pattern is zero value assigned. (The sampling system filters them out of the stream.)
- **Step 2:** Sample the event stream for the time interval T and monitor the results of URV and OPV.

- **Step 3:** If the URV is not within the required value $URV \pm \eta$ (η is a specified tolerance value), then continue to *Step-4* or *Step-6*.
- **Step 4:** If the URV is within the required value, from the event type set select an event type (E_i) which has the maximum weighting in the stream entropy. If the set of event patterns contains no elements then it is terminated.
- **Step 5:** Change the sampling value for E_i, to ($\pm FineCalibration$) and continue with a new iteration until we know the probability values that generates improved results for OPV. For this event type, assign the sampling probability value which produces the nearest results for the OPV.

In Algorithm 3 the terms are:

- S is the original event stream.
- $Query$ is the event detection query.
- P is the set of sampling probabilities for each event types.
- T is trail time interval in which we repeat the sampling probability calculation.
- $MaxTrailNumber$ is the maximum number of trails, specified by the user.
- URV is the User Requested Result Value.
- OPV is the Desired Optimization Result Value.
- $Calibration$ is the calibration value used for calibrating sampling probabilities.
- $FineCalibration$ is the fine calibration value (a value smaller than the "calibration").
- $Result^t_{URV}$ is the results at trail time t.

The complexity of the algorithm provided depends on the number of event types in the event pattern (N_{types}) and the maximum number of trails $MaxTrailNumber$. The total number of cycles in the algorithm for calculating of sampling rates is $N_{types} * MaxTrailNumber$. The number of steps (time complexity) of the algorithm is linear to the number of event types and maximum number of trails.

One advantage of our algorithm is its adaptability to changes in the stream entropy. If the entropy of the stream is changed after a time interval, the algorithm can recalculate the sampling probabilities (using certain calculations trails) for the event types so that it can optimize the result value.

10.5 Correctness of Detected Complex Events

Depending on event detection queries, sampling the event stream results in the fact that some of the complex events extracted from the sampled stream are not the same complex events as they can be extracted from the original stream. In some cases on the sampled stream different complex events might be detected that are different from those in the original stream, because some of the event instances are dropped from the stream and not included in the sampled stream. Based on use case requirements this kind of complex event detection can be considered as incorrectly detected complex events since they cannot be detected on the original stream (and does not exist in reality).

Operator	Events detected from Original	Events from Sampled
every (A -> B)	(a1, b1), (a2, b3), (a4, b4)	(a1, b2), (a2, b4)
every A -> B	(a1, b1), (a2, b3), (a3, b3), (a4, b4)	(a1, b2), (a2, b4), (a4, b4)
A -> every B	(a1, b1), (a1, b2), (a1, b3), (a1, b4)	(a1, b2), (a1, b4)
every A -> every B	(a1, b1), (a1, b2), (a1, b3), (a2, b3), (a3, b3), (a1, b4), (a2, b4), (a3, b4), (a4, b4)	(a1, b2), (a1, b4), (a2, b4), (a4, b4)

Table 10.1: Example of Detected Complex Events from Original and Sampled Event Stream

Let us consider $S_O = \{e^o_1, \dots e^o_n\}$ as the original event stream and $S_{Sampled} = \{e^s_1, \dots e^s_m\}$ as the sampled stream so that $n < m$. By using an event detection pattern it is possible to detect a set of complex events CE_O from the original stream, and from the sampled stream a set of complex events $CE_{Sampled}$. We define correctly detected complex events from the sampled stream as follows:

Definition 10.3. ***(Exactly Sampled Complex Events)*** *If and only if the $CE_{Sampled} \subseteq CE_O$ we call the detected complex events from the sampled stream Exactly Sampled Complex Events. (Complex events are identical to each other if they have the same event instances and the same timestamps.)*

Consider that we have an event stream generated from the event Types (A, B, C, D, E, F) and at an specific processing moment the event processing agent has in its data window the event instances shown in Figure 10.3. Based on the definition of each event algebra operator different occurrences of complex events can be observed in the event window given in Figure 10.3.

The sequence operator (SEQ) can be handled differently depending on the operational semantics of the operation. For example, in the Esper event processing language[2], the sequence pattern can be used together with the key operator *Every*. The *Every* operator determines when the sequence matcher should be restarted to match the next sequence (specifies the event consumption policy).

As shown in Table 10.1, different usage of the *Every* operator can detect in the event stream different sets of complex events. If the pattern matcher restarts and looks for the next A event in the stream, we will have 3 matches $(a_1, b_1), (a_2, b_3), (a_4, b_4)$. The patterns *(every (A -> B)),(every A -> B) , (A -> every B)* detect different sets of complex events in the original stream compared to the sampled stream. Some of the detected complex events in the sampled stream are the same as those on the original stream but not all of them. Only the pattern *(every A -> every B)* detects in the sampled stream a subset of complex events that are detected in the original stream.

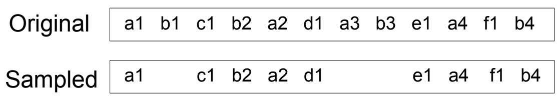

Original
a1 b1 c1 b2 a2 d1 a3 b3 e1 a4 f1 b4
Sampled
a1 c1 b2 a2 d1 e1 a4 f1 b4

Figure 10.3: Example of an Event Window, the Original and the Sampled Stream

[2] http://esper.codehaus.org/esper-4.6.0/doc/reference/en-US/html/event_patterns.html

The usage of sampling approaches for event detection should be controlled for each event detection queries to check whether a detection in the sampled stream can produce wrong event detection or it can produce *exactly sampled complex events*. We do not provide a solution to this problem in this research, rather we monitor event detection queries for the correct detection in the sampled stream. We consider user queries that can detect correct complex events from both sampled and original stream.

10.5.1 The Effect of Different Event Operators

In this section we describe the effects of different event algebra operators like AND, OR, SEQ and NOT on the detection of complex events from the sampled stream. Based on their exact operational semantics event detection operators might detect the same complex events in the sampled stream or they might detect different events.

In additional to the detection operators, the type of data windowing on the event stream and its evaluation might produce different results. In the following we discuss the effect of each operator (the basic four operations SEQ, AND, OR, NOT), and subsequently get into the details of data windowing and window evaluation.

SEQ Operator: As shown in Figure 10.3 and Table 10.1, the sequence operator may produce different complex events on the sampled stream, due to missing event components for the detection of complex events. The sequence operator depends heavily on each of the event instances and their sequence. The operation is highly sensitive to missed event instances. However, if the sequence operator has to match every possible sequence of events, then this is an *exactly sampled complex event*.

AND Operator: The AND Operator is similar to the sequence operator without the ordering of events. The AND operator is less sensitive to sampling in the event stream when the operation should be evaluated in a data window in time intervals. In a window there might be several instances of the same event type and missing of some of the event instances have no effect on the detected complex event. In some cases the AND operation can produce *exactly sampled complex events*, e.g., if it matches every existing event instances (“Every” operator similar to the “Every” operational semantic in Esper language).

OR Operator: The OR operator is the least sensitive to the sampling of events. The complex events produced are in most cases a subset of the complex events produce in the original stream and it produces in most cases *exactly sampled complex events*.

NOT Operator: The NOT operator matches event instances not existing in the event stream. In the sampling process some of the events are dropped and excluded from the sampled stream. In this way the NOT operator can be matched on sampled stream but not in the original stream. It means matching in the sampled stream with NOT operator causes false positives in the detection of complex events.

However, we have no false negatives with the NOT operator, i.e., if the NOT operator does not match in the sampled stream it is definitely not matched in the original stream, because the sampled stream is a subset of the original stream. In the example shown in Figure 10.2, if the pattern (NOT A) is not matched in the data window in the sampled stream, it also does not match in the original stream because no instance of event type A (a) exists in the sampled event window.

10.5.2 Data Window Evaluation

The evaluation of events existing in the selected data window has an important role for event detection of the sampled event stream. Based on the windowing type (tumbling or sliding windows) and the time point of matching (rhythm of matching) different results can be observed in the sampling stream. Some of the event processing systems process events in the window on each new arriving event instance. Some other systems evaluate the data in time intervals. In some of the event processing language like Esper EPL, the windowing and processing time point can be specified in the event detection pattern.

Let us consider the event window in Figure 10.3 which shows a window sequence of events and the sampled stream of the same window. For example, whether the processing of the sequence operator (SEQ), defined as a single notification for all existing sequences in the window, using the pattern (A SEQ B) on both original and sampled stream generates the same notifications? And whether the notification is sent at the processing time and not at the arrival time of the events are both identical? Such questions can only be answered based on the operational semantics of query language and the event consumption policies. These issues should be considered before applying sampling on raw event stream.

This will be the same case for the AND, OR, and NOT operators, if only one notification is generated for the whole window then they might produce the same notification.

10.6 Evaluation

We conducted several experiments to evaluate our algorithm for sampling the event stream. We evaluated our algorithm for sampling probability calculation for both optimization problems. The event throughput was reduced down to a specific rate while maximizing the number of complex events, and the rate of complex events were maximized to a specific rate while minimizing the throughput rate of sampled event.

10.6.1 Methodology and Experiment Setups:

We have implemented a prototype of our sampling approach. For the event detection steps, we used the Esper[3] event detection engine. To cleanly separate the impact of our approach from those of the underlying implementation and configuration choices, we compared the evaluation metrics against the same implementation setup. We ran both the event sampling agent and event processing agent on the sample host.

As raw event stream we use in our experiments randomly generated synthetic event streams so that we can control the event stream entropy and event stream homogeneity. We generated different event streams by using different event types and different probabilities so that we controlled the entropy of the stream.

In all the experiments for the probability calculation trails, we sent 10,000 events through the event sampling and event processing system and measured the throughput of the sampled stream and calculated the number of detected complex events.

As our evaluation metrics do not depend on the run-time environment, the hardware setup and configurations used in the experimentation do not impact our evaluation results, for this reason we compare the different approaches against each other. All of the event detection queries used are listed in Appendix C.

[3] `http://esper.codehaus.org`, version 4.6.0

10.6.2 Experiments with Sampled Event Stream

We have conducted experiments with different event patterns and sampling methods. In the preliminary experiment, we checked to see if our approach for type-based sampling in comparison to uniform sampling had an advantage in the detection of complex events. In the latter experiments, we shown how our type-based sampling algorithm for the calculation of sampling probability worked on different event patterns. Furthermore, we experimented with the number of event types used in a pattern and its effects on sampling rates. Finally, the last experiment was about event patterns including bounding event types, the effect of event types at the beginning and end of event detection patterns. In the following we get into the details of each experiment:

Experiment 1: Unify Filtering vs. Type-Based Filtering

In this experiment we compared the effect of unify sampling with type-based sampling based on the number of detected complex events.

We generated an event stream from two type of events, type A and B. The entropy of stream is 20% A and 80% B. We detected complex events on this stream using a simple sequence pattern (A SEQ B) (Esper Pattern every (A -> B), query C.1 in Appendix C). We conducted the event detection one time with uniform sampling and one time with type-based sampling. We used the sampling rate on 99% of event type A, and 17.5% on event type B. In this experiment, we sent 10,000 events through the system with the given entropy and aimed to sample the stream only for the detection of 50% of existing complex events.

Our experiment result in Figure 10.4 shows that the type-based event sampling can be used to detect the same 50% of complex events from much fewer raw events than uniform sampling. The experiment shows type-based filtering can generated a better sampled event stream for optimized event detection.

Experiment 2: Effect of Event Operators

In the following experiments we showed the effects of different event operators (AND, OR, SEQ and NOT). We generated an event stream from two type of events, type A and B. The entropy of the stream was 20% A and 80% B. We detected complex events on this stream using a simple sequence patter (A SEQ B) (Esper Pattern every (A -> B), query C.1 in Appendix C). We sent 10,000 events through the system generated with the entropy given and aimed to sample the stream only for the detection of 50% of existing complex events.

Figure 10.6a shows how our algorithm calculates the sampling rate for each event types. The sampling agent starts from a sampling rate of 100% for each of the types, and do several trails to reduce the sampling rate to achieve the required rate of the complex events. It calibrates the sampling rates in several interval trails, as shown on the x-axis. The algorithm starts from the event type which has the highest impact on the detection of complex events based on stream entropy and detection pattern.

Figure 10.6b shows the result of our algorithm using the AND operator. We sampled the stream for the event detection using query C.2 in Appendix C. The effects of the AND operator is similar to the sequence operator with minimal differences. The sampling system starts calibrating the sampling rate of one of the event types and continues calibrating other events until it achieves the requested rate for the complex events. As shown, the system can detect 50% of the complex events with about 30% of the raw event stream.

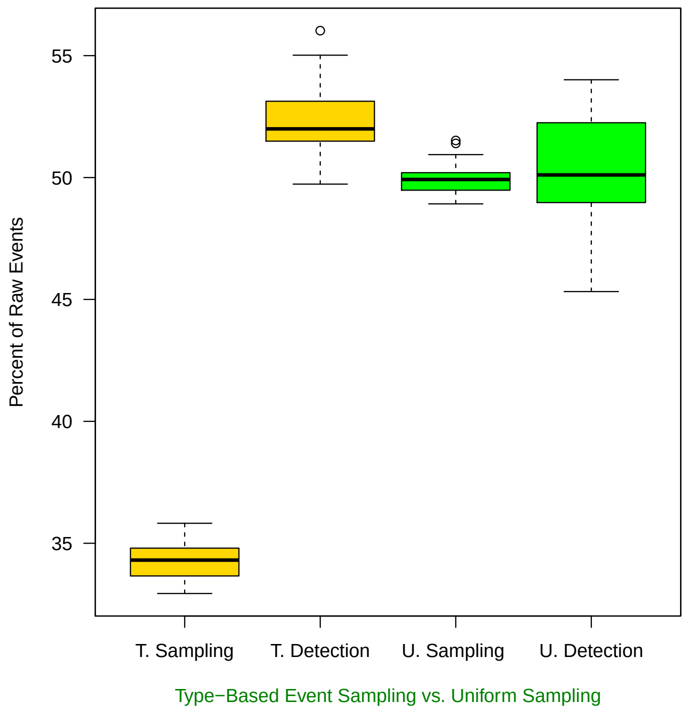
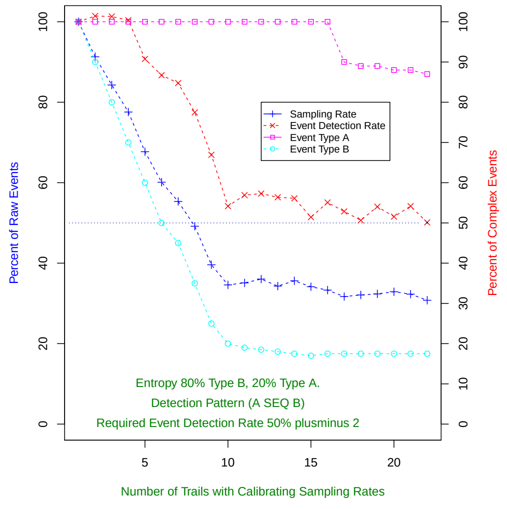


Figure 10.4: Uniform vs. Type-Based Sampling for Sequence Pattern Detection

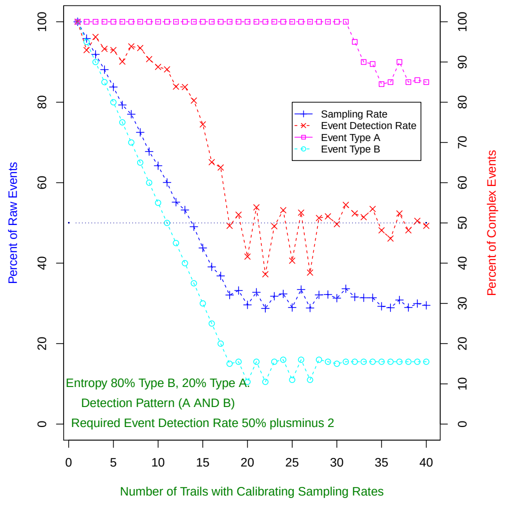
The usage of the OR operator is shown in Figure 10.6c using query C.3 in Appendix C. As we have expected, the rate of detected complex events is exactly the same as the rate of raw events, because of the usage of the OR operator. The sampling system starts with an event type which has higher entropy weightings, and fixes the requested rate of complex events with the other event type. Because of the OR operator it does not matter which one of the event types is reduced in the sampled stream.

Figure 10.6d shows the usage of the NOT operator with the AND operator using query C.4 in Appendix C. We combined the NOT operator with AND operator because the NOT operator alone needs a data window definition. As shown on the diagram, the system needs about 60% of the raw event stream until it can detect 50% of existing complex events.

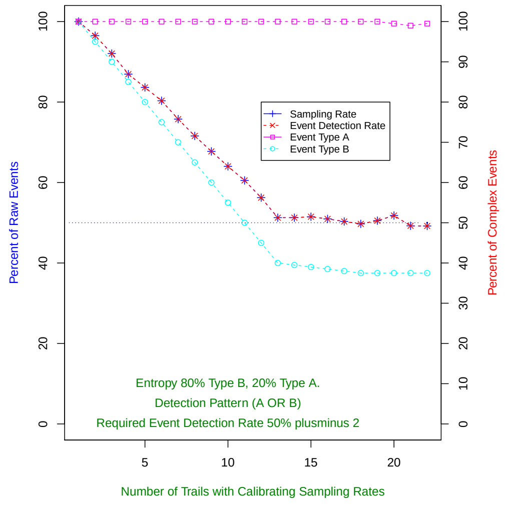
We conducted the inverse of the experiments for the sampling of raw events at 50% and maximize the number of detected complex events. Our algorithm calibrates the sampling rates for different event types and maximizes the number of complex events while needing only half of the raw events stream. These experiments for 50% sampling and maximizing the detected complex events are shown in figures 10.6a, 10.6b, 10.6c, 10.6d for the same event patterns (SEQ, AND, OR, NOT).



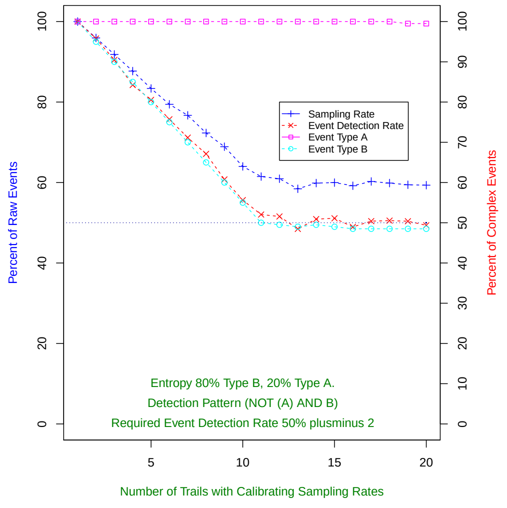
(a) Sequence Operator (A SEQ B)



(b) AND Operator (A AND B)



(c) OR Operator (A OR B)

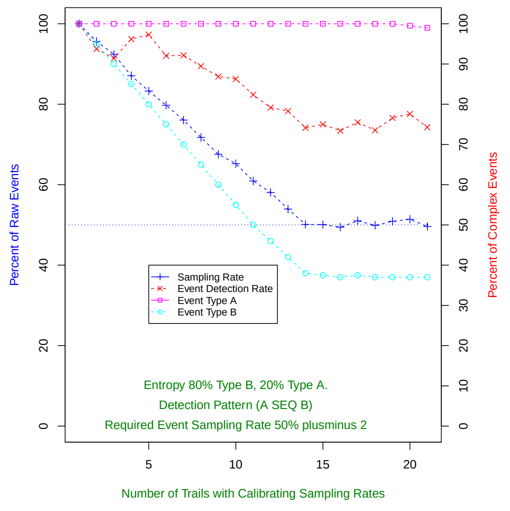


(d) NOT Operator (Not A AND B)

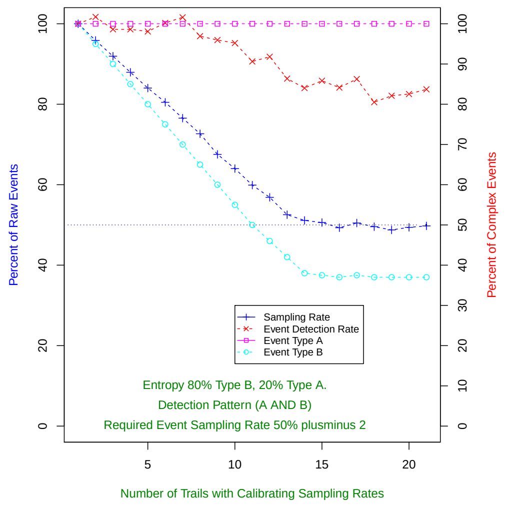
Figure 10.5: Event Stream Sampling for Different Event Detection Patterns (50% of Complex Events)

Experiment 3: Repetition of an Event Type in the Pattern

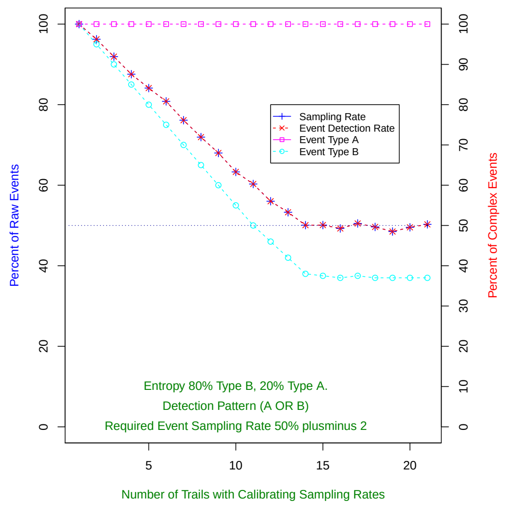
The effect of repetition of an event type in the event pattern have been experimented on. Repetition is an impact factor for the calculation of the event sampling rate. Figure 10.7 shows our experiments that ran event sampling and detection using the SEQ operator with different repetition numbers of event types (9 patterns), pattern (A SEQ B) up to (A SEQ A SEQ A ... B) 9 times A in the pattern. We used the same stream entropy for all runs and calculated the sampling rates which produced 50% of complex events. Our experiment shows that with the increase of repetition of an event type in a pattern, the event sampling rate required has to be increased to detect the same number of complex events. If we need about 25% of the A events for the pattern (A SEQ B), for the pattern (A . . . A SEQ B) (9 times A) we need about 45% of the A events.



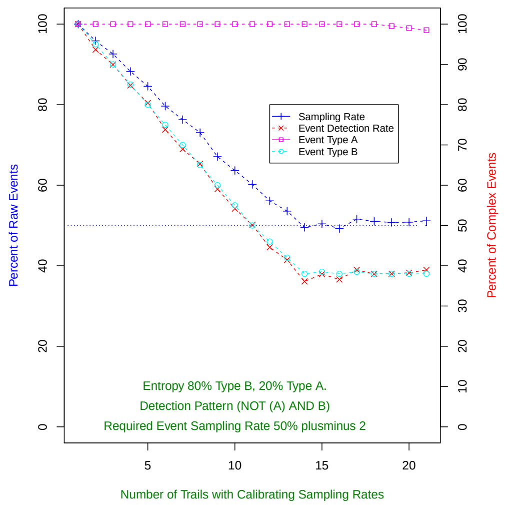
(a) Sequence Operator (A SEQ B)



(b) AND Operator (A AND B)



(c) OR Operator (A OR B)



(d) NOT Operator (Not A AND B)

Figure 10.6: Event Stream Sampling for Different Event Detection Patterns (50% Sampling Of Stream)

Experiment 4: Bounding Patterns on Sampled Stream

Sometimes in an event processing pattern an event type is bounded (surrounded) by two other event types that play important roles on the detection of the complex event. In this experiment we check to see if bounding event types have an effect on the sampling rate of other event types. Figures 10.8 and 10.9 show two experiments with bounding event types. These experiments illustrate our algorithm uses one of the event types to fix the sampling rate, and subsequently fine grain the sampling rate with other event types.

Bounding event types have no effect on the sampling rate calculation on our algorithm because it starts by fixing one of the event types and calibrates the sampling rate requested, and finally continues with other types.

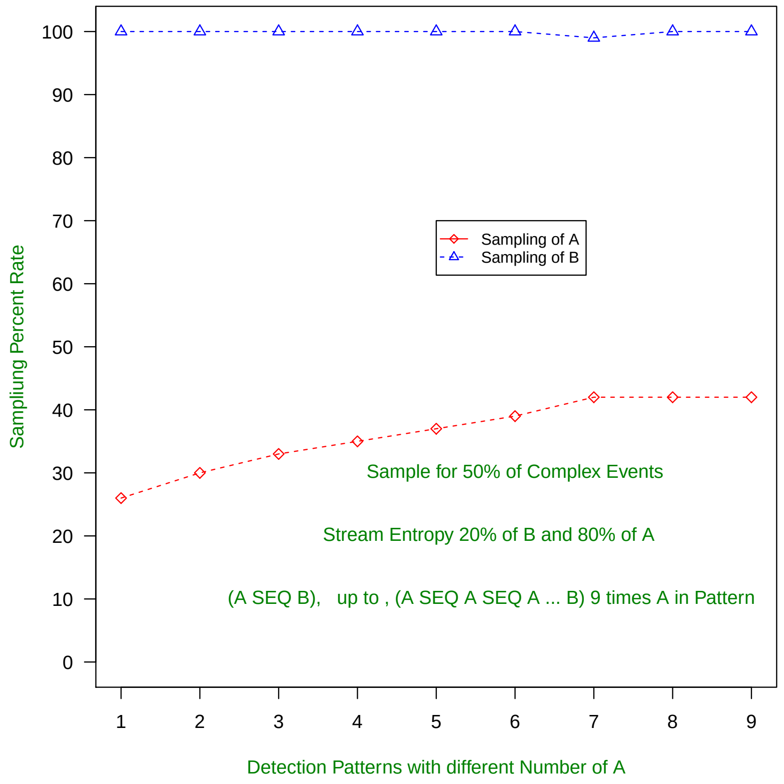
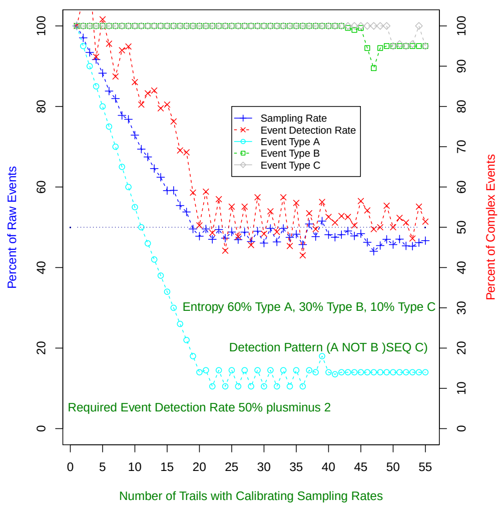


Figure 10.7: Sampling for Sequence Patterns (A SEQ B) up to (A SEQ A SEQ A ... B) 9 times A in Event Dection Pattern



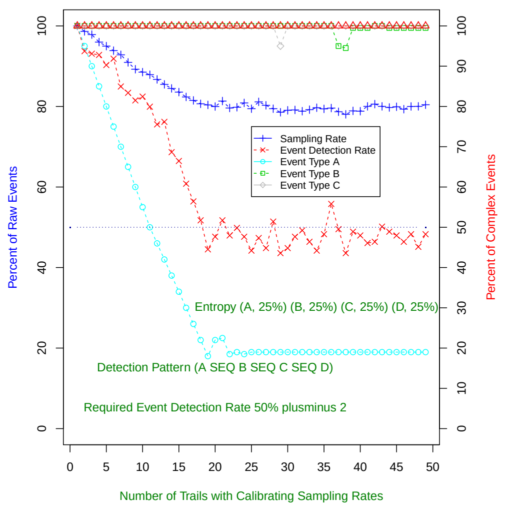


Figure 10.8: Sampling for Patterns, (A AND NOT B SEQ C)

Figure 10.9: Sampling for Patterns, (A SEQ B SEQ C SEQ D)

10.7 Future Work on Event Stream Sampling

In this section we describe some of the ideas developed for sampling event streams which might produce better results. We describe only the basic initial ideas. The implementation and evaluation of these ideas are out of the scope of this thesis.

- **Sampling using a Reservoir:** Using a reservoir of events to sample event stream might be an effective approach. The selection of the reservoir of events might have a big impact on the results. The reservoir might have the same size as the detection window specified by the user query.

  If events are marked with two timestamps (e.g., start and end timestamps), we might have several event instances within the same time interval and with the same event type. Consider an event instance $e_1(t_1, t_2)$ from event type E_1, if there is another event instance $e_2(t_3, t_4)$ from the same event type E_1, and $(t_3 \geq t_1) \wedge (t_4 \leq t_2)$, event e_2 can get a higher sampling probability than other events of this type. Such kinds of event duplicates might be sampled so that the throughput of the raw event stream is reduced.

  The disadvantage of this approach is the memory and computation required for keeping events in the reservoir. The costs can be higher than the computation required for uniform sampling. Intensive computation for sampling affects the throughput of the sampling approach. An analysis of the reservoir sampling approach should ascertain the quality of event detection on the sampled stream generated by reservoir sampling.

- **Sampling without Event Type Identification:** Events can be sampled without type identification. Type identification is a time and resource consuming task which affects overall performance. The event stream might be sampled without checking the types and only based on the arriving sequence of events. Which events are included in the sampled stream should be decided based on the sequence of events and the time of their arrivals. This is like a random walk in the event stream and selection of events. The events sampled have different sampling probability which depend on their sequence and not on their types.

  One extension might be to check only the identity of some of the instances, e.g., when the sampling processor reads the event instance it will check the event type, otherwise it jumps to the next number of events without checking and reading some of the events.

- **Filtering Rates Based on Background Knowledge:** In the knowledge-based event processing approach event types are defined based on background knowledge. The exact event type of each event instance is specified based on relations in the background knowledge base.

  In the integration of sampling event stream and knowledge-based event processing, it might be possible to define sampling probabilities based on relations in the knowledge base, e.g., based on the predicates used in each of the triple patterns (BGPs). Using the sampling system we can estimate how likely an event instance is matched to the arriving event stream.

10.8 Summary and Discussion

In this chapter we presented our approach for sampling event stream to optimize event detection in the sampled stream. We have shown that if the event types and event entropy of an event stream is known, it is possible to optimize the sampling of the event stream for complex event processing. This is, for example, the case when the event processing system has specific processing capacity or only a part of the existing complex events required by the use case.

We have shown that the calculation of sampling probability rates based on event types improves event detection in comparison to uniform sampling. Type-based sampling improves the number of events detected for complex events and can better reduce raw event throughput than uniform sampling. Event pattern and stream entropy should be used to calculate sampling rates.

One of our basic assumptions was event processing system had limited processing throughput capacity so that we would have to reduce the raw event stream to a specific degree. Because of this, there was no need to compare our sampling and event processing system with the scenario of having no sampling step as the first processing step.

In knowledge-based event processing the capacity of the semantic event processing system is limited and processing has higher computation and data transportation costs. The sampling of event processing is an applicable approach for some of the use cases, since not all of the existing complex events are required by the use case.

Part IV

Conclusion and Future Work

Chapter 11
Conclusion and Outlook

This chapter concludes our research on knowledge-based event processing. First, we summarize the contributions of the thesis and describe the most important lessons learned from this research. At the end, we describe some of the future prospects of our research.

Contents

11.1 Summary

In this research, we have investigated the usage of domain background knowledge for complex event processing. We have identified some of the event processing use cases that can profit from the usage of external knowledge bases. The use cases presented in this research describe example applications using external background knowledge in event processing. We have described different requirements for each of the use cases like real-time event processing, scalability, reasoning level on background knowledge and expressiveness of event detection rules and possible approximated complex event detection.

We studied different existing event representation methods and went into details about event ontologies. Based on this study, we proposed using top-level ontologies to represent events and other relevant event processing concepts so as to reuse and extend them in different event processing use cases.

We described which kinds of knowledge representation is required to enable knowledge-based event detection patterns that can detect events based on resources relations in background knowledge. Based on the study of state-of-the-art event processing approaches, we proposed different approaches for the integration of external KBs with event streams.

Considering use case requirements, we provide three different event processing approaches, event stream enrichment, event query pre-processing and sampling of the event stream. We evaluated each of the proposed approaches based on their event processing performance and processing costs.

We conclude that it is not possible to identify a single event processing approach applicable to all the event processing use cases. Based on use case expectations and requirements, one of the approaches or a combination of them can be applied for target use case.

In the following sections, we describe the different properties of the proposed approaches and their usage in the event processing of use cases.

11.2 Lessons Learned

The main lessons learned from this research can be summarized as follows:

- **Knowledge Representation for Semantic CEP:** In our study of existing ontologies for events, situations, actions, actors and other related concepts, it is not possible to identify a single ontology that can satisfy all of the requirements of event processing applications. We have learned it is possible to consider these ontologies as top-level ontologies that can be extended depending on domain and application requirements. (Chapter 6)
- **Representation of Complex Event Patterns:** For the detection of events based on relations in background knowledge, we have identified a combinatorial event detection query that include query parts for describing graph patterns and which can be combined with event algebra operators. (Chapter 7)
- **Multi-Step Processing:** It is possible to drastically reduce the cost of semantic event processing if we can process the events in multi-pass processing rather than single-pass processing. If user latency expectations allows multi-pass processing, event processing in multi-passing can reduce processing costs. (Chapter 8)
- **Plan-Based Event Processing:** Planning event enrichment for event detection can help to reduce costs and improve overall performance of the CEP system. (Chapter 8)
- **Event Query Enrichment:** Compared to the event stream enrichment approach, the pre-processing of event detection queries and their rewriting using background knowledge can highly improve the event processing performance. However, in some cases it is a challenging task to include the latest updates on the external knowledge bases so that the system can provide guarantees for the accurate detection of complex events. (Chapter 9)
- **Event Processing on Sampled Stream:** The sampling of the event stream can highly improve the performance of the event processing system, due to the reduction of the raw event stream throughput. We have learned that, in comparison to the uniform sampling, the calculation of sampling rates based on event types improves event detection. Type-based sampling can perform better for event stream sampling for the purpose of event detection. (Chapter 10)

11.3 Usage of Proposed Event Processing Approaches

The proposed event processing approaches can only be used in applications in which use cases requirements are provided. Each of these approaches has different properties that can be suitable for one or more of the CEP use case requirements. We have identified the important properties of the approaches proposed, like performance, scalability, elasticity, reasoning, precision, recall and the ability to integrate knowledge updates on external KBs.

These properties are identified as follows:

- **Performance/Latency:** It is the ability to process high throughput event streams in use case specific time (specified use case real-time).
- **Scalability:** The ability of the event processing system to scale out and up. In some of the use cases there are event processing systems that can scale up to process high throughput of events and integrate huge amounts of knowledge facts.
- **Elasticity/Adaptability:** In some cases the throughput of event stream or the knowledge base may vary dynamically, and can increase or decrease rapidly. The CEP system should be an elastic system to process the event processing depending on the stream load.
- **Reasoning Complexity:** Reasoning is the ability of the system to do reasoning on the external knowledge base. Depending on the ontology applied reasoning can have different levels of complexity. The proposed event processing approaches can use external reasoners to do description logic reasoning on external knowledge bases. This factor describes the different reasoning abilities of the approaches proposed for knowledge-based event processing.
- **Precision of the Event Detection:** The precision of event detection is a factor that describes the correct detection of complex events. This factor is about truth maintenance in event processing systems. It shows if the detection of complex events is done correctly based on correct facts. As we have discussed in previous chapters, some of the proposed event processing approaches might detect events which have not been correctly detected, i.e., they do not happen in reality but are detected in the system. We discussed the correctness of event detections in event processing in sampled stream (Section 10.5).
- **Recall of the Event Detection:** The ability of event processing to detect all existing complex events from the stream of events. In some of the proposed event processing approaches it is not possible to detect all existing complex events from the event stream. The completeness of the detected complex events is not guaranteed by the event processing system. Such cases are discussed in Section 10.5 and 9.3.1.
- **Updates of Knowledge:** The knowledge stored in the external knowledge base is not static reference knowledge and can have updates. These updates can be handled by some of the proposed approaches so that complex events can be correctly detected. Other approaches do not provide the guarantee the latest version of external knowledge for event detection is included.

Table 11.1 provides a comparison of the properties of the event processing methods proposed (naive polling of KB, semantic enrichment of event streams (SEES), query rewriting, two-phase knowledge base communication (2KBQ) and sampling of event stream). The requirements of some of the event processing use cases are listed in Table 4.4. These requirements have to be matched to the properties of the approaches proposed for the fusion of background knowledge with the event stream.

	KB Polling	**SEES**	**Rewriting**	**2KBQ**	**Sampling**
Performance	low	high	high	high	high
Scalability	limited	limited	high	high	high
Elasticity	limited	limited	high	high	high
Reasoning	high	limited	limited	high	high
Precision/Recall of Event Detection	high	limited	limited	limited	limited
KB Changes	high	high	limited	high	limited

Table 11.1: Comparison of Different Event Processing Approaches

11.4 Differences to RDF Stream Reasoning Approaches

The differences between our approach for knowledge-based complex event processing and the state-of-the-art RDF streaming approach (described in 3.9.1) can be described in the following points:

1. Although some of the RDF stream reasoning systems can use 'static' reference knowledge along with RDF streams, the amount of static reference knowledge is limited to the main memory of the reasoner because they have to incorporate knowledge into the memory of the reasoner. In our approach, reasoning is delegated to a highly optimized external reasoner. We assume the external KB is a highly scalable triplestore with a scalable reasoner (a distributed triple store and reasoner). The event mapping engine can query the external knowledge base and activate the reasoner for external knowledge. The result of the reasoning is then enriched to the event stream and forwarded for event pattern matching in the subsequent event detection phase.

2. In our approach the event stream is not mapped to an RDF stream. The event stream is based on an event model, e.g., a name/value pair stream, which is usual in most event processing use cases. The stream is enriched with additional values, e.g., for a stock market event the data about a company's products, supplies for production processes and number of employees can be enriched to the stream. The external knowledge base is not only a simple triple store, but also includes a reasoner (e.g., a DL-Reasoner) that can reason on the stored triples.

11.5 Research Perspectives

As event processing is one of the fastest growing segments in the last years [Etz10a, ept08], research on complex event processing have fanned out in different directions.

In this section, we describe some of the potential research extensions of our research. These research topics are in the intersection of semantic technologies, event-driven business process management, database technology and event stream processing.

11.5.1 Background Knowledge Learning from Event Stream

One of the extensions might be to learn from the event stream and add the knowledge learned to background knowledge which then can be used for the detection of upcoming events. Also, the history of the detected complex events can be used to learn new additional knowledge about the application domain. The knowledge learned can be added as knowledge updates to the existing KB or can be treated in a separate knowledge base for special reasoning purposes. In this way, the detection of complex events is realized on the basis of background knowledge and the additional knowledge gleaned from the event stream history.

Figure 11.1 shows our vision for a knowledge-based CEP system with a learning curve. Such systems might be capable of enlarging the knowledge base with knowledge learned from the history of raw events and detected complex events. As shown in Figure 11.1, the knowledge learned can be added to the KB as updates.

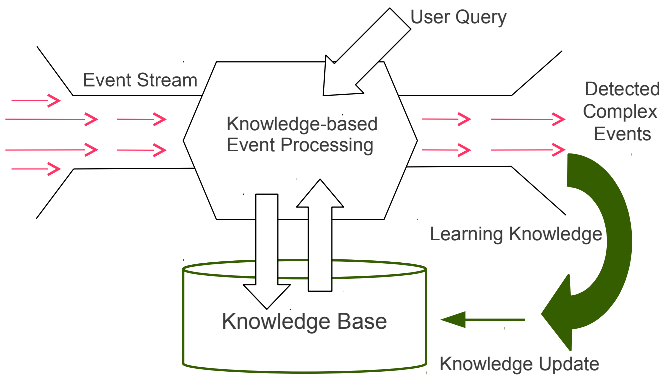


Figure 11.1: Knowledge-based Event Processing with Background Knowledge Learning Loop from Event Stream

Some of the possible research questions that should be investigated are:

- What kind of knowledge learned from the event stream history can be added to the background knowledge?
- What kind of complex events can be detected based on additional knowledge learned from the history of the events?
- To what extent are detected complex events valid when the knowledge base updates over time? (Can we also include T-Box changes?)
- The detection of complex events is only valid within a period of time that defines a specific event history and KB history. How should the validations of event history be specified for usage in CEP?

11.5.2 Mining Complex Event Patterns

In some of the use cases the patterns of complex events are so complicated that it is difficult for people to define them. On the other side, people in different industries are interested to know what other patterns are useful for monitoring their business processes. Mining complex event patterns is about extracting patterns for complex event detection, which the user is unaware of but might be interested in.

Most existing approaches for pattern mining use state-of-the-art approaches from machine learning to extract event detection rules. Margara et al. [Mar13, Mar14a] propose an approach for event pattern mining called iCEP for the meaningful pattern extraction from historical traces/logs of events. In their approach, the Information Gain Ratio [Har02] is used to measure the influences of events. They proposed an event model with event types and attributes and, among others proposed methods for the extraction of event types, attributes and windows. Existing machine learning algorithms are used for pattern extraction based on correlation detection between primitive and composite events.

Further approaches can be developed for the mining of event detection patterns based on the learning approaches like association rule learning [Agr93], Generalized Sequential Pattern Mining Algorithm [Han07], Sequential Pattern Discovery [Zak01, Pei01] and Closed Pattern Mining (CloSpan and ClaSP) [Yan03, Gom13].

One of the potential extensions of mining complex event patterns is the usage of domain background knowledge with existing algorithms for pattern mining. It is possible to enrich raw events with background knowledge prior to event mining, so that patterns can be detected not only based on event correlations but also on the relations of events in the background knowledge. The result of pattern mining is a detection pattern specifically based on relations in background knowledge that is similar to knowledge-based event processing queries.

11.5.3 Noisy and Uncertain Events

In real world applications of event processing event producers generate event streams which are frequently noisy. In use cases like sensor network the event stream is noisy and cannot be assumed as a valid and stable stream. A large body of research work has been done on the processing of noisy event streams [Li13, Sto13, Li13, Agr08] and complex event processing under uncertainty [Was08, Was12].

Background knowledge can help to learn more about noise sources and uncertain event instances. Usage of background knowledge about noise resources from event producers and event transmission channels can improve the quality of event processing from noisy event streams.

Some of the abstract research questions about the usage of background knowledge for processing of noisy event stream can be:

- How can background knowledge about the noise sources and noisy event transmission channels be represented?
- Background knowledge about noise resources are dynamic knowledge due to changes in the event producers, environment and event transmission channels. How can one adapt knowledge updates for event processing?
- To what extent can the usage of background knowledge improve the quality of noisy event stream processing?

- How can one treat the truth maintenance of event detections while taking into consideration background knowledge updates; how do they affect the precision of event detection?

11.5.4 Knowledge-Based Process Mining

Process Mining [1] [vdA11] techniques allow the automatic extraction of information from event logs. Process mining enables three kinds of applications: process analysis, process design and process enactment.

The usage of background knowledge in process mining [AdM09] can enable a more knowledge-able mining technique. One potential extension of the process mining technique can be enrichment with background knowledge of event logs prior to the mining process by the addition of payload to event instances. This kind of event enrichment might improve the quality of event mining.

Furthermore, the usage of complex events, detection based on background knowledge, can improve the quality of process mining because complex events are abstract high-level events specifically based on relations of resources in the knowledge base.

11.5.5 Knowledge-Based Event Stream Sampling

Beyond the future work on event stream sampling that we described in the Section 10.7, further extensions are possible, like using background knowledge to sample the event stream based on the relations of events in background knowledge. Research questions might include finding out which relation types are significant for sampling of event streams. How can relations be identified based on the network of background knowledge graphs or on the semantics of the relations? Such extensions can help to sample the event stream based on their importance for the target use case rather than sampling based on random variables defined by the sequence of events or their types.

11.6 Final Remarks

As a final remark, we will return to the research questions laid out for this dissertation and describe briefly our contributions to the gaps and limitations mentioned:

1. **Lack of Knowledge Representation Methods:** We address this gap by providing models for the representation of related concepts and for the description of the event detection queries. We proposed upper-level ontology modules to be used for the ontological representation of related concepts like events and actions (Chapter 6).

   We have shown how event detection queries based on conceptual relations in the background knowledge can be expressed. We delved into the details of the combination of event detection operators and knowledge graph patterns (Chapter 7). However, a detailed design of such event detection query was not our aim, which can be better done in co-operation with the research community and domain experts who can specify the different requirements of the use cases.

[1] http://www.processmining.org/

2. **Limitation of Processing Methods for Fusion of Events and Background Knowledge:** To address this research question we provided three different event processing approaches: event stream enrichment (Chapter 8), event query pre-processing (Chapter 9) and sampling of the event stream (Chapter 10). We have shown that using these approaches, the fusion of event streams and background knowledge, is possible and the trade-off between expressive reasoning and real-time event processing based on use case requirements and special features can be optimized.

For the evaluation of the approaches proposed we have decided to provide an experimental evaluation, because the formal worst-case complexity analysis for our event processing approaches depends on external factors, like the complexity of event detection queries, complexity of knowledge bases and reasoners, stream entropy and data homogeneity stored in the KB. In our particular case formal analysis is possible with some limitation so that it is only applicable for simple queries with strict assumptions. In our evaluation we experimented with different properties of our approach using real world data and synthetically generated data stream to stress test the system while conducting different queries with different complexities (listed in the Appendix).

Part V

Appendix

Code Listings

Appendix A
Acronyms and Glossary

ACID	ACID Properties (Atomicity, Consistency, Isolation, Durability)
BASE	BASE Properties (Basic Availability, Soft-state, Eventual consistency)
BGP	RDF Basic Triple Pattern
CEP	Complex Event Processing
CQL	Continuous Query Language
CSV	Comma Separated Values
DB	DataBase
DBMS	DataBase Management System
DL	Description Logic
DSMS	Data Stream Management System
ECA	Event-Condition-Action
EMA	Event Mapping Agent
EPA	Event Processing Agent
EPL	Event Processing Language
EPN	Event Processing Network
EPTS	Event Processing Technical Society (http://www.ep-ts.com)
EQPR	Event Query Pre-processing and Rewriting
ESP	Event Stream Processing
KB	Knowledge Base
NFA	Non-Deterministic Finite Automaton
OWL	Ontology Web Language
RDF	Resource Description Framework
RDFS	Resource Description Framework Schema
RSS	Really Simple Syndication
SCEP	Semantic-Enabled Complex Event Processing
SEES	Semantic Enrichment of Event Stream
SPARQL	SPARQL Protocol And RDF Query Language
SQL	Simple Query Language
URL	Uniform Resource Locator
URI	Uniform Resource Identifier
W3C	World Wide Web Consortium
XML	eXtensible Markup Language

Appendix B

List of Queries for Semantic Event Enrichment and Detection

In this appendix, we list the event detection queries used in our experiments.

```
PREFIX dcterms: <http://purl.org/dc/terms/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX dbpprop: <http://dbpedia.org/property/>
PREFIX dbpedia: <http://dbpedia.org/resource/>
PREFIX dbpedia-owl: <http://dbpedia.org/ontology/>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX yago: <http://dbpedia.org/class/yago/>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX skos: <http://www.w3.org/2004/02/skos/core#>
PREFIX category:  <http://dbpedia.org/resource/Category>
```

Listing B.1: Prefixes Used in Our SPARQL Queries

B.1 SPARQL Queries for Event Enrichment

In the following queries, the "$$" is replaced with event instances and results are enriched to the event stream. We used the following queries to extract background knowledge from external knowledge bases.

```
 SELECT * WHERE { <$$> dbpedia-owl:location ?location }
```

Listing B.2: SPARQL Query sq_m1

```
SELECT * WHERE { <$$> dbpedia-owl:industry ?industry }
```

Listing B.3: SPARQL Query sq_m2

```
SELECT * WHERE { <$$> dbpedia-owl:numberOfEmployees ?employees  }
```

Listing B.4: SPARQL Query sq_m3

```
SELECT * WHERE { <$$> dbpprop:products ?products  }
```

Listing B.5: SPARQL Query sq_m4

```
SELECT * WHERE { <$$>  dbpedia-owl:subsidiary ?subsidiary }
```

Listing B.6: SPARQL Query sq_m5

```
SELECT * WHERE { <$$> rdf:type ?rdftype }
```

Listing B.7: SPARQL Query sq_m6

```
SELECT * WHERE { <$$> dcterms:subject ?dcterms }
```

Listing B.8: SPARQL Query sq_m7

```
 SELECT * WHERE { <$$> dbpedia-owl:location ?location
   {  <$$> dbpedia-owl:industry ?industry  . } }
```

Listing B.9: SPARQL Query sq_s2

```
SELECT * WHERE { <$$> dbpedia-owl:location ?location
   {  <$$> dbpedia-owl:industry ?industry  . }
   {  <$$> dbpedia-owl:numberOfEmployees ?employees   . } }
```

Listing B.10: SPARQL Query sq_s3

```
SELECT * WHERE { <$$> dbpedia-owl:location ?location
   {  <$$> dbpedia-owl:industry ?industry  . }
   {  <$$> dbpedia-owl:numberOfEmployees ?employees   . }
   {  <$$> dbpprop:products ?products   . } }
```

Listing B.11: SPARQL Query sq_s4

```
SELECT * WHERE { <$$> dbpedia-owl:location ?location
   {  <$$> dbpedia-owl:industry ?industry  . }
   {  <$$> dbpedia-owl:numberOfEmployees ?employees   . }
   {  <$$> dbpprop:products ?products   . }
   {  <$$>  dbpedia-owl:subsidiary ?subsidiary  . } }
```

Listing B.12: SPARQL Query sq_s5

```
SELECT * WHERE { <$$> dbpedia-owl:location ?location
   {  <$$> dbpedia-owl:industry ?industry  . }
   {  <$$> dbpedia-owl:numberOfEmployees ?employees   . }
   {  <$$> dbpprop:products ?products   . }
   {  <$$>  dbpedia-owl:subsidiary ?subsidiary  . }
   {  <$$> rdf:type ?rdftype  . } }
```

Listing B.13: SPARQL Query sq_s6

```
SELECT * WHERE { <$$> dbpedia-owl:location ?location
   {  <$$> dbpedia-owl:industry ?industry  . }
   {  <$$> dbpedia-owl:numberOfEmployees ?employees   . }
   {  <$$> dbpprop:products ?products   . }
   {  <$$>  dbpedia-owl:subsidiary ?subsidiary  . }
   {  <$$> rdf:type ?rdftype  . }
   {  <$$> dcterms:subject ?dcterms  . } }
```

Listing B.14: SPARQL Query sq_s7

```
SELECT * WHERE { {  <$$> dbpedia-owl:location ?location  . }
 UNION {  <$$> dbpedia-owl:industry ?industry  . } }
```

Listing B.15: SPARQL Query sq_orm2

```
SELECT * WHERE { {  <$$> dbpedia-owl:location ?location  . }
 UNION {  <$$> dbpedia-owl:industry ?industry  . }
 UNION {  <$$> dbpedia-owl:numberOfEmployees ?employees   . } }
```

Listing B.16: SPARQL Query sq_orm3

```
SELECT * WHERE { {  <$$> dbpedia-owl:location ?location  . }
 UNION {  <$$> dbpedia-owl:industry ?industry  . }
 UNION {  <$$> dbpedia-owl:numberOfEmployees ?employees   . }
 UNION {  <$$> dbpprop:products ?products   . } }
```

Listing B.17: SPARQL Query sq_orm4

```
SELECT * WHERE { {  <$$> dbpedia-owl:location ?location  . }
 UNION {  <$$> dbpedia-owl:industry ?industry  . }
 UNION {  <$$> dbpedia-owl:numberOfEmployees ?employees  . }
 UNION {  <$$> dbpprop:products ?products  . }
 UNION {  <$$> dbpedia-owl:subsidiary  ?subsidiary  . } }
```

Listing B.18: SPARQL Query sq_orm5

```
SELECT * WHERE { {  <$$> dbpedia-owl:location ?location  . }
 UNION {  <$$> dbpedia-owl:industry ?industry  . }
 UNION {  <$$> dbpedia-owl:numberOfEmployees ?employees  . }
 UNION {  <$$> dbpprop:products ?products  . }
 UNION {  <$$> dbpedia-owl:subsidiary  ?subsidiary  . }
 UNION {  <$$> rdf:type ?rdftype  . } }
```

Listing B.19: SPARQL Query sq_orm6

```
SELECT * WHERE { {  <$$> dbpedia-owl:location ?location  . }
 UNION {  <$$> dbpedia-owl:industry ?industry  . }
 UNION {  <$$> dbpedia-owl:numberOfEmployees ?employees  . }
 UNION {  <$$> dbpprop:products ?products  . }
 UNION {  <$$> dbpedia-owl:subsidiary  ?subsidiary  . }
 UNION {  <$$> rdf:type ?rdftype  . }
 UNION {  <$$> dcterms:subject ?dcterms  . } }
```

Listing B.20: SPARQL Query sq_orm7

B.2 Esper Queries for Event Detection

```
select * from StockEvent(value('location_:_http://dbpedia.org/resource/Round_Rock,_Texas
    ')=true)
```

Listing B.21: Esper Query eq_m1

```
select * from StockEvent(value('industry_:_http://dbpedia.org/resource/Computer')=true)
```

Listing B.22: Esper Query eq_m2

```
select * from StockEvent(nbigger('employees_:_1000')=true)
```

Listing B.23: Esper Query eq_m3

```
select * from StockEvent(value('products_:_http://dbpedia.org/resource/Smartphone')=true
    )
```

Listing B.24: Esper Query eq_m4

```
select * from StockEvent(value('subsidiary_:_http://dbpedia.org/resource/Alienware')=
    true)
```

Listing B.25: Esper Query eq_m5

```
select * from StockEvent(value('rdftype_:_http://dbpedia.org/ontology/Company')=true)
```

Listing B.26: Esper Query eq_m6

```
select * from StockEvent(value('dcterms_:_http://dbpedia.org/resource/Category:
    Mobile_phone_manufacturers')=true)
```

Listing B.27: Esper Query eq_m7

```
select * from StockEvent(value('location : http://dbpedia.org/resource/Round_Rock, Texas
    ')=true, value('industry : http://dbpedia.org/resource/Computer')=true)
```

Listing B.28: Esper Query eq_s2

```
select * from StockEvent(value('location : http://dbpedia.org/resource/Round_Rock, Texas
    ')=true, value('industry : http://dbpedia.org/resource/Computer')=true, nbigger('
    employees : 1000')=true)
```

Listing B.29: Esper Query eq_s3

```
select * from StockEvent(value('location : http://dbpedia.org/resource/Round_Rock, Texas
    ')=true, value('industry : http://dbpedia.org/resource/Computer')=true, nbigger('
    employees : 1000')=true, value('products : http://dbpedia.org/resource/Smartphone')=
    true)
```

Listing B.30: Esper Query eq_s4

```
select * from StockEvent(value('location : http://dbpedia.org/resource/Round_Rock, Texas
    ')=true, value('industry : http://dbpedia.org/resource/Computer')=true, nbigger('
    employees : 1000')=true, value('products : http://dbpedia.org/resource/Smartphone')=
    true, value('subsidiary : http://dbpedia.org/resource/Alienware')=true)
```

Listing B.31: Esper Query eq_s5

```
select * from StockEvent(value('location : http://dbpedia.org/resource/Round_Rock, Texas
    ')=true, value('industry : http://dbpedia.org/resource/Computer')=true, nbigger('
    employees : 1000')=true, value('products : http://dbpedia.org/resource/Smartphone')=
    true, value('subsidiary : http://dbpedia.org/resource/Alienware')=true, value('
    rdftype : http://dbpedia.org/ontology/Company')=true)
```

Listing B.32: Esper Query eq_s6

```
select * from StockEvent(value('location : http://dbpedia.org/resource/Round_Rock, Texas
    ')=true, value('industry : http://dbpedia.org/resource/Computer')=true, nbigger('
    employees : 1000')=true, value('products : http://dbpedia.org/resource/Smartphone')=
    true, value('subsidiary : http://dbpedia.org/resource/Alienware')=true, value('
    rdftype : http://dbpedia.org/ontology/Company')=true, value('dcterms : http://dbpedia
    .org/resource/Category:Mobile_phone_manufacturers')=true)
```

Listing B.33: Esper Query eq_s7

B.3 Esper Queries with Operation

```
select * from pattern [ every a=StockEvent(value('location : http://dbpedia.org/resource
    /Round_Rock, Texas')=true]
```

Listing B.34: ESPER Query eq_opm1

```
select * from pattern [ every a=StockEvent(value('industry : http://dbpedia.org/resource
    /Computer')=true]
```

Listing B.35: ESPER Query eq_opm2

```
select * from pattern [ every a=StockEvent(nbigger('employees : 1000')=true]
```

Listing B.36: ESPER Query eq_opm3

```
select * from pattern [ every a=StockEvent(value('products : http://dbpedia.org/resource
    /Smartphone')=true]
```

Listing B.37: ESPER Query eq_opm4

```
select * from pattern [ every a=StockEvent(value('subsidiary_:_http://dbpedia.org/
    resource/Alienware')=true]
```

Listing B.38: ESPER Query eq_opm5

```
select * from pattern [ every a=StockEvent(value('rdftype_:_http://dbpedia.org/ontology/
    Company')=true]
```

Listing B.39: ESPER Query eq_opm6

```
select * from pattern [ every a=StockEvent(value('dcterms_:_http://dbpedia.org/resource/
    Category:Mobile_phone_manufacturers')=true]
```

Listing B.40: ESPER Query eq_opm7

```
select * from pattern [ every a=StockEvent(value('location_:_http://dbpedia.org/resource
    /Round_Rock,_Texas')=true)]
```

Listing B.41: ESPER Query with OR Operation eq_orm1

```
select * from pattern [ every a=StockEvent(value('location_:_http://dbpedia.org/resource
    /Round_Rock,_Texas')=true)  or  b=StockEvent(value('industry_:_http://dbpedia.org/
    resource/Computer')=true)]
```

Listing B.42: ESPER Query with OR Operation eq_orm2

```
select * from pattern [ every a=StockEvent(value('location_:_http://dbpedia.org/resource
    /Round_Rock,_Texas')=true)  or  b=StockEvent(value('industry_:_http://dbpedia.org/
    resource/Computer')=true)  or  c=StockEvent(nbigger('employees_:_1000')=true)]
```

Listing B.43: ESPER Query with OR Operation eq_orm3

```
select * from pattern [ every a=StockEvent(value('location_:_http://dbpedia.org/resource
    /Round_Rock,_Texas')=true)  or  b=StockEvent(value('industry_:_http://dbpedia.org/
    resource/Computer')=true)  or  c=StockEvent(nbigger('employees_:_1000')=true)  or  d
    =StockEvent(value('products_:_http://dbpedia.org/resource/Smartphone')=true)]
```

Listing B.44: ESPER Query with OR Operation eq_orm4

```
select * from pattern [ every a=StockEvent(value('location_:_http://dbpedia.org/resource
    /Round_Rock,_Texas')=true)  or  b=StockEvent(value('industry_:_http://dbpedia.org/
    resource/Computer')=true)  or  c=StockEvent(nbigger('employees_:_1000')=true)  or  d
    =StockEvent(value('products_:_http://dbpedia.org/resource/Smartphone')=true)  or  e=
    StockEvent(value('subsidiary_:_http://dbpedia.org/resource/Alienware')=true)]
```

Listing B.45: ESPER Query with OR Operation eq_orm5

```
select * from pattern [ every a=StockEvent(value('location_:_http://dbpedia.org/resource
    /Round_Rock,_Texas')=true)  or  b=StockEvent(value('industry_:_http://dbpedia.org/
    resource/Computer')=true)  or  c=StockEvent(nbigger('employees_:_1000')=true)  or  d
    =StockEvent(value('products_:_http://dbpedia.org/resource/Smartphone')=true)  or  e=
    StockEvent(value('subsidiary_:_http://dbpedia.org/resource/Alienware')=true)  or  f=
    StockEvent(value('rdftype_:_http://dbpedia.org/ontology/Company')=true)]
```

Listing B.46: ESPER Query with OR Operation eq_orm6

```
select * from pattern [ every a=StockEvent(value('location_:_http://dbpedia.org/resource
    /Round_Rock,_Texas')=true)  or  b=StockEvent(value('industry_:_http://dbpedia.org/
    resource/Computer')=true)  or  c=StockEvent(nbigger('employees_:_1000')=true)  or  d
    =StockEvent(value('products_:_http://dbpedia.org/resource/Smartphone')=true)  or  e=
    StockEvent(value('subsidiary_:_http://dbpedia.org/resource/Alienware')=true)  or  f=
    StockEvent(value('rdftype_:_http://dbpedia.org/ontology/Company')=true)  or  g=
    StockEvent(value('dcterms_:_http://dbpedia.org/resource/Category:
    Mobile_phone_manufacturers')=true)]
```

Listing B.47: ESPER Query with OR Operation eq_orm7

Appendix C

List of Queries for Event Processing on Sampled Stream

In this appendix, we list the event detection queries used for our experiments on sampled event streams.

C.1 Esper Queries for Event Detection

```
select * from pattern[every (A = TestEvent(name = 'a') -> B = TestEvent(name = 'b')) ]
```

Listing C.1: Esper Query SQ_1

```
select * from pattern[every (A = TestEvent(name = 'a') and B = TestEvent(name = 'b')) ]
```

Listing C.2: Esper Query SQ_2

```
select * from pattern[every (A = TestEvent(name = 'a') or B = TestEvent(name = 'b')) ]
```

Listing C.3: Esper Query SQ_3

```
select * from pattern[every ( (A = TestEvent(name = 'a') and not B = TestEvent(name = 'b
    ')) -> C = TestEvent(name = 'c') ) ]
```

Listing C.4: Esper Query SQ_4

Appendix D
Zusammenfassung

Diese Dissertation beschäftigt sich mit der Fragestellung der Nutzung von Hintergrundwissen zur Ermöglichung wissensbasierter Ereignisverarbeitung. Der Einsatz von ontologischem Hintergrundwissen erhöht die Ausdruckmächtigkeit der Erkennungsmuster, verbessert die Semantik der Ereignisverarbeitung und ermöglicht die Spezifikation von Erkennungsmustern auf einer höheren Abstraktionsschicht.

In dieser Arbeit stellen wir ein Framework für das Semantische Complex Event Processing (SCEP) vor. Dabei bieten wir Lösungen für zwei grundlegende Probleme. Das erste Problem, mit dem wir konfrontiert sind, sind fehlende Wissensrepräsentationsmethoden zur Darstellung von Ereignissen und anderen relevanten Konzepten. Wir schlagen modulare Upper-Level Ontologien vor und beschreiben, warum die Modularität benötigt wird und wie diese Ontologien modular basiert auf Aspekten des Ontologie-Engineering aufgebaut werden. Für die Darstellung von komplexen Ereignissen beschreiben wir ein kombinatorisches Event Detection Pattern auf Basis von Knowledge Graph Patterns und den Kompositionsoperatoren.

Die zweite Herausforderung, die wir behandeln, ist die begrenzte Möglichkeit der Verarbeitungsmethoden für die Fusion von Ereignisströmen und den Hintergrunddaten. Typischerweise gibt es einen Kompromiss (Trade-off) zwischen der höheren Aussagekraft der verwendeten Ontologiesprache, die zu einer höheren Rechenkomplexität führt, und Leistungsanforderungen wie Echtzeit-Ereignisverarbeitung. Obwohl einige der bestehenden Ansätze direkt auf den Datenströmen mit Hilfe von Kompositionsoperatoren komplexe Ereignisse erkennen können, können sie die Ereignisse auf der Basis von Schlussfolgerungen auf Hintergrundwissen nicht erkennen.

Für die Fusion von Hintergrundwissen mit dem Ereignisstrom schlagen wir drei unterschiedliche Ansätze vor, die basiert auf dem Anwendungsfall ausgewählt oder kombiniert werden können. Der erste Ansatz ist die semantische Anreicherung von Ereignisströmen, dies bietet ein Konzept für die Anreicherung der Ereignisdaten mit Hintergrundwissen vor dem Ereignisverarbeitungsschritt.

Der zweite Ansatz ist die Anreicherung von komplexen Erkennungsmustern mit Hintergrundwissen. Die komplexen Erkennungsmuster, die nicht ohne erforderliches Wissen verarbeitet werden können, werden mit Hilfe von der Wissensdaten aus der Wissensbasis angereichert und in neue einfache Formen umgeschrieben. Die Umschreibung geschieht basiered auf den verwendeten Kompositionsoperatoren und den relevanten Relationen in der Wissensbasis.

Zur Verringerung der Ereignisverarbeitungslast schlagen wir einen weiteren Ansatz vor. Dabei werden die Datensätze basiert auf randomisierten Faktoren ausgewählt, sodass die Event Processing Engine nur eine Teilmenge des ursprünglichen Datenstroms beobachten darf. Ansätze für die Probenahme von Datenströmen werden vor allem für die Berechnungsfunktionen verwendet und sind nicht zur Mustererkennung ausgelegt. Wir schlagen einen Ansatz für die typ-basierte Probenahme von Ereignisströmen vor, sodass die Probenahme auf der Basis der Entropie des Ereignisstroms und der Eigenschaften des benutzerdefinierten Erkennungsmusters funktioniert. Unser Ansatz kann Proben des ursprünglichen Ereignisstroms durch die Kalibrierung der Ereignistypen berechnen, um den angeforderten Durchsatz des Roh-Ereignisstroms zu erreichen oder eine bestimmte Rate von komplexen Ereignissen zu erkennen.

Appendix E
About the Author

Kia Teymourian studied computer science at the Berlin University of Technology (TU-Berlin, 2000-2006). He received his Masters of Computer Science (equivalent to Master of Science "in German: Diplom-Informatiker Gesamturteil: -sehr gut -", US grading system: A) from the TU-Berlin in December 2006.

Previously, he had studied civil engineering at Tehran Azad University (1993-1997) with a major in hydraulics and water supply engineering. He received his B.Sc. in civil engineering in July 1997 and worked as a civil engineer for consulting engineering companies (1997-2000).

From May 2007 until December 2014, he worked as a research associate at the Networked Information Systems Research Group headed by Prof. Robert Tolksdorf and the Corporate Semantic Web Research Group overseen by Prof. Adrian Paschke, both of which are part of the Institute of Computer Science of the Free University of Berlin. He worked on '*'TripCom", "Digipolis", "Corporate Semantic Web"* and *"Corporate Smart Content"* research projects.

His research interests include data streaming processing, complex event processing, reactive systems and reactive rule languages, extended database technologies and distributed systems in general.

Bibliography

[Aba03] ABADI, Daniel J.; CARNEY, Don; ÇETINTEMEL, Ugur; CHERNIACK, Mitch; CONVEY, Christian; LEE, Sangdon; STONEBRAKER, Michael; TATBUL, Nesime and ZDONIK, Stan: Aurora: a new model and architecture for data stream management. *The VLDB Journal* (2003), vol. 12(2):pp. 120–139, URL http://dx.doi.org/10.1007/s00778-003-0095-z

[Abd07] ABDELMOTY, A. I.; SMART, P. and JONES, C. B.: Building place ontologies for the semantic web:: issues and approaches, in: *GIR '07: Proceedings of the 4th ACM workshop on Geographical information retrieval*, ACM, New York, NY, USA, pp. 7–12

[Ada06] ADAIKKALAVAN, Raman and CHAKRAVARTHY, Sharma: SnoopIB: Interval-based event specification and detection for active databases. *Data & Knowledge Engineering* (2006), vol. 59(1):pp. 139 – 165, URL http://www.sciencedirect.com/science/article/pii/S0169023X05001096

[Adi04] ADI, Asaf and ETZION, Opher: Amit - the situation manager. *The VLDB Journal* (2004), vol. 13(2):pp. 177–203, URL http://dx.doi.org/10.1007/s00778-003-0108-y

[AdM09] ALVES DE MEDEIROS, A.K. and AALST, W.M.P.: Process Mining towards Semantics, in: TharamS. Dillon; Elizabeth Chang; Robert Meersman and Katia Sycara (Editors) *Advances in Web Semantics I*, vol. 4891 of *Lecture Notes in Computer Science*, Springer Berlin Heidelberg (2009), pp. 35–80, URL http://dx.doi.org/10.1007/978-3-540-89784-2_3

[Agg07] AGGARWAL, Charu C. (Editor): *Data Streams - Models and Algorithms*, vol. 31 of *Advances in Database Systems*, Springer (2007), URL http://dx.doi.org/10.1007/978-0-387-47534-9

[Agr89] AGRAWAL, R. and GEHANI, N. H.: ODE (Object Database and Environment): the language and the data model. *SIGMOD Rec.* (1989), vol. 18(2):pp. 36–45, URL http://doi.acm.org/10.1145/66926.66930

[Agr93] AGRAWAL, Rakesh; IMIELIŃSKI, Tomasz and SWAMI, Arun: Mining Association Rules Between Sets of Items in Large Databases. *SIGMOD Rec.* (1993), vol. 22(2):pp. 207–216, URL http://doi.acm.org/10.1145/170036.170072

[Agr00] AGRAWAL, Sanjay; CHAUDHURI, Surajit and NARASAYYA, Vivek R.: Automated Selection of Materialized Views and Indexes in SQL Databases, in: *Proceedings of the 26th International Conference on Very Large Data Bases*, VLDB '00, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, pp. 496–505, URL http://dl.acm.org/citation.cfm?id=645926.671701

[Agr08] AGRAWAL, Jagrati; DIAO, Yanlei; GYLLSTROM, Daniel and IMMERMAN, Neil: Efficient Pattern Matching over Event Streams, in: *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, SIGMOD '08, ACM, New York, NY, USA, pp. 147–160, URL http://doi.acm.org/10.1145/1376616.1376634

[Akd08] AKDERE, Mert; ÇETINTEMEL, Ugur and TATBUL, Nesime: Plan-based complex event detection across distributed sources. *Proc. VLDB Endow.* (2008), vol. 1:pp. 66–77, URL http://www.vldb.org/pvldb/1/1453869.pdf

[Ani11a] ANICIC, Darko; FODOR, Paul; RUDOLPH, Sebastian and STOJANOVIC, Nenad: EP-SPARQL: a unified language for event processing and stream reasoning, in: *Proceedings of the 20th international conference on World Wide Web*, WWW '11, ACM, New York, NY, USA, pp. 635–644, URL http://doi.acm.org/10.1145/1963405.1963495

[Ani11b] ANICIC, Darko; FODOR, Paul; RUDOLPH, Sebastian; STÜHMER, Roland; STOJANOVIC, Nenad and STUDER, Rudi: ETALIS: Rule-Based Reasoning in Event Processing, in: Sven Helmer; Alexandra Poulovassilis and Fatos Xhafa (Editors) *Reasoning in Event-Based Distributed Systems*, vol. 347 of *Studies in Computational Intelligence*, Springer Berlin / Heidelberg (2011), pp. 99–124

[Ant04] ANTONIOU, Grigoris and VANHARMELEN, Frank: *A Semantic Web Primer*, MIT Press (2004)

[Ara06] ARASU, Arvind; BABU, Shivnath and WIDOM, Jennifer: The CQL continuous query language: semantic foundations and query execution. *The VLDB Journal* (2006), vol. 15(2):pp. 121–142, URL http://dx.doi.org/10.1007/s00778-004-0147-z

[Art98] ARTALE, Alessandro and FRANCONI, Enrico: A Temporal Description Logic for Reasoning about Actions and Plans. *J. Artif. Int. Res.* (1998), vol. 9(1):pp. 463–506, URL http://dl.acm.org/citation.cfm?id=1622797.1622809

[Art01] ARTALE, Alessandro and FRANCONI, Enrico: A Survey of Temporal Extensions of Description Logics. *Annals of Mathematics and Artificial Intelligence* (2001), vol. 30(1-4):pp. 171–210, URL http://dx.doi.org/10.1023/A:1016636131405

[Art09] ARTIKIS, Alexander and PALIOURAS, Georgios: Behaviour Recognition using the Event Calculus, in: Lazaros S. Iliadis; Ilias Maglogiannis; Grigorios Tsoumakas; Ioannis P. Vlahavas and Max Bramer (Editors) *Artificial Intelligence Applications and Innovations III, Proceedings of the 5TH IFIP Conference on Artificial Intelligence Applications and Innovations (AIAI'2009), April 23-25, 2009, Thessaloniki, Greece*, vol. 296 of *IFIP Advances in Information and Communication Technology*, Springer (2009), pp. 469–478, URL http://dx.doi.org/10.1007/978-1-4419-0221-4_55

[Baa03] BAADER, Franz; CALVANESE, Diego; MCGUINNESS, Deborah L.; NARDI, Daniele and PATEL-SCHNEIDER, Peter F. (Editors): *The Description Logic Handbook: Theory, Implementation, and Applications*, Cambridge University Press, New York, NY, USA (2003)

[Baa08] BAADER, Franz; HORROCKS, Ian and SATTLER, Ulrike: Description Logics, in: Frank van Harmelen; Vladimir Lifschitz and Bruce Porter (Editors) *Handbook of Knowledge Representation*, chap. 3, Elsevier (2008), pp. 135–180, URL download/2007/BaHS07a.pdf

[Bab02] BABCOCK, Brian; DATAR, Mayur and MOTWANI, Rajeev: Sampling from a moving window over streaming data, in: *Proceedings of the thirteenth annual ACM-SIAM symposium on Discrete algorithms*, SODA '02, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, pp. 633–634, URL http://dl.acm.org/citation.cfm?id=545381.545465

[Bal05] BALDONI, Roberto and VIRGILLITO, Antonino: Distributed Event Routing in Publish/Subscribe Communication Systems: a survey, Technical Report 15-05, Dipartimento di Informatica e Sistemistica, Università di Roma "La Sapienzia", Rome, Italy (2005), URL DistributedEventRoutinginPublish/SubscribeCommunicationSystems:asurvey

[Bar10] BARBIERI, Davide Francesco; BRAGA, Daniele; CERI, Stefano and GROSSNIKLAUS, Michael: An execution environment for C-SPARQL queries, in: *Proceedings of the 13th International Conference on Extending Database Technology*, EDBT '10, ACM, New York, NY, USA, pp. 441–452, URL http://doi.acm.org/10.1145/1739041.1739095

[Bas93] BASSEVILLE, Michèle and NIKIFOROV, Igor V.: *Detection of Abrupt Changes: Theory and Application*, Prentice-Hall, Inc., Upper Saddle River, NJ, USA (1993)

[Bas09] BASKERVILLE, Richard; PRIES-HEJE, Jan and VENABLE, John: Soft design science methodology, in: *Proceedings of the 4th International Conference on Design Science Research in Information Systems and Technology*, DESRIST '09, ACM, New York, NY, USA, pp. 9:1–9:11, URL http://doi.acm.org/10.1145/1555619.1555631

[Bee91] BEERI, Catriel and MILO, Tova: A Model for Active Object Oriented Databases, in: *Proceedings of the 17th International Conference on Very Large Data Bases*, VLDB '91, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, pp. 337–349, URL http://dl.acm.org/citation.cfm?id=645917.672146

[Ber92] BERNDTSSON, Mikael and LINGS, Brian: On Developing Reactive Object-Oriented Databases, Tech. Rep. TR-92-04, Department of Computer Science, University of Skoevde (1992)

[Ber01] BERNAT, Guillem; BURNS, Alan and LLAMOSI, Albert: Weakly Hard Real-Time Systems. *IEEE Trans. Comput.* (2001), vol. 50(4):pp. 308–321, URL http://dx.doi.org/10.1109/12.919277

[Ber07] BERNHARDT, Thomas and VASSEUR, Alexandre: Esper: Event Stream Processing and Correlation, Online article (2007), URL http://onjava.com/pub/a/onjava/2007/03/07/esper-event-stream-processing-and-correlation.html

[Ber08] BERNHARDT, Thomas and VASSEUR, Alexandre: Esper-complex event processing, Online article (2008), URL http://esper.codehaus.org/

[Bhu06] BHUVANAGIRI, Lakshminath and GANGULY, Sumit: Estimating Entropy over Data Streams, in: Yossi Azar and Thomas Erlebach (Editors) *Algorithms – ESA 2006*, vol. 4168 of *Lecture Notes in Computer Science*, Springer Berlin Heidelberg (2006), pp. 148–159, URL http://dx.doi.org/10.1007/11841036_16

[BL98] BERNERS-LEE, Tim: The Semantic Web as a language of logic (1998), URL http://www.w3.org/DesignIssues/Logic.html

[BL01] BERNERS-LEE, Tim; HENDLER, James and LASSILA, Ora: The Semantic Web. *Scientific American Magazine* (2001), URL http://www.scientificamerican.com/article.cfm?id=the-semantic-web

[Bol08] BOLLES, Andre; GRAWUNDER, Marco and JACOBI, Jonas: Streaming SPARQL extending SPARQL to process data streams, in: *Proceedings of the 5th European semantic web conference on The semantic web: research and applications*, ESWC'08, Springer-Verlag, Berlin, Heidelberg, pp. 448–462, URL http://dl.acm.org/citation.cfm?id=1789394.1789438

[Boo53] BOOLE, George: *Investigation of The Laws of Thought on Which Are Founded the Mathematical Theories of Logic and Probabilities* (1853), URL http://www.gutenberg.org/etext/15114

[Bor03] BORGIDA, Alex and SERAFINI, Luciano: Distributed Description Logics: Assimilating Information from Peer Sources. *Journal of Data Semantics* (2003), vol. 1:p. 2003

[Bra12] BRAVO, José; HERVÁS, Ramón and RODRÍGUEZ, Marcela (Editors): *Ambient Assisted Living and Home Care - 4th International Workshop, IWAAL 2012, Vitoria-Gasteiz, Spain, December 3-5, 2012. Proceedings*, vol. 7657 of *Lecture Notes in Computer Science*, Springer (2012), URL http://dx.doi.org/10.1007/978-3-642-35395-6

[Bre07] BRENNA, Lars; DEMERS, Alan; GEHRKE, Johannes; HONG, Mingsheng; OSSHER, Joel; PANDA, Biswanath; RIEDEWALD, Mirek; THATTE, Mohit and WHITE, Walker: Cayuga: a high-performance event processing engine, in: *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, SIGMOD '07, ACM, New York, NY, USA, pp. 1100–1102, URL http://doi.acm.org/10.1145/1247480.1247620

[Bru10] BRUNS, R. and DUNKEL, J.: *Event-Driven Architecture: Softwarearchitektur für ereignisgesteuerte Geschäftsprozesse*, Springer (2010)

[Bry07] BRY, François and ECKERT, Michael: 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)*, vol. 4524 of *LNCS*, pp. 16–30, URL http://rewerse.net/publications/rewerse-publications.html#REWERSE-RP-2007-035

[Bry09] BRY, François; ECKERT, Michael; ETZION, Opher; PASCHKE, Adrian and RIECK, Jon: Event Processing Language Tutorial (2009)

[Buc95] BUCHMANN, A. P.; DEUTSCH, A.; ZIMMERMANN, J. and HIGA, M.: The REACH active OODBMS. *SIGMOD Rec.* (1995), vol. 24(2):pp. 476–, URL http://doi.acm.org/10.1145/568271.223889

[Cal10] CALBIMONTE, Jean-Paul; CORCHO, Oscar and GRAY, Alasdair J. G.: Enabling ontology-based access to streaming data sources, in: *Proceedings of the 9th international semantic web conference on The semantic web - Volume Part I*, ISWC'10, Springer-Verlag, Berlin, Heidelberg, pp. 96–111, URL http://dl.acm.org/citation.cfm?id=1940281.1940289

[Car01] CARZANIGA, Antonio; ROSENBLUM, David S. and WOLF, Alexander L.: Design and evaluation of a wide-area event notification service. *ACM Trans. Comput. Syst.* (2001), vol. 19(3):pp. 332–383, URL http://doi.acm.org/10.1145/380749.380767

[Cas10] CASTILLO, Roger and LESER, Ulf: Selecting Materialized Views for RDF Data, in: *Proceedings of the 10th International Conference on Current Trends in Web Engineering*, ICWE'10, Springer-Verlag, Berlin, Heidelberg, pp. 126–137, URL http://dl.acm.org/citation.cfm?id=1927229.1927244

[Cha93] CHAKRAVARTHY, S.; KRISHNAPRASAD, V.; ANWAR, E. and K KIM, S.: Anatomy of a Composite Event Detector, Tech. Rep. UF-CIS-TR-93-039, Computer and Information Sciences, University of Florida (1993)

[Cha94a] CHAKRAVARTHY, S. and MISHRA, D.: Snoop: an expressive event specification language for active databases. *Data Knowl. Eng.* (1994), vol. 14:pp. 1–26, URL http://portal.acm.org/citation.cfm?id=197053.197054

[Cha94b] CHAKRAVARTHY, Sharma; KRISHNAPRASAD, V.; ANWAR, Eman and KIM, S.-K.: Composite Events for Active Databases: Semantics, Contexts and Detection, in: *Proceedings of the 20th International Conference on Very Large Data Bases*, VLDB '94, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, pp. 606–617, URL http://dl.acm.org/citation.cfm?id=645920.672994

[Cha95] CHAKRAVARTHY, Sharma; KRISHNAPRASAD, V.; TAMIZUDDIN, Z. and BADANI, R. H.: ECA Rule Integration into an OODBMS: Architecture and Implementation, in: *Proceedings of the Eleventh International Conference on Data Engineering*, ICDE '95, IEEE Computer Society, Washington, DC, USA, pp. 341–348, URL http://dl.acm.org/citation.cfm?id=645480.655427

[Cha97] CHAKRAVARTHY, S.: Sentinel: an object-oriented DBMS with event-based rules, in: *Proceedings of the 1997 ACM SIGMOD international conference on Management of data*, SIGMOD '97, ACM, New York, NY, USA, pp. 572–575, URL http://doi.acm.org/10.1145/253260.253409

[Cha03] CHANDRASEKARAN, Sirish; COOPER, Owen; DESHPANDE, Amol; FRANKLIN, Michael J.; HELLERSTEIN, Joseph M.; HONG, Wei; KRISHNAMURTHY, Sailesh; MADDEN, Samuel R.; REISS, Fred and SHAH, Mehul A.: TelegraphCQ: Continuous Dataflow Processing for an Uncertain World, in: *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, SIGMOD '03, ACM, New York, NY, USA, pp. 668–668, URL http://doi.acm.org/10.1145/872757.872857

[Cha09] CHAKRAVARTHY, Sharma and JIANG, Qingchun: *Stream Data Processing: A Quality of Service Perspective Modeling, Scheduling, Load Shedding, and Complex Event Processing*, Springer Publishing Company, Incorporated, 1st edn. (2009)

[Cha10] CHAKRABARTI, Amit; CORMODE, Graham and MCGREGOR, Andrew: A near-optimal algorithm for estimating the entropy of a stream. *ACM Trans. Algorithms* (2010), vol. 6(3):pp. 51:1–51:21, URL http://doi.acm.org/10.1145/1798596.1798604

[Cra03] CRANOR, Chuck; JOHNSON, Theodore; SPATASCHEK, Oliver and SHKAPENYUK, Vladislav: Gigascope: a stream database for network applications, in: *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, SIGMOD '03, ACM, New York, NY, USA, pp. 647–651, URL http://doi.acm.org/10.1145/872757.872838

[Dan12] DANH, L. P.; MINH, D. T.; DUC, P. Minh; BONCZ, P. A.; THOMAS, E. and MICHAEL, F.: Linked Stream Data Processing: Facts and Figures (2012)

[Day88] DAYAL, U.; BLAUSTEIN, B.; BUCHMANN, A.; CHAKRAVARTHY, U.; HSU, M.; LEDIN, R.; MCCARTHY, D.; ROSENTHAL, A.; SARIN, S.; CAREY, M. J.; LIVNY, M. and JAUHARI, R.: The HiPAC project: combining active databases and timing constraints. *SIGMOD Rec.* (1988), vol. 17(1):pp. 51–70, URL http://doi.acm.org/10.1145/44203.44208

[Dea08] DEAN, Jeffrey and GHEMAWAT, Sanjay: MapReduce: simplified data processing on large clusters. *Commun. ACM* (2008), vol. 51(1):pp. 107–113, URL http://doi.acm.org/10.1145/1327452.1327492

[Del09] DELLA VALLE, Emanuele; CELINO, Irene and DELL'AGLIO, Daniele: The alpha Urban LarKC - a Semantic Urban Computing application, in: *In the Posters and Demo session at the 8th International Semantic Web Conference (ISWC 2009), Washington, DC, 25-29 October 2009*, URL http://www.larkc.eu/wp-content/uploads/2008/01/alpha-urban-larkc_iswc-poster-demo.pdf

[Del10] DELLA VALLE, Emanuele; CELINO, Irene and DELL'AGLIO, Daniele: The Experience of Realizing a Semantic Web Urban Computing Application. *T. GIS* (2010), vol. 14(2):pp. 163–181, URL http://dx.doi.org/10.1111/j.1467-9671.2010.01189.x

[Dem06] DEMERS, Alan; GEHRKE, Johannes; HONG, Mingsheng; RIEDEWALD, Mirek and WHITE, Walker: Towards expressive publish/subscribe systems, in: *Proceedings of the 10th international conference on Advances in Database Technology*, EDBT'06, Springer-Verlag, Berlin, Heidelberg, pp. 627–644, URL http://dx.doi.org/10.1007/11687238_38

[Dem07] DEMERS, Alan J.; GEHRKE, Johannes; PANDA, Biswanath; RIEDEWALD, Mirek; SHARMA, Varun and WHITE, Walker M.: Cayuga: A General Purpose Event Monitoring System, in: *CIDR 2007, Third Biennial Conference on Innovative Data Systems Research, Asilomar, CA, USA, January 7-10, 2007, Online Proceedings*, www.cidrdb.org (2007), pp. 412–422, URL http://www.cidrdb.org/cidr2007/papers/cidr07p47.pdf

[Dia91] DIAZ, Oscar; PATON, Norman W. and GRAY, Peter M. D.: Rule Management in Object Oriented Databases: A Uniform Approach, in: *Proceedings of the 17th International Conference on Very Large Data Bases*, VLDB '91, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, pp. 317–326, URL http://dl.acm.org/citation.cfm?id=645917.672325

[Dia07] DIAO, Yanlei; IMMERMAN, Neil and GYLLSTROM, Daniel: SASE+: An agile language for kleene closure over event streams, Tech. Rep., University of Massachusetts, Amherst (2007), 07-03

[Dit86] DITTRICH, Klaus R.; KOTZ, Angelika M. and MÜLLE, Jutta A.: An event/trigger mechanism to enforce complex consistency constraints in design databases. *SIGMOD Rec.* (1986), vol. 15(3):pp. 22–36, URL http://doi.acm.org/10.1145/15833.15836

[DL11] DAVID LUCKHAM, Roy Schulte: Event Processing Glossary - Version 2.0, Event Processing Technical Society (2011), URL http://www.ep-ts.com

[Doe03] DOERR, Martin: The CIDOC conceptual reference module: an ontological approach to semantic interoperability of metadata. *AI Mag.* (2003), vol. 24(3):pp. 75–92

[Ege04] EGENHOFER, Max J.; FREKSA, Christian and MILLER, Harvey J. (Editors): *GIScience*, vol. 3234 of *Lecture Notes in Computer Science*, Springer (2004)

[ept08] Event Glossary 1.1, Event Processing Technical Society, http://www.ep-ts.com (2008)

[Etz10a] ETZION, Opher: Event Processing: Past, Present and Future. *Proc. VLDB Endow.* (2010), vol. 3(1-2):pp. 1651–1652, URL http://dl.acm.org/citation.cfm?id=1920841.1921065

[Etz10b] ETZION, Opher and NIBLETT, Peter: *Event Processing in Action*, Manning Publications Co., Greenwich, CT, USA, 1st edn. (2010)

[Eug03] EUGSTER, Patrick Th.; FELBER, Pascal A.; GUERRAOUI, Rachid and KERMARREC, Anne-Marie: The many faces of publish/subscribe. *ACM Comput. Surv.* (2003), vol. 35(2):pp. 114–131, URL http://doi.acm.org/10.1145/857076.857078

[Fen07] FENSEL, Dieter; LAUSEN, Holger; POLLERES, Axel; BRUIJN, Jos de; STOLLBERG, Michael; ROMAN, Dumitru and DOMINGUE, John: *Enabling Semantic Web Services: The Web Service Modeling Ontology*, Springer (2007)

[Fer97] FERNANDEZ, Mariano; GOMEZ-PEREZ, Asuncion and JURISTO, Natalia: METHONTOLOGY: from Ontological Art towards Ontological Engineering, in: *Proceedings of the AAAI97 Spring Symposium Series on Ontological Engineering*, Stanford, USA, pp. 33–40

[Fid05] FIDLER, E.; JACOBSEN, Hans-Arno; LI, Guoli and MANKOVSKI, Serge: The PADRES Distributed Publish/Subscribe System., in: Stephan Reiff-Marganiec and Mark Ryan (Editors) *Proceedings of the 8th international Conference on Feature Interactions in Telecommunications and Software Systems (FIW05)*, IOS Press, pp. 12–30, URL http://www.cs.toronto.edu/~gli/icfi05.pdf

[Fik02] FIKES, Richard and ZHOU, Qing: A Reusable Time Ontology, Tech. Rep., AAAI (2002), URL http://www.aaai.org/Papers/Workshops/2002/WS-02-11/WS02-11-015.pdf, technical Report WS-02-11

[Fik04] FIKES, Richard; HAYES, Patrick and HORROCKS, Ian: OWL-QL-a language for deductive query answering on the Semantic Web. *Web Semant.* (2004), vol. 2(1):pp. 19–29, URL http://dx.doi.org/10.1016/j.websem.2004.07.002

[For79] FORGY, Charles Lanny: *On the efficient implementation of production systems.*, Ph.D. thesis, Pittsburgh, PA, USA (1979), aAI7919143

[For90] FORGY, Charles L.: Expert systems, chap. Rete: a fast algorithm for the many pattern/many object pattern match problem, IEEE Computer Society Press, Los Alamitos, CA, USA (1990), pp. 324–341, URL http://dl.acm.org/citation.cfm?id=115710.115736

[Gan03] GANGEMI, Aldo and MIKA, Peter: Understanding the Semantic Web through Descriptions and Situations, in: Robert Meersman; Zahir Tari and Douglas C. Schmidt (Editors) *On The Move to Meaningful Internet Systems 2003: CoopIS, DOA, and ODBASE - OTM Confederated International Conferences, CoopIS, DOA, and ODBASE 2003, Catania, Sicily, Italy, November 3-7, 2003*, vol. 2888 of *Lecture Notes in Computer Science*, Springer (2003), pp. 689–706, URL http://springerlink.metapress.com/openurl.asp?genre=article{&}issn=0302-9743{&}volume=2888{&}spage=689

[Gat92] GATZIU, Stella and DITTRICH, Klaus R.: SAMOS: an Active Object-Oriented Database System, in: *IEEE Quartely Bulletin on Data Engineering*, IEEE (1992), pp. 23–39

[Gat94] GATZIU, Stella and DITTRICH, Klaus R.: Detecting Composite Events in Active Database Systems Using Petri Nets, in: Jennifer Widom and Sharma Chakravarthy (Editors) *Fourth International Workshop on Research Issues in Data Engineering: Active Database Systems, Houston, Texas, February 14-15, 1994*, IEEE Computer Society (1994), pp. 2–9, URL http://dx.doi.org/10.1109/RIDE.1994.282859

[Geh92a] GEHANI, N. H.; JAGADISH, H. V. and SHMUELI, O.: Event specification in an active object-oriented database. *SIGMOD Rec.* (1992), vol. 21(2):pp. 81–90, URL http://doi.acm.org/10.1145/141484.130300

[Geh92b] GEHANI, Narain H.; JAGADISH, H. V. and SHMUELI, Oded: Composite Event Specification in Active Databases: Model & Implementation, in: *VLDB '92: Proceedings of the 18th International Conference on Very Large Data Bases*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, pp. 327–338

[Geh93] GEHANI, Narain H.; JAGADISH, H. V. and SHMUELI, Oded: COMPOSE: A System For Composite Specification And Detection, in: *Advanced Database Systems*, Springer-Verlag, London, UK (1993), pp. 3–15, URL http://dl.acm.org/citation.cfm?id=647416.725348

[Gem06] GEMULLA, Rainer; LEHNER, Wolfgang and HAAS, Peter J.: A dip in the reservoir: maintaining sample synopses of evolving datasets, in: *Proceedings of the 32nd international conference on Very large data bases*, VLDB '06, VLDB Endowment, pp. 595–606, URL http://dl.acm.org/citation.cfm?id=1182635.1164179

[Gha06] GHANEM, Thanaa M.; AREF, Walid G. and ELMAGARMID, Ahmed K.: Exploiting predicate-window semantics over data streams. *SIGMOD Rec.* (2006), vol. 35(1):pp. 3–8, URL http://doi.acm.org/10.1145/1121995.1121996

[Gib07] GIBERT, Karina; VALLS, Aida and CASALS, Joan: Enlarging a Medical Actor Profile Ontology with New Care Units, in: David Riaño (Editor) *K4CARE*, vol. 4924 of *Lecture Notes in Computer Science*, Springer (2007), pp. 101–116, URL http://dx.doi.org/10.1007/978-3-540-78624-5_8

[Gil02] GILBERT, Seth and LYNCH, Nancy: Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. *SIGACT News* (2002), vol. 33(2):pp. 51–59, URL http://doi.acm.org/10.1145/564585.564601

[Glo13] GLOMBIEWSKI, Nikolaus; HOSSBACH, Bastian; MORGEN, Andreas; RITTER, Franz and SEEGER, Bernhard: Event Processing on your own Database, in: Gunter Saake; Andreas Henrich; Wolfgang Lehner; Thomas Neumann and Veit Köppen (Editors) *Datenbanksysteme für Business, Technologie und Web (BTW), - Workshopband, 15. Fachtagung des GI-Fachbereichs "Datenbanken und Informationssysteme" (DBIS), 11.-15.3.2013 in Magdeburg, Germany. Proceedings*, LNI, GI (2013), pp. 33–42, URL http://www.btw-2013.de/proceedings/Event%20Processing%20on%20your%20own%20Database.pdf

[GM06] GERO MÜHL, Peter Pietzuch, Ludger Fiege: *Distributed Event-Based Systems*, springer (2006)

[Gom13] GOMARIZ, Antonio; CAMPOS, Manuel; MARÍN, Roque and GOETHALS, Bart: ClaSP: An Efficient Algorithm for Mining Frequent Closed Sequences, in: Jian Pei; Vincent S. Tseng; Longbing Cao; Hiroshi Motoda and Guandong Xu (Editors) *Advances in Knowledge Discovery and Data Mining, 17th Pacific-Asia Conference, PAKDD 2013, Gold Coast, Australia, April 14-17, 2013, Proceedings, Part I*, vol. 7818 of *Lecture Notes in Computer Science*, Springer (2013), pp. 50–61, URL http://dx.doi.org/10.1007/978-3-642-37453-1_5

[GP04] GÓMEZ-PÉREZ, Asunción; FERNÁNDEZ-LÓPEZ, Mariano and CORCHO, Oscar: *Ontological Engineering: with examples from the areas of Knowledge Management, e-Commerce and the Semantic Web*, Springer (2004)

[Gra81] GRAY, Jim: The transaction concept: virtues and limitations (invited paper), in: *Proceedings of the seventh international conference on Very Large Data Bases - Volume 7*, VLDB '81, VLDB Endowment, pp. 144–154, URL http://dl.acm.org/citation.cfm?id=1286831.1286846

[Gra06] GRAU, Bernardo Cuenca; PARSIA, Bijan; SIRIN, Evren and KALYANPUR, Aditya: Modularity and Web Ontologies, in: Patrick Doherty; John Mylopoulos and Christopher A. Welty (Editors) *Proceedings, Tenth International Conference on Principles of Knowledge Representation and Reasoning, Lake District of the United Kingdom, June 2-5, 2006*, AAAI Press (2006), pp. 198–209

[Gro03] GROUP, The STREAM: STREAM: The Stanford Stream Data Manager, Technical Report 2003-21, Stanford InfoLab (2003), URL http://ilpubs.stanford.edu:8090/583/

[Gru93] GRUBER, Thomas R.: A translation approach to portable ontology specifications. *Knowl. Acquis.* (1993), vol. 5(2):pp. 199–220

[Gu05] GU, Yu; MCCALLUM, Andrew and TOWSLEY, Don: Detecting Anomalies in Network Traffic Using Maximum Entropy Estimation, in: *Proceedings of the 5th ACM SIGCOMM Conference on Internet Measurement*, IMC '05, USENIX Association, Berkeley, CA, USA, pp. 32–32, URL http://dl.acm.org/citation.cfm?id=1251086.1251118

[Gup99] GUPTA, Ashish and MUMICK, Inderpal Singh: Materialized Views, chap. Maintenance of Materialized Views: Problems, Techniques, and Applications, MIT Press, Cambridge, MA, USA (1999), pp. 145–157, URL http://dl.acm.org/citation.cfm?id=310709.310737

[Gyl06] GYLLSTROM, Daniel; WU, Eugene; CHAE, Hee-Jin; DIAO, Yanlei; STAHLBERG, Patrick and ANDERSON, Gordon: SASE: Complex Event Processing over Streams. *CoRR* (2006), vol. abs/cs/0612128, URL http://arxiv.org/abs/cs/0612128

[Haa01] HAARSLEV, Volker and MÖLLER, Ralf: RACER System Description, in: *IJCAR '01: Proceedings of the First International Joint Conference on Automated Reasoning*, Springer-Verlag, London, UK, pp. 701–706

[Haa08] HAARSLEV, Volker and MÖLLER, Ralf: On the Scalability of Description Logic Instance Retrieval. *J. Autom. Reason.* (2008), vol. 41(2):pp. 99–142, URL http://dx.doi.org/10.1007/s10817-008-9104-7

[Han07] HAN, Jiawei; CHENG, Hong; XIN, Dong and YAN, Xifeng: Frequent Pattern Mining: Current Status and Future Directions. *Data Min. Knowl. Discov.* (2007), vol. 15(1):pp. 55–86, URL http://dx.doi.org/10.1007/s10618-006-0059-1

[Han11] HAN, Jing; HAIHONG, E.; LE, Guan and DU, Jian: Survey on NoSQL database, in: *Pervasive Computing and Applications (ICPCA), 2011 6th International Conference on*, pp. 363–366

[Har02] HARRIS, Earl: Information Gain Versus Gain Ratio: A Study of Split Method Biases, in: *International Symposium on Artificial Intelligence and Mathematics (AI&M 2002), Fort Lauderdale, Florida, USA, January 2-4, 2002* (2002), URL http://rutcor.rutgers.edu/$\sim$amai/aimath02/PAPERS/14.pdf

[He14] HE, Yeye; BARMAN, Siddharth and NAUGHTON, Jeffrey F.: On Load Shedding in Complex Event Processing, in: Nicole Schweikardt; Vassilis Christophides and Vincent Leroy (Editors) *Proc. 17th International Conference on Database Theory (ICDT), Athens, Greece, March 24-28, 2014*, OpenProceedings.org (2014), pp. 213–224, URL http://dx.doi.org/10.5441/002/icdt.2014.23

[Hev04] HEVNER, A.R.; MARCH, S.T.; PARK, J. and RAM, S.: Design science in information systems research. *Management Information Systems Quarterly* (2004), vol. 28(1):pp. 75–106, URL http://www.hec.unil.ch/yp/HCI/articles/hevner04.pdf

[Hin10] HINZE, A. and BUCHMANN, A.P.: *Principles and Applications of Distributed Event-Based Systems*, IGI Global research collection, Information Science Reference (2010), URL http://books.google.de/books?id=Fk5-Q0J0Cp0C

[Hir09] HIRZEL, Martin; ANDRADE, Henrique; GEDIK, Bugra; KUMAR, Vibhore; LOSA, Giuliano; SOULÉ, Robert and WU, Kun-Lung: SPADE Language Specification, Technical report, IBM (2009)

[Hob04] HOBBS, Jerry R. and PAN, Feng: An ontology of time for the semantic web. *ACM Trans. Asian Lang. Inf. Process.* (2004), vol. 3(1):pp. 66–85, URL http://doi.acm.org/10.1145/1017068.1017073

[Hor05] HORROCKS, Ian; PARSIA, Bijan; PATEL-SCHNEIDER, Peter and HENDLER, James: Semantic Web Architecture: Stack or Two Towers, in: *Proceedings of Principles and Practice of Semantic Web Reasoning, Third International Workshop*, Springer (2005), pp. 37–41

[Hut03] HUTWAGNER, L.; THOMPSON, W.; SEEMAN, G. M. and TREADWELL, T.: The bioterrorism preparedness and response Early Aberration Reporting System (EARS). *J Urban Health* (2003), vol. 80:pp. i89–96, URL http://www.ncbi.nlm.nih.gov/entrez/query.fcgi?cmd=Retrieve&db=PubMed&dopt=Citation&list_uids=12791783

[Hyv07] HYVÖNEN, Eero; LINDROOS, Robin; KAUPPINEN, Tomi and HENRIKSSON, Riikka: An Ontology Service for Geographical Content, in: *6th International and 2nd Asian Semantic Web Conference (ISWC2007+ASWC2007)*, pp. 33–34, URL http://data.semanticweb.org/conference/iswc-aswc/2007/tracks/posters-demos/papers/372

[Hyv09] HYVÖNEN, Eero; MÄKELÄ, Eetu; KAUPPINEN, Tomi; ALM, Olli; KURKI, Jussi; RUOTSALO, Tuukka; SEPPÄLÄ, Katri; TAKALA, Joeli; PUPUTTI, Kimmo; KUITTINEN, Heini; VILJANEN, Kim; TUOMINEN, Jouni; PALONEN, Tuomas; FROSTERUS, Matias; SINKKILÄ, Reetta; PAAKKARINEN, Panu; LAITIO, Joonas and NYBERG, Katariina: CultureSampo: A National Publication System of Cultural Heritage on the Semantic Web 2.0, in: Lora Aroyo; Paolo Traverso; Fabio Ciravegna; Philipp Cimiano; Tom Heath; Eero Hyvönen; Riichiro Mizoguchi; Eyal Oren; Marta Sabou and Elena Simperl (Editors) *The Semantic Web: Research and Applications*, vol. 5554 of *Lecture Notes in Computer Science*, Springer Berlin Heidelberg (2009), pp. 851–856, URL http://dx.doi.org/10.1007/978-3-642-02121-3_69

[Jen96] JENSEN, Kurt: *Coloured Petri nets (2nd ed.): basic concepts, analysis methods and practical use: volume 1*, Springer-Verlag, London, UK, UK (1996)

[Jia04] JIANG, Qingchun; ADAIKKALAVAN, Raman and CHAKRAVARTHY, Sharma: Estreams: Towards an Integrated Model for Event and Stream Processing, Tech. Rep. CSE-2004-3, Department of Computer Science and Engineering, University of Texas at Arlington (2004)

[Jia05] JIANG, Qingchun; ADAIKKALAVAN, Raman and CHAKRAVARTHY, Sharma: NFMi: An Inter-domain Network Fault Management System, in: *Proceedings of the 21st International Conference on Data Engineering*, ICDE '05, IEEE Computer Society, Washington, DC, USA, pp. 1036–1047, URL http://dx.doi.org/10.1109/ICDE.2005.94

[Jon01] JONES, Christopher B.; ALANI, Harith and TUDHOPE, Douglas: Geographical Information Retrieval with Ontologies of Place, in: *in Spatial Information Theory, LNCS 2205*, Springer (2001), pp. 322–335

[Kal08] KALLMAN, Robert; KIMURA, Hideaki; NATKINS, Jonathan; PAVLO, Andrew; RASIN, Alexander; ZDONIK, Stanley; JONES, Evan P. C.; MADDEN, Samuel; STONEBRAKER, Michael; ZHANG, Yang; HUGG, John and ABADI, Daniel J.: H-store: a high-performance, distributed main memory transaction processing system. *Proc. VLDB Endow.* (2008), vol. 1(2):pp. 1496–1499, URL http://dl.acm.org/citation.cfm?id=1454159.1454211

[Kem12] KEMPER, Alfons; NEUMANN, Thomas; FUNKE, Florian; LEIS, Viktor and MÜHE, Henrik: HyPer: Adapting Columnar Main-Memory Data Management for Transactional AND Query Processing. *IEEE Data Eng. Bull.* (2012), vol. 35(1):pp. 46–51, URL http://sites.computer.org/debull/A12mar/hyper.pdf

[Kol12] KOLLIA, Ilianna and GLIMM, Birte: Cost Based Query Ordering over OWL Ontologies, in: Philippe Cudré-Mauroux; Jeff Heflin; Evren Sirin; Tania Tudorache; Jérôme Euzenat; Manfred Hauswirth; JosianeXavier Parreira; Jim Hendler; Guus Schreiber; Abraham Bernstein and Eva Blomqvist (Editors) *The Semantic Web – ISWC 2012*, vol. 7649 of *Lecture Notes in Computer Science*, Springer Berlin Heidelberg (2012), pp. 231–246, URL http://dx.doi.org/10.1007/978-3-642-35176-1_15

[Kop97] KOPETZ, Hermann: *Real-Time Systems: Design Principles for Distributed Embedded Applications*, Kluwer Academic Publishers, Norwell, MA, USA, 1st edn. (1997)

[Kow74] KOWALSKI, Robert A.: Predicate Logic as Programming Language, in: *IFIP Congress*, North-Holland Pub Co, Amsterdam (1974), pp. 569–574

[Kow86] KOWALSKI, R and SERGOT, M: A Logic-based Calculus of Events. *New Gen. Comput.* (1986), vol. 4(1):pp. 67–95, URL http://dx.doi.org/10.1007/BF03037383

[Kow92] KOWALSKI, Robert: Database Updates in the Event Calculus. *J. Log. Program.* (1992), vol. 12(1-2):pp. 121–146, URL http://dx.doi.org/10.1016/0743-1066(92)90041-Z

[Koz06] KOZLENKOV, Alex; PENALOZA, Rafael; NIGAM, Vivek; ROYER, Loïc; DAWELBAIT, Gihan and SCHROEDER, Michael: Prova: rule-based java scripting for distributed web applications, in: *Proceedings of the 2006 international conference on Current Trends in Database Technology*, EDBT'06, Springer-Verlag, Berlin, Heidelberg, pp. 899–908, URL http://dx.doi.org/10.1007/11896548_68

[Kra04] KRAEMER, Juergen and SEEGER, Bernhard: PIPES: a public infrastructure for processing and exploring streams, in: *Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, SIGMOD '04, ACM, New York, NY, USA, pp. 925–926, URL http://doi.acm.org/10.1145/1007568.1007699

[Kra09] KRAEMER, Juergen and SEEGER, Bernhard: Semantics and Implementation of Continuous Sliding Window Queries over Data Streams. *ACM Trans. Database Syst.* (2009), vol. 34(1):pp. 4:1–4:49, URL http://doi.acm.org/10.1145/1508857.1508861

[Lac05] LACLAVIK, Michal: AgentOWL - Agents with OWL ontology models using JADE agent system and Jena (2005), URL http://eprints.agentlink.org/5561/;http://agentowl.sourceforge.net/

[Lag01] LAGOZE, Carl and HUNTER, Jane: The ABC Ontology and Model, in: *DCMI '01: Proceedings of the International Conference on Dublin Core and Metadata Applications 2001*, National Institute of Informatics, Tokyo, Japan, pp. 160–176

[Lal06] LALL, Ashwin; SEKAR, Vyas; OGIHARA, Mitsunori; XU, Jun and ZHANG, Hui: Data Streaming Algorithms for Estimating Entropy of Network Traffic, in: *Proceedings of the Joint International Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS '06/Performance '06, ACM, New York, NY, USA, pp. 145–156, URL http://doi.acm.org/10.1145/1140277.1140295

[Lam78] LAMPORT, Leslie: Time, clocks, and the ordering of events in a distributed system. *Commun. ACM* (1978), vol. 21(7):pp. 558–565, URL http://doi.acm.org/10.1145/359545.359563

[Lam94] LAMPORT, Leslie: The Temporal Logic of Actions. *ACM Trans. Program. Lang. Syst.* (1994), vol. 16(3):pp. 872–923, URL http://doi.acm.org/10.1145/177492.177726

[Lam10] LAMPOS, Vasileios and CRISTIANINI, Nello: Tracking the flu pandemic by monitoring the Social Web, in: *2nd IAPR Workshop on Cognitive Information Processing (CIP 2010)*, IEEE Press, pp. 411–416

[Lap92] LAPLANTE, Phillip A.: *Real-Time Systems Design and Analysis: An Engineer's Handbook*, IEEE Press, Piscataway, NJ, USA (1992)

[Li13] LI, Zheng; GE, Tingjian and CHEN, Cindy X.: ϵ-Matching: event processing over noisy sequences in real time, in: *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, SIGMOD '13, ACM, New York, NY, USA, pp. 601–612, URL http://doi.acm.org/10.1145/2463676.2463715

[Liu03] LIU, Ying and PLALE, Beth: Survey of Publish Subscribe Event Systems, Tech. Rep., Indiana University (2003), URL http://www.cs.indiana.edu/cgi-bin/techreports/TRNNN.cgi?trnum=TR574

[Loh91] LOHMAN, Guy M.; LINDSAY, Bruce; PIRAHESH, Hamid and SCHIEFER, K. Bernhard: Extensions to Starburst: objects, types, functions, and rules. *Commun. ACM* (1991), vol. 34(10):pp. 94–109, URL http://doi.acm.org/10.1145/125223.125266

[LP11] LE-PHUOC, Danh; DAO-TRAN, Minh; PARREIRA, Josiane Xavier and HAUSWIRTH, Manfred: A native and adaptive approach for unified processing of linked streams and linked data, in: *Proceedings of the 10th international conference on The semantic web - Volume Part I*, ISWC'11, Springer-Verlag, Berlin, Heidelberg, pp. 370–388, URL http://dl.acm.org/citation.cfm?id=2063016.2063041

[Luc01] LUCKHAM, David C.: *The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems*, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA (2001)

[Man92] MANNA, Zohar and PNUELI, Amir: *The Temporal Logic of Reactive and Concurrent Systems*, Springer-Verlag New York, Inc., New York, NY, USA (1992)

[Man09a] MANEGOLD, Stefan; KERSTEN, Martin L. and BONCZ, Peter: Database architecture evolution: mammals flourished long before dinosaurs became extinct. *Proc. VLDB Endow.* (2009), vol. 2(2):pp. 1648–1653, URL http://dl.acm.org/citation.cfm?id=1687553.1687618

[Man09b] MANI, Chandy K. and ROY, Schulte W.: *Event Processing: Designing IT Systems for Agile Companies*, McGraw-Hill, Inc., New York, NY, USA (2009)

[Mar11] MARGARA, Alessandro and CUGOLA, Gianpaolo: Processing flows of information: from data stream to complex event processing, in: *Proceedings of the 5th ACM international conference on Distributed event-based system*, DEBS '11, ACM, New York, NY, USA, pp. 359–360, URL http://doi.acm.org/10.1145/2002259.2002307

[Mar13] MARGARA, Alessandro; CUGOLA, Gianpaolo and TAMBURRELLI, Giordano: Towards Automated Rule Learning for Complex Event Processing (2013)

[Mar14a] MARGARA, Alessandro; CUGOLA, Gianpaolo and TAMBURRELLI, Giordano: Learning from the Past: Automated Rule Generation for Complex Event Processing, in: *Proceedings of the 8th ACM International Conference on Distributed Event-Based Systems*, DEBS '14, ACM, New York, NY, USA, pp. 47–58, URL http://doi.acm.org/10.1145/2611286.2611289

[Mar14b] MARGARA, Alessandro; URBANI, Jacopo; VAN HARMELEN, Frank and BAL, Henri: Streaming the Web: Reasoning over dynamic data. *Web Semantics: Science, Services and Agents on the World Wide Web* (2014), URL http://www.sciencedirect.com/science/article/pii/S1570826814000067

[McC69] MCCARTHY, John and HAYES, Patrick J.: Some Philosophical Problems from the Standpoint of Artificial Intelligence, in: B. Meltzer and D. Michie (Editors) *Machine Intelligence 4*, Edinburgh University Press (1969), pp. 463–502, reprinted in McC90

[McC89] MCCARTHY, Dennis and DAYAL, Umeshwar: The architecture of an active database management system. *SIGMOD Rec.* (1989), vol. 18(2):pp. 215–224, URL http://doi.acm.org/10.1145/66926.66946

[McL83] MCLEOD, A.I. and BELLHOUSE, D.R.: A convenient algorithm for drawing a simplerandom sample. *Applied Statistics* (1983), vol. 32(2):pp. 182–184

[Mea55] MEALY, G. H.: A method for synthesizing sequential circuits. *Bell System Technical Journal* (1955), vol. 34(5):pp. 1045–1079

[Men87] MENDELSON, Elliott: *Introduction to Mathematical Logic; (3rd Ed.)*, Wadsworth and Brooks/Cole Advanced Books & Software, Monterey, CA, USA (1987)

[Min69] MINSKY, Marvin L.: *Semantic Information Processing*, The MIT Press (1969)

[Min74] MINSKY, Marvin: A Framework for Representing Knowledge, Tech. Rep., Cambridge, MA, USA (1974)

[Mir87] MIRANKER, Daniel P.: TREAT: A Better Match Algorithm for AI Production System Matching, in: Kenneth D. Forbus and Howard E. Shrobe (Editors) *Proceedings of the 6th National Conference on Artificial Intelligence. Seattle, WA, July 1987*, Morgan Kaufmann (1987), pp. 42–47, URL http://www.aaai.org/Library/AAAI/1987/aaai87-008.php

[Mot95] MOTAKIS, Iakovos and ZANIOLO, Carlo: Composite Temporal Events in Active Databases: A Formal Semantics, in: James Clifford and Alexander Tuzhilin (Editors) *Recent Advances in Temporal Databases, Proceedings of the International Workshop on Temporal Databases, Zürich, Switzerland, 17-18 September 1995*, Workshops in Computing, Springer (1995), pp. 332–351

[Moz10] MOZAFARI, B. and ZANIOLO, C.: Optimal Load Shedding with Aggregates and Mining Queries, in: *Data Engineering (ICDE), 2010 IEEE 26th International Conference on*, pp. 76–88

[Müh06] MÜHL, Gero; FIEGE, Ludger and PIETZUCH, Peter R.: *Distributed event-based systems*, Springer (2006), URL http://dx.doi.org/10.1007/3-540-32653-7

[Mut03] MUTHUKRISHNAN, S.: Data streams: algorithms and applications, in: *Proceedings of the fourteenth annual ACM-SIAM symposium on Discrete algorithms*, SODA '03, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, pp. 413–413, URL http://dl.acm.org/citation.cfm?id=644108.644174

[Nat04] NATH, Suman; GIBBONS, Phillip B.; SESHAN, Srinivasan and ANDERSON, Zachary R.: Synopsis diffusion for robust aggregation in sensor networks, in: *Proceedings of the 2nd international conference on Embedded networked sensor systems*, SenSys '04, ACM, New York, NY, USA, pp. 250–262, URL http://doi.acm.org/10.1145/1031495.1031525

[Nit07] NITZSCHE, Jörg; WUTKE, Daniel and VAN LESSEN, Tammo: An Ontology for Executable Business Processes, in: Martin Hepp; Knut Hinkelmann; Dimitris Karagiannis; Rüdiger Klein and Nenad Stojanovic (Editors) *Proceedings of the Workshop on Semantic Business Process and Product Lifecycle Management (SBPM 2007) held in conjunction with the 3rd European Semantic Web Conference (ESWC 2007) Innsbruck, Austria, June 7, 2007*, Artikel in Tagungsband, Innsbruck: CEUR Workshop Proceedings (2007), pp. 52–63

[Par10] PARZYJEGLA, Helge; GRAFF, Daniel; SCHRÖTER, Arnd; RICHLING, Jan and MÜHL, Gero: From active data management to event-based systems and more, chap. Design and implementation of the Rebeca publish/subscribe middleware, Springer-Verlag, Berlin, Heidelberg (2010), pp. 124–140, URL http://dl.acm.org/citation.cfm?id=1985625.1985635

[Pas04] PASSIN, Thomas Thomas B.: *Explorer's Guide to the Semantic Web*, Manning Publications Co. (2004)

[Pas06a] PASCHKE, Adrian: ECA-LP / ECA-RuleML: A Homogeneous Event-Condition-Action Logic Programming Language, in: *RuleML-2006*, Athens, Georgia, USA

[Pas06b] PASCHKE, Adrian: ECA-RuleML: An Approach combining ECA Rules with temporal interval-based KR Event/Action Logics and Transactional Update Logics. *CoRR* (2006), vol. abs/cs/0610167, URL http://arxiv.org/abs/cs/0610167

[Pas08] PASCHKE, Adrian and BICHLER, Martin: Knowledge representation concepts for automated SLA management. *Decis. Support Syst.* (2008), vol. 46(1):pp. 187–205

[Pas10] PASCHKE, Adrian; KOZLENKOV, Alexander and BOLEY, Harold: A Homogeneous Reaction Rule Language for Complex Event Processing, vol. abs/1008.0823 (2010), URL http://arxiv.org/abs/1008.0823

[Pas11] PASCHKE, Adrian: Rules and Logic Programming for the Web, in: Polleres et al [Pol11] (2011), pp. 326–381, URL http://dx.doi.org/10.1007/978-3-642-23032-5_6

[Pat91] PATON, N. W. and DIAZ, O.: Object-oriented databases and frame-based system: comparison. *Inf. Softw. Technol.* (1991), vol. 33(5):pp. 357–365, URL http://dx.doi.org/10.1016/0950-5849(91)90105-K

[Pat99] PATON, Norman W. and DÍAZ, Oscar: Active database systems. *ACM Comput. Surv.* (1999), vol. 31(1):pp. 63–103, URL http://doi.acm.org/10.1145/311531.311623

[Ped08] PEDRINACI, Carlos; DOMINGUE, John and DE MEDEIROS, Ana Karla Alves: A Core Ontology for Business Process Analysis, in: Sean Bechhofer; Manfred Hauswirth; Jörg Hoffmann and Manolis Koubarakis (Editors) *The Semantic Web: Research and Applications, 5th European Semantic Web Conference, ESWC 2008, Tenerife, Canary Islands, Spain, June 1-5, 2008, Proceedings*, vol. 5021 of *Lecture Notes in Computer Science*, Springer, pp. 49–64, URL http://dx.doi.org/10.1007/978-3-540-68234-9_7

[Pei01] PEI, Jian; HAN, Jiawei; MORTAZAVI-ASL, B.; PINTO, H.; CHEN, Qiming; DAYAL, U. and HSU, Mei-Chun: PrefixSpan,: mining sequential patterns efficiently by prefix-projected pattern growth, in: *Data Engineering, 2001. Proceedings. 17th International Conference on*, pp. 215–224

[Pet03] PETROVIC, Milenko; BURCEA, Ioana and JACOBSEN, Hans-Arno: S-ToPSS: Semantic Toronto Publish/Subscribe System, in: *Proceedings of the 29th international conference on Very large data bases - Volume 29*, VLDB '2003, VLDB Endowment, pp. 1101–1104, URL http://dl.acm.org/citation.cfm?id=1315451.1315559

[Pie03a] PIETZUCH, Peter R. and BACON, Jean: Peer-to-peer overlay broker networks in an event-based middleware, in: *Proceedings of the 2nd international workshop on Distributed event-based systems*, DEBS '03, ACM, New York, NY, USA, pp. 1–8, URL http://doi.acm.org/10.1145/966618.966628

[Pie03b] PIETZUCH, Peter R.; SHAND, Brian and BACON, Jean: A framework for event composition in distributed systems, in: *Proceedings of the ACM/IFIP/USENIX 2003 International Conference on Middleware*, Middleware '03, Springer-Verlag New York, Inc., New York, NY, USA, pp. 62–82, URL http://dl.acm.org/citation.cfm?id=1515915.1515921

[Pol11] POLLERES, Axel; D'AMATO, Claudia; ARENAS, Marcelo; HANDSCHUH, Siegfried; KRONER, Paula; OSSOWSKI, Sascha and PATEL-SCHNEIDER, Peter F. (Editors): *Reasoning Web*, vol. 6848 of *Lecture Notes in Computer Science*, Springer (2011), URL http://dx.doi.org/10.1007/978-3-642-23032-5

[Pru08] PRUD'HOMMEAUX, Eric and SEABORNE, Andy: SPARQL Query Language for RDF, W3C Recommendation (2008)

[Qun12] QUNZHI, Zhou; YOGESH, Simmhan and VIKTOR, Prasanna: SCEPter: Semantic Complex Event Processing over End-to-End Data Flows, Tech. Rep., Department of Computer Science, Ming Hsieh Department of Electrical Engineering, University of Southern California (2012)

[Rai07] RAIMOND, Yves and ABDALLAH, Samer: The Event Ontology, http://motools.sourceforge.net/event/event.html (2007)

[Rei91] REITER, Raymond: Artificial Intelligence and Mathematical Theory of Computation, chap. The Frame Problem in Situation the Calculus: A Simple Solution (Sometimes) and a Completeness Result for Goal Regression, Academic Press Professional, Inc., San Diego, CA, USA (1991), pp. 359–380, URL http://dl.acm.org/citation.cfm?id=132218.132239

[Row01] ROWSTRON, Antony I. T.; KERMARREC, Anne-Marie; CASTRO, Miguel and DRUSCHEL, Peter: SCRIBE: The Design of a Large-Scale Event Notification Infrastructure, in: *Proceedings of the Third International COST264 Workshop on Networked Group Communication*, NGC '01, Springer-Verlag, London, UK, UK, pp. 30–43, URL http://dl.acm.org/citation.cfm?id=648089.747486

[Rud11] RUDOLPH, Sebastian: Foundations of Description Logics, in: Polleres et al [Pol11] (2011), pp. 76–136, URL http://dx.doi.org/10.1007/978-3-642-23032-5_2

[Ruo07] RUOTSALO, Tuukka and HYVÖNEN, Eero: A Method for Determining Ontology-Based Semantic Relevance, in: Roland Wagner; Norman Revell and Günther Pernul (Editors) *Database and Expert Systems Applications, 18th International Conference, DEXA 2007, Regensburg, Germany, September 3-7, 2007, Proceedings*, vol. 4653 of *Lecture Notes in Computer Science*, Springer, pp. 680–688, URL http://dx.doi.org/10.1007/978-3-540-74469-6_66

[Sch08] SCHMIDT, Kay-Uwe; ANICIC, Darko and STÜHMER, Roland: Event-driven Reactivity: A Survey and Requirements Analysis, in: *SBPM2008: 3rd international Workshop on Semantic Business Process Management in conjunction with the 5th European Semantic Web Conference (ESWC'08)*, CEUR Workshop Proceedings (CEUR-WS.org, ISSN 1613-0073), URL http://sbpm2008.fzi.de/paper/paper7.pdf

[Sch09] SCHERP, Ansgar; FRANZ, Thomas; SAATHOFF, Carsten and STAAB, Steffen: F–a model of events based on the foundational ontology dolce+DnS ultralight, in: *K-CAP '09: Proceedings of the fifth international conference on Knowledge capture*, ACM, New York, NY, USA, pp. 137–144

[Sch13] SCHAPRANOW, Matthieu-P. and PLATTNER, Hasso: *In-Memory Technology Enables History-Based Access Control for RFID-Aided Supply Chains* (2013), the Secure Information Society: Ethical, Legal and Political Challenges, pp. 187-213, ISBN13 978-1-4471-4762-6

[Seq09] SEQUEDA, Juan and CORCHO, Óscar: Linked Stream Data: A Position Paper, in: Kerry Taylor and David De Roure (Editors) *Proceedings of the 2nd International Workshop on Semantic Sensor Networks (SSN09), collocated with the 8th International Semantic Web Conference (ISWC-2009), Washington DC, USA, October 26, 2009*, vol. 522 of *CEUR Workshop Proceedings*, CEUR-WS.org (2009), pp. 148–157, URL http://ceur-ws.org/Vol-522/p13.pdf

[SF08] SUAREZ-FIGUEROA, Mari Carmen and GOMEZ-PEREZ, Asuncion: NeOn Methodology: Scenarios for Building Networks of Ontologies, in: Knowledge Management Knowledge Patterns EKAW 16th International Conference on Knowledge Engineering (Editor) *Proceedings of the International Conference on SOFTWARE, SERVICES and SEMANTIC TECHNOLOGIES* (2008)

[Sha09] SHAW, Ryan; TRONCY, Raphaël and HARDMAN, Lynda: LODE: Linking Open Descriptions of Events (2009), (Paper 2009-036)

[Sim09] SIMPERL, Elena: Reusing Ontologies on the Semantic Web: A Feasibility Study. *Data Knowl. Eng.* (2009), vol. 68(10):pp. 905–925, URL http://dx.doi.org/10.1016/j.datak.2009.02.002

[Sir06] SIRIN, Evren and PARSIA, Bijan: Optimizations for answering conjunctive abox queries: First results, in: *In Proc. of the Int. Description Logics Workshop (DL)* (2006)

[SM09] SCHULTZ-MØLLER, Nicholas Poul; MIGLIAVACCA, Matteo and PIETZUCH, Peter: Distributed complex event processing with query rewriting, in: *Proceedings of the Third ACM International Conference on Distributed Event-Based Systems*, DEBS '09, ACM, New York, NY, USA, pp. 4:1–4:12, URL http://doi.acm.org/10.1145/1619258.1619264

[spa08] SPARQL Protocol for RDF, W3C Recommendation (2008), URL http://www.w3.org/TR/rdf-sparql-protocol/

[Ste12] STEWART, Avaré and DIAZ, Ernesto: Epidemic intelligence: for the crowd, by the crowd (Tutorial), in: *Proceedings of the 12th international conference on Web Engineering*, ICWE'12, Springer-Verlag, Berlin, Heidelberg, pp. 504–505, URL http://dx.doi.org/10.1007/978-3-642-31753-8_55

[Sto88] STONEBRAKER, Michael; HANSON, Eric N. and POTAMIANOS, Spyros: The POSTGRES Rule Manager. *IEEE Trans. Softw. Eng.* (1988), vol. 14(7):pp. 897–907, URL http://dx.doi.org/10.1109/32.42733

[Sto91] STONEBRAKER, Michael and KEMNITZ, Greg: The POSTGRES next generation database management system. *Commun. ACM* (1991), vol. 34(10):pp. 78–92, URL http://doi.acm.org/10.1145/125223.125262

[Sto07] STONEBRAKER, Michael; MADDEN, Samuel; ABADI, Daniel J.; HARIZOPOULOS, Stavros; HACHEM, Nabil and HELLAND, Pat: The end of an architectural era: (it's time for a complete rewrite), in: *Proceedings of the 33rd international conference on Very large data bases*, VLDB '07, VLDB Endowment, pp. 1150–1160, URL http://dl.acm.org/citation.cfm?id=1325851.1325981

[Sto13] STOWELL, Dan and PLUMBLEY, Mark D.: Segregating Event Streams and Noise with a Markov Renewal Process Model. *J. Mach. Learn. Res.* (2013), vol. 14(1):pp. 2213–2238, URL http://dl.acm.org/citation.cfm?id=2567709.2567732

[Str98] STROM, Robert E.; BANAVAR, Guruduth; CHANDRA, Tushar Deepak; KAPLAN, Marc A.; MILLER, Kevan; MUKHERJEE, Bodhi; STURMAN, Daniel C. and WARD, Michael: Gryphon: An Information Flow Based Approach to Message Brokering. *CoRR* (1998), vol. cs.DC/9810019, URL http://arxiv.org/abs/cs.DC/9810019

[Stu98] STUDER, Rudi; BENJAMINS, V. Richard and FENSEL, Dieter: Knowledge engineering: principles and methods. *Data Knowl. Eng.* (1998), vol. 25(1-2):pp. 161–197

[Stu05] STUCKENSCHMIDT, Heiner and HARMELEN, Frank van: *Information Sharing on the Semantic Web*, Springer (2005)

[Stu07] STUDER, Rudi; GRIMM, Stephan and ABECKER, Andreas: *Semantic Web Services: Concepts, Technologies, and Applications*, Springer (2007)

[Stu09] STUCKENSCHMIDT, Heiner; PARENT, Christine and SPACCAPIETRA, Stefano (Editors): *Modular Ontologies: Concepts, Theories and Techniques for Knowledge Modularization*, vol. 5445 of *Lecture Notes in Computer Science*, Springer, Berlin (2009)

[Sur03] SURE, York; STAAB, Steffen; STUDER, Rudi and GMBH, Ontoprise: On-To-Knowledge Methodology (OTKM), in: *Handbook on Ontologies, International Handbooks on Information Systems*, Springer (2003), pp. 117–132

[Tay09] TAYLOR, H.; YOCHEM, A.; PHILLIPS, L. and MARTINEZ, F.: *Event-Driven Architecture: How SOA Enables the Real-Time Enterprise*, Addison-Wesley (2009)

[Tey09a] TEYMOURIAN, Kia and PASCHKE, Adrian: Semantic Rule-Based Complex Event Processing, in: *RuleML 2009: Proceedings of the International RuleML Symposium on Rule Interchange and Applications*

[Tey09b] TEYMOURIAN, Kia and PASCHKE, Adrian: Towards semantic event processing, in: *DEBS '09: Proceedings of the Third ACM International Conference on Distributed Event-Based Systems*, ACM, New York, NY, USA, pp. 1–2

[Tey09c] TEYMOURIAN, Kia; STREIBEL, Olga; PASCHKE, Adrian; ALNEMR, Rehab and MEINEL, Christoph: Towards Semantic Event-Driven Systems, in: Khaldoun Al Agha; Mohamad Badra and Gregory B. Newby (Editors) *NTMS 2009, 3rd International Conference on New Technologies, Mobility and Security, 20-23 December 2009, Cairo, Egypt*, IEEE (2009), pp. 1–6, URL http://dx.doi.org/10.1109/NTMS.2009.5384713

[Tey10a] TEYMOURIAN, Kia; COSKUN, Gökhan and PASCHKE, Adrian: Modular Upper-Level Ontologies for Semantic Complex Event Processing, in: Oliver Kutz; Joana Hois; Jie Bao and Bernardo Cuenca Grau (Editors) *Modular Ontologies - Proceedings of the Fourth International Workshop, WoMO 2010, Toronto, ON, Canada, May 11, 2010*, vol. 210 of *Frontiers in Artificial Intelligence and Applications*, IOS Press (2010), pp. 81–93, URL http://dx.doi.org10.3233/978-1-60750-544-0-81

[Tey10b] TEYMOURIAN, Kia and PASCHKE, Adrian: Enabling knowledge-based complex event processing, in: *Proceedings of the 2010 EDBT/ICDT Workshops, Lausanne, Switzerland, March 22-26, 2010*, ACM International Conference Proceeding Series, ACM, URL http://doi.acm.org/10.1145/1754239.1754281

[Tey11a] TEYMOURIAN, Kia; ROHDE, Malte; HASSAN-HAIDAR, Ahmad and PASCHKE, Adrian: Processing of Complex Stock Market Events Using Background Knowledge, in: *Proceedings of the Workhop on Detection, Representation, and Exploitation of Events in the Semantic Web (DeRiVE 2011)*, Bonn, Germany, URL http://ceur-ws.org/Vol-779/

[Tey11b] TEYMOURIAN, Kia; ROHDE, Malte and PASCHKE, Adrian: Processing of Complex Stock Market Events Using Background Knowledge, in: *Proceedings of the 5th International RuleML2011@BRF Challenge, co-located with the 5th International Rule Symposium*, Fort Lauderdale, Florida, USA, URL http://ceur-ws.org/Vol-799/

[Tey12a] TEYMOURIAN, Kia and PASCHKE, Adrian: Semantic Processing of Sensor Event Stream by Using External Knowledge Bases (Short Paper), in: Cory A. Henson; Kerry Taylor and Óscar Corcho (Editors) *Proceedings of the 5th International Workshop on Semantic Sensor Networks, SSN12, Boston, Massachusetts, USA, November 12, 2012*, vol. 904 of *CEUR Workshop Proceedings*, CEUR-WS.org (2012), pp. 121–126, URL http://ceur-ws.org/Vol-904/paper8.pdf

[Tey12b] TEYMOURIAN, Kia; ROHDE, Malte and PASCHKE, Adrian: Fusion of Background Knowledge and Streams of Events, in: *Proceedings of the 6th ACM International Conference on Distributed Event-Based Systems*, DEBS '12, ACM, New York, NY, USA, pp. 302–313, URL http://doi.acm.org/10.1145/2335484.2335517

[Tey12c] TEYMOURIAN, Kia; ROHDE, Malte and PASCHKE, Adrian: Knowledge-based processing of complex stock market events, in: *15th International Conference on Extending Database Technology, EDBT '12, Berlin, Germany, March 27-30, 2012, Proceedings*, ACM, pp. 594–597, URL http://doi.acm.org/10.1145/2247596.2247674

[Tey14] TEYMOURIAN, Kia and PASCHKE, Adrian: Plan-Based Semantic Enrichment of Event Streams, in: *11th Extended Semantic Web Conference (ESWC 2014)*, Crete, Greece, URL http://2014.eswc-conferences.org/sites/default/files/ESWC2014_submission_68_to_be_published_on_the_ESWC_website.pdf

[Tur48] TURING, Alan Mathison: Intelligent Machinery, Report, National Physical Laboratory, Teddington, UK (1948)

[Vai07] VAISHNAVI, Vijay K. and KUECHLER, William, Jr.: *Design Science Research Methods and Patterns: Innovating Information and Communication Technology*, Auerbach Publications, Boston, MA, USA, 1st edn. (2007)

[Val09] VALLE, Emanuele Della; CERI, Stefano; HARMELEN, Frank van and FENSEL, Dieter: It's a Streaming World! Reasoning upon Rapidly Changing Information. *IEEE Intelligent Systems* (2009), vol. 24(6):pp. 83–89, URL http://dx.doi.org/10.1109/MIS.2009.125

[vdA11] VAN DER AALST, Wil M. P.: *Process Mining: Discovery, Conformance and Enhancement of Business Processes*, Springer Publishing Company, Incorporated, 1st edn. (2011)

[Ven10] VENABLE, John: Design Science Research Post Hevner et al.: Criteria, Standards, Guidelines, and Expectations, in: Robert Winter; J. Zhao and Stephan Aier (Editors) *Global Perspectives on Design Science Research*, vol. 6105 of *Lecture Notes in Computer Science*, Springer Berlin (2010), pp. 109–123, URL http://dx.doi.org/10.1007/978-3-642-13335-0_8

[vH07] VAN HARMELEN, Frank; VAN HARMELEN, Frank; LIFSCHITZ, Vladimir and PORTER, Bruce: *Handbook of Knowledge Representation*, Elsevier Science, San Diego, USA (2007)

[Vit85] VITTER, Jeffrey S.: Random sampling with a reservoir. *ACM Trans. Math. Softw.* (1985), vol. 11(1):pp. 37–57, URL http://doi.acm.org/10.1145/3147.3165

[Vog08] VOGELS, Werner: Eventually Consistent. *Queue* (2008), vol. 6(6):pp. 14–19, URL http://doi.acm.org/10.1145/1466443.1466448

[Voi11] VOISARD, Agnès and ZIEKOW, Holger: ARCHITECT: A layered framework for classifying technologies of event-based systems. *Inf. Syst.* (2011), vol. 36(6):pp. 937–957, URL http://dx.doi.org/10.1016/j.is.2011.03.006

[W3C04] OWL Web Ontology Language Overview, W3C Recommendation (2004), URL http://www.w3.org/TR/owl-features/

[Wag05] WAGNER, Arno and PLATTNER, Bernhard: Entropy Based Worm and Anomaly Detection in Fast IP Networks, in: *Proceedings of the 14th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprise*, WETICE '05, IEEE Computer Society, Washington, DC, USA, pp. 172–177, URL http://dx.doi.org/10.1109/WETICE.2005.35

[Wal07a] WALTON, Christopher D.: *Agency and the Semantic Web*, Oxford University Press, New York, NY, USA (2007)

[Wal07b] WALZER, Karen; SCHILL, Alexander and LÖSER, Alexander: Temporal constraints for rule-based event processing, in: *Proceedings of the ACM first Ph.D. workshop in CIKM*, PIKM '07, ACM, New York, NY, USA, pp. 93–100, URL http://doi.acm.org/10.1145/1316874.1316890

[Wan11] WANG, Di; RUNDENSTEINER, Elke A. and ELLISON, Richard T., III: Active Complex Event Processing over Event Streams. *Proc. VLDB Endow.* (2011), vol. 4(10):pp. 634–645, URL http://dl.acm.org/citation.cfm?id=2021017.2021021

[Was08] WASSERKRUG, Segev; GAL, Avigdor; ETZION, Opher and TURCHIN, Yulia: Complex event processing over uncertain data, in: *Proceedings of the second international conference on Distributed event-based systems*, DEBS '08, ACM, New York, NY, USA, pp. 253–264, URL http://doi.acm.org/10.1145/1385989.1386022

[Was12] WASSERKRUG, Segev; GAL, Avigdor; ETZION, Opher and TURCHIN, Yulia: Efficient Processing of Uncertain Events in Rule-Based Systems. *IEEE Trans. Knowl. Data Eng.* (2012), vol. 24(1):pp. 45–58, URL http://doi.ieeecomputersociety.org/10.1109/TKDE.2010.204

[Wie10] WIENHOFEN, Leendert W. M. and TOUSSAINT, Pieter J.: Enriching events to support hospital care, in: *Proceedings of the 7th Middleware Doctoral Symposium*, MDS '10, ACM, New York, NY, USA, pp. 26–30, URL http://doi.acm.org/10.1145/1891748.1891753

[Wil09] WILLHALM, Thomas; POPOVICI, Nicolae; BOSHMAF, Yazan; PLATTNER, Hasso; ZEIER, Alexander and SCHAFFNER, Jan: SIMD-scan: ultra fast in-memory table scan using on-chip vector processing units. *Proc. VLDB Endow.* (2009), vol. 2(1):pp. 385–394, URL http://dl.acm.org/citation.cfm?id=1687627.1687671

[Wor04] WORBOYS, Michael F. and HORNSBY, Kathleen: From Objects to Events: GEM, the Geospatial Event Model, in: Egenhofer et al [Ege04] (2004), pp. 327–344, URL http://springerlink.metapress.com/openurl.asp?genre=article{&}issn=0302-9743{&}volume=3234{&}spage=327

[Xia04] XIAORONG, Scott Christley; XIANG, Xiaorong and MADEY, Greg: An Ontology For Agent-Based Modeling And Simulation (2004), URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.60.2685;http://www.nd.edu/~schristl/papers/ontology_agent2004.pdf

[Xu05] XU, Kuai; ZHANG, Zhi-Li and BHATTACHARYYA, Supratik: Profiling Internet Backbone Traffic: Behavior Models and Applications, in: *Proceedings of the 2005 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM '05, ACM, New York, NY, USA, pp. 169–180, URL http://doi.acm.org/10.1145/1080091.1080112

[Yan97] YANG, Jian; KARLAPALEM, Kamalakar and LI, Qing: Algorithms for Materialized View Design in Data Warehousing Environment, in: *Proceedings of the 23rd International Conference on Very Large Data Bases*, VLDB '97, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, pp. 136–145, URL http://dl.acm.org/citation.cfm?id=645923.673657

[Yan03] YAN, Xifeng; HAN, Jiawei and AFSHAR, Ramin: CloSpan: Mining Closed Sequential Patterns in Large Datasets, in: *SIAM Int. Conf. Data Mining (SDM'03)*, SIAM (2003), pp. 166–177

[Zak01] ZAKI, Mohammed J.: SPADE: An Efficient Algorithm for Mining Frequent Sequences. *Mach. Learn.* (2001), vol. 42(1-2):pp. 31–60, URL http://dx.doi.org/10.1023/A:1007652502315

[Zha11] ZHANG, Chunxia; CAO, Cungen; SUI, Yuefei and WU, Xindong: A Chinese time ontology for the Semantic Web. *Knowledge-Based Systems* (2011), vol. 24(7):pp. 1057 – 1074, URL http://www.sciencedirect.com/science/article/pii/S0950705111000876

[Zha12] ZHANG, Ying; DUC, Pham Minh; GROFFEN, Fabian; LIAROU, Erietta; BONCZ, Peter; KERSTEN, Martin; CALBIMONTE, Jean-Paul and CORCHO, Oscar: Benchmarking RDF Storage Engines. Deliverable D1.2, Tech. Rep., PlanetData FP7 (2012)

[Zhu01] ZHUANG, Shelley Q.; ZHAO, Ben Y.; JOSEPH, Anthony D.; KATZ, Randy H. and KUBIATOWICZ, John D.: Bayeux: an architecture for scalable and fault-tolerant wide-area data dissemination, in: *Proceedings of the 11th international workshop on Network and operating systems support for digital audio and video*, NOSSDAV '01, ACM, New York, NY, USA, pp. 11–20, URL http://doi.acm.org/10.1145/378344.378347