

Situation Management with Complex Event Processing

Zur Erlangung des akademischen Grades
Doktor der Ingenieurwissenschaften
der Fakultät für Wirtschaftswissenschaften
Karlsruher Institut für Technologie (KIT)

genehmigte
Dissertation
von

Dipl.-Inform. Suad Sejdovic

Tag der mündlichen Prüfung: 05. September 2018
Referent: Prof. Dr. York Sure-Vetter
Korreferent: Prof. Dr. Thomas Setzer

Abstract

With the broader dissemination of digital technologies, visionary concepts like the Internet of Things also affect an increasing number of use cases with interfaces to humans, e.g. manufacturing environments with technical operators monitoring the processes. This leads to additional challenges, as besides the technical issues also human aspects have to be considered for a successful implementation of strategic initiatives like Industrie 4.0. From a technical perspective, complex event processing has proven itself in practice to be capable of integrating and analyzing huge amounts of heterogeneous data and establishing a basic level of situation awareness by detecting situations of interests. Whereas this reactive nature of complex event processing systems may be sufficient for machine-to-machine use cases, the new characteristic of application fields with humans remaining in the control loop leads to an increasing action distance and delayed reactions. Taking human aspects into consideration leads to new requirements, with transparency and comprehensibility of the processing of events being the most important ones. Improving the comprehensibility of complex event processing and extending its capabilities towards an effective support of human operators allows tackling technical and non-technical challenges at the same time.

The main contribution of this thesis answers the question of how to evolve state-of-the-art complex event processing from its reactive nature towards a transparent and holistic situation management system. The goal is to improve the interaction among systems and humans in use cases with interfaces between both worlds. Realizing a holistic situation management requires three missing capabilities to be introduced by the contributions of this thesis: First, based on the achieved transparency, the retrospective analysis of situations is enabled by collecting information related to a situation's occurrence and development. Therefore, CEP engine-specific situation descriptions are transformed into a common model, allowing the automatic decomposition of the underlying patterns to derive partial patterns describing the intermediate states of processing. Second, by introducing the psychological model of situation awareness into complex event processing, human aspects of information processing are taken into consideration and introduced into the complex event processing paradigm. Based on this model, an extended situation life-cycle and transition method are derived. The introduced concepts and methods allow the implementation of the controlling function of situation management and enable the effective acquisition and maintenance of situation awareness for human operators to purposefully direct their attention towards upcoming situations. Finally, completing the set of capabilities for situation management, an approach is presented to support the generation and integration of prediction models for predictive situation management. Therefore, methods are introduced to automatically label and extract relevant data for the generation of prediction models and to enable the embedding of the resulting models for an automatic evaluation and execution. The contributions are introduced, applied and evaluated along a scenario from the manufacturing domain.

Contents

I	Introduction	1
1	Introduction	3
1.1	Research Questions	4
1.2	Research Methodology	6
1.3	Contributions of this Thesis	6
1.4	Previous Publications	7
1.5	Structure of this Thesis	8
2	Motivation	11
2.1	Real-Time Processing with Complex Event Processing	11
2.2	Running Example	12
2.3	Problem Statements	15
2.3.1	Transparency	15
2.3.2	Situation Awareness	16
2.3.3	Proactivity	17
2.3.4	Conclusion	18
II	Preliminaries	19
3	Foundations	21
3.1	Stream Processing	21
3.2	Complex Event Processing	22
3.2.1	Events	22
3.2.2	Event Streams	24
3.3	Event Processing Networks	25
3.3.1	Event Producer	26
3.3.2	Event Channel	26
3.3.3	Event Processing Agent	26
3.3.4	Event Consumer	28
3.4	Event Patterns	28
3.4.1	Basic Event Patterns	28
3.4.2	Dimensional Patterns	29
3.4.3	Windows	30
3.5	Complex Event Processing Engines	30
4	State of the Art	33
4.1	Transparency: Partial Pattern Fulfillment	33
4.2	Situation-Aware Systems	36
4.3	Proactive Event-Driven Computing	37
4.4	Conclusion	39

III Main Part	41
5 Requirements	43
5.1 Requirements Elicitation Process	43
5.2 Capability Requirements	43
5.3 System-related requirements	46
6 Transparent Complex Event Processing	49
6.1 Transparency	49
6.1.1 Transparency and Complex Event Processing	50
6.1.2 Partial Pattern Fulfillment	51
6.2 Observing Situations of Interest	52
6.2.1 Model for Situations of Interest	52
6.2.2 Decomposing Queries	59
6.3 Analyzing Situations of Interest	63
6.3.1 (Sub-)Query Analysis	64
6.3.2 Event Object Analysis	65
6.4 Architecture	65
6.4.1 Generating Sub-Queries	67
6.4.2 Analyzing Situations	68
6.5 Running Example	69
6.6 Summary	73
7 Situation-Aware Complex Event Processing	75
7.1 Situation Awareness	75
7.2 Situation-aware Complex Event Processing	79
7.2.1 Extended Situation Lifecycle	80
7.2.2 Transitions in the Situation Life-Cycle	81
7.3 Development of Situation-Aware CEP Applications	82
7.4 Architecture	86
7.5 Running Example	87
7.6 Summary	90
8 Predictive Complex Event Processing	91
8.1 Predictive Analytics	91
8.2 Event Trace Labeling and Event Export Selection	94
8.3 Continuous Precision Monitoring	97
8.4 Architecture	98
8.5 Running Example	100
8.6 Summary	103
IV Finale	105
9 Evaluation	107
9.1 Evaluation Strategy	107
9.2 Conceptual Investigation	109
9.2.1 Fulfillment of Requirements	109

9.2.2	Discussion	112
9.3	User Study	112
9.3.1	Design	112
9.3.2	Results	114
9.3.3	Discussion	118
9.4	Performance Experiments	119
9.4.1	Setup	119
9.4.2	Results	120
9.4.3	Discussion	126
10	Conclusion	129
10.1	Summary	129
10.2	Significance of the Results	132
10.3	Outlook	132
A	Appendix A	135
A.1	Survey Results	135
A.2	Questionnaire	136

Part I

Introduction

1

Introduction

The progressive infiltration of every day life, private and industrial, with digital systems leads to more and more data to be generated. New and affordable sensor technology allows more extensive capturing of relevant information than ever before. This development has led to paradigms like the Internet of Things (IoT), that envisions a pervasive network of connected devices through standard communication protocols [Wortmann and Flüchter, 2015]. The goal is the collection and integration of data from multiple sources to gain new insights into processes to identify optimization potentials. Due to its promising opportunities within the domain of industrial production, the IoT paradigm was adopted and the Industrial Internet of Things (IIoT) aims at the interconnection of all related machines and their environment [Jeschke et al., 2017]. Typical IIoT applications target the reduction of machine down-times, e.g. through optimizing maintenance operations. The implementation of IIoT applications leads to an increase in Machine-to-Machine (M2M) interactions generating even more data.

The processing of this data poses a challenge, but in return allows valuable insights. In 2001, Laney [Laney, 2001] identified trends in several dimensions that issued a challenge to the effective processing of data in the future: volume, velocity and variety. Volume refers to the growing size of data sets, whereas velocity indicates the increasing speed of data generation and consumption. Variety describes the issue of a rising amount of heterogeneous data types and sources. This perspective on data and its processing is the initiator of the Big Data era. The posed challenges by Laney are still valid nowadays, even if many works have tackled them in the past, as through the broad dissemination of new sensor technology and high-throughput instruments the amount of data keeps growing and data sets increase at exponential rate [Chen and Zhang, 2014]. The impact of Big Data led to a fourth scientific paradigm in research: data-intensive science is complementing the previous paradigms, namely empirical science, theoretical science and computational science [Bell et al., 2009]. In the entrepreneurial context, Big Data also represents a highly relevant topic, as it has significant implications on business processes and business models [Schroeck et al., 2012]. According to Chen [Chen and Zhang, 2014], the introduced dimensions also significantly impact knowledge discovery processes, like the Cross-industry Standard Process for Data Mining (CRISP-DM). Implementing those processes in real-time for Big Data represents one of the most significant competitive advantages of the modern age [Balboni and Cook, 2012].

With the broader dissemination of digital technologies, an increasing number of use cases is identified and addressed that has interfaces to users, e.g. operators and domain experts. The black-box mode of operation of current monitoring systems has the potential to reduce the comprehensibility of results for human operators. This incomprehension might lead to a lack of trust and bring recommendations of the system into question, if automatic reactions are impossible or unwanted. Consequently, systems are required that transparently process the data and offer an effective situation management that supports human operators with situation awareness and allows a timely reaction with the goal to achieve proactivity.

One of the first domains to experience and approach the challenges of Big Data and situation management was the financial industry. It required the processing of an increasing amount of data to detect promising business situations in real-time and automatically react to it. Besides many isolated applications for individual problems,

Complex Event Processing (CEP) evolved as the technology to tackle the mentioned technical challenges for arbitrary use cases and allow real-time business intelligence [Hang and Fong, 2010]. It enables the integration of heterogeneous data from multiple sources to continuously process a high-velocity stream of data. In event processing systems, data elements are considered as events, i.e. notifications of happenings within a specific domain [Etzion and Niblett, 2011]. The primary goal of CEP systems is the detection of situations of interest, i.e. defined states of the environment indicating a higher-level happening in the monitored surrounding. The internal processing logic is implemented by an event processing network, which consists of processing nodes, called event processing agents, that implement filtering, transformation and pattern detect functionality. One main driver of the success of CEP systems is their ability to handle large amounts of events by defining relationships among them, e.g. sequences, and avoiding complex world models [Engel and Etzion, 2011]. The application logic is described by means of a declarative language, e.g. SQL-based, and does not require programming skills. This gives non-technical domain experts access to describe their information needs. Additionally, action statements are defined to be automatically triggered once the situation is detected.

In brief, event processing systems are capable of tackling the technical challenges of Big Data and are already used to handle huge amounts of heterogeneous data. From a pure technical perspective, the existing systems meet the arising challenges of (I)IoT applications. The previously mentioned aspects of human interfaces are left out in this technical consideration. As soon as human operators get involved in the real-time monitoring of complex technical environments and their management, new dimensions need to be tackled. The incomprehension of the mode of operation and the resulting recommendations might lead to a delayed reaction towards a situation, reducing the added value of real-time systems. Although complex event processing has all attributes to take a central role in the era of Big Data, its intransparency is seen as a limitation for its further enhancement towards a holistic situation management that supports both worlds: machine-to-machine interactions and the involvement of humans.

Within the course of this thesis, the enhancement of complex event processing is promoted on basis of human aspects. Establishing a holistic situation management for operators pursues three targets. First, increasing the transparency during run-time to query the state of processing at any time and additionally to capture situation statistics for monitoring and resolving situations. Second, extending the level of situational awareness in CEP systems to facilitate a systematic direction of an operator's attention. Third, enabling projections into the future to minimize the consequences of delays due to manual reactions in human-in-the-loop environments.

1.1 Research Questions

The primary goal of this thesis is the incorporation of aspects of human information processing into the handling of Big Data with complex event processing. This enhancement of CEP is related to technical challenges which have to be solved. The basic requirement for further extensions is establishing transparency during the processing. Based on the new insights, CEP systems are extended with capabilities to further support the situation management.

The principal research question focused on in this thesis is:

How can complex event processing be further evolved to a holistic situation management system?

The research questions comprises three main aspects: situation awareness, situation management and event stream processing. Situation awareness as a recognized psychological model was described in detail by Endsley [Endsley, 1995]. It defines the phenomenon and its role in dynamic decision making. It is based on the human factors that influence the processing of information in a specific environment and describes the mental process to establish a state of knowledge. The successful assessment of situations for human operators is affected by

information overload. Situation management aims to support operators with their monitoring tasks and ease the situation assessment process by better explaining occurred situations. Situation awareness builds the basis for situation management, which encompasses three relevant dimensions to successfully handle situations: the current state of a situation, its past and future development. Achieving an initial level of situation awareness is key for the success of complex event processing systems. By describing complex relationships between events, CEP systems are capable to identify occurring situations and trigger reactions. The complex event processing paradigm tackles many of the technological challenges that emerge with Big Data, e.g. its volume, velocity and variety, but lacks on the other side in taking human factors into consideration in its conceptual design. Its situation awareness arises implicitly from its mode of operation and is not grounded on the psychological model behind it.

To sum it up, the principal research question aims at the merger of both domains with the goal to retain the technical advantages of complex event processing and further evolve its conceptual design to a human centered processing. The principal question is broken down into three sub-questions, each targeting different dimensions of the main research question. Each sub-question is motivated in detail in Section 2.3.

Research Question 1 (Transparency). *How can transparency be introduced in complex event processing during run-time?*

The first research question deals with a weakness of current complex event processing engines: their black-box mode of operation while detecting defined situations. Answering this question requires analyzing the current modus operandi and deriving a concept of transparency for CEP systems. A holistic situation management system requires deep knowledge about situations of interest and their occurrence. Therefore, specific characteristics of the system have to become observable to allow the enhancement of further capabilities towards situation management. As several CEP systems are in use nowadays, the main challenge is the design of a generalizable solution for transparency. Transparent complex event processing is the fundamental requirement to enable a holistic situation management. This research question is answered in Chapter 6 and evaluated in Chapter 9.

Research Question 2 (Situation Awareness). *How can situation-aware complex event processing be established?*

The second research question covers the challenge how complex event processing has to be extended conceptually to support the psychological model of situation awareness to its full extent. Situation awareness is an elementary prerequisite for situation management, as the successful situation assessment represents the most important capability of it. Therefore, the implicit situation life-cycle of current CEP applications has to be extended to incorporate all levels of situation awareness and allow the tracking of situations through it. As various domains rely on CEP systems, the extended life-cycle has to abstract from domain specific states and be independent of the use case. Thus, a generalizable solution is needed that supports the tracking by abstracting from the specific situation and introducing abstract intermediate states suitable for different domains, like manufacturing and finance. Nevertheless, the solution shall allow some degrees of freedom to enable the incorporation of use case specific knowledge, e.g. differing criticality levels. This research question is answered in Chapter 7 and evaluated in Chapter 9.

Research Question 3 (Prediction Support). *How can situation-aware complex event processing be used to support projections into the future?*

The third research question deals with the highest level of situation awareness and a mandatory capability of a situation management system: the projection of a situation's development into the future. As stated earlier, a holistic situation management relies on all three levels of situation awareness. As prediction models are highly dependent on the use case at hand, no general solution is derivable that fits all problems. Nevertheless, by leveraging the insights from transparent and situation-aware complex event processing, the generation of a prediction model is

supportable. The goal for this research question lies in assisting the training of a prediction model with the collection and extraction of relevant data and integrating the manually trained model in the processing workflow. This research question is answered in Chapter 8 and evaluated in Chapter 9.

1.2 Research Methodology

The underlying research methodology to tackle the presented research questions follows the design science paradigm for information systems research [Hevner et al., 2004]. The conceptual framework follows an approach where business needs are derived from an environment of interest, e.g. people, organizations and technologies. Information systems research develops IT artifacts and theories to meet these business needs by applying existing knowledge from a knowledge base that comprises existing foundations and methodologies [Hevner et al., 2004]. The term "artifact" in this context stands for software, hardware, methods and models used to fulfill the business needs. Design science as a systematic problem solving process consists of seven guidelines shown in Table 1.1.

ID	Guideline
G1	Design as an Artifact
G2	Problem Relevance
G3	Design Evaluation
G4	Research Contributions
G5	Research Rigor
G6	Design as a Search Process
G7	Communication of Research

Table 1.1: Guidelines in Design Science for Information Systems Research

Following the design science principle, all seven guidelines are taken into consideration within this thesis. In reference to **G1**, different artifacts have been developed to tackle the various needs relating to the research questions. The artifacts comprise methods, models and their implementation in form of software and are described in Chapter 6, 7 and 8. The problem relevance, according to **G2**, is further elaborated in Chapter 2. An evaluation in the end of this thesis respects **G3** and examines different aspects of the developed artifacts. The research contributions (**G4**) of this thesis are methods and concepts to fulfill the derived business needs and are incorporated into a software artifact implementing the holistic situation management for situations. Foundations and existing methodologies are considered in Chapters 3 and 4 to derive requirements in Chapter 5. Additionally, the evaluation results of a conceptual investigation, a user study and performance experiments are presented in Chapter 9 to conform **G5** and **G6**. The communication of the research results (**G7**) has been conducted at different conferences to discuss the results with experts in the relevant fields. The previous publications are referenced in Section 1.4.

1.3 Contributions of this Thesis

This thesis contributes concepts, methods and models for situation management on basis of complex event processing. In addition, the conceptual artifacts have been implemented in a prototype. The developed system encompasses the following contributions, addressing the presented research questions:

- **RQ 1 - Transparency** The introduction of transparency in complex event processing systems based on partial pattern fulfillment requires the identification of relevant intermediate states to be observed. Therefore, the following contributions have been made with regard to RQ 1:

- **(C1) Model for Situations of Interest** An abstract representation for a situation of interest and its related query is developed to become independent of the underlying CEP engine to support multiple implementations of event processing architectures.
- **(C2) Decomposition Method** On basis of the developed abstract model for situations, a decomposition method is introduced to derive all relevant intermediate states by recursively analyzing and breaking down the original query.
- **(C3) Software Architecture** In the course of this thesis, a software architecture is designed and implemented to support the developed concept of transparency in CEP systems by adding a new processing layer that does not affect the existing architecture.
- **RQ 2 - Situation Awareness** Extending the basic level of situation awareness that existing CEP systems already offer, requires an explicit definition of a situation life-cycle and its adoption afterwards. Additionally, implementing the situation life-cycle has to provide a degree of freedom to enable the capturing of use case specifics. With regard to RQ 2, the following contributions have been made:
 - **(C4) Extended Situation Life-Cycle** The implicitly defined life-cycle of existing CEP applications is made explicit and extended afterwards to allow the fine-grained differentiation between application states to support a human operator with the acquisition and maintenance of situation awareness.
 - **(C5) Implementation Guideline** The introduced situation life-cycle is kept abstract allowing the coverage of a broad field of use cases. The implementation of the life-cycle, respectively of its transitions between the states allows the adjustment towards the requirements of the use case at hand. Therefore, this thesis describes a guideline which aspects to consider for an implementation of a situation-aware CEP application. The default implementation is based on the concept of partial pattern fulfillment.
 - **(C6) Architectural Extension** As part of this thesis, the developed architecture for transparent complex event processing (see C3) is extended to support situation awareness on basis of partial pattern fulfillment. Therefore, a component is implemented to handle situation states and its transitions.
- **RQ 3 - Prediction Support** As projections into the future are part of situation management, the solution within this thesis supports predictions in the situation life-cycle. Depending on the use case, different prediction models are required, therefore, the approach supports domain experts with the preparation of data for the training of a prediction model, its embedding in the solution and its continuous monitoring.
 - **(C7) Architectural Extension** Based on the previously introduced concepts and architecture, components are designed and implemented that support the collection, labeling and export of relevant information that is collected at run-time and serves as an input for domain experts to derive a prediction model. The manually trained prediction models are embedded in the architecture and automatically included in the handling of situation states in the life-cycle.

All contributions combined together implement higher-level capabilities for a holistic situation management system. The added value of those capabilities is evaluated within Chapter 9 and shows that the goal of establishing a holistic situation management is achieved.

1.4 Previous Publications

The core contributions of this thesis have been conceptually designed and developed within a German-funded research project. The implementation of the solution has been presented at various project meetings. In addition, the core contributions are peer-reviewed and published as described below.

BigPro (Big Data and Event-based Adjustment for the Configuration of Resilient Production Systems, 09/2014-11/2017, BMBF)

The major part of this thesis has been developed within the research project BigPro. The goal of the project was the development of a Big Data platform supporting proactive disturbance management in manufacturing environments. The project comprised the following tasks: i) creation of an information landscape to determine relevant data, ii) evolution of complex event processing towards a situation management system, iii) development of a reaction management and iv) implementation of a new, user-oriented visualization concept [Stich et al., 2015]. The central component of the platform is a CEP engine and all relevant events are received via an event bus. On basis of the underlying architecture, the solution presented in this thesis has been introduced to extend the capabilities of the platform. The newly claimed transparency and situation-awareness has been used to improve the reaction management for machine outages and to deliver valuable input for a situation-dashboard.

Publications

- Suad Sejdovic. **Partial Pattern Fulfillment and its Application in Event Processing**. Proceedings of the 10th ACM International Conference on Distributed and Event-Based Systems (DEBS). 2016, Los Angeles, USA. (see [Sejdovic, 2016]).
- Suad Sejdovic, Yvonne Hegenbarth, Gerald H Ristow, Roland Schmidt. **Proactive Disruption Management System: How not to be surprised by upcoming Situations**. Proceedings of the 10th ACM International Conference on Distributed and Event-Based Systems (DEBS). 2016, Los Angeles, USA. (see [Sejdovic et al., 2016]).
- Suad Sejdovic, Natalja Kleiner. **Proactive and Dynamic Event-driven Disruption Management in the Manufacturing Domain**. Proceedings of the IEEE 14th International Conference on Industrial Informatics (INDIN). 2016, Poitiers, France. (see [Sejdovic and Kleiner, 2016]).
- Suad Sejdovic, Sven Euting, Dominik Riemer, York Sure-Vetter. **Considering Human Factors in the Development of Situation-Aware CEP Applications**. Proceedings of the 11th ACM International Conference on Distributed and Event-Based Systems (DEBS). 2017, Barcelona, Spain. (see [Sejdovic et al., 2017]).

Additionally, the research challenges have been discussed in a doctoral consortium at the 9th International Conference on Distributed Event-Based Systems (DEBS), 2015, Oslo, Norway.

1.5 Structure of this Thesis

First, the underlying terms, concepts and technologies with regard to stream processing and complex event processing are introduced as foundations of this thesis. The central aspects of this thesis are motivated in Chapter 2 by describing the evolution of relevant applications fields for CEP and introducing a running example for illustration purposes. Along the use case, three problems of existing systems are derived and transformed into requirements for this thesis. Chapter 4 gives an overview of related work in the fields of partial pattern fulfillment and its application, situation-aware systems and proactive event-driven computing. Based on the motivation, the running example and the related work, requirements towards this thesis are derived and described in Chapter 5. Chapter 6 explains the solution and artifacts for transparent complex event processing, whereas Chapter 7 describes a conceptual extension to establish situation awareness based on transparency. The predictive aspects of situation awareness are given attention in Chapter 8 by describing an approach to support the development of predictive models for the projection of situations. Chapter 9 presents evaluation results and is followed by a conclusion and outlook for future work in Chapter 10. Figure 1.1 illustrates the structure of this thesis.

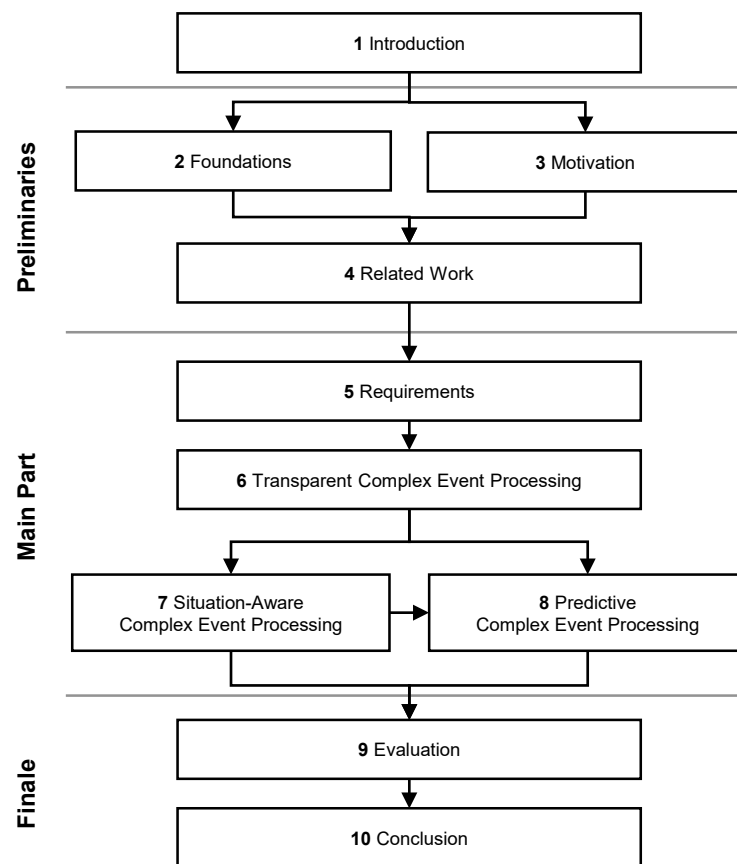


Figure 1.1: Structure of the Thesis

2

Motivation

In this chapter, the thesis is motivated by describing the evolution of real-time processing from a practical perspective and discussing its development in science over the last decades. Section 2.1 gives an overview of the origin of complex event processing and how its application field has changed over time. In Section 2.2 a running example is introduced to be used throughout the thesis to illustrate various concepts and approaches. Finally, in Section 2.3, gaps are identified that justify the further enhancement of the complex event processing paradigm towards a new evolutionary step, the human centric complex event processing.

2.1 Real-Time Processing with Complex Event Processing

Since the introduction of enterprise applications in companies, process and business related data is being collected to better understand the internals and make well-founded decisions. In the past, this data was stored and due to its size and resource restrictions analyzed in fixed intervals in a batch manner, e.g. quarterly. As pointed out in the introduction, the continuous improvements of computer systems led to an immense increase of generated data. The financial industry was one of the first fields to experience the identified Big Data challenges. Around the year 2000, more and more trading data became digitally available, but traditional processing capabilities were not sufficient to effectively handle it. Thus, financial companies first recognized real-time data processing as a key technology to significantly increase their competitive advantage [Madia, 2014]. Identifying profitable situations, e.g. reversal of a trend for specific markets, became the task of complex event processing systems, that emerged in science around the same time. The goal was to correlate stock and market data to identify potential investment options and trade corresponding financial products automatically, as microseconds distinguished between winning and losing money. This business requirement led to high-performance CEP systems with very low latency. According to Madia [Madia, 2014], by 2001 up to 20 % of the trades were computer assisted, whereas in 2016, up to 90 % were done fully electronic [Nagel, 2016]. The research on CEP systems within this period was strongly focused on performance and application driven.

During that time, the growth of CEP systems outside the financial industry was relatively moderate [Madia, 2014], nevertheless in parallel, further research topics arose, e.g. incorporating uncertain events. In recent years, CEP was progressively applied for use cases across other domains, e.g. for transportation [Baumgrass et al., 2014], nevertheless no domain as a whole emerged to profit from the CEP paradigm as the financial domain did. This has changed recently, with the onset of IoT and its application in industrial settings. IoT represents a paradigm with the goal to connect physical devices by standard communication protocols enabling the interaction among each other [Bassi and Horn, 2008]. As the costs of electrical components are continuously decreasing, more and more manufacturers integrate additional sensor and communication technology into their products. The goal is to collect data about the usage of products and to allow the interconnection to offer higher-level functionalities. Besides improvements in the personal and social domain, Atzori [Atzori et al., 2010] identified several applications in the domains of transportation, logistics, healthcare and smart environments.

One important environment missing in Atzori's listing is the manufacturing domain, as it recently became the focus of attention. A strategic initiative closely related to the Industrial Internet of Things (IIoT) and its implementation in European countries is Industrie 4.0 [Kagermann et al., 2013]. It stands as a synonym for the fourth industrial revolution and highlights the potential that is seen in IIoT and its impact on the economy [Jeschke et al., 2017]. The basis for the industrial revolution builds the convergence of the physical and virtual world, which leads to a mixed form of cyber-physical-systems [Kagermann et al., 2013].

Solving the arising challenges through the fine-meshed interconnectedness is highly relevant for the success of strategic initiatives like Industrie 4.0 and its impact on the economy. Complex event processing with its capabilities to detect complex relationships between events, its low latency and maturity presents an appropriate way to handle the huge and further increasing amount of heterogeneous data. As proposed by Chen [Chen and Zhang, 2014], CEP engines are capable to implement the central processing layer in IoT architectures.

The new application field differs significantly in one aspect compared to the domain CEP systems originated from: not all situations within the manufacturing field allow an automatic reaction to be triggered, in many use cases humans remain in the control loop. In contrast to stock trading, cyber-physical-systems require actions to be taken on-site, directly at the physical component. Consequently, a human operator is involved in the reaction procedure and follows the system's recommendation. According to Luckham [Luckham, 2002], this aspect leads to the challenge, how humans are kept involved and informed in today's information systems to better comprehend the mode of operation and results.

In the following sections, this challenge is further elaborated on by first introducing a running example from the manufacturing domain and afterwards analyzing the gaps to be bridged for an improved complex event processing as a critical success factor for the implementation of the Internet of Things.

2.2 Running Example

For a better understanding, a running example is introduced, which is used throughout the thesis to illustrate the developed concepts and approaches. The use case is derived from the manufacturing domain and describes a prominent technology frequently used in industrial production: punching. It was chosen, because it represents a simple and comprehensible environment, nevertheless includes all relevant aspects of many manufacturing environments to illustrate the problems of state-of-the-art monitoring systems.

The punching environment comprises several components that in coaction produce large quantities of identical geometric forms. Figure 2.1 illustrates a punching environment and shows how the various components are tied together to implement the punching process. Initially, a three millimeter thick aluminum sheet is placed on a conveyor belt, where a bar-code reader scans the label on the workpiece to verify that the correct metal sheet is chosen for the current production job. Appropriate metal sheets are automatically forwarded to the heart of the process, the punching machine. Depending on the currently deployed production plan, the punching machine is equipped with different punch shapes. The machine forces the equipped punch through the material by applying a specific tonnage, which varies with regard to the material used for the sheets. A container, placed underneath the machine, is used to collect the punched forms. Monitoring the production environment requires suitable sensor technology to be installed appropriately. Within this use case, a vibration sensor is attached to the workbench to measure the occurring vibrations during the punching procedure. Another sensor is installed in the punching machine and observes the force applied to workpieces. In general, punching machines have an emergency stop, in case the applied tonnage exceeds the maximum permitted punching force. An additional source for further information is the scale, as the weight of the container is continuously measured.

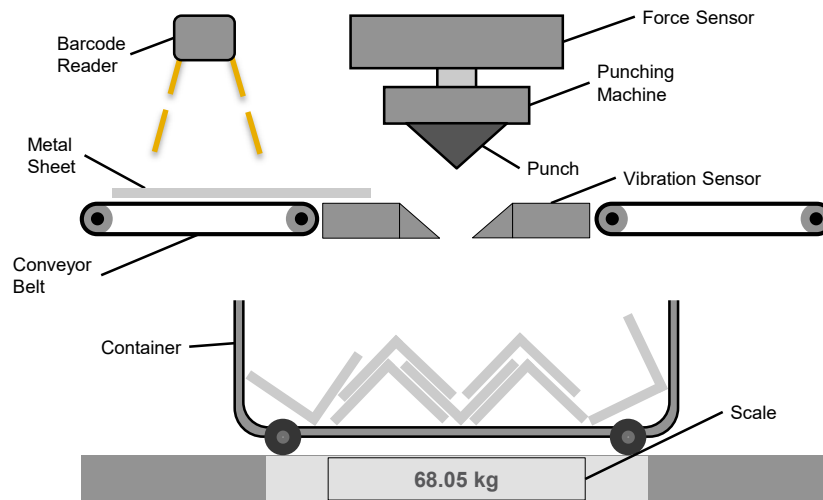


Figure 2.1: Punching Example

The production environment is already monitored by means of a complex event processing system. The presented sensors are connected to an event bus and deliver their observations continuously through the production process. First, the barcode reader emits an event indicating which metal sheet was scanned on the conveyor belt. Afterwards, the punching machine applies the necessary force to the workpiece and the force sensor determines the used tonnage to be send to the event bus. During the punching procedure, the vibration sensor measures the occurring vibrations and emits a vibration event, before lastly the scale measures the current weight of the container and generates a weight event.

Situation of Interest

By using the introduced events, a situation of interest is monitored by using a CEP engine during the production. The sharpness of the punch is crucial for the quality of the punched forms. A dull punch produces forms with rough edges, which require manual rework. Through mechanical wear, the punch progressively becomes dull. The development of the dullness depends on various factors, e.g. the material and thickness of the sheets. Replacing the punch prematurely results in higher production costs. On the other side, if the punch is replaced belatedly, a machine outage is risked. As illustrated in Figure 2.2, a dull tool leads to an increase in punching force required to be applied to the metal sheet, which can exceed the punching machine capacity and trigger an emergency stop.

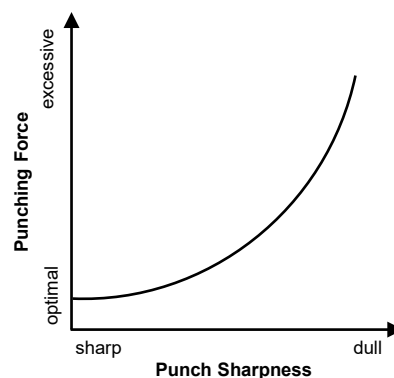


Figure 2.2: Development of Punching Force vs. Punch Sharpness

Based on the simple events coming from the deployed sensors, a pattern, described by a domain expert, expresses the situation of a dull tool. Listing 2.1 shows the resulting query for the underlying CEP engine, in this example the query is written in the Apama Query Language (AQL) and uses simple events, aggregated events and event operators to describe the situation. A query comprises at least three important sections to describe a situation of interest. The *inputs*-section describes which events are processed and whether filters have to be applied to the incoming event stream. The *find*-section describes the relationship between the events and whether a temporal constraint has to be applied, whereas the *send*-section defines the reaction once the event pattern was detected in the event stream. Basically, the dull tool pattern defines the condition that after identifying a high level of punching force a rising trend is recognized and the *MaxForce* event occurs. The rising trend is calculated by a simple aggregation within the CEP engine by averaging the punching force over the last 15 punches and comparing two aggregated events with each other.

```
query DullToolSituation {
  inputs {
    Force(force > 16.8)      retain 1;
    ForceRisingTrend()      retain 1;
    MaxForce()              retain 1;
  }

  find Force:level -> ForceRisingTrend:trend and MaxForce:maxForce within 2 seconds {
    send DullTool(maxForce.id, maxForce.force) to "um:NotificationChannel";
  }
}
```

Listing 2.1: Apama Query for Detection of Dull Tool

The *MaxForce* event is generated by another pattern, shown in Listing 2.2 describing a sequence of a vibration alert and a significantly higher punching force, that still lies below the maximum capacity of the punching machine. As this situation is also used in other use cases, it was defined in a separate query. It comprises the observation that through the dull punch the production process leads to increased vibration in the workbench, which is detected by the vibration sensor.

```
query MaxForceSituation {
  inputs {
    Force(force >= 17.5)    retain 1;
    VibrationAlert()       within 2 seconds retain 1;
  }

  find Force:force -> VibrationAlert:vibration {
    send MaxForce(force.force, vibration.vibration) to "um:NotificationChannel";
  }
}
```

Listing 2.2: Apama Query for Detection of Max Force

Consequently, the introduced query detects a dull tool before the maximum capacity of the punching machine is reached. The monitoring system interacts with the operator of the environment and informs about the dull tool by generating a complex event indicating that a reaction is required. Thus, the punch is replaced before the machine triggers an emergency stop and leads to an unplanned maintenance, which results in higher maintenance costs. This gives a machine operator the possibility to replace the dull tool and restart the production process. Therefore, spare punches have to be hold available for replacement. This use case comprises all required aspects to illustrate the concepts and solutions within this thesis. Additionally, it is used in the following section to illustrate the gaps of nowadays CEP environments as a motivation and further explanation of the presented research questions.

2.3 Problem Statements

In general, manufacturing environments comprise more than a single production machine and have significantly more sensors in the environment deployed to collect huge amounts of data. As stated earlier, in contrast to fully digitized domains, like stock trading, production environments consist of cyber-physical-systems that often require a human operator to remain in the control loop. The necessity for human interaction arises for various reasons. For instance, the replacement of parts in a machine needs to be done manually, as the automatic replacement is not feasible and an operator has to fetch spare parts from the stock of inventory. In some cases, automatic reactions might be possible, but are only suggested to operators as a recommendation, e.g. due to the criticality of the system, missing trust in the results or insufficient comprehensibility of the mode of operation of the monitoring system.

Complex event processing is suitable to tackle many of the challenges that arise through the broad digitization in the current era of Industrie 4.0 and the increasing dissemination of IoT. Event processing systems have a long history of processing huge amounts of data and establishing a basic level of situation awareness by detecting occurring situations with the goal to automatically react towards them. With the new expansion into the world of cyber-physical-systems, that additionally have interfaces to human operators, complex event processing has to be enhanced further to better integrate this aspect into its mode of operation. The goal is to leverage the capabilities of CEP engines and link them with the way of human information processing to establish a holistic situation management. The following subsections identify existing gaps between both worlds relevant for this thesis and derive the requirements for a new level of complex event processing, incorporating the concept of humans in the loop. One aspect that is also relevant in use cases with human interaction, but is not considered in the course of this thesis, is the design of graphical user interfaces.

2.3.1 Transparency

The first aspect to be discussed is the transparency of complex event processing systems. Transparency refers to the property of a system that allows the observation of internals during its processing of data. In fully automated use cases, the transparency of the processing logic at run-time might not be required, nevertheless represents an additional benefit for the comprehensibility of the system. For use cases with human interaction, transparency is a basic necessity, as it leads to trust in a computer system [Corritore et al., 2003]. Dependent on the existing knowledge that users have, the required level of transparency to accept recommendations of a system fluctuates [Kantowitz et al., 1997]. Users with more experience have a higher standard towards transparency as users with less experience, who build trust in a shorter time. A user survey conducted by Glass [Glass et al., 2008] revealed that transparency was considered as the most significant aspect to support building trust in a system and that intransparent, opaque system behavior reduces the trustworthiness of a system. By allowing to retrace the analysis of the data, a computer system supports the interpretation of the results for decision-makers, which additionally increases the trust in the system [Jagadish et al., 2014].

By analyzing the introduced running example, the goal is to evaluate how state-of-the-art CEP systems behave with regard to transparency at run-time. Therefore, the behavior is observed while processing an excerpt from an event stream, illustrated in Figure 2.3. The analysis comprises the comparison of both perspectives, from the viewpoint of the CEP engine and from the viewpoint of a human operator.

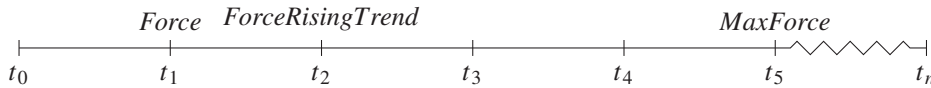


Figure 2.3: Excerpt of Event Stream

At t_0 no pattern-related event is detected in the event stream, consequently, for the human operator and the CEP engine the situation stays undetected, i.e. the internal processing of the CEP system is not triggered. With the arrival of the *Force* event at t_1 , the internal processing of the CEP engine starts with the evaluation of the deployed query, as the detected event is part of the defined pattern. As long, as the human operator is not directly monitoring the input events on the event bus, this development stays unnoticed. Equally, the event detection at t_2 stays intransparent for the human operator, whereas the CEP engine continues with its progressive evaluation of the deployed query. As no pattern-relevant events are detected at t_3 and t_4 , nothing changes with regard to the previous state. This changes at t_5 with the occurrence of the *MaxForce* event, that is part of the query representing the situation of a dull tool. Now the CEP engine finishes the pattern detection and triggers its action statement, which generates a notification event to be sent to a specific notification channel. By means of this notification, the human operator finally becomes aware of the situation and has now to replace the dull tool to avoid a machine outage.

At no point of time the operator was aware of the upcoming situation, although evidentiary observations were made with regard to the defined situation. Also, the operator does not have any possibility to query the current state of the monitoring environment during run-time. Operating in a black-box manner represents an issue in environments where human operators are involved in the monitoring and operation of technical systems. Thus, by increasing the transparency of CEP systems and allowing the observation of the current internal state, the added value of CEP is further expandable. Therefore, a mechanism is needed to enable the observation of partial pattern matches, as they occur within the CEP engine. Manually defining the intermediate states, in terms of queries representing those states, is not practicable, as it represents a non-intuitive task for domain experts. In contrast, defining patterns for situations is an intuitive task, thus the mechanism should be based on existing situation descriptions. Finally, transparency represents an important prerequisite for many optimization tasks [Jagadish et al., 2014] and builds the basis for further improvements and enhancements of CEP capabilities, as discussed later.

In summary, state-of-the-art CEP systems operate in a black-box mode which deteriorates the comprehensibility of the system as its current state is nontransparent for the operator. By allowing the observation of its current state during the processing, the transparency and comprehensibility of the system is improvable.

2.3.2 Situation Awareness

Monitoring complex technical environments, like in manufacturing, represents a cognitively demanding task for operators. Today's production facilities consist of up to several thousand components that require monitoring and correspondent handling. Extracting the full potential of Big Data requires consideration of scalability not only from the perspective of a system, but even more from the perspective of humans [Jagadish et al., 2014], i.e. which data is available and how is it presented. Monitoring systems are deployed to support the operator and collect different relevant aspects of the environment to give an overview of the current state. As long as no outages occur, operators browse through collected information to stay up to date with the current developments in the environment. Complex event processing systems represent an extension to existing monitoring solutions and add a basic level of situational awareness, as they are capable of evaluating complex relationships between events in the environment and signify the occurrence of a situation of interest. This behavior represents a valuable and meaningful add-on, especially for environments with huge amounts of data to be processed.

As already stated by Simon in 1971 [Simon, 1971], one disadvantage of data rich environments is the phenomenon that additional information consumes the attention of its recipients and leads to the requirement of allocating the attention. Thus, through the further increase of data and information nowadays, an attention management is required. In general, the goal of monitoring systems is to direct the attention of operators to relevant happenings and trigger a timely reaction to resolve issues. Therefore, monitoring systems take the task of processing every single signal from the operator and aggregate this information to higher level insights. In principle this describes the basic mode of operation of CEP systems, where simple events are aggregated to complex events indicating the occurrence of a situation of interest.

With regard to the characteristics of human information processing, CEP systems are improvable. State-of-the-art systems differentiate only between two states with regard to situations: detected and not detected. Human operators differentiate also between intermediate states, which allows the efficient and timely direction of attention towards situations. By introducing aspects of the psychological model of situation awareness and further refining the perspective on situations from the view of an operator, an improved attention management is implementable. An increased transparency during run-time, as discussed in Section 2.3.1, is seen as a basic requirement for a better direction of an operator's attention.

In summary, state-of-the-art CEP systems only partially support the psychological model of situation awareness leading to an inefficient direction of an operator's attention. By extending its support to full situation awareness, the attention management is improvable.

2.3.3 Proactivity

One eligible question with regard to monitoring environments is their value proposition. In terms of complex event processing the added value lies in the (near) real-time detection of a situation of interest and triggering a reaction. The value of this notification and reaction highly correlates with the time dimension. The sooner a situation is identified, the better are the chances to mitigate the disadvantageous effects of it, respectively in case of an advantageous situation to intensify them. Figure 2.4 illustrates the development of the information value for complex event processing according to Fülöp [Fülöp et al., 2012]. Due to its reactive nature, detecting situations at the time of their occurrence has the highest information value for CEP and represents the real-time aspect of such systems. As a consequence, automatic reactions are triggered in real-time, which builds the basis for the successful development of complex event processing and for the gaining popularity as a real-time processing paradigm.

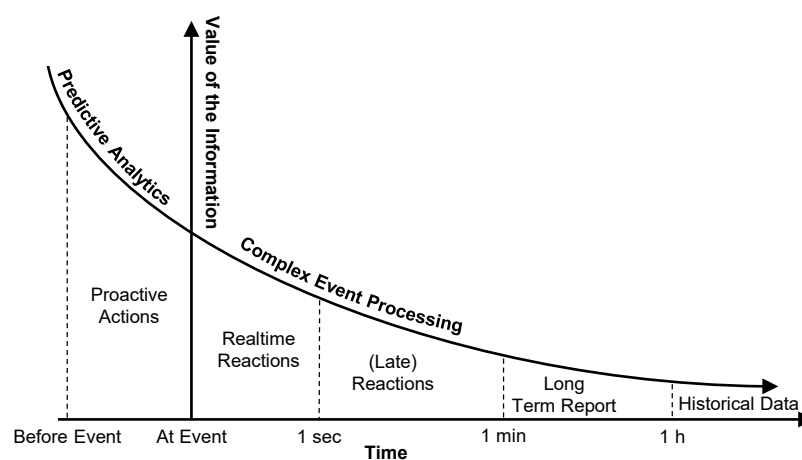


Figure 2.4: Development of the Value of an Information according to [Fülöp et al., 2012]

The development of the information value in Figure 2.4 is depicted for fully automated environments and loses its validity for mixed environments, where human operators are in the control loop and CEP engines solely notify about situations. Nevertheless, a short reaction time represents a significant competitive advantage. Business value is decreased, if the action distance to an observation, i.e. the time needed to process the data and decide which action to take, is too high [Schwegmann et al., 2013]. Thus, the question arises how to adopt the mode of operation of CEP systems to reduce the action distance, respectively the reaction time with humans in the loop. The latency in taking actions due to the involvement of human operators and physical components leads to the requirement for projections into the (near) future to further increase the beneficial impact of complex event processing in such environments and allow a proactive reaction.

Considering the introduced running example, a human operator is notified about a dull tool once the query is fully evaluated and the defined pattern is detected in the event stream. The time between the notification and resolution of the situation may vary significantly depending on various factors: i) time until the notification is noticed by a human operator, ii) whether the necessary spare parts are available or have to be ordered, iii) whether available spare parts are prepared for a replacement in case they require a pre-processing and iv) whether the human operator has the necessary skills for the replacement task. One significant improvement in the running example is to inform the human operator about upcoming situations as early as possible to utilize the remaining time for preparations.

As discussed in the previous subsection, Section 2.3.1, CEP engines evaluate queries progressively. By using the incremental information available within the CEP systems, it is possible to give operators a heads up about a potential situation. As shown in Figure 2.4, an information before the actual situation occurs has an increased value. Certainly, early notifications are afflicted with uncertainty, nevertheless they might improve the reaction time, as operators become prepared for potential situations, e.g. by pre-ordering and preparing required spare parts. Leveraging the progressive processing of queries within CEP engines, the reaction time in human-in-the-loop environments might be improved. Therefore, a mechanism is required, that allows the observation of situation developments within a CEP engine. This requirement correlates with the need for an increased transparency, which represents a basic prerequisite for it.

In summary, state-of-the-art CEP systems follow a reactive mode of operation that limits its added value in environments with humans in the loop, as the action distance is increased. Therefore, predictions should be supported to reduce the action distance and enable a proactive mode of operation.

2.3.4 Conclusion

In this chapter the underlying motivation of this thesis has been presented. Through the evolution of the application areas for complex event processing systems, new needs were derived based on identified gaps of current systems. In conclusion, complex event processing requires further development towards a more human centric way of event processing by considering human factors into its mode of operation. The missing transparency of current CEP systems was identified as one central prerequisite to bridge the gap and allow the further enhancement of CEP towards its next evolutionary step.

Part II

Preliminaries

3

Foundations

This chapter describes the basic definitions, terms and technologies that build the foundations of this thesis. At first, in Section 3.1 the field of *Stream Processing* and its evolution through multiple domains is introduced. The main content of this thesis is based in the field of *Complex Event Processing*, which is generally introduced in Section 3.2 together with the fundamentals of *Events* and *Event Streams*. Important building blocks of Complex Event Processing, namely *Event Processing Networks* and *Patterns*, are introduced in Sections 3.3 and 3.4. In the end, in Section 3.5, *Complex Event Processing Engines*, as the technical implementation of the Complex Event Processing paradigm, and their run-time behavior are explained in detail.

3.1 Stream Processing

As nowadays more and more data is generated, an increasing number of application fields requires the continuous processing of streams of data from distributed sources. Real-time data processing systems are part of the stream processing paradigm, which according to Cugola [Cugola and Margara, 2012] comprises two concepts: timeliness and flow processing. Timeliness is geared towards a real-time (business real-time) processing of data and the immediate extraction of relevant knowledge. Flow processing requires data to be processed continuously as an unbounded stream of data. The concept of flow processing comes along with the idea of not necessarily storing all data, as relevant insights are gained in real-time.

Different technologies emerged as implementations of the stream processing paradigm: *Active Database Systems*, *Data Stream Management Systems (DSMS)*, *Publish/Subscribe* and *Complex Event Processing (CEP)*. Traditional database management systems (DBMS) are passive and require the user to actively trigger requests to gain insights from the data. Active databases are an extension to traditional databases and move the reactive behavior into the database itself [McCarthy and Dayal, 1989]. Active databases follow the Event-Condition-Action (ECA) paradigm [Dayal et al., 1988], where events trigger the evaluation of conditions and actions define a set of tasks that are executed as a response. Within their core, active databases still rely on a persistent storage which negatively impacts their performance in high-volume use cases. Even though they are able to evaluate conditions with regard to an event, stateful operations, containing several events and complex relationships, are not supported.

Another technology, which overcomes the limitations of active databases, are data stream management systems [Babcock et al., 2002]. DSMSs operate on unbounded streams of data which are ordered by time and support one-time processing as the typical way of processing data. DSMSs follow the *Database-Active Human-Passive (DAHP)* interaction style [Abadi et al., 2003] and for that purpose make use of standing queries, i.e. queries that are deployed once and continuously evaluated to produce results [Cugola and Margara, 2012]. In contrast to traditional databases, where dynamic queries are matched against a static dataset, dynamic data streams are matched against deployed queries. Users are continuously informed about the results of the query, which makes DSMS suitable for continuous monitoring tasks. The detection of complex relationships between the elements in a data stream involving sequences and ordering relations are usually out of scope of these systems [Cugola and Margara, 2012].

The next development within the field of stream processing are systems supporting the publish/subscribe concept: systems subscribe to specific types of data to express their interest while sources publish data without knowing any details about the receivers. Publish/subscribe engines are in charge of delivering the data to subscribers according to their expressed interests. With regard to the expression of interests, two different types of publish/subscribe systems can be distinguished: topic- and content-based systems [Eugster et al., 2003]. Whereas topic-based systems only allow to subscribe to predefined topics, content-based systems allow the definition of complex filters to express interesting contents. These subscriptions only refer to single data elements on the stream and do not allow the definition of complex relationships between them.

Complex event processing represents an extension of DSMSs and publish/subscribe systems as it is based on numerous foundations of those technologies. As a central paradigm and technology within this thesis, CEP is introduced in detail in the following section together with the general concepts of event stream processing.

3.2 Complex Event Processing

The discussed technologies, like DSMSs, show limitations with the identification of complex relationships between data elements on a data stream. According to Cugola [Cugola and Margara, 2012], this limitation originates from the fact that their systems' design is generic and does not associate a meaning to the data being processed, i.e. elements on the data stream are only considered as data items without knowing what they represent in the context of the processing. *Event processing* differentiates in that point from classical stream processing by associating a specific meaning to the data element that is being processed. Instead of just processing general pieces of data, *Events* are introduced that associate a meaning to the elements on the stream and represent a happening in a related environment, e.g. a machine outage.

The underlying assumption in *Complex Event Processing* (CEP) is that events in an *Event Stream* have to be filtered and combined to understand what is happening. Accordingly, CEP systems have a special focus on detecting complex relationships between diverse events to recognize *Patterns* of interest that describe *Situations of Interest* (SOI). For the detection of complex relationships, CEP strongly relies on time aspects, as some powerful patterns require for consideration of time, e.g. sequences. One reason, why complex event processing has a direct impact on real-world problems and businesses is its ability to relate to elements of those environments, i.e. events describe happenings in them, just as SOIs describe a business-related situation that is of interest and can be identified by means of complex event processing. By exploring the temporal and causal relationships between events, business conditions can be monitored and in case of changing conditions reactions can be triggered in real-time.

In the following, the fundamental elements of event processing are introduced and described in detail. Events and event streams form the basis for the mightiness and broad usage of complex event processing and the event processing paradigm in general, as they build the bridge between computer systems and the real-world.

3.2.1 Events

Derived from the Latin word *eventus*, an *event* describes the occurrence of something in an environment. Within the context of computing systems, an event represents the record of an activity in a system [Luckham, 2002], i.e. an event represents the information that something happened in any kind of system. The source of an event can be of any type, e.g. a software system like an ERP system or a hardware system like a temperature sensor. In [Etzion and Niblett, 2011] Etzion gives a broader definition of an event:

"An event is an occurrence within a particular system or domain; it is something that has happened, or is contemplated as having happened in that domain. The word event is also used to mean a programming entity that represents such an occurrence in a computing system."

Within this definition three different aspects of events are mentioned. First, the term event refers to something that has happened in the real-world or a virtual environment. Second, the term refers to an entity within a computer system that represents the occurrence from the real- or virtual-world. And third, an event can also be contemplated as having happened, i.e. an event entity in a system refers to the occurrence of an event in the real- or virtual-world, where no event happened at all. This situation can be considered as a false positive. According to Luckham [Luckham, 2002] three additional aspects of an event have to be acknowledged:

- **Form:** An event's form describes an object that can contain a set of data components
- **Significance:** An event signifies an activity and its form contains data components describing the activity
- **Relativity:** As activities can be related to other activities by time, causality and aggregation, also events can have the same relationships as the activities they signify

The goal of working with events is to process them in a computer system. Therefore, a suitable event representation is required. As stated by Etzion [Etzion and Niblett, 2011], the three main building blocks of an event are *event objects*, *event attributes* and *event types*. An event object is the specific representation of the occurrence of an event in a computer system. The contents of an event object, which are a set of data components, are considered as the event attributes. Each attribute has a name and a data type. Several event objects with the same structure are described by an event type, specifying the intent and structure. Event objects are instances of this event type and conform the defined structure.

When defining the content of an event object, three different kinds of information can be distinguished according to Etzion [Etzion and Niblett, 2011]: *header*, *payload* and *open content*. The header describes meta-information about the event, e.g. the occurrence time of the event as events are strongly related with time or some additional information about the event producing sensor. The payload contains a set of data components describing the occurrence of the event. The set of data components is defined by the event type and consists of attribute names and the according data types. The open content part of an event serves as a free-format container, which is not constrained by the event type definition. Interchanging events requires a format depending on the implementation for event processing and the supported formats. Nowadays, most event processing systems support standard data formats like the Extensible Markup Language (XML) or the JavaScript Object Notation (JSON).

Figure 3.1 illustrates an example showing all described aspects of events: turning on the heater as an activity triggers the heating control system to generate an event signifying the occurred activity. The generated event is transferred to a processing system by using a JSON message as the event representation. Within the processing system the event is treated as an event object whose inner structure and event attributes conform a previously defined event type. The event type clearly describes how events of its kind look like and which event attributes they contain. By just considering the event representation, the event object and the event type, only the form aspect of an event is covered. According to Luckham [Luckham, 2002] this leads to the confusion that an event represents just a message. Therefore, it is important to consider the other two crucial aspects described previously: significance and relativity. In the example, the events clearly signify an activity in the real-world and are also related to each other as the activities in the real-world are related to each other. Thus, event processing has to deal with the relationships between events.

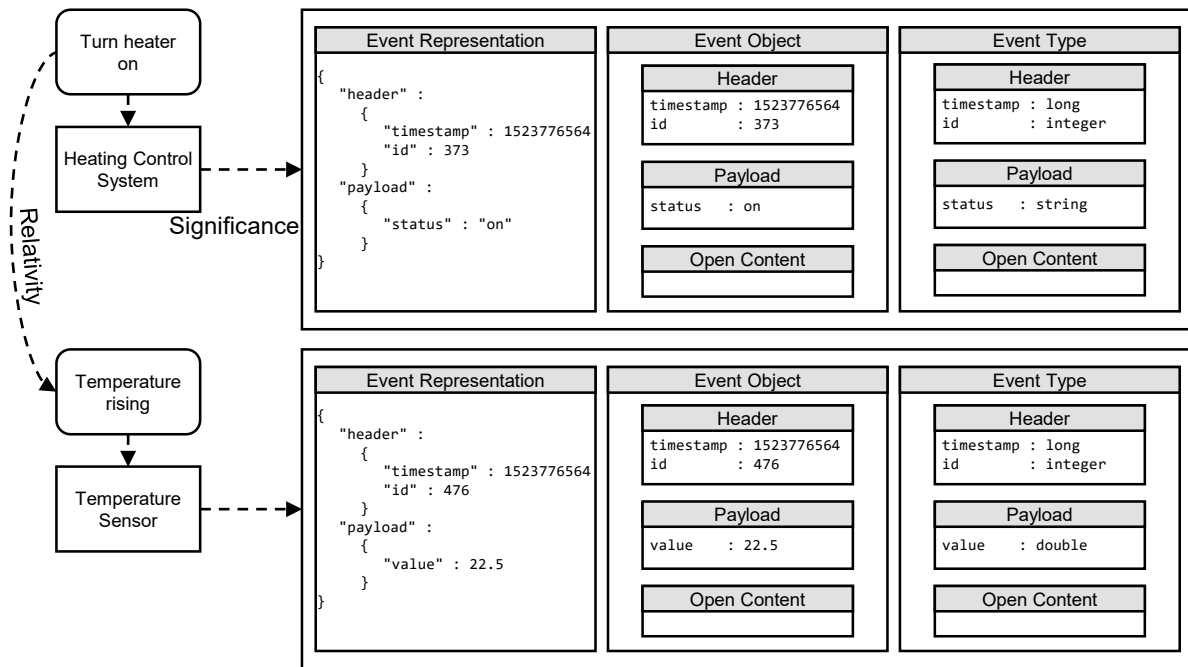


Figure 3.1: Event Definition Example

Simple Events vs. Complex Events

Within the field of complex event processing, two general event types are further distinguished: *Simple Events* and *Complex Events*. Whereas simple events relate to raw events coming directly from event producers, like sensors, complex events refer to higher level events that are derived through the processing of simple events. Complex events signify a higher level event described by a pattern, respectively a situation that has occurred and is observable through the correlation of simple events.

3.2.2 Event Streams

Events form the smallest unit in event stream processing and are processed in different ways. *Stateless event processing* considers only single events to implement different functions, e.g. filtering, logging or transforming events. Such operators have one single event as input and deliver a single or multiple events as output. Obviously, such operators do not have the ability to process sets of events in order to consider their relationship. There is another class of operators, supporting *stateful event processing*, e.g. aggregating, stateful transforming or splitting event sets. Such operators perform their processing logic on so called *event streams*, which are defined by Etzion [Etzion and Niblett, 2011] as:

"An event stream is a set of associated events. It is often a temporally totally ordered set (that is to say, there is a well-defined timestamp-based order to the events in the stream). A stream in which all the events must be of the same type is called a homogeneous event stream; a stream in which the events may be of different types is referred to as a heterogeneous event stream."

According to this definition, event streams are a continuous set of event objects that are produced incrementally over time and ordered according to their timestamps. Event streams come with some special properties as described by Bruns [Bruns and Dunkel, 2015], which have to be considered at their processing:

- **Live data:** Every event object in an event stream represents the occurrence of the related event in the real- or virtual-world in (near) real-time.
- **Simplicity:** In general, event streams carry many simple events that do not have a high complexity, but get their complexity and higher meaning through applying processing logic.
- **Frequency:** Most event streams have a high volume and a high frequency.
- **Unboundness:** In contrast to bound data which is finite and can always be processed once all data elements are known, event streams are unbound and infinite.
- **Volatility:** Due to the high volume, events and event streams are not stored, thus they have to be processed in real-time as they occur, else the information disappears.
- **Relationship:** Events in an event stream have implicit relationships between single events, which is not obvious by just looking on the event stream.

Complex event processing takes up all those properties and presents a solution to handle them accordingly. The processing of events in complex event processing is done on a heterogeneous event stream, where, besides different event types, also complex events are part of the processing.

3.3 Event Processing Networks

Event processing is about performing operations on event objects. One rather abstract possibility to model this processing of event objects is an *Event Processing Network* (EPN). It is used to describe the architecture of an event processing system and comprises four components, namely *Event Producer* (EP), *Event Processing Agent* (EPA), *Event Consumer* (EC) and as an intermediary element *Event Channel*. In general, an EPN describes how events are received from event producers and directed to the event consumers through a network of processing agents [Sharon and Etzion, 2007]. Figure 3.2 illustrates a simplified example of an event processing network showing all high-level EPN components. In more complex EPNs, all components may have multiple connections between each other.

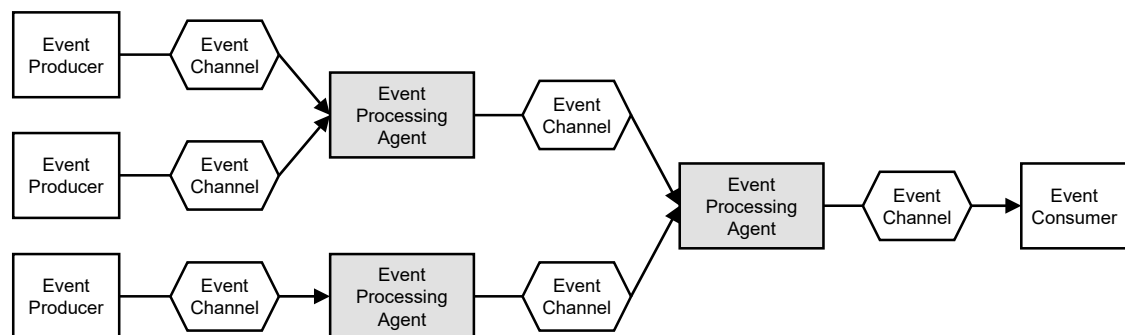


Figure 3.2: Event Processing Network Example

Hierarchies are useful when it comes to designing an EPN. Event processing networks may be nested one inside another, where one node in the EPN represents a nested event processing network instead of a single agent. The input and output of the nested EPN corresponds to producers and consumers of the EPN that it contains [Etzion and Niblett, 2011]. Additionally, EPNs may contain feedback loops, where the output of an agent is fed back into a preceding agent. In the following, the different components of an EPN are described in detail.

3.3.1 Event Producer

To process event streams at least one EP is required that observes relevant properties in the real- or virtual-world and serves as a source for an EPN. Producers monitor specific properties in the environment they belong to and according to a defined logic in the EPs decide when to generate a signifying event that relates to an activity. According to Luckham [Luckham, 2002], observations should be benign, i.e. they must not change the behavior of the observed system. Three categories of sources for events are distinguishable:

- **Hardware Event Producers:** Sensors that generate events reporting on aspects of the physical environment they are observing, e.g. a temperature sensor in a manufacturing environment.
- **Software Event Producers:** Software applications designed to act as a software sensor observing specific aspects within applications and generating events once defined states are achieved, e.g. approval and release of an order in the system. Complex event processing systems act as software EPs, too.
- **Human Interaction:** Events directly created by humans, either by interacting with a system and causing the event or by directly specifying the event, e.g. by entering the results of a laboratory test into a system.

In the example in Figure 3.1 the heater control system and the temperature sensor act as EPs observing real-world activities, like the start-up of the heating system or the increase of the room temperature. Once relevant activities are observed, they generate a signifying event and transform it into a machine-readable format for processing.

3.3.2 Event Channel

Produced events in an EPN are transported through the network to be processed by other components. Event channels act as a processing element receiving events from one or more source elements, deciding on how to route this event and then sending the input event to one or more target elements [Sharon and Etzion, 2007]. Event channels can be implemented in different ways depending on whether a single channel is used to handle all the routing between elements or several channels are used. A single channel is called an *event bus* and offers less flexibility with regard to possible requirements for specific processing components, e.g. individual access privileges.

3.3.3 Event Processing Agent

Event Processing Agents are one of the central building blocks in EPNs as they are mainly responsible for the processing of events. Externally an EPA has one specific function, but internally an EPA can represent three different logical stages to implement the external, higher functionality: *filtering*, *matching* and *derivation*. By filtering, the EPA decides which of the inputs are part of the further processing, whereas matching is used to detect patterns among events, before the derivation stage generates new events and defines their content. In general, EPAs take events from EPs as their input, apply their processing logic to transform them and forward the derived output to the next components in the EPN. Additionally, EPAs can be stateless or stateful, depending on whether only the current event is processed or preceding/succeeding events have to be considered.

Etzion [Etzion and Niblett, 2011] introduces a hierarchy of different EPA types that are illustrated in Figure 3.3 and described in the following. On the first level of hierarchy, three different types of EPAs can be distinguished:

- **Filter EPAs** are used to filter an input event stream and deliver a subset as output. The filter EPA represents a stateless EPA as only the current event object is considered. The filtering functionality is based on the event type or event attributes. A filter EPA does not transform the input event, as it only makes use of the internal filtering stage of an EPA and does not have a matching or derivation step.

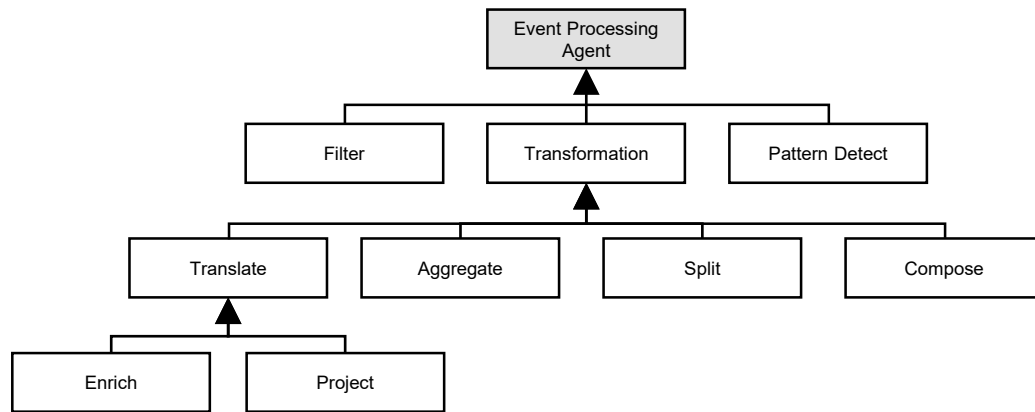


Figure 3.3: Hierarchy of Event Processing Agents

- **Transformation EPAs** are used to modify the input event by applying internally a derivation function and optionally a filtering function. The transformation EPA can be stateless or stateful and process a single event object or a set of event objects, before emitting single or multiple event objects for the further processing.
- **Pattern Detect EPAs** are used to detect the occurrence of defined patterns on the incoming stream of event objects. The pattern matching function can be applied to more than one input stream. Once the defined pattern is detected, it emits one or more derived events to inform about the matched pattern. Pattern detect EPAs are stateful as collections of multiple events are processed to detect patterns. A pattern detect EPA makes use of all three internal logical functions if it additionally implements a filtering. The result of the internal matching function is a matching set which contains all events that match the pattern. This set serves as an input to the derivation function to generate the output of the EPA.

As mentioned, transformation EPAs are either stateless or stateful depending on the type of transformation they perform. The differing transformation types are distinguished based on the cardinality of their in- and outputs:

- **Translate EPAs** are stateless in their functionality as they take each incoming event object and operate on it independently. In general, translate EPAs are used to transform events by modifying event attributes or converting events from one event type into another. Furthermore, within the class of translate EPAs two special instances are distinguished: **Enrich** and **Project** EPAs. The former is used to add information to an event or to correct wrong values of event attributes. For this purpose, Enrich EPAs are able to query data from a *global state element*, e.g. a database containing additional information about an order. Project EPAs are used to select a subset of the attributes of an input event and derive a new event object containing only the selected subset of event attributes.
- **Aggregate EPAs** are stateful, as they take a stream of incoming event objects and operate on them to finally emit a single event object. Aggregations are calculated incrementally or the aggregation is computed on a set of events after all relevant events are available. The input set of events that is required to perform the aggregation is temporarily stored within the aggregate EPA.
- **Split EPAs** take single event objects as their input and emit multiple event objects. Split EPAs are used to create more than one output stream in order to route event objects to several succeeding processing components. Each of the event objects is a clone or a projection of the original event object.
- **Compose EPAs** take two streams of incoming event objects and operate on them to emit one stream of output objects. The functionality is similar to join operations known from database systems. Additionally, compose EPAs may use a matching criterion to create derived events based on the matched events.

3.3.4 Event Consumer

In general, produced events are consumed by event consumers that represent sinks in an event processing network. An event consumer accepts event objects from the preceding components in the event processing network and processes them according to some internally defined logic. Actions that should be invoked by receiving the event are described within this logic. Event consumers are classified into three categories:

- **Hardware Event Consumer:** The counterpart of hardware sensors, observing physical aspects of an environment, are actuators that are capable of affecting the physical aspects of the environment, e.g. an actuator that is capable of regulating the room temperature according to a defined logic.
- **Software Event Consumer:** Software applications act as event consumers and trigger several actions in reaction to a received event, e.g. trigger a business process like the ordering of a transportation for a previously approved order. Complex event processing systems are another example for event consumers.
- **Human Interaction:** In many use cases produced events are consumed by human interaction. Event consumers generate some kind of alert about a critical situation to inform a user about it and give a chance to react accordingly. Examples for such an event consumption are dashboards and SMS or e-mail notifications.

3.4 Event Patterns

Being able to detect patterns in event streams represents one of the most powerful mechanisms in event processing [Luckham, 2002]. Patterns are regarded as descriptions of complex relationships in the real-world, describing an occurring situation and not just a single activity. Sets of events that match the defined conditions in the pattern satisfy it. Patterns are at the core of the previously introduced pattern detect EPAs, as the focus lies on the pattern matching step. Usually, they implement also the filtering step to sort out irrelevant events according to a *relevant event types list*, as only relevant event types are to be considered during the match making process. Events that are examined in the matching step are called *participant events*. Once the EPA finds a subset of events that match the pattern, the situation was *detected*. Sets of events that match the defined conditions are called *pattern matching sets* and are passed to the derivation step. Patterns are considered as the technical representation of a situation of interest. Two different types of patterns are distinguished: basic event patterns and dimensional patterns.

3.4.1 Basic Event Patterns

Basic patterns relate to frequently used, simple operations that are applied on single events or on collections of events. In general, they do not depend on the timing or ordering of events, but can be used with a temporal context. In the following the most frequently used are introduced: logical operator patterns and threshold patterns.

With regard to the operators in patterns, two general types are differentiated: blocking and non-blocking operators. According to Babcock [Babcock et al., 2002], blocking operators are unable to produce results until the entire input is available. With unbound streams of data, it is impossible to analyze the entire input. Therefore, blocking operators require a binding, which is done by means of windows to partition the event stream. Non-blocking operators on the other side process unbound streams of data and are evaluated without any specific binding. The performance of a system can be significantly improved by using windows also for non-blocking operators.

Logical operator patterns belong to the basic and most frequently used patterns and define conditions by combining multiple events. Logic operators are order independent, i.e. they only evaluate whether events have occurred or not. The following types of logical operator patterns are distinguished:

- **Conjunction Pattern (AND):** A conjunction pattern describes a condition, where involved events are linked by and-operators and all events have to be detected to fulfill the pattern. The and-operator represents a non-blocking operator. An example for a conjunction pattern is the condition $A \text{ AND } B$, which is satisfied once the events A and B are detected in the event stream.
- **Disjunction Pattern (OR):** The or-operator is used to describe disjunction patterns between events. It is a non-blocking operator and the pattern is fulfilled as soon as one of the involved events is detected in the event stream. An example for a disjunction pattern is represented by the condition $A \text{ OR } B$, which is satisfied once the event A or B occur in the event stream.
- **Negation Pattern (NOT):** The not-operator is the only blocking logical operator and represents a single-item operator. Negation patterns are fulfilled, if the involved element is not detected in the event stream. In order to evaluate the pattern, negation patterns require a bounding to an excerpt of the event stream by a window. An example for a negation condition is $\text{NOT}(A)$, which is satisfied if the event A does not occur on the considered excerpt of the event stream.

In many use cases, a combination of logical operator patterns is found, as it represents a powerful possibility to describe complex relationships among events, whenever the order of event occurrence is immaterial.

In contrast to logical operator patterns, threshold patterns involve aggregation operations performed on a set of participant events. The aggregation results are used for a comparison with defined threshold values by using one of the relations $=, \leq, \geq, <, >, \neq$. Different threshold patterns are distinguished:

- **Count Patterns** use the count-operator to determine the number of events within the set of participant events and to compare the result with a given threshold. If the statement evaluates to true, the pattern is detected.
- **Minimum and Maximum Patterns** determine the min and max value within the set of participant events and compare it with a given threshold. If the statement evaluates to true, the pattern is detected.
- **Average Patterns** calculate the average over a set of participant events and compare it with a given threshold. If the statement evaluates to true, the pattern is detected.

The introduced threshold patterns can be evaluated on unbound event streams, as the aggregation functions represent non-blocking operators. State-of-the-art CEP systems support the definition of custom aggregation functions. The combination of threshold and logical operator patterns is used for complex relationships between events.

3.4.2 Dimensional Patterns

According to Etzion [Etzion and Niblett, 2011], dimensional patterns relate to the time dimension, the space dimension and a combination of both. In most real-world use cases, temporal patterns are used. Within this thesis, the focus lies on temporal patterns, as they are capable of describing complex time-dependent correlations between events. One of the most important operators in state-of-the-art complex event processing systems is the sequence-operator [Mousheimish et al., 2017], which is used to define *Sequence Patterns*.

The sequence-operator takes into consideration the temporal order of events in the participant event set to define a relationship between events. Thus, the time of arrival of the events in the event stream is relevant for the processing. A sequence pattern describes an ordered set of events and is fulfilled once all involved events are detected in the specified order. An example for a sequence condition is $A \rightarrow B$, which is satisfied once the events A and B are detected in the event stream and event B occurs after the occurrence of event A .

3.4.3 Windows

As described earlier, event streams represent a continuous set of event objects and depending on the used operators, a partitioning of the event stream might be necessary. The partitioning is done by applying a concept called *temporal context*. According to Etzion [Etzion and Niblett, 2011], a context is a named specification of conditions to group events and process them in a related way. The most relevant context dimension to partition event streams is the temporal dimension, which also builds the basis for complex event processing. As defined earlier, every event is related to time and carries a timestamp. Etzion defines a temporal context as a set of one or more time intervals, where each time interval contains events that happened during this interval.

Windows form the implementation of the temporal context concept in event processing systems and are crucial, as for blocking operators a partition of the event stream is obligatory. In addition, windows are used to further refine a relationship between events, as some relations in the real-world underlie temporal constraints. According to Golab [Golab and Özsu, 2005], two types of windows can be differentiated: *time-based* and *count-based*. A time-based window retains only those elements that arrived in the defined time interval, whereas a count-based window always retains the defined number of most recent items. An example for a time-based window is the restriction to only consider events that arrived within the last five minutes, whereas an example for a count-based window is to always examine the last five elements. The most frequent implementation for both windows is a sliding window, where the partition moves over the event stream by time and new events push old elements out of the window.

3.5 Complex Event Processing Engines

Defining the processing logic of an event processing application by means of EPNs represents an abstract, high-level description independent of implementation details. Depending on the processing engine used to execute the logic, EPN components are already implemented and need to be orchestrated according to the EPN. One technology capable of implementing processing logic for event stream processing is complex event processing.

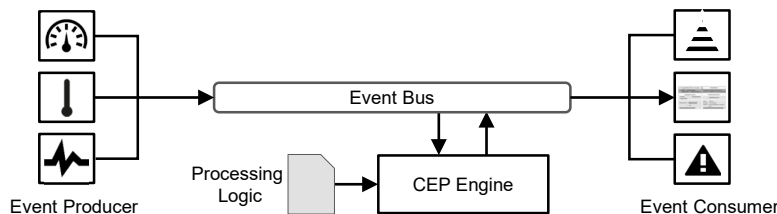


Figure 3.4: Basic Complex Event Processing Architecture

Figure 3.4 shows the components of state-of-the-art CEP architectures: the event producers, an event bus, a complex event processing engine, a description of the event pattern and the event consumers. Event producers observe relevant aspects of an environment and produce events to signify a happening. Events are handed over to an event bus, where other components access the events to process them. Events on the event bus are consumed by event consumers, e.g. to inform a user about a happening in the environment. Complex event processing engines build the fundament of the CEP model as they realize the actual processing of events and incorporate all introduced principles related to it. Within the architecture, a CEP engine acts as an event consumer as well as an event producer. The CEP engine serves as a processing system, which is capable of executing the processing logic to detect patterns that were previously described with an EPN. This processing logic is described in a platform dependent and specific way, e.g. program code or queries, that can be deployed and executed in an engine.

To evaluate the deployed patterns, CEP engines use different approaches to represent the processing logic internally. The two most frequent approaches of CEP engines are non-deterministic finite automata (NFA) and trees [Flouris et al., 2017]. Within a NFA, the states represent the complex events or parts of it and each transition indicates that a related event of a pattern was found in the event stream. Within trees, the leafs represent simple events and internal nodes the operators that define the relationship between those events. Independent of the internal representation, state-of-the-art CEP engines are optimized for the processing of high-volume event streams and are capable of processing hundreds of thousands of events per second.

Complex event processing engines are responsible to filter and combine events accordingly to the defined processing logic. They perform the filtering, aggregation and correlation on-the-fly on the event stream as the events flow from their producers to consumers in order to detect patterns [Cugola and Margara, 2012]. Therefore, two steps are performed every time a new item enters the engine: detection and production, where the logic defines what has to be detected and what has to be done once it was detected. In the following, different approaches to describe patterns and actions are presented and discussed, as the processing logic and its representation are one of the central components in a CEP architecture.

3.5.1 Describing Patterns

Processing event streams correctly requires a description of the processing logic. Three different programming models are distinguished and illustrated in Figure 3.5: imperative languages, declarative languages and visual interfaces. As shown, all three differentiate especially in the dimensions of simplicity and flexibility. All approaches have their advantages and disadvantages that impact their suitability for efficient complex event processing. Flexibility refers to the ability of a programming model to allow the definition of arbitrarily complex processing logic. Simplicity refers to the usability of the programming model and how easy it is to define the processing logic. This aspect is very important, as the logic and patterns are defined in an expert-driven manner by human beings. Languages used to define event processing logic and especially patterns are called *Event Pattern Languages* (EPL).

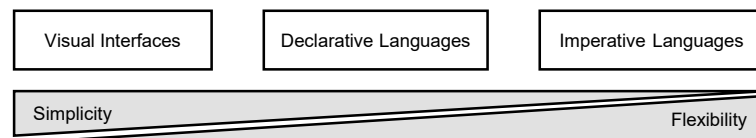


Figure 3.5: Comparison of Programming Models

According to Luckham [Luckham, 2002], [Luckham, 2011], EPLs should fulfill the following requirements:

- **Power of Expression:** Specifying complex patterns over events requires powerful operators. An EPL must support powerful operators to describe relationships between events, e.g. relational operators, temporal operators, filtering and time related constraints.
- **Notational Simplicity:** An EPL should allow users to define complex patterns easily and succinctly. Also, it should be typed to avoid errors in writing patterns, i.e. event objects belong to defined event types.
- **Precise Semantics:** An EPL should have a mathematically precise concept of match making, i.e. the event sets that match a pattern are known.
- **Scalable Pattern Matching:** An EPL should come along with an efficient pattern matching engine that is scalable with regard to high-volume event streams. The design of the language should be chosen so that no negative impacts on the performance arise.

Comparing the programming models with regard to the requirements, many differences are observable. As shown, imperative languages offer the highest flexibility, as the complete processing logic is written in program code and developers specify how events are processed. This flexibility and mightiness comes at the cost of poor notational simplicity and usability, as it can be overwhelming for users without programming experience. This limitation restricts the usability of the system to technical users. The complexity of imperative languages makes the process of pattern definition very time-consuming. Lastly, it is also difficult to automatically optimize the resulting code for scalable pattern matching, as the sheer amount of possibilities offers too many variations.

The other extreme are visual interfaces, which offer a graphical tool. They come with a high usability and notational simplicity, but at the cost of flexibility. Visual interfaces are intended for users without any technical experience and allow to definition of simple patterns by guiding through the process. This simplicity comes at the cost of restricted mightiness in the processing logic and processing possibilities.

Declarative languages are a reasonable compromise between imperative languages and visual interfaces, as they allow defining what to do without describing how to do it. Etzion [Etzion and Niblett, 2011] differentiates two language types of declarative languages: *rule-oriented languages* and *stream-oriented languages*.

Production rules, active rules and logic programming rules are all three different approaches of rule-oriented languages. Production rules work in a forward-chaining way and are of the type *if ... then ...*. Active rules have their roots in active databases and follow the event-condition-action principle: with the occurrence of an event, e.g. an update of a specific database table, a condition is evaluated and if it is satisfied, the action is triggered. Lastly, logic programming rules are based on logical assertions.

Stream-oriented languages on the other side are based on data flow graphs, where nodes represent processing elements and edges represent the data flow. Continuous queries are represented with stream-oriented languages and allow more flexibility than visual interfaces. Stream-oriented languages have a clear syntax that often has a straightforward structure and are therefore easy to use. The languages are often based on well-known existing languages, like the Structured Query Language (SQL) and can therefore be adapted very quickly, even by users without deep technical skills. As a declarative language, stream-oriented languages allow the automatic query optimization within the CEP engine, as the engine decides how to execute the defined logic in an optimal way. The resulting queries are deployed on a CEP engine to be executed. The execution within CEP engines follows a similar principle as the execution of standing queries in DSMSs, where the query is static and the dynamic data stream is routed through the query. According to [Engel and Etzion, 2011], the major breakthrough that turned event-driven applications pervasive and led to a huge attention, was the development of models and tools to easily express an information need and have it executed by a reactive system with a very good performance. Additionally, according to Chandy [Chandy and Schulte, 2009], the declarative definition of event patterns supports highly the effectiveness of CEP systems, as they reduce the development time significantly and still offer a high expressiveness.

With regard to the previously mentioned advantages of declarative languages and especially the possibility for non-technical users to define very powerful queries, this thesis focuses on queries as the main approach to define patterns, respectively situations of interest. Queries are similar to rules, as they also define a complex condition between events in an event stream and an action that describes how to react in case of an occurrence. Additionally, queries also describe further aspects, like the required input streams that are considered for the pattern detection and whether further filters are required for the input stream.

4

State of the Art

This chapter gives an overview of relevant work in fields of interest for this thesis. In Section 4.1, approaches are presented that emphasize the importance of partial patterns and make use of them for various use cases. Situation awareness and its application is presented in Section 4.2, whereas Section 4.3 covers existing work in the field of proactive event-driven computing. Each section is followed by a discussion, analyzing the approaches and their gaps that are tackled within this thesis to implement an event-driven situation management system.

Figure 4.1 illustrates the relationship between identified problems in Chapter 2 and the fields of interest. Partial pattern fulfillment is considered as the mechanism of choice to introduce transparency into CEP engines during run-time. Transparency itself represents a fundamental prerequisite for situation awareness and predictions. Situation management bases its elementary mode of operation on situation awareness, therefore existing situation awareness systems are analyzed. Proactive event-driven computing is examined to tackle the challenge of predictions, as they form the highest level of situation awareness and are required for a holistic situation management system.

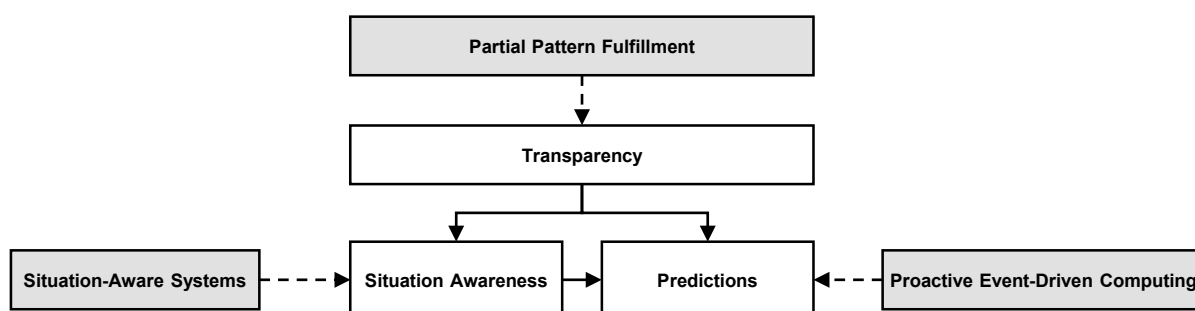


Figure 4.1: Relationship between Fields of Interest and Goals of Thesis

4.1 Transparency: Partial Pattern Fulfillment

Examining the partial fulfillment of patterns is a powerful analysis method to implement higher-level capabilities. This section analyzes existing work that uses the partial execution of patterns and processes to improve the pattern management in CEP systems, optimize their query execution or predict single aspects, like the remaining cycle time of a process.

One application field for partial pattern fulfillment is pattern management. The goal of pattern management is the efficient generation, maintenance and evolution of patterns [Sen and Stojanovic, 2010]. In [Sen et al., 2010], the authors describe an approach for a recommendation-based pattern generation to increase the relevance and efficiency of new patterns and reuse existing knowledge. An RDFS-based representation of patterns is used to calculate similarities between them in order to identify existent patterns to be reused and spot missing ones. The partial or total reuse of patterns is done in three steps: i) the domain expert defines a sub-pattern with the newly introduced representation of patterns that is expected to be found in a repository, ii) the underlying system finds

equal and similar patterns and presents the results by using a ranking function, iii) the user selects which patterns or sub-patterns should be reused. Additionally, this approach is also used to identify missing patterns. Here, two levels are distinguished, the conceptual level and the instance level. Differences on the conceptual level refer to a structural mismatch, while the instance level describes differences on the level of property value definitions. This approach uses partial patterns at design time, whereas [Sen and Stojanovic, 2010] build upon the defined pattern model and present an approach for the refinement and evolution of patterns during run-time. Therefore, after the deployment in a CEP engine, execution statistics are collected to give insights to a pattern engineer, which parts of a pattern might be old and require a modification, as for example in the past they were executed more frequently. To enable the collection of execution statistics, the authors adopt a state-of-the-art CEP engine to notify an additional component once a transition is triggered in the underlying finite state machine used for the evaluation of patterns. By means of this mechanics, partial pattern matches become observable.

Another application field using the insights from sub-patterns is the optimization of query executions in CEP engines. Various works, like [Wu et al., 2006], [Agrawal et al., 2008], [Ma et al., 2015] and [Zhang et al., 2017] follow the assumption that multiple queries in one environment share certain aspects and can be optimized by reducing redundant computations. In [Zhang et al., 2017] for example, the authors describe three sharing techniques to improve the execution, namely merge, decomposition and operator transformation. The merge sharing technique follows the principle to merge the results of one query into another, which is possible for queries with the same pattern and identical time window constraints. Sharing results between non-identical, but similar queries is implemented through the decomposition sharing technique, where one pattern is broken down into multiple sub-patterns. The goal is to find common sub-patterns between two queries and to share the result of the sub-pattern execution, thus, it has to be executed only once. The sub-query decomposition is transformed into the task of finding all common sub-strings, i.e. not all sub-patterns are generated, but only the common ones between two queries. The last sharing technique (the operator transformation) translates operators into other representations to find more common sub-patterns, as only combinations are found that use identical operators. The whole solution is implemented and tested for SAP ESP. Similar to this approach, in [Ma et al., 2015] the authors use sub-pattern, prefix and suffix relationships between multiple patterns to improve the evaluation performance. Therefore, based on the relationships between various patterns, a new automaton is derived by summarizing shared automata states between patterns to execute multiple sub-patterns and patterns at once with every single evaluation step. As previously, only common sub-patterns of queries are taken into consideration. The approach led to the development of a new CEP engine that still has to prove its value in real monitoring environments.

The prediction of specific aspects based on partial pattern fulfillment is another relevant application field. The authors in [Terroso-Sáenz et al., 2016] introduce an approach using existing regular reactive patterns to derive predictive patterns for situations. The solution supports sequence and conjunction patterns and tracks the partial execution of patterns to keep track of occurring frequencies. The probability of occurrence is calculated by comparing the frequencies of sub-patterns and their related pattern. Once the probability is higher than a defined threshold, the derived event is synthetically generated, annotated with the probability of occurrence and send to the event bus. The handling of probabilistic features of the newly derived events is not described further. The solution is implemented in a custom CEP engine. A similar approach in another domain is introduced in [van Dongen et al., 2008] and [Van der Aalst et al., 2011] for predictions in business process management systems. In both approaches, the execution of a process is tracked and the transition times between different tasks in the process are logged. Based on the history of past process executions, a transition system is generated describing the different partial process executions and their time characteristics. To predict the completion time of a specific process instance, its partial trace (i.e. the sequence of tasks executed so far) is mapped to a state in the transition system and the prediction is calculated on basis of the state's time statistics.

Discussion

Considering all the efforts that have been made in the past, the potential of the evaluation of partial patterns has been recognized for various purposes. All presented publications have one common aspect: the increased transparency by analyzing sub-patterns allows existing systems to be further enhanced with new capabilities. In [Sen et al., 2010] and [Sen and Stojanovic, 2010], the authors use partial pattern fulfillment for pattern management. The approach is implemented for a single CEP engine by modifying the core of the engine, the correlator. The developed solution in this thesis uses partial pattern fulfillment to realize transparency and observe intermediate states of processing. In contrast to the presented approach, the approach in this thesis does not require any modifications of the CEP engine to extract run-time statistics for patterns. It follows a generalizable approach that works with multiple CEP engines and is based on the pattern definition of a situation. Therefore, it tackles the challenging question how various engines with differing event pattern languages can be supported in one approach. The step-wise extension to support further engines requires only a negligible manual one-time effort per engine. This aspect removes the limitation of the previous approaches that one specific CEP engine has to be used and overcomes a barrier for the further usage of advanced complex event processing systems. The functionality of evaluating partial pattern fulfillment in this thesis is implemented by analyzing and decomposing queries of situations of interest. The decomposition strategies used in the presented query optimization works, like [Ma et al., 2015] and [Zhang et al., 2017], are not suitable for the purposes of this thesis. Contrary to optimization tasks, the observation of intermediate states during the pattern execution requires all valid sub-patterns to be known, whereas for optimizations it is sufficient to derive the common sub-patterns of multiple patterns. Consequently, this thesis introduces a decomposition approach based on the pattern definition to generate all sub-patterns of interest.

With regard to predictions based on partial matches, the approach in this thesis is capable of delivering the same functionality, as all relevant statistics, that are used in the presented work, are captured. Different to [Terroso-Sáenz et al., 2016], the approach in this thesis does not generate a synthetic derived event with a probability, but notifies relevant components by materializing a prediction event, as otherwise every event consumer has to be adopted to process probabilistic events correctly. The prediction on basis of the observed history allows a fast and simple projection into the future, but due to its simplicity also represents a limitation as it has the potential to lead to many false positives. The performance of the prediction model varies significantly from use case to use case. In the developed approach in this thesis more flexibility is offered by allowing the prediction model to be more sophisticated, e.g. neural networks, and by fully integrating the prediction aspect into the dynamic life-cycle extension that is adoptable to the needs of the current use case. With regard to the partial evaluation of pattern executions in [Terroso-Sáenz et al., 2016], a comparison is not possible, as the authors do not describe their partial execution in detail. It is only known, that a new CEP engine has been developed to incorporate the functionality.

Concluding from the existent work, employing partial pattern fulfillment leads to a variety of possible enhancements for complex event processing. The transparency increase through the evaluation of partial pattern matches builds the core for all further extensions in this thesis, as it allows the observation of intermediate states during the execution of queries in a CEP engine. Therefore, a conceptual extension is required that enables the observation of intermediate states in various CEP engines.

4.2 Situation-Aware Systems

Situation awareness is mentioned in countless publications, indeed it is not always used as a defined concept but rather as common knowledge. [Stojanovic and Artikis, 2011] gives an overview of existing research challenges in applying complex event processing for real-time situational awareness, referring to the ability of CEP systems to recognize situations of interest in real-time. The concept behind the term "situation awareness" is not specifically introduced. One aspect identified by the authors, is the potential of CEP systems to become situation-aware on a further level. Instead of just recognizing the occurrence of a situation, CEP systems need to be further developed to exploit the insights that are already gained during the query execution to find weak signals that indicate a situation. By refining the mode of operation, CEP systems are capable to become future-situation-aware, which conforms the ideas of the psychological model of situation awareness.

Various publications, like [Jakobson et al., 2006], [Nilsson et al., 2008], [Salfinger et al., 2014], [Costa et al., 2016] are based on the concept of situation awareness defined by Endsley in [Endsley, 1995], introducing the psychological model. In [Nilsson et al., 2008], the authors present a user study to conceptualize knowledge for a rule-based application supporting operators in maritime surveillance with situation awareness. On basis of a non-specified agent framework, a prototype is developed to describe relations between entities by means of a visual rule editor. The rules are used during run-time to recognize occurring situations and inform human operators. In [Salfinger et al., 2014], a situation evolution model is introduced for a rule-based system to capture a situation's dynamic development. Therefore, a situation is tracked through different states of evolution, namely the trigger, climax and clearance phase. The authors introduce transitions between the phases by manually defining the conditions that lead to a specific phase. The prototype is implemented on basis of the JBoss Drools rule engine and specifically suited for a traffic monitoring use case. A survey on situation-aware systems has been compiled in [Salfinger et al., 2013]. With regard to the activities of acquiring situation awareness and maintaining it, the authors identify criteria to evaluate systems with regard to both dimensions. The authors evaluated eleven existing solutions, coming to the conclusion that no system supports the acquirement and maintenance to its full extent.

A strongly related field to situation awareness is situation management. It is understood as an integration of situation awareness into a framework that is suitable for an implementation through a computer system. In [Jakobson et al., 2006], a basic framework for situation management is introduced and the basic elements of situation modeling are defined. Finally, the authors describe how a possible implementation of a situation management system might be implemented on basis of event correlation without developing a prototype or specifying a technology. In [Costa et al., 2016], the authors implement a situation management system on basis of complex event processing, following the ideas of Jakobson in [Jakobson et al., 2006]. Therefore, the authors introduce a life-cycle for situations and implement the possibility to track an active situation through this life-cycle on basis of the JBoss Drools engine. The implementation uses annotations in rules to configure the situation tracking. In addition, the action statements of rules are adopted to register occurring situations in the situation management component. The tracking is conducted by observing the event attribute values after the detection of the situation.

Discussion

This thesis aims to combine the mental capabilities of operators with the real-time monitoring capabilities of complex event processing. Therefore, in contrast to [Stojanovic and Artikis, 2011], this thesis is based on the fundamental definition of situation awareness as given by Endsley in [Endsley, 1995], to explicitly define its meaning in CEP systems. The goal is to extend it further to allow situation management according to [Jakobson et al., 2006], which also integrates the projection into the future as identified as a research challenge by Stojanovic.

The approach within this thesis aims to be generalizable and valid for CEP applications in general. Therefore, in contrast to [Nilsson et al., 2008] and [Salfinger et al., 2014], situation awareness and the ability to track a situation's evolution are based on the conceptual level of patterns and situations and are independent of specific applications for single use cases. By establishing an approach on the abstract concept of a situation of interest, its related query and its life-cycle, the solution abstracts from the specific use case and allows its usage in various domains. The introduction of abstract states in the situation life-cycle allows the solution in this thesis to automatically derive all relevant situations that describe an intermediate condition. Thus, contrary to [Salfinger et al., 2014], no manual definition of interesting intermediate states is necessary.

With regard to situation management systems, Jakobson defines a framework that builds the basis for this thesis and describes the basic requirements. A possible implementation is described by means of events, which represents the goal of this thesis: implementing a holistic situation management system based on complex event processing. The holistic management of situations comprises three different aspects: the investigative, predictive and controlling situation management. A first step towards an implementation is done by Costa, covering only a sub-set of required capabilities. For a better controlling situation management, Costa introduces a situation life-cycle to track the evolution of abstract situations, respectively their related attribute values after their occurrence. Compared to this approach, the introduced solution in this thesis enables a fine differentiation between situation states before its occurrence. One additional limitation is given through the dependency towards specific rule engines in [Nilsson et al., 2008], [Salfinger et al., 2014] and [Costa et al., 2016]. The developed solution in course of this thesis supports multiple CEP engines by transforming engine-specific query representations into an abstract unifying model for situations and deriving all required sub-queries for the observation of intermediate states.

In summary, all presented work delivers valuable insights for this thesis and the further development of a holistic situation management that is able to handle big data. The introduced work is in particular used to derive several requirements on situation-aware systems that are further described in Chapter 5.

4.3 Proactive Event-Driven Computing

The disadvantage of the reactive nature of state-of-the-art CEP systems was perceived in the scientific CEP-community. This led to the lately trending topic of evolving CEP from a reactive view to a proactive one. Several publications describe a vast number of approaches for proactive CEP applications, e.g. [Tóth et al., 2010], [Engel and Etzion, 2011], [Wang and Cao, 2014], [Fournier et al., 2015] and [Mousheimish et al., 2017].

The first works considering complex event processing and predictive analytics together are [Fülöp et al., 2010] and [Tóth et al., 2010]. The former is a survey of both fields coming to the conclusion that a combination of both has significant potential to improve complex event processing by enabling proactivity and predictive analytics by capabilities to work with big data. This idea is picked up by Tóth to describe a conceptual architecture to connect a predictive component with an existing CEP system by using complex events for the training of multiple predictors. This approach is extended in [Fülöp et al., 2012] to sketch a possible implementation of the conceptual architecture. Therefore, the authors introduce a new event processing network, namely predictive EPN, which is responsible to forward composite events to a predictive component allocated outside of the CEP system. Following the approach of separating the event processing from the prediction, the authors in [Schwegmann et al., 2013], introduce an architecture where solely the outputs of the CEP engine are used for the training of a predictive model and no further insights into the execution are required. The approach has its limits, as it is only used to predict specific key performance indicators for business processes that are monitored by means of a CEP application.

In contrast to outsourcing the prediction capabilities, various publications are based on the abstract concept presented in [Engel and Etzion, 2011]. The authors introduce a new event processing agent. The proactive agent is kept abstract on purpose to allow multiple implementations of predictive models. On basis of the new proactive agent, in [Engel et al., 2012] the authors describe a basic model for proactive event-driven computing. The traditional operational pattern of CEP systems is complemented by a new phase: Detect – *Forecast* – Decide – Act. The basic model distinguishes between reactive situations, as known from CEP systems, and proactive situations, that deal with their prediction and proactive reaction. Therefore, in contrast to reactive situations, proactive ones are extended by a probability of occurrence and if possible the costs of a situation occurrence to enable intelligent decisions. The basic model is applied with modifications and differing predictive models in various use cases. In [Metzger et al., 2012] and [Feldman et al., 2013] a domain-specific event processing platform for the transport and logistics domain is introduced. The FInest platform is based on the concept of proactive agents described by Engel. Therefore, the authors extend an existing event processing engine to enable probabilistic rules to describe new predictive patterns for situations of interest. Within the predictive patterns, the results from prediction models are used as their input. Another platform, described in [Artikis et al., 2014], [Fournier et al., 2015] and [Kibangou et al., 2017], is the SPEEDD platform for proactive event-driven traffic management. The authors describe a proactive architecture consisting of four building blocks: event processing, event recognition & forecasting, time decision making and visualization. The core of the prediction capabilities is implemented by IBM Proactive Technology Online (Proton)¹ as its CEP engine.

A third way to implement proactivity in CEP systems is to base it on the existing and already used reactive queries to detect situations of interests. One recent approach is presented in [Mousheimish et al., 2017]. The authors describe a solution combining Early Classification on Time Series (ECTS) techniques with complex event processing. The input for the solution are classified multivariate time series. Historic event traces within the CEP system are used as time series data and are classified manually by a domain expert. The goal is to learn predictive patterns to classify a scenario as early as possible. In a succeeding step, the predictive patterns are automatically transformed into CEP queries, which are deployed in an engine. The underlying technique to establish proactivity is based on shapelets, i.e. special shapes in time series data that discriminate different classes of data [Ye and Keogh, 2009]. Therefore, the first step comprises the univariate extraction of shapelets, whereas in the second step, sequences of shapelets are generated to represent the multivariate time series that serves as input. The new representation defines an abstract version of the pattern that also matches similar patterns in the event stream. In the last step, shapelets are transformed to simple CEP patterns representing the single shapelets and complex CEP patterns to describe the sequence of shapelets. The derived patterns are enrolled in a CEP engine to classify the event stream with regard to the trained situations.

Discussion

The projection into the future is an important requirement for a holistic situation management system as targeted within this thesis. As became apparent by analyzing related work for the prediction of situations, currently there is no general solution that fits all use cases related to the detection of situation occurrences. Dependent on the requirements, different predictive models are applied and adopted to the specifics of the problem at hand. Therefore, this thesis follows the approach of Engel in [Engel and Etzion, 2011] and does not predetermine a predictive model, but introduces supportive capabilities to ease the training of adequate predictors and their monitoring. In contrast to Engel, this thesis follows the approach of separating the prediction from the event processing as described in [Fülöp et al., 2010] and applied in [Tóth et al., 2010], [Fülöp et al., 2012] and [Schwegmann et al., 2013].

¹ <https://github.com/ishkin/Proton>

Decoupling the predictive capability from the reactive nature of complex event processing comes with the advantage that the engine itself does not have to be modified. All related work that is based on the introduction of a proactive event processing agent as shown in [Engel and Etzion, 2011] encompasses a modification of the core technology of the underlying CEP engine, which comes with the risk of negatively impacting relevant characteristics, e.g. the throughput. The assumption within this thesis is, that extending an CEP architecture by an additional layer without impacting the basic system leads to a broader dissemination, as the existing system is kept running. Additionally, solutions with predetermined models are limited to specific use cases, e.g. as the presented solutions in [Metzger et al., 2012] and [Kibangou et al., 2017]. While in [Metzger et al., 2012] and [Feldman et al., 2013], probabilistic rules are introduced to define predictive patterns manually, [Mousheimish et al., 2017] follows the approach to use reactive patterns to automatically derive predictive patterns. Within this thesis no specific prediction model is introduced, as depending on the use case different models are suitable. Nevertheless, the approach follows the idea of Mousheimish, that reactive patterns allow significant insights for prediction models.

The presented publications form an important basis for situation-aware applications, as they already tackle the challenge of missing projections. Within this thesis, basic design principles, like the decoupling of predictions and event processing, are taken over from the related work. In contrast, this thesis does not define a predictive model, as every use case requires an individual model. Nevertheless, the developed approach supports domain experts in deriving relevant data to train an appropriate model and evaluate it in a live environment with continuous monitoring support. The resulting architecture is designed flexible, thus existing predictors can be integrated.

4.4 Conclusion

The overall goal of this thesis is the implementation of a situation management system by considering the human aspects of situation awareness and combining them with the technical possibilities of complex event processing by extending the existing state-of-the-art of event processing. This requires a better understanding of situation awareness as it is a prerequisite for a holistic management of situations. Supporting situation awareness to its full extent requires the provision of prediction capabilities. Therefore, proactive event-driven computing has been analyzed to extract suitable concepts for the further development in the course of this thesis. The analysis of situation-aware systems has targeted the goal to derive suitable approaches and requirements for the further enhancement of CEP systems. The analysis has shown, that especially the tracking of evolving situations is an important, but missing capability of existing environments and requires an increased level of transparency, which leads to the field of partial pattern fulfillment. Being able to transparently track a situation's evolution through an extended situation life-cycle requires the observation of intermediate states during its detection. Sub-patterns are a suitable method to implement this requirement, but are not originally supported by CEP engines. Consequently, an extension is required for a general observation support for partial pattern matches across multiple CEP systems.

Part III

Main Part

5

Requirements

Within this chapter, requirements towards the approach in this thesis are derived. The research questions defined in Chapter 1 build the basis for the requirements elicitation process. Additionally, requirements are refined by analyzing the motivating scenario and identified problems in Chapter 2 and the related work in Chapter 4. First, a short introduction is given into the requirements elicitation process and how requirements can be derived from existing artifacts, before describing the resulting requirements.

5.1 Requirements Elicitation Process

Requirements play an essential role in the development of software systems. In [IEEE, 1990], a requirement is defined as "*a condition or capability needed by a user to solve a problem or achieve an objective*". Knowing the requirements towards a system is prerequisite for the development and design of the system and its capabilities. The requirements elicitation process comprises multiple tasks to uncover the needs for a system. According to [Zowghi and Coulin, 2005], five general tasks during the process are distinguished: i) understanding the domain, ii) identifying sources of requirements, iii) analyzing the stakeholders, iv) selection of techniques for the elicitation and finally v) eliciting the requirements.

Within this thesis, the tasks required for the elicitation are conducted in different chapters and sections. Understanding the domain refers to the contents of the introduction in this thesis, in particular Chapters 1, 2 and 3 give detailed insights into the domain from different perspectives. Existing work, systems and concepts from related fields serve as a source for requirements and are analyzed in Chapter 4. Potential stakeholders are one important source for requirements, therefore existing surveys eliciting stakeholder needs are taken into consideration. As described in Section 1.2, this thesis follows the design science paradigm for information systems research that relies on analyzing the existing knowledge base. Therefore, the technique used for the elicitation is a deep analysis of existing works to derive further requirements towards the solution. Figure 5.1 illustrates the mapping between derived requirements and their related research questions. Based on the described activities and their implementation in this thesis, the requirements for a situation management system on basis of complex event processing are described in the next sections.

5.2 Capability Requirements

The requirements within this thesis are categorized into two groups. The first set of requirements refers to the capabilities of the solution and is presented in the following, whereas the next section covers the needs towards the system design and architecture. As presented earlier, in [Salfinger et al., 2013] the authors compiled a survey by evaluating eleven situation aware systems according to a catalog of criteria. These criteria is now used to derive required capabilities. The requirements are classified into two groups with regard to the task they focus on: gaining situation awareness or maintaining it.

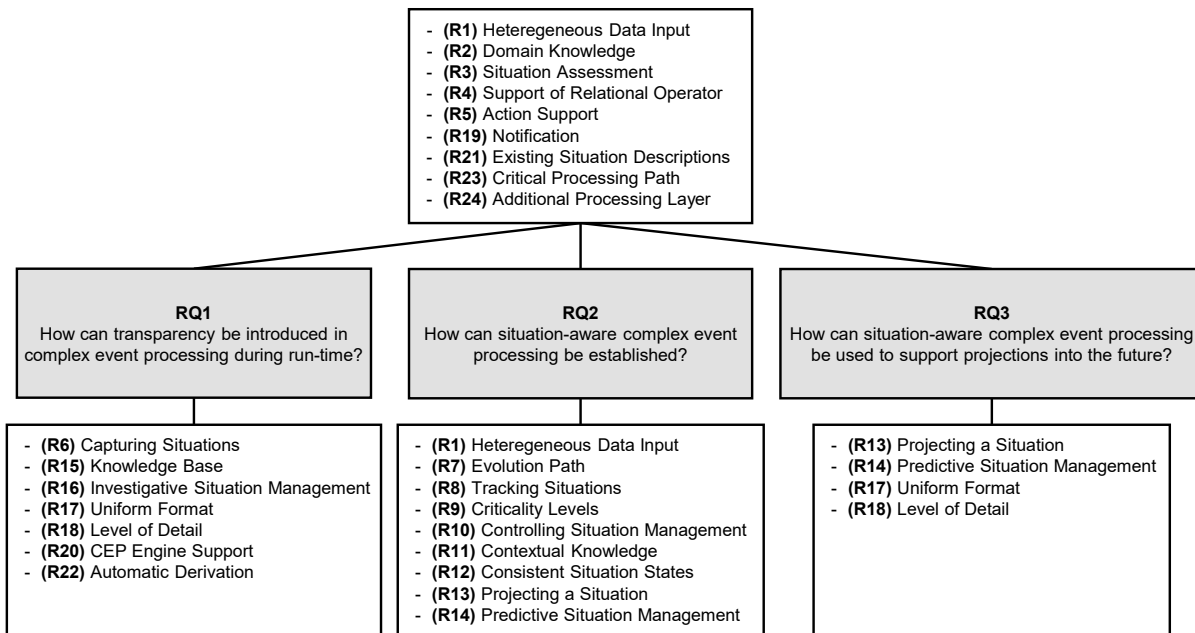


Figure 5.1: Research Questions and related Requirements

Gaining situation awareness strongly relies on the processing of relevant data. To support various application fields, the solution needs to be flexible with regard to the representation of use case specific aspects. In manufacturing scenarios as presented in Section 2.2, various sensors with differing data types are used to monitor the environment and their measurements are interrelated to define a situation of interest. Supporting heterogeneous data types is an important aspect for dynamic systems. It increases their complexity, but in parallel improves flexibility to make systems suitable for various use cases and broaden their field of application. Another important capability for acquiring situation awareness is the incorporation of domain knowledge into the acquisition process, i.e. allowing domain experts to model their knowledge to describe the situations which form the basis for situation awareness. Thus, to support a broad field of use cases, the following two requirements have to be fulfilled:

Requirement R1: Heterogeneous Data Input

Heterogeneous data types are supported as input by the system.

Requirement R2: Domain Knowledge

Use case-related domain knowledge can be incorporated in the system.

Gaining situation awareness refers to the capability of detecting an occurred situation. Therefore, a system has to evaluate whether data has been detected that indicates an occurrence under consideration of the underlying data. According to Salfinger, most systems conform a template-based approach to describe an a priori specification of a situation of interest [Salfinger et al., 2013]. In the introduced running example in Section 2.2, a rule-based implementation defines the situation of a dull tool. Describing conditions that signify a situation requires the system to support typical relational operators as introduced in Chapter 3. The running example makes use of two relational operators: sequence and conjunction. By linking situation detection to actions as a reaction to it, situation aware systems are able to implement Boyd's Observe-Orient-Decide-Act (OODA) loop [Boyd, 1996]. An action of a system might also be just a notification of the operator, as presented in the running example. The described demands result in the following requirements:

Requirement R3: Situation Assessment

Situation occurrences can be detected by the system through evaluating the data.

Requirement R4: Support of Relational Operators

Typical relational operators are supported by the system.

Requirement R5: Action Support

The system gives recommendations for actions or triggers them as a reaction to a detected situation.

According to [Nilsson et al., 2008], time represents an important aspect, as situation awareness fluctuates with the evolution of the monitored environment. Therefore, it is not sufficient just to gain situation awareness, but to maintain it over time. This requires a method to capture a situation's current state and to track its evolution according to potential evolution paths. These needs also relate to the identified problems in Section 2.3.1 and 2.3.2. Within the running example, a dull tool situation does not occur suddenly, but evolves over time by passing multiple intermediate states. For an effective support of human operators, the system has to support different criticality levels that relate to the intermediate states during the evolution of a situation. This aspect assists in the purposeful direction of attention. From a situation management perspective, the system uses the mentioned capabilities to implement a higher level functionality in terms of allowing the controlling of situations [Jakobson et al., 2006].

Requirement R6: Capturing Situations

The system captures a situation's current state.

Requirement R7: Evolution Path

The system knows potential evolution paths for a situation.

Requirement R8: Tracking Situations

The system is capable of tracking a situation's state through evolution paths.

Requirement R9: Criticality Levels

The system supports different criticality levels for the evolution of a situation.

Requirement R10: Controlling Situation Management

The system supports the controlling of situations.

A generalizable solution needs to abstract from use case specifics, nevertheless has to allow the incorporation of contextual knowledge in the process of maintaining situation awareness. Contextual knowledge in this case refers to the experience of domain experts to adjust the system based on environmental characteristics, e.g. readjust thresholds in accordance to it. In contrast to requirement R2 (Domain Knowledge), which targets the description of a situation of interest by a domain expert, this requirement aims for the experience of domain experts. A situation might be defined equally for various manufacturing environments, but the situation management system requires an individual configuration, as for example a machine outage is more expensive in another setting. Therefore, the system requires a mechanism to support the fine-tuning of the whole situation management, e.g. by adjusting the evolution path and criticality levels dependent on the use case at hand and the required information level.

Requirement R11: Contextual Knowledge

The system supports the incorporation of contextual knowledge.

During the tracking of the evolution of a situation, inconsistencies may occur, e.g. due to incompleteness or expiring time windows. As an example, the situation's state is captured and tracked through the evolution path but does not develop further from a specific point. This might be because of missing information that the situation has further evolved or the situation has stopped its evolution. The system needs to ensure that situations are reset to a consistent state after a certain time to signify the human operator that no evidentiary observations are made that indicate a potential occurrence of the situation.

Requirement R12: Consistent Situation States

The system ensures the consistency of situation states.

A projection into the future forms the highest level of situation awareness. It represents the most difficult level to achieve [Salfinger et al., 2013], as a wide variety of prediction models exist. The evolution paths of a situation need to take projections into consideration. From the perspective of situation management the system needs to support predictive situation management [Jakobson et al., 2006], which encompasses the integration of prediction models to predict a situation's potential development. Introducing projections leads to proactivity, consequently, a human operator gets informed about an upcoming situation and is able to react prior to its occurrence.

Requirement R13: Projecting a Situation

The system supports the concept of projections in a situation's evolution paths.

Requirement R14: Predictive Situation Management

The system supports the prediction of the future state of a situation.

An additional demand resulting from the situation management perspective is the capability for investigative situation management [Jakobson et al., 2006]. It allows the retrospective analysis of occurred situations to derive further knowledge and draw conclusions for refining situation descriptions. Allowing the analysis requires the system to collect relevant data for defined situations to form a knowledge base and persist historic data. Due to the amount of generated data nowadays, only situation-relevant data has to be stored. The knowledge base also serves as a source of data for the definition and continuous refinement of prediction models to implement the previously mentioned projection capability. In addition, the system eases the analysis of collected data by offering a structured format, as differing data formats represent a barrier for the analysis process [Schwegmann et al., 2013]. With regard to the level of detail that is collected by the system, [Jagadish et al., 2014] states that different levels have to be supported by the system to satisfy differing information needs and allow a drill-down into the data. In contrast to domain knowledge and contextual knowledge, the knowledge base refers to collected run-time data.

Requirement R15: Knowledge Base

The system supports the collection of situation related data to form a knowledge base.

Requirement R16: Investigative Situation Management

The system supports the retrospective analysis of occurred situations.

Requirement R17: Uniform Format

The system supports the data analysis by offering a uniform format.

Requirement R18: Level of Detail

The system collects data on different levels of detail and allows a drill-down.

According to [Nilsson et al., 2008], trust and understanding forms the basis for acceptance of situation awareness systems. Therefore, the system has to be able to communicate its findings to a human operator. This leads to the requirement of notifying a user about the current development of a situation and the basis for this evolution.

Requirement R19: Notification

The system informs a user about a situation's development and the underlying data for this observation.

5.3 System-related requirements

This section discusses requirements related to the system design and its architecture. These requirements affect the technical aspects of the solution design. Complex event processing represents a solid basis for a situation management that is capable to handle big data. To implement the capability requirements presented in the previous section, the architecture of a basic complex event processing architecture has to be extended and complemented. Sources for the elicitation of system-related requirements by deep analysis are related work from the fields of partial

pattern fulfillment, situation aware systems and complex event processing presented in Chapter 4. Additionally, requirements are derived from the motivation in Chapter 2 and the foundations of event processing in Chapter 3.

To ensure a broad dissemination of the developed solution, an architecture is required that incorporates the capabilities of a potential existing environment. Therefore, the system has to support existing CEP engines and to incorporate their basic capabilities for situation detection and enhance it for all further functionalities. Ideally, the support is implemented by minimal modifications without the need to adopt all components.

Requirement R20: CEP Engine Support

The system offers a mechanism to support arbitrary existing complex event processing engines.

With regard to the success factors of complex event processing, [Engel and Etzion, 2011] identified the easy expression of an information need even by non-technical users as one breakthrough, that turned event-driven applications pervasive. By using declarative languages to capture processing logic, the programming costs and development times are reduced [Stonebraker et al., 2005]. To leverage this aspect, the system has to use the existing possibilities for describing situations and build its succeeding processing on it. In case further artifacts are required to implement the described capabilities of a situation management system, the developed solution has to derive as much of it as possible automatically. The single point of entry for a user to introduce a situation of interest into the system is the situation description by means of a query.

Requirement R21: Existing Situation Descriptions

The system operates on existing situation descriptions.

Requirement R22: Automatic Derivation

Based on situation descriptions, the system derives further aspects automatically (wherever possible).

The central functionality of an existing CEP environment is the detection of occurring situations. This represents the critical path of the system. At the heart of a real-time processing system is the requirement to keep data moving and avoid costly operations on the critical processing path [Stonebraker et al., 2005]. Any extension implementing the enhanced capabilities of a situation management system has to minimize its negative effects on this path.

Requirement R23: Critical Processing Path

The system minimizes its negative effect on the critical processing path.

Associated with the preceding requirement, [Fülöp et al., 2012] suggests separating the functionality of complex event processing and a predictive component to avoid an affectation on the maintainability of the CEP engine by keeping the complexity low and avoiding a tight coupling of different capabilities. This requirement is also valid for all other capability enhancements, which leads to the requirement of a separate processing layer on top of the underlying basic CEP architecture.

Requirement R24: Additional Processing Layer

The system implements its enhanced capabilities within an additional architectural layer.

6

Transparent Complex Event Processing

In order to tackle the problems introduced in the first part of this thesis, this chapter aims to introduce transparency into complex event processing systems during run-time. The goal is to continuously grant access to the current state of the system and the intermediate results with the intention to further support the understanding of the internals and results. Therefore, Section 6.1 discusses the general concept of *Transparency*, its implications on complex event processing and a conceptual approach to introduce transparency in complex event processing. Section 6.2 describes an approach to implement the previously introduced concept of *Partial Pattern Fulfillment*, whereas Section 6.3 makes use of the newly claimed transparency to analyze the current state and offer users and developers an overview of the system and its current state. In Section 6.4, the technical implementation and its resulting *architecture* are described. Next, in Section 6.5 the introduced concepts are illustrated along the running example in this thesis, before finalizing this chapter with a summary in Section 6.6.

6.1 Transparency

As discussed in Chapter 2, transparency serves as a source of trust and changes the way how users think about the trustworthiness of a system [Kizilcec, 2016]. Understanding the internals of a computer system increases the trust and comprehensibility of results and recommendations given by the system [Glass et al., 2008]. Besides trustworthiness, transparency is a prerequisite for the identified requirements (see Chapter 5) towards a holistic situation management system, e.g. to allow the tracking of its current state. According to Johnson [Johnson and Johnson, 1993], an intelligent computer system should provide knowledge and explanations on the results that are necessary for a user to carry out a given task. Increasing transparency during the processing of event streams, initially requires an understanding of transparency as a concept.

When talking about transparency in computer systems normally intransparency is meant, i.e. the ability of a computer system to hide the internal mechanics for the outer world. The idea behind this ability is to abstract from the specific computing logic and give the user a defined interface that does not change while the internal behavior might change over time. Thus, the user has only to care about the correct usage of the interface and not about how the computing logic is implemented in detail. Originally, transparency coming from the Latin word *transparens* means the property of an object that light can pass through it and one is able to see through it. Depending on the level of knowledge about the internal mechanics, one also talks about *white-box* knowledge if the internal structures and computing processes are known. The term *black-box* refers to the opposite, where the internals of a system are unknown. *Grey-box* describes the combination of black-box and white-box components. Figure 6.1 illustrates the different levels of transparency depending on the type of knowledge.

Within this thesis, whenever the term transparency is used, it means the detailed knowledge about internal happenings. Transparency always refers to a defined *set of characteristics* that should be taken into consideration, e.g. the internal data flow within an application or the intermediate results of some calculations. The first step to gain transparency is to specify an *object under study* and decide which aspects are of interest for a closer examination.

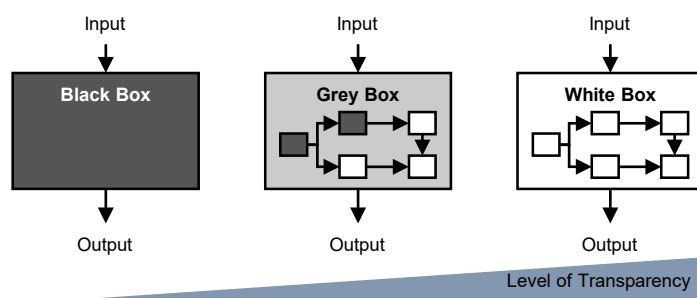


Figure 6.1: Levels of Transparency

In order to become transparent regarding the object under study an a priori knowledge of the object is required [Kalmykov and Kalmykov, 2015]. Based on this knowledge a *model* of the object under study is derived. The model represents a prerequisite for the observation of specific characteristics of an object and for the derivation of further knowledge based on observations. As stated by Seidewitz [Seidewitz, 2003], the model is defined as a set of statements about the object under study and is considered correct if the statements correspond to observations made in the objects environment.

Observations play an important role for transparency, as they evaluate whether a defined model is correct or not. Consequently, it is crucial to make the characteristics described by a model observable, i.e. to make them measurable. Dependent on the domain and the characteristics, observations are made in a broad variety of ways, e.g. through log-files containing intermediate computing results that describe the behavior of a computer system.

Transferring the concept of transparency to computer systems, two different perspectives are taken: transparency at design-time and at run-time. For an application, both views are of interest, as they represent two different types of transparency. Nevertheless, both are heavily related with each other, as transparency at run-time is not possible without transparency at design-time describing the relevant aspects to observe. At design-time the knowledge about internal components and their wirings is of interest, which is given by the previously described model of an object under study, e.g. in terms of program code. At run-time the internal behavior is of interest while the application is processing a set of data. In reverse, the ability to make observations at run-time is very important as it helps to evaluate the statements about the object under study and to check whether the described model at design-time is correct. Beyond that, observations are used to give users continuous transparency at run-time by showing current calculations along with intermediate results to increase the comprehensibility of it.

Developers of computer systems with white-box knowledge of the computing logic often use observations during run-time to debug systems and review the correctness of the program code or architecture (which are considered as the model). Thus, transparency plays an important role for the testing and debugging of computer systems. Certainly, the output of a system itself might be used to evaluate whether it works correctly with regard to a given input. However, in many cases further investigations are necessary to evaluate intermediate steps, build trust and finally guarantee the correctness of it. Depending on the complexity of the computing logic, the evaluation of correctness can represent a tedious task.

6.1.1 Transparency and Complex Event Processing

Considering the previously described concept of transparency and its prerequisites, complex event processing is very well suited to support transparency at run-time with regard to the pattern matching capability. As illustrated in Figure 6.2, it has a clear object under study, namely the situation of interest, and it is appropriately described by

a query, which is considered as the model of the object under study. The query clearly describes the relevant characteristics for the detection of a situation, i.e. the involved events, filters to be applied and the pattern of interest. The task of the query is apparent: the detection of a pattern in event streams to identify situations. Furthermore, queries are described in an event pattern language that has clear meaning and a predefined structure.

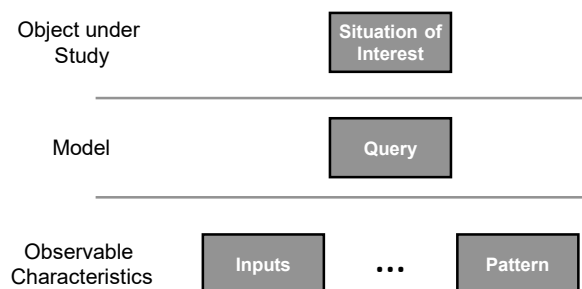


Figure 6.2: Transparency Concept in Complex Event Processing

Nevertheless, state-of-the-art CEP engines are operating in a black-box manner and only allow the observation of the final state of processing, i.e. the identification of a situation occurrence. Although all relevant aspects that allow the observation of a situation and its development are already present, the user is only provided with partial transparency. Therefore, the question arises, how the relevant aspects are made *continuously observable* during run-time. Logging as a concept is considered as best practice for continuous transparency at run-time, i.e. the developer implements a logging functionality at specific points of the processing to understand the intermediate results of the application and track its progress. In the next subsection a similar mechanism is discussed to enable the continuous transparency at run-time for CEP systems and to allow the observation of a situation.

6.1.2 Partial Pattern Fulfillment

Logging as an approach to achieve transparency has a long tradition in the development of computer systems. It is often used to gain insights into the processing either to debug the application or to continuously track the intermediate results and to enable a user to verify results on their correctness. The challenge of logging is, that an expert knowing the processing logic has to define the relevant states which need to be logged. This delimits the possibilities of fully automatic logging.

When it comes to pattern matching, several intermediate results are interesting: the result of a filtering function, an aggregation and the pattern matching process itself. Particularly the results of the match making process are of special interest, as the goal is to observe the situation development on its way to a full match. Reconsidering the way of operation of CEP engines during pattern detection, provides insights to derive the requirement for a transparent processing: within state-of-the-art CEP engines, a pattern is progressively evaluated. Therefore, queries are represented as NFAs or trees to perform match making on these structures. Before a pattern is fully matched, partial matches take place. According to Flouris [Flouris et al., 2017], these partial matches should be monitored, as they express the potential for an imminent full pattern match. In most cases, the internals of the CEP engine are a black-box and the partial matches cannot be observed. Therefore, the question arises, how partial pattern matches are made observable in general, without knowing the internal details of the underlying engine.

Monitoring the execution of queries and their patterns is essential to increase the transparency at run-time. Therefore, in a first step it is necessary to define what has to be observed, i.e. which characteristics are of interest. As the goal of pattern detection is apparent and a pattern cannot match without its sub-patterns being matched before, the intermediate states are derivable from the pattern itself.

Following the basic idea of logging, the relevant intermediate states might be derived automatically and mapped to partial matches, which represent an indicator for emerging situations and that something relevant might happen in the near future. To recapitulate, tracking partial matches means tracking matches of valid sub-sets, respectively sub-patterns of a pattern.

The concept of partial pattern fulfillment allows the observation of relevant intermediate states for the situation detection, which increases the transparency at run-time and allows the continuous tracking of an emerging situation. Deriving automatically relevant intermediate states requires an analysis of patterns and their automatic decomposition into valid sub-patterns, respectively their queries and sub-queries.

6.2 Observing Situations of Interest

Making situations of interest continuously observable requires the continuous tracking of their intermediate states. The automatic decomposition of situation-describing queries into their sub-queries, which define the intermediate states, represents a non-trivial task. Automatically deriving relevant intermediate states comprises two steps: understanding given queries and decomposing them into valid sub-queries. With the sheer variety of CEP engines and event pattern languages, a general decomposition approach requires a unifying model that supports the abstract representation of queries independent of their specific implementation. This section starts with the derivation of a common model to represent situations of interest and their corresponding queries for an automatic decomposition. Afterwards, an automatic decomposition approach is presented operating on the unifying model.

6.2.1 Model for Situations of Interest

As discussed in Section 3.5, event pattern languages differ significantly in the way how queries and related situations of interest are described. Generalizing the approach of transparency through partial pattern fulfillment in complex event processing requires a suitable and abstract representation regardless of the underlying CEP engine and its related event pattern language. With regard to this representation, three aspects have to be fulfilled:

1. The common model allows an easy definition of all pattern related aspects
2. The common model eases the subsequent processing of pattern-detecting queries
3. The common model allows an easy transformation of working queries into an engine-specific language

Figure 6.3 shows an overview of the final model with all of its elements. The model is depicted as a UML (Unified Modeling Language) class diagram with annotated associations to describe the relationship between the various model elements. *Situations of interest* form the central object of interest in complex event processing, as they describe the relevant happening in the monitored environment. Situations of interest represent the meaning of a specific event set that was detected in the event stream. They also form the root of the unifying model and represent the central element in this thesis. Another object of interest, that is strongly related to a situation of interest is the associated *Query*. The query *defines* a situation as it contains all relevant aspects, which describe a situation precisely. The query clearly defines the involved events (*Input Stream*), the relationship between those single events in the input event stream (*Pattern Detector*) and an *Action* to be triggered once the pattern is detected. The situation gives this defined relationship a meaning for the specific use case. In the following, the development of the unifying model is described, design decisions are elaborated and all elements are explained in detail.

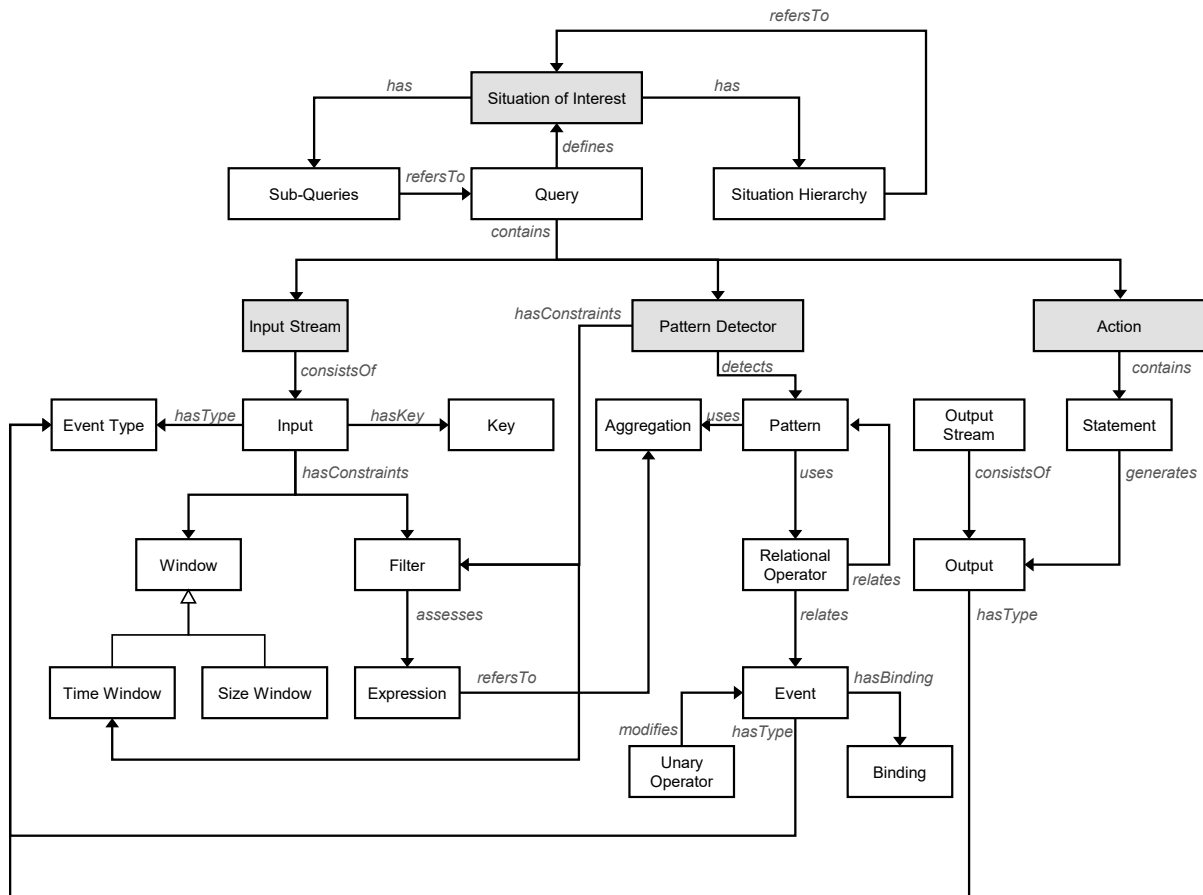


Figure 6.3: Model Overview

Sub-Queries

As discussed earlier, different approaches lead to an observable pattern detection functionality. One solution suitable for a general approach is the introduction of sub-queries to observe partial pattern matches. Sub-queries help to make the intermediate steps of the pattern detection observable. The core of a sub-query is, just as well as for queries in general, the relevant pattern. The sub-queries are derived during the decomposition step described in Section 6.2.2. In addition to its fundamental query, every situation of interest has a set of *Sub-Queries*, which follow the same structure as queries. The derived sub-queries *refer* to their original query to create a link to their origin. The different relationships are illustrated in Figure 6.4.

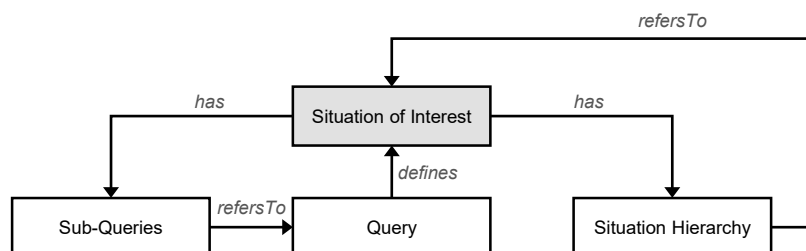


Figure 6.4: Model of an Situation of Interest

Situation Hierarchy

One fundamental aspect of CEP is the ability to represent hierarchies of activities, i.e. of event patterns. Figure 6.5 illustrates this aspect: whereas on the lowest level only simple event objects are taken into consideration to describe a relevant situation, Level 2 patterns already use complex events comprising the simple events from Level 1. Thus, the situation detection, i.e. the detection of a situation-related pattern, is based on Level 1 patterns. All pattern-relevant events on Level 1 need to have an event producer and need to be observable. All higher level patterns are an aggregation of simple and complex events from the levels below.

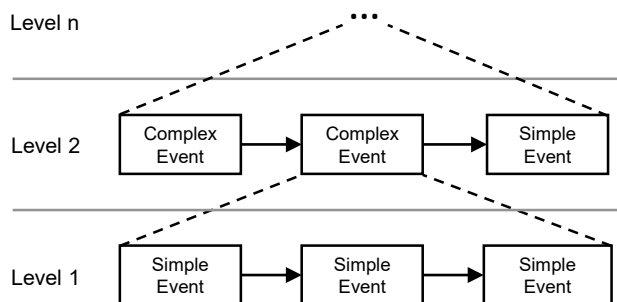


Figure 6.5: Example for a Hierarchical Relationship

Capturing this relationship between situations of interest requires an addition to the common model: the *Situation Hierarchy*, as shown in Figure 6.6. It *refers* to those situations of interest that have a hierarchical relationship with the situation that is currently under consideration, i.e. those situations whose query output is used as an input in the examined situation. The situation of interest representing a dull tool uses a complex event that is derived by executing the query describing the *MaxForce* situation, as introduced in the running example in Section 2.2. Figure 6.6 illustrates the relationship between the specific query and situation hierarchy in the model.

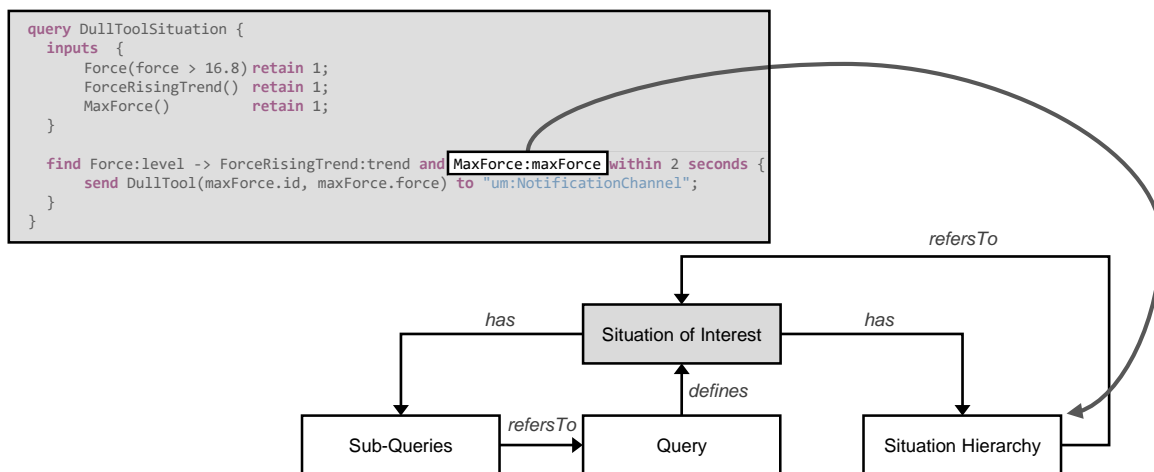


Figure 6.6: Example for a Situation Hierarchy in a Query

Query

Queries are the central artifact of every situation of interest, as they describe all aspects that are relevant for the processing of an event stream in order to identify those situations, respectively the occurrence of their pattern. With regard to the previously described concept of pattern detect event processing agents (see Section 3.3.3), three relevant steps of a pattern matching functionality have to be considered: filtering, matching and derivation.

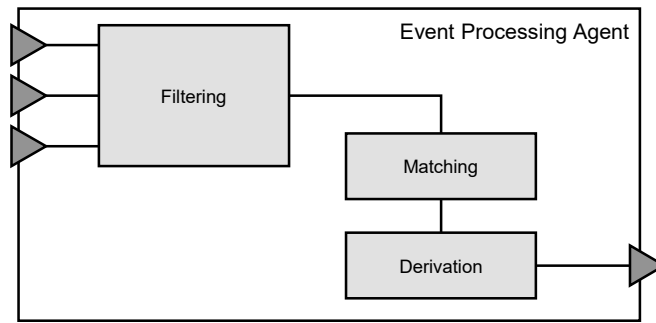


Figure 6.7: Internals of a generic Event Processing Agent according to [Etzion and Niblett, 2011]

By analyzing the internal structure of the pattern detect EPA in Figure 6.7, aspects are derived to be included in the implementation-independent unifying model to fully describe the pattern matching capability. Additionally, by using the abstract concept of EPAs to derive relevant elements of the model, it is ensured to cover most state-of-the-art event pattern languages, as they base on the fundamental concept of event processing networks and event processing agents described in Section 3.3.

First of all, an EPA supports multiple inputs, introducing event objects for further processing. Afterwards, the introduced event input is forwarded to a filter that applies filtering conditions to the inputs to identify the participant events. The filtered stream is forwarded to a match making component that matches the incoming event stream against a defined pattern. Once a pattern matching set of events is found in the matching component, the identified event sets are forwarded to the derivation step. Lastly, the derivation step applies the defined processing logic to the pattern matching events and generates an output to inform participants in the event processing network about the result. Reconsidering the contents of queries and the internals of a pattern detect EPA, it is essential to fully cover the aspects of a query in the common model, as it describes all components relevant for the pattern matching.

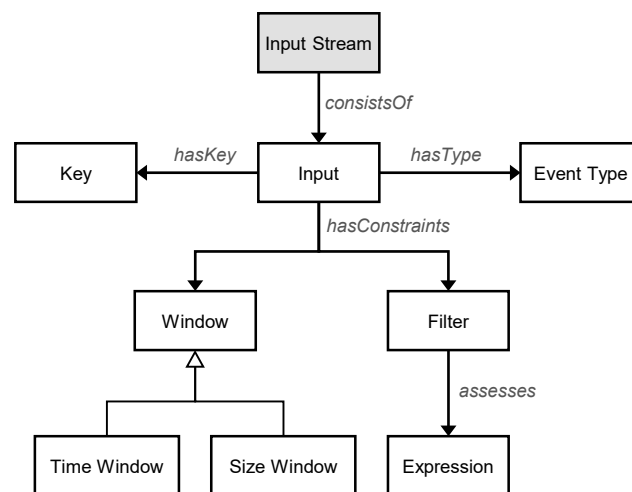


Figure 6.8: Model of an Input Stream

The first element to be added to the *Query* in the model is the *Input Stream*. It represents the set of events that serve as an input to the match making component and *consists* of several *Inputs*. The input stream gives already an overview which event types have an implicit relationship among each other. Each input itself represents an event stream of a specific *Event Type* that correlates with the situation of interest. In general, event types do not directly correlate to a situation, but rather a specific subset of event instances with particular event attribute values has to be considered. Representing this constraint in the model requires attaching a *Filter* to the *Input*.

A filter takes an event object from the incoming event stream and decides whether this object is to be considered for further processing or not. At this stage, filters allow only stateless filtering of event objects by *assessing* a filter expression, that evaluates a *Boolean Expression* against the event object. If the expression is true, the event passes the filter, if not, the event is discarded. Extended filter possibilities are added in a subsequent step.

The next extension of the *Input* adds the *Key* element. In several event pattern languages a key is used as a unique identifier to define a partition for the processing of event streams. Thus, a stream gets partitioned based on its key and the partitioned event streams get separately processed in the succeeding steps. Partitions play an important role during the execution of queries, but not at all with regard to their decomposition. Nevertheless, being able to reconstruct a functional, engine-specific query requires the key field to be added to the model. The last important extension of the *Input* adds a further constraint, the *Window* element. Windows for inputs represent an additional filter and reduce the throughput that an EPA has to handle. Two different types of windows are distinguished: time-based and count-based. A *Time-Based Window* forwards only these event objects to the succeeding components that arrive within the sliding time window. *Count-Based Windows*, also called size windows, always forward a defined number of the most recent event objects for further processing. Finally, all input details relevant for the further processing are covered in the model and illustrated in Figure 6.8.

Figure 6.9 shows the running example and the relation between the query input elements and the relating model elements. The input section of the query describes the input stream for the pattern matching. Within this section, single event streams are defined by stating which event types are to be considered. In addition, all three event streams encompass a window constraint, stating that only the most current event object has to be used. One event stream, consisting of force events, additionally defines a filter expression based on its event attribute.

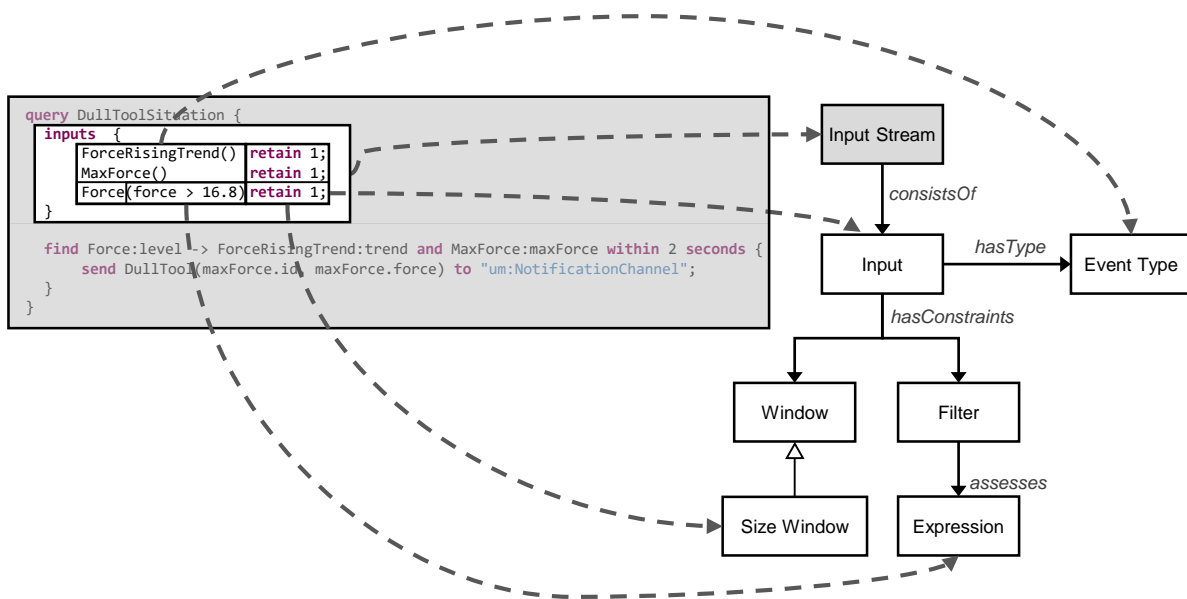


Figure 6.9: Example for an Input Stream in a Query

After the definition of the inputs, the processing of this stream and the relevant details for the match making are covered in the model. Therefore, one element is added representing the pattern detection functionality. The *Pattern Detector* covers the whole processing logic around a described relationship between event objects in the input stream. The resulting model excerpt is illustrated in Figure 6.10. The central element of the pattern detector is the *Pattern*, which represents the central logic for the match-making step and clearly describes the relationship between the participant events identified in the *Input*. The *Relational Operator* describes the type of relationship between two elements and can be categorized in two ways: structural and logical.

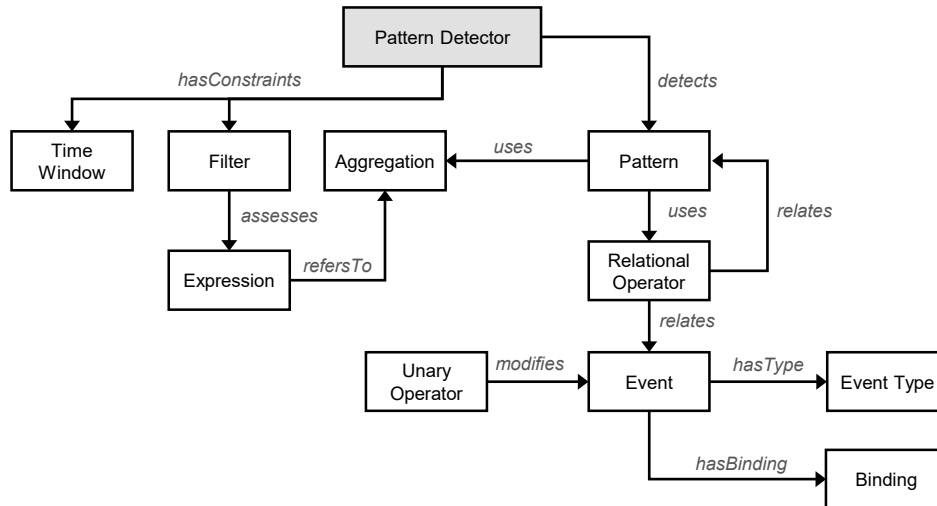


Figure 6.10: Model of a Pattern Detector

Both relation types are of interest for the further processing and derivation of correct sub-queries, their relevance and consequences are discussed later. As relational operators represent binary operators, the left- and right-hand-side part of the relationship has to be covered in the model. Two different elements are *related* with each other by relation operators: *Events* and *Patterns* themselves. By allowing relationships between events and patterns, an arbitrary concatenation of relational operators are facilitated. In case no further concatenation has to be modeled, the member of a relation is represented as an *Event*, representing the single event object, its *Event Type* and its *Binding*. Additionally, events are *modifiable* by a *Unary Operator*, e.g. the negation operator.

Next, three missing aspects of a pattern are added to the model: *Aggregations*, *Filters* and *Time Windows*. *Aggregations* are *used* to compare the attribute value of an event object with an aggregated value calculated over a set of event objects. Beside well-known aggregation functions, like the average, also custom aggregation functions are represented with the *Aggregation* element in the model. In difference to the previously mentioned input filter that is only capable of processing single event objects, at this stage filters allow the stateful filtering of event objects. Therefore, multiple event objects of different event types are compared with each other by defining an *Expression*. In case an expression is true, the event objects remain in the set of participant events, else they are discarded and not further considered for the pattern matching. With regard to the stateful filtering, two different filtering approaches are distinguished: content-based and context-based filtering. Content-based filters compare the values of multiple event attributes with each other, e.g. whether they refer to the same object id, whereas context-based filters compare results of additional method calls that take sets of event objects as their input, e.g. whether an event attribute of an event object is below or above the current average value. Thus, filters can also be defined for aggregations. An additional condition for patterns is definable by time windows. As described, the pattern detector collects all events that passed the input filter to apply the pattern matching logic. Time windows restrict the set of participant events to events that occur in the same time interval. As the pattern detector operates on unbound streams of events, the processing is theoretically infinite and even events with a huge time difference in between get evaluated together. Depending on the use case, this behavior is useful, in all other cases time windows are used to restrict the processing. Time windows are obligatory in case of blocking operators, e.g. negations, as a pattern detector is not able to evaluate a blocking operator on an unbound stream of events. Therefore, time windows are used to limit the scope of blocking operators to finite portions of the event stream.

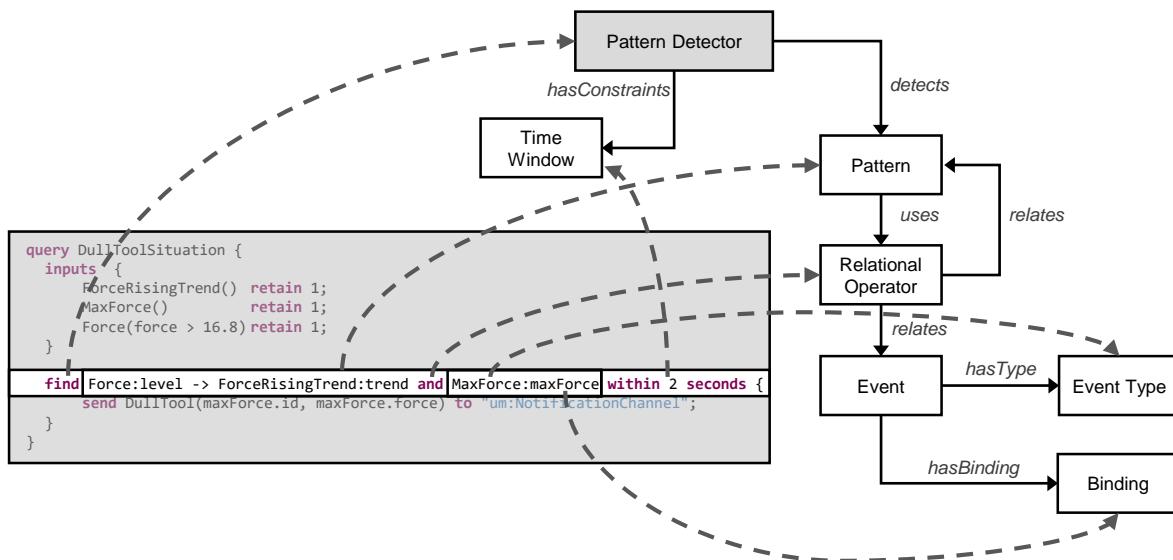


Figure 6.11: Example for a Pattern in a Query

Figure 6.11 illustrates the relationship between the pattern detection section in the query of the running example and the introduced model for a pattern detector. The pattern describes a relationship between three events by usage of two different relational operators, namely sequence and conjunction operator. Within the pattern definition the events are described with their event type and a binding that allows their referencing in the derivation step. Lastly, the pattern encompasses a time window, which represents an additional constraint.

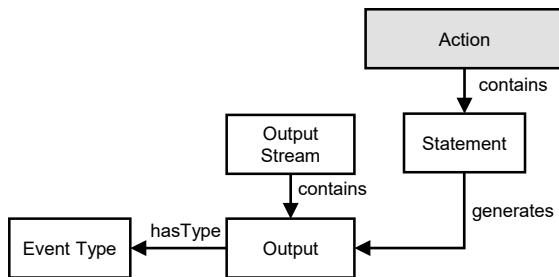


Figure 6.12: Model of an Output

Figure 6.12 illustrates the last extension of the query element: the *Output*. The last step within a pattern detect EPA is the derivation, which describes a transformation to generate the output of the EPA. In general, situation-describing queries and their related patterns also describe the action to be taken once the pattern is detected in the input stream. Queries can contain multiple *Actions* to be executed. An action itself consists of a *Statement* defining the logic to be executed. It describes which and whether event objects from the pattern matching set are used to derive a new complex event to be forwarded to further processing components. With regard to query hierarchies the question arises which event objects are created by the action statement. An additional *Event Type* in the model represents the type of the derived event object.

Figure 6.13 shows the last step within the query in the running example, the action section. It comprises one statement that generates an output stream by creating a new complex event *DullTool*, that references values from the previously defined events in the pattern and sending the new event to a channel on the event bus.

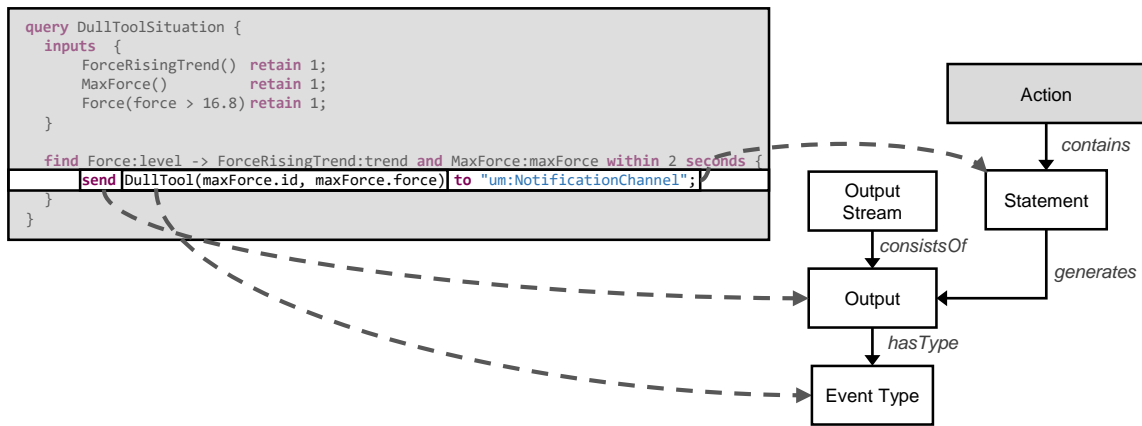


Figure 6.13: Example for an Action in a Query

By using the introduced model of a situation of interest all situation-related aspects of a query are described without relying on an engine-specific implementation of the query. Additionally, the model contains valuable information about the relationship with other situations and the resulting hierarchy. With regard to the defined goals at the beginning of this subsection, the model fulfills all of them: i) it allows the implementation-independent representation of a situation and its related query, ii) the model is laid out to ease the following decomposition and iii) all relevant aspects are covered to derive functionally working patterns in the source event pattern language.

6.2.2 Decomposing Queries

As discussed earlier, the goal of partial pattern fulfillment is to observe the occurrence of partial matches of a defined pattern in a query. In order to observe partial matches of a pattern, it is necessary to define the valid sub-patterns leading to partial matches. After converting the specific query representation into the common model, the decomposition of the query is targeted. Goal of the decomposition is the analysis of the pattern, the derivation of valid sub-patterns and based on this the generation of valid sub-queries. Hereby, the decomposition only considers the structural aspects of a pattern described by relational operators. All other conditions, like filter expressions and time windows stay untouched, as this would change the meaning of the described situation and does not support the observation of intermediate processing steps. In the following, the decomposition process of a pattern into its sub-patterns is introduced and it is explained how the different relational patterns behave during a decomposition. Afterwards, the generation of valid sub-queries for the further analysis of partial matches is described.

Decomposition of Patterns

Decomposing a pattern into its sub-patterns highly relies on the type of relational operators used and on the length of the pattern. Depending on the type of operator (sequence, conjunction and disjunction), the pattern decomposition represents a non-trivial task. Before describing the decomposition process, an overview is given on how the different operators behave with regard to a decomposition into sub-patterns and what their characteristics are.

The **Sequence-Operator** captures the arrival of a set of event objects with a specific order of arrival and is the most frequently used type to describe relationships between events. Since the order of arrival of events is relevant, the sequence-operator represents a non-commutative operator. Commutativity plays a significant role for the decomposition, as it has direct implications on its complexity. The decomposition of a sequence-pattern is carried out straight forward by removing events step by step beginning with the last event in the sequence.

Figure 6.14 illustrates a simple sequence-pattern consisting of three event objects *A*, *B* and *C*. The resulting sub-patterns still respect and fulfill the structural constraints defined in the pattern, yet allow the observation of the development of a sequence-pattern step by step.

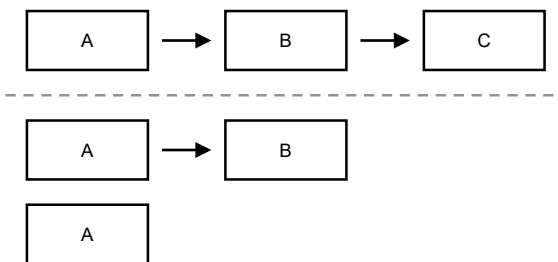


Figure 6.14: Example for a Sequence Pattern Decomposition

Another frequently used operator is the **AND-Operator**. It requires a set of participant events to contain all by an AND-operator connected event objects to match the pattern. The order of occurrence of the event objects is not taken into consideration. Thus, the AND-operator represents a commutative operator, which has a significant impact on the pattern decomposition. Figure 6.15 illustrates a simple AND-pattern and its resulting sub-patterns. Through commutativity, for AND-patterns of the length n all combinations of the lengths $l = 1, 2, \dots, n - 1$ have to be derived and considered as relevant, whereby duplicates are removed based on the properties of commutativity.

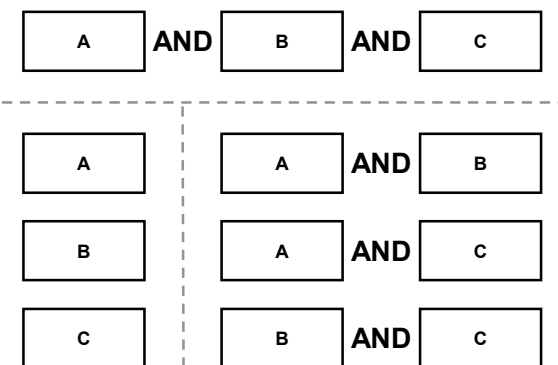


Figure 6.15: Example for an AND-Pattern Decomposition

A common operator that is less frequently used and also less frequently supported by CEP engines is the **OR-Operator**. OR-patterns describe a disjunction of events and are fulfilled when at least one event, that is part of the pattern, occurs as a participant event, i.e. the order of arrival has no impact on the result. The OR-operator is a commutative operator. With regard to the decomposition of patterns and its focus on relational operators, the OR-operator plays an insignificant role. Figure 6.16 illustrates the decomposition of an OR-pattern into three related sub-patterns. Due to the fact that as soon as one of the sub-patterns matches also the pattern matches, the sub-patterns do not allow the tracking of intermediate states and are discarded for further consideration.

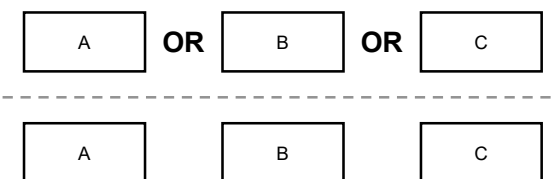


Figure 6.16: Example for an OR-Pattern Decomposition

As shown in the examples, the structure of a pattern, its length and the types of involved relationship operators significantly impact the number of sub-patterns to be considered for the observation of emerging situations. The number of required sub-patterns is determined by the number of involved operators, as shown in Formula 6.1.

$$\text{Number of Partial Patterns} = \#sequenceOperators + \sum_{sp \in SP_{AND}} \sum_{k=1}^{n-1} \binom{\#events \text{ in } sp}{k} \quad (6.1)$$

$$SP_{AND} = \{sp \mid \text{longest possible sub-patterns that are only connected with AND-operators}\}$$

$n = \text{length of } sp$

As shown, the number of sequence-operators has a minor influence on the number of required partial patterns, whereas AND-operators have a significant impact. According to their position in the pattern and the number of used AND-operators, several combinations have to be considered additionally. The decomposition of a pattern is handled recursively by taking the different characteristics of the operators into consideration.

Algorithm 1 `decomposePATTERN`

```

1: procedure DECOMPOSEPATTERN(pattern)
2:   if pattern.relation ∈ RelationalOperators then
3:     if pattern.relation == sequenceOperator then                                ▷ Handling a Sequence relation
4:       left ← pattern.leftPart
5:       right ← pattern.rightPart
6:       subQueryList.add(left)
7:       if left.relation ∈ RelationalOperators then
8:         if left.relation == sequenceOperator then                                ▷ Handling a Sequence in a Sequence
9:           subQueryList.add(decomposePATTERN(left))
10:        end if
11:       if left.relation == andOperator then                                       ▷ Handling a Conjunction in a Sequence
12:         subQueryList.add(decomposeAND(left))
13:       end if
14:     end if
15:     if right.relation ∈ RelationalOperators then
16:       if right.relation == andOperator then
17:         tempList ← decomposeAND(right)
18:         for i = 0; i < tempList.size(); i = i + 1 do
19:           tempPattern ← pattern
20:           tempPattern.leftPart ← left
21:           tempPattern.rightPart ← tempList[i]
22:           subQueryList.add(tempPattern)
23:         end for
24:       end if
25:     end if
26:   end if
27:   if pattern.relation == andOperator then
28:     subQueryList.add(decomposeAND(pattern))
29:   end if
30: end if
31: return subQueryList
32: end procedure

```

The pseudo code shown in Algorithm 1 represents the entry point into the decomposition process. The decomposition algorithm takes the model of a pattern, as described in Section 6.2.1, as its input. First, in line 2, the algorithm confirms that the pattern is further decomposable, i.e. the root of the pattern is a relational operator.

In this case, the algorithm checks whether the next processing step is the decomposition of a sequence-operator or an AND-operator (see lines 3 and 27). For a sequence-operator, the algorithm adds the left-hand-side of the relation to the list of relevant sub-patterns and checks whether a further decomposition of the child nodes is necessary in lines 7 and 15. In case the child node contains a relational operator and describes a further pattern, the decomposition is called recursively for another sequence-operator or the *decompose_{AND}* procedure is executed for the decomposition of an AND-pattern. Another noteworthy part in the procedure is shown between lines 16 and 22: in case the right-hand-side of a relation contains an AND-pattern, the results of its decomposition have to be combined with the left-hand-side of the relation, to derive further valid sub-patterns.

Algorithm 2 *decompose_{AND}*

```

1: procedure DECOMPOSEAND(pattern)
2:   left ← pattern.leftPart
3:   right ← pattern.rightPart
4:   subqueryList.add(left)
5:   subqueryList.add(right)
6:   patternsLeft.add(left)
7:   patternsRight.add(right)
8:   if left.relation ∈ RelationalOperators then
9:     if left.relation == andOperator then
10:      tempList.addAll(decomposeAND(left))
11:      subQueryList.addAll(tempList)
12:      for i = 0; i < tempList.size(); i = i + 1 do
13:        for j = 0; j < patternsRight.size(); j = j + 1 do
14:          tempPattern ← left
15:          tempPattern.leftPart ← tempList[i]
16:          tempPattern.rightPart ← patternsRight[j]
17:          subqueryList.add(tempPattern)
18:        end for
19:      end for
20:    end if
21:  end if
22:  if right.relation ∈ RelationalOperators then
23:    if right.relation == andOperator then
24:      tempList.addAll(decomposeAND(right))
25:      subQueryList.addAll(tempList)
26:      for i = 0; i < tempList.size(); i = i + 1 do
27:        for j = 0; j < patternsLeft.size(); j = j + 1 do
28:          tempPattern ← right
29:          tempPattern.leftPart ← patternsLeft[j]
30:          tempPattern.rightPart ← tempList[i]
31:          subqueryList.add(tempPattern)
32:        end for
33:      end for
34:    end if
35:  end if
36:  return subqueryList
37: end procedure

```

The decomposition procedure for AND-patterns is shown in Algorithm 2. Processing AND-patterns requires the consideration of the impact that the commutative nature of the operator has. Therefore, first a recursive drill-down of the AND-pattern is done until both child nodes contain only symbols. Afterwards, on its way back to the initial procedure call, the results from the lower level calls are combined with sub-patterns derived in the previous step. This approach ensures that all possible and relevant sub-pattern combinations are taken into consideration.

Derivation of Sub-Queries

The result of the *decomposePATTERN* procedure is a list of relevant sub-patterns for a given pattern. During the decomposition, only the pattern was considered, all other aspects of the query, e.g. inputs and input filters, have to be adjusted in the next step. A temporal sub-query is generated that contains the sub-pattern as its pattern and takes over all other elements of the initial query. Deriving valid sub-queries requires the further processing of the temporal sub-query. Validity in this context means that all constraints defined in the query are still valid for the partial patterns, in case the constrained elements are still part of it and all other aspects have to be cleaned up.

Therefore, the different constraints within the sub-query are checked and modified in several steps to ensure validity. The input stream is cleaned up and all inputs that are no longer relevant for the pattern detection are removed from the sub-query. Next, the stateful filters in the sub-pattern are checked for uninvolved elements that are still part of a filter statement and adjusted accordingly. Additionally, the output statement of the sub-query is adjusted to derive a complex event indicating the partial match and referencing the related query and situation. Finally, the resulting sub-queries are valid and can be used to observe partial matches in order to track the development of the pattern, respectively the situation. An example in Section 6.5 illustrates the decomposition process.

6.3 Analyzing Situations of Interest

As stated earlier, the transparency during run-time is intended to allow users and developers of CEP applications to track the evolution of a situation and to gain insights in the application during the processing of event streams. The goal is to give users the possibility to examine the current state of the system. With regard to the information that is captured through the transparent execution of patterns, different levels of information are distinguished. Figure 6.17 illustrates the different levels and the involved elements.

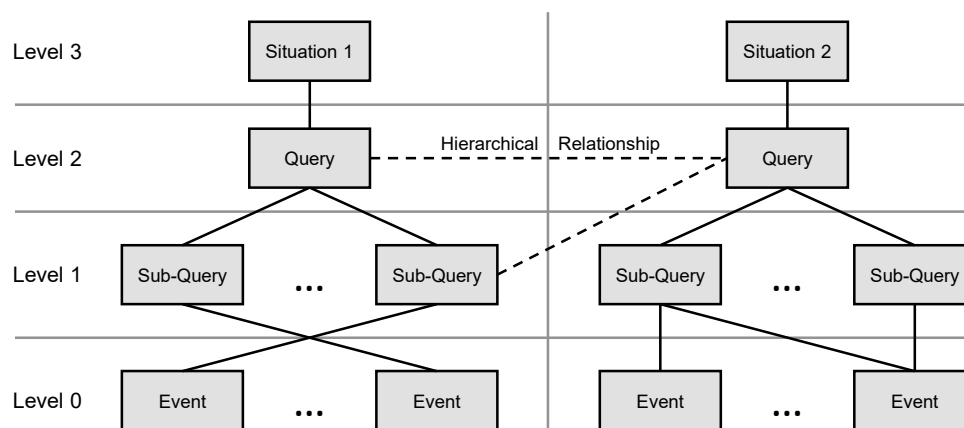


Figure 6.17: Levels of Information

Information on *Level 3* describes the situation of interest and builds the entry point. It contains high-level details about the situation, e.g. when it was detected the last time and how often it occurred. On *Level 2*, the information

captures details about the query and pattern. *Level 1* information uses the new transparency and considers the observable intermediate states at run-time to give insights into a situation even before it occurs, as up to that point in time only sub-queries may have occurred. On *Level 0*, information about the involved event objects is captured to give the deepest possible insights into the situation. The described information level hierarchy allows the drill-down from a situation down to the single event objects involved in the situation evolution. Additionally, references between (sub-)queries, give additional information about a hierarchical relationship between situations.

The next step comprises using this information to generate insights into queries and events that might be of interest and to put sub-patterns into context to the situation by reviewing the history and the development from a sub-pattern to its pattern. In the following, interesting aspects of queries, respectively sub-queries, and event objects are described that are of interest for users to better understand situations and their development over time.

6.3.1 (Sub-)Query Analysis

Queries form the entry point to the analysis as the central component describing the situation of interest. Queries and sub-queries are analyzed during the execution of a CEP application to continuously allow insights into the processing. Therefore, different aspects of queries and sub-queries are considered to deliver information about a situation: i) occurrence aspects, ii) time-related aspects and iii) relationship aspects.

Occurrence Aspects

During the execution of a CEP application, the frequency of occurrences of queries and sub-queries, respectively their pattern matches, is captured. The frequency of occurrence helps users and developers to get a feeling how important a situation is and about the probability of occurrence. In addition, the frequency of executed sub-queries gives developers valuable information whether parts of a query are out-of-date and need a refactoring, as for example in the past they were executed more frequently than nowadays (see [Sen, 2013]). The frequencies are also important for the relationship statistics between sub-queries in order to better describe the development possibilities between a sub-pattern and its superior pattern.

Time-related Aspects

Queries and especially sub-queries, respectively their related patterns occur frequently in event streams. One interesting aspect to gather data about are the time aspects of these occurrences. Therefore, the mean time interval between two occurrences, as well as the minimum and maximum time interval is determined. Additionally, it is interesting to capture the time aspects between sub-queries, i.e. what are the time-related statistics between two sub-queries that are part of the same query. Thus, the mean, minimum and maximum time intervals between sub-queries are additionally measured to give a time estimation for the development of a sub-query to a more mature sub-query, respectively its superior query itself.

Relationship Aspects

A query and its sub-queries have a relationship between each other as well as sub-queries have one between each other. An interesting aspect to capture during analysis is the ratio between the occurrences of each sub-query and the sub-queries describing further evolved states of the sub-query under study. By means of this information it is possible to calculate a transition probability between the sub-queries. Together with the previously captured time-related aspects between sub-queries, interesting insights are observable for users to get a feeling what might happen in the future. Additionally, all queries and sub-queries that have a hierarchical relationship with other queries as a part of another situation, might be used to follow this relationship in order to get further information about the related situation.

6.3.2 Event Object Analysis

For every query and sub-query, respectively pattern and sub-pattern the relevant event types are known and analyzed during run-time on-the-fly in the event stream. The data about the event objects form the deepest information level and allow users to further drill-down from the level of queries and sub-queries. Coming from the level of queries and sub-queries, the event objects that are part of the pattern, respectively sub-pattern match are analyzed further. Therefore, the following aspects are captured:

- **For Event Types:** For every involved event type, the frequency and time-related aspects, e.g. mean time interval, minimum time interval and maximum time interval, are captured
- **For Event Attributes:** For every event attribute, the mean, minimum and maximum of its values is calculated during the processing of the event stream. As the calculation is done on the unbound event stream, the computation has to be done incrementally online on the infinite stream. Additionally, the frequency of every event attribute is determined to get an overview on the value distribution. Lastly, time-related aspects, e.g. mean, minimum and maximum time intervals, are captured for event attribute values.

6.4 Architecture

After finalizing the conceptual approach to enable transparent complex event processing, questions related to the system architecture and its design are discussed. The general concept how to introduce transparency plays a major role. Existing CEP systems have evolved over years and are capable to process event streams with a low latency. Thus, one important requirement is to keep the existing characteristic of a system. With regard to the design, different approaches have to be discussed: i) developing a transparent CEP engine from scratch, ii) modifying existing CEP engines or iii) adding a layer to state-of-the-art architectures.

The first option comes with some advantages, as the partial matching of patterns is implementable directly into the correlator of the engine, i.e. the central component within an engine implementing the pattern detecting capability and handling the correlation of events. Thus, the transformation of queries into an abstract representation becomes obsolete. The development of a new engine represents a complex task to accomplish, as all requirements towards an engine have to be considered, e.g. out-of-order handling of events, processing logic for operators, time window handling and ensuring a high throughput of events. Current state-of-the-art CEP engines are developed and tuned over years to satisfactorily fulfill all requirements. Also, in order to benefit from the new transparent processing of event streams, the new engine would have to be introduced in an existing environment by replacing the previous engine. Consequently, existing queries would have to be translated into the new event pattern language. All these efforts are considered as barriers for the dissemination of transparent complex event processing.

The first question that arises with regard to the second option is whether the correlator of the CEP engine is modifiable. Most state-of-the-art CEP engines do not have an open core, thus do not allow the modification of its core technology for pattern matching. Additionally, the question arises, whether all partial matches are easily observable, as dependent on the basic technology for pattern matching in the core, the modifications might be profound. Another aspect to consider, is whether fundamental features of the CEP engine are still supported after the modification. As stated earlier, engines are highly tuned to fulfill different requirements, e.g. low latency, and a modification of the core might have an impact on this capability. So, depending on the use case, the newly modified engine might not be taken into consideration as a replacement for the previous engine. And if, again the existing queries have to be translated into the event pattern language of the newly introduced CEP engine.

In contrast to the two mentioned options, which target a modified implementation of CEP engines, the third option aims at a new architecture design. Therefore, this thesis targets the extension of state-of-the-art CEP architectures by an additional processing layer to establish transparency which has numerous advantages. First, the existing CEP engine in an existing environment is retained, thus no expensive transition projects have to be conducted. Second, the existing queries are further used and do not have to be adapted for a new event pattern language and engine. Also, the new architecture is deployable in a way so that the existing architecture has only a small overlap with regard to the processing logic, which leads to the advantage that the additional layer does not influence the processing performance of the existing system and does not affect the maintainability of the CEP system. As the new layer has only a few requirements in order to work with existing CEP engines, the new functionality is made accessible to a broader audience and additionally supports the further usage of complex event processing. Lastly, the newly available information can be easily used to further enhance the capabilities of complex event processing architectures step by step without modifying the heart of the architecture, the CEP engine.

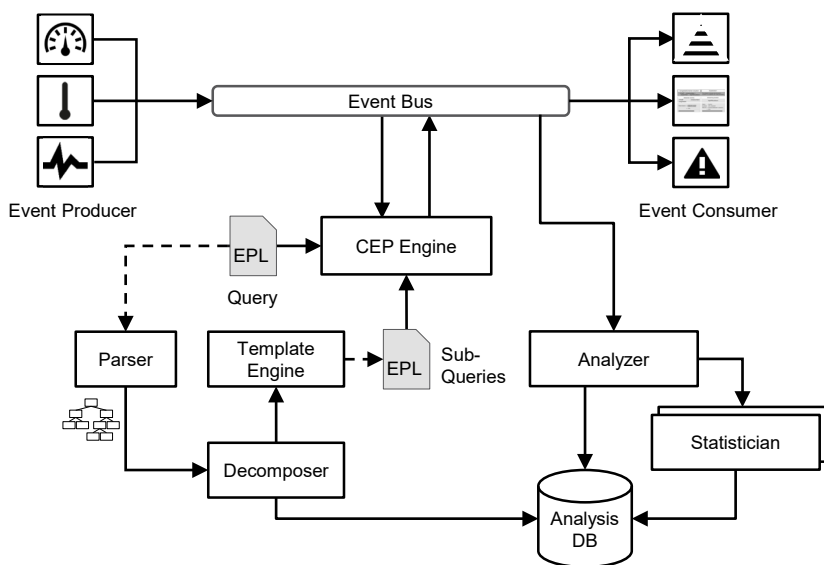


Figure 6.18: The General Architecture

Figure 6.18 shows an overview of the resulting architecture and the involved components. The upper part of the illustration shows the previously introduced state-of-the-art architecture of a complex event processing environment (event producer, event bus, CEP engine, event consumer) as it can be found in many use cases. Event producers observe relevant aspects of the use case and send their observations of an activity to an event bus. Event consumers on the other side react directly on the events on the event bus or on results coming from the processing of the events. The CEP engine has a notable role in the architecture, as it acts as an event consumer and an event producer.

The lower left part of the illustration shows the components (*Parser*, *Decomposer*, *Template Engine*) involved in the transformation of a query into an abstract representation, the decomposition of this representation and the derivation of valid sub-queries. The derived sub-queries are then deployed back to the CEP engine in the architecture, which now materializes a complex event signifying the occurrence of a partial match. Handing the detection of partial matches back to the underlying CEP engine has one significant advantage in comparison to other solutions: the expressivity and mightiness of the supported operators is ensured. In other cases the mode of operation of operators could be different, thus leading to inconsistent results. All other approaches may increase the complexity, as it needs to be ensured that operators have the identical operational meaning and processing logic.

The lower right part illustrates the components (*Analyzer, Statistician, Analysis DB*) involved in the analysis of the observations. Two things are analyzed during run-time: the partial matches and the related events. The analysis results give users of complex event processing systems the possibility to gain more insights during run-time. All components and their interplay are described in detail in the following subsections.

6.4.1 Generating Sub-Queries

The first part of the extended architecture handles the generation of valid sub-queries and comprises three steps: i) transforming the query into the abstract representation, ii) decomposing the abstract representation into valid sub-queries and iii) deriving the EPL-specific queries in order to deploy them back to the underlying CEP engine.

At first, the query describing the situation has to be transformed into the previously introduced model. Therefore, the query description in the specific event pattern language for the used CEP engine has to be parsed and processed. The *Parser* component in the architecture takes over this task. Figure 6.19 illustrates the transformation process.

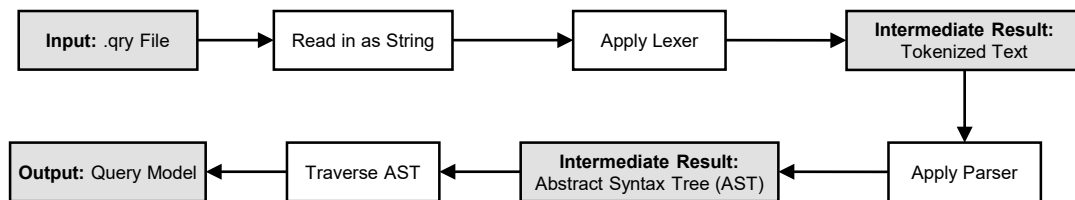


Figure 6.19: Transformation Process of a Query-File into a Query-Model

The query is read in as a string and handed over to the lexer. The lexer transforms the sequence of characters into tokens, which are suitable as an input for a parser. The parser processes the tokens, combines them and groups them together to form an abstract syntax tree (AST), which is used to derive the final model by visiting all nodes in the AST and mapping them to elements of a query in the model. The generation of a lexer and a parser is done automatically by means of a parser generator which takes a grammar of the event pattern language as its input. For every event pattern language that should be supported, a grammar is required. Depending on the CEP engine to be supported, the grammar might be already available, in other cases it has to be defined manually, but represents a one time effort per CEP engine. Depending on the run-time behavior of event operators, a modification of the transformation logic might be necessary. Figure 6.20 illustrates the data flow between the various components.

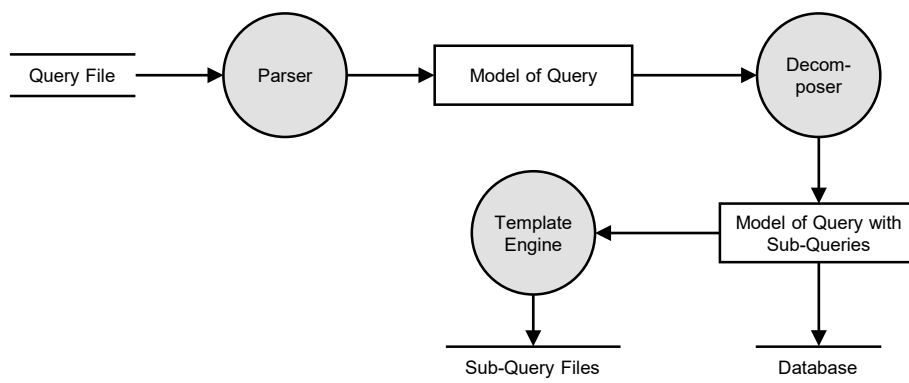


Figure 6.20: Data Flow for Decomposition of Queries

The new representation of the query serves as the input for the next component, the decomposer. The decomposition of the query model follows the algorithms introduced in Section 6.2.2, where the pattern of the query is recursively decomposed into all sub-patterns. After the decomposition, the sub-patterns are used to derive valid sub-queries, i.e. the additional filters, input and output statements are adjusted. The valid sub-query models are handed over to the next component to derive the engine specific representation of the sub-queries. Additionally, the model of a situation, including the query model and all of its sub-query models is stored for further usage in the database, which also serves as a situation registry to later identify relevant situations for further analysis.

The derivation of language-specific sub-queries for the underlying CEP engine is done by a template engine. The goal is to derive valid documents according to a predefined template, which includes placeholders for dynamic aspects of a document and a specific format. Template engines then take the template and a data model as their inputs to replace the dynamic aspects with specific content from the model and to derive a valid document representing the content of the data model in a specific format. Similar to the generation of a parser, a template for queries is required for every event pattern language to be supported. Then, the automatically derived sub-queries are directly deployed back in the CEP engine.

The introduced approach for query decomposition is implemented for any CEP engine and requires only three adjustments: First, a grammar is required describing the event pattern language used to describe queries. Second, depending on the mode of operation and characteristics of the CEP engine and its EPL, modifications might be necessary for the mapping logic from an AST into the common model representation. Third, a template definition is required for the template engine in order to produce valid sub-queries after the decomposition.

6.4.2 Analyzing Situations

The second part of the extended architecture is devoted to the analysis of relevant simple and complex events. The goal is to make use of the newly claimed transparency and give a summarized picture of the complex event processing at run-time. The architecture is therefore extended with the following new components: an analyzer and statisticians. Figure 6.21 illustrates the data flow between the components for the analysis of queries.

The analyzer forms the entry point into the analysis and represents also the main component that is in charge to orchestrate the analysis process. The analyzer component is connected with the event bus and listens to relevant events, i.e. events that are related to situations of interest that were previously decomposed and are registered with their model in the database. The filtering on relevant events is done on basis of the input streams of the queries and according to their situation hierarchy. The goal is to take all events into consideration that are relevant for the detection of a situation, respectively its pattern.

The resulting event stream for analysis in the statistician components contains two types of events: events that are required as input for the queries, respectively sub-queries, and events indicating a partial pattern match. Depending on the type, the events are differently treated. Events that are part of a query or sub-query are stored in the database to be available for further analysis. Additionally, the events are examined by a component responsible for the event statistics. The statistics are calculated on-the-fly on the events on their way from the analyzer to the database. The calculated statistics contain the information described in Section 6.3.2 and are stored in the database.

Events indicating a partial match have to be enriched before analyzing them. In order to keep the overhead generated through the additional matching of sub-queries low, complex events signifying a partial match only carry references to the related query, respectively situation of interest. But for further insights, it is necessary to know which specific events are part of the partial match. Therefore, the analysis component does an enrichment of partial

matching events and determines which already stored events are involved in matching the sub-pattern. This information is added to the partial matching event. The enriched partial matches are forwarded to a statistic component, which calculates the statistics for sub-patterns and especially their relationship among each other, as described in Section 6.3.1. The sub-query statistics are also stored in the database, where they can be accessed by users to give further insights about situations of interest. The analysis is independent from the specific CEP engine, as it solely relies on the model of a situation including its queries and sub-queries. With regard to the interfaces to the existing, basic architecture, the analysis component only requires read-access to the event bus.

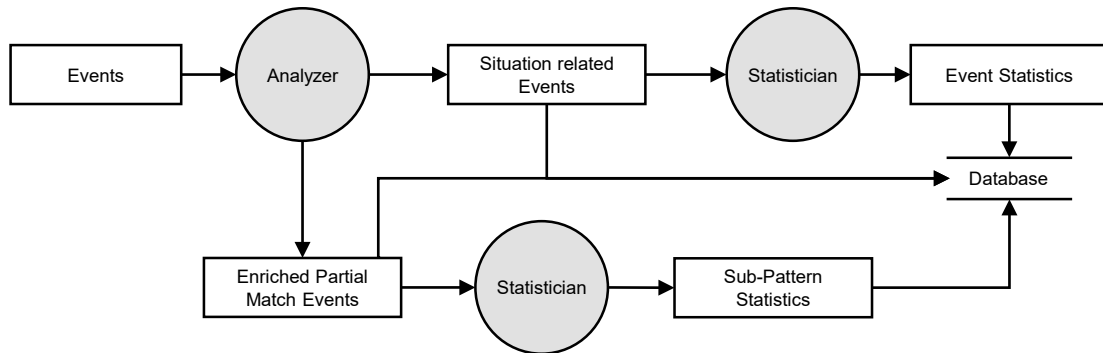


Figure 6.21: Data Flow for Analysis of Queries

6.5 Running Example

For a better understanding, an illustration of especially the decomposition and its concepts is shown by using the running example introduced in Section 2.2. The illustration comprises the transformation and decomposition process, as well as an excerpt of the analysis part, showing only the sub-query statistics. The situation of interest, the usage of a dull tool, is described with a query in the Apama Query Language (AQL), which represents the event pattern language of the Apama¹ CEP engine. Consequently, the engine-specific representation needs to be transformed into the previously introduced unifying model for a further processing in the decomposition step. Figure 6.22 presents the query and identifies the relevant aspects of the query for a transformation into the model.

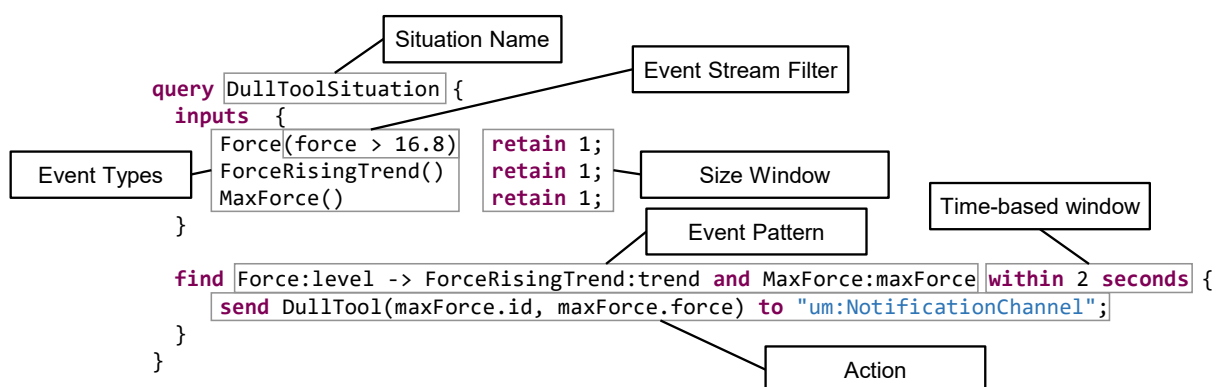


Figure 6.22: Example Query with Annotations

¹ <http://www.apamacommunity.com/>

The name of the situation of interest is derived from the query name. Relevant inputs for the pattern detection are extracted from the *inputs* section of the query, which also defines further filters to be applied. The pattern description itself is found by analyzing the *find* instruction of the query, where also a time window is defined. Lastly, the action statement is examined to derive the output element in the model. Transferring the query into the common model requires a grammar describing the query language, which was manually created for AQL. Based on this grammar a parser is derived by means of a parser generator to extract an abstract syntax tree. The resulting AST is traversed by a component mapping its contents to the model elements. Finally, the query is transferred into the common model. This process is done for all relevant situations defined in the environment.

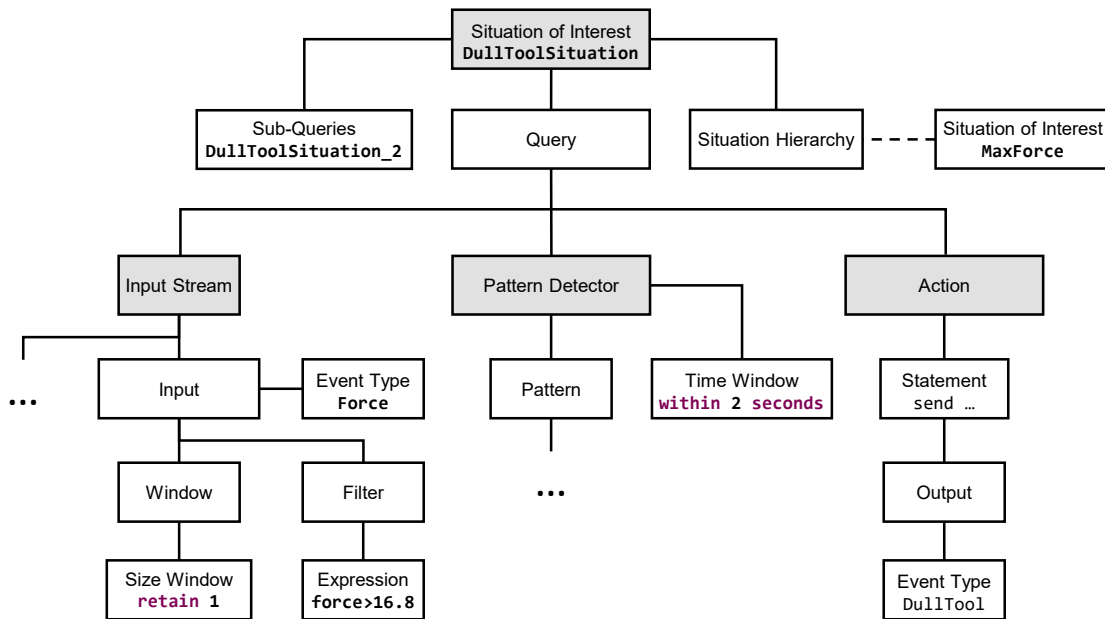


Figure 6.23: Example Situation Model

Figure 6.23 illustrates an excerpt of the resulting model. The situation of interest, as the main object, is the root of the model. Another situation of interest having a relationship with *DullToolSituation* is *MaxForceSituation*, as the output of *MaxForceSituation* is used as input in *DullToolSituation*. Hierarchies are only modeled for complex events created through a relational pattern, as the evaluation of partial matches is restricted to them. Thus, complex events created by just an aggregation, are not linked in the situation hierarchy. The input stream contains all information from the input section of the query. The example shows how the event type *Force* is added as an input to the model with its related restrictions: filtering on one of its attribute values, namely the force value, and a size window indicating to only use the most recent element in the event stream. On the other side, the output contains the information that an event of the event type *DullTool* is derived once the pattern matches. The statement contains the exact instruction used by the query to derive the complex event.

The pattern of the query is shown separately in Figure 6.24, which describes the pattern section with all related aspects. First, a time window is added to the pattern detection element, indicating that the processing window for the pattern match comprises a two second time window of event stream data. Next, the pattern structure is described by including further nested elements. At the lowest level of the pattern element, the leaf nodes contain the events that are matched according to the defined relations and comprises the event type and its binding in the query. The leaf nodes are connected by a relation operator defining the relationship between them.

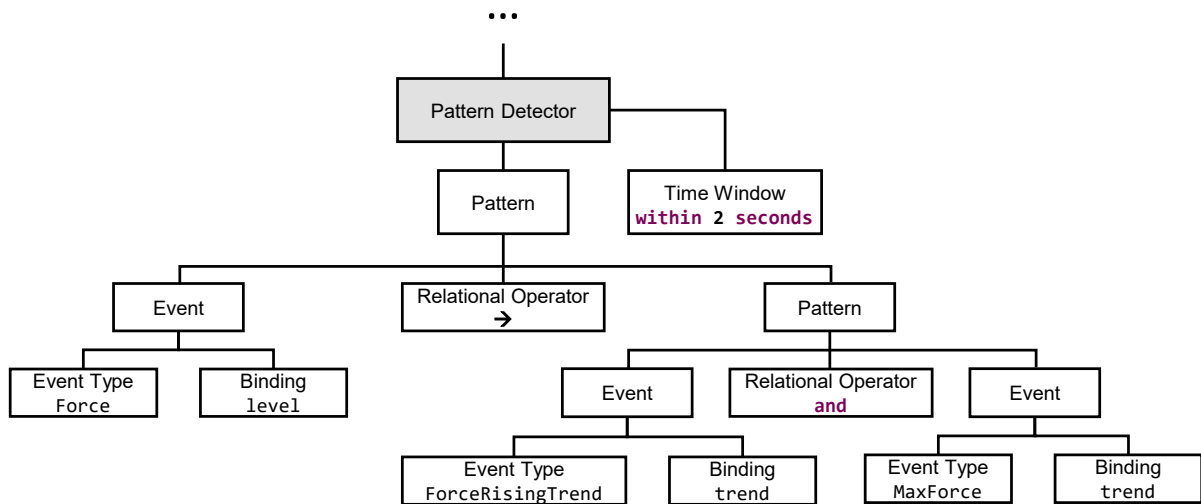


Figure 6.24: Example Pattern Model

The resulting model of the pattern is used as input for the decomposition procedure described in Section 6.2.2. The decomposition is done in a recursive way and is illustrated in Figure 6.25. In the first step, the sequence is decomposed into its left- and right-hand-side (LHS/RHS) parts, where the left-hand-side part represents the first sub-pattern, comprising only the *Force* event. The right-hand-side part contains another relational operator, the conjunction operator defining an AND-pattern. As described in the decomposition algorithm, this pattern is again recursively decomposed into its LHS and RHS. By taking into consideration the commutative character, the decomposed parts have to be combined with the LHS of the sequence operator to derive further sub-patterns. The decomposition process stops, as the leaf nodes only consist of events, so no further decomposition is possible. The procedure derives three possible intermediate states during the detection of the *DullToolSituation*:

1. *Force* : *level*
2. *Force* : *level* → *ForceRisingTrend* : *trend*
3. *Force* : *level* → *MaxForce* : *maxForce*

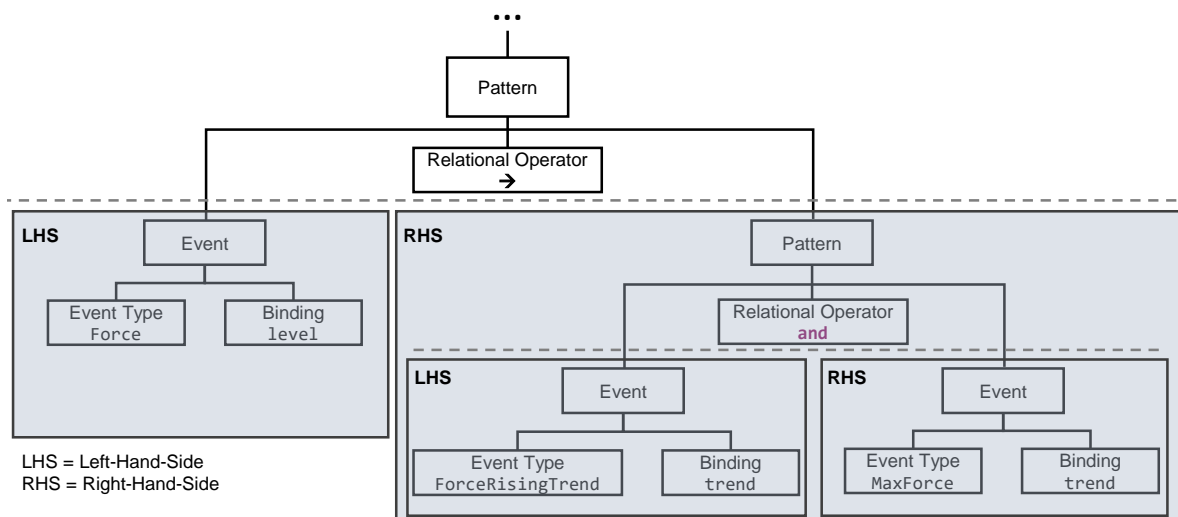


Figure 6.25: Decomposition in Running Example

The decomposition algorithm clones the original query and decomposes the included pattern. All other aspects of the query are not adjusted during the decomposition process. Therefore, after the derivation of the sub-patterns, the sub-queries have to be cleaned up by removing non-valid filters and input statements. Considering the first sub-pattern, comprising only the *Force* event, the input statements for *ForceRisingTrend* and *MaxForce* have to be removed. In addition, for every sub-query the action statement has to be adjusted to generate a complex event used internally in the analysis component of the architecture to correctly relate the partial match to a sub-query, respectively its superior query. The final sub-query model is transformed back into the engine-specific event processing language by means of a template engine. Beforehand, a template was created describing the structure of AQL queries. This task, like the creation of a grammar, has to be done only once per CEP engine that should be supported. The new sub-query is deployed in the existing CEP engine and used to track partial matches of the initial pattern. Listing 6.1 shows one of the derived sub-queries, describing the sub-pattern *Force : level* \rightarrow *ForceRisingTrend : trend*.

```
query DullToolSituation_2 {
  inputs {
    Force(force>16.8)    retain 1;
    ForceRisingTrend()  retain 1;
  }

  find Force:level -> ForceRisingTrend:trend within 2 seconds {
    send PartialMatch_5ae1d727a02199717caf6d0a("5ae1d727a02199717caf6d0b", "5ae1d727a02199717caf6d0a")
    to "um:AnalysisChannel";
  }
}
```

Listing 6.1: Derived Sub-Query in AQL

On basis of the derived sub-queries, the execution is analyzed continuously and the statistics introduced in Section 6.3 are calculated. The situation statistics give insights about the occurrence frequency of a situation and its last materialization. Figure 6.26 shows the exemplary statistics for the presented sub-query in Listing 6.1. It comprises several statistical information relevant for analyzing the sub-query and events that are part of it.

On the highest level, the statistic gives insights in the occurrence frequency of the sub-query, its membership to a situation, the event statistics, time statistics and the relationship to other sub-queries, respectively the original query. The partial pattern was detected 321 times and the field *Relationship* indicates that the probability to evolve to the situation itself, respectively a full pattern materialization, is 43.61%. The event statistic is captured for two events that are part of the sub-query: *Force* and *ForceRisingTrend*. It shows the different event values that have been part of the 321 partial matches. In Figure 6.26, the event statistic is shown for the *Force* event and indicates, that the values 16.8 and 17.2 occurred during run-time. The *Force* event has as well been detected 321 times, resulting in a percentage of 100%. Event statistics also comprise time statistics, which are focused on the single event under consideration. Behind each value of an event, information is gathered on how often this value has been part of a partial match, which percentage this represents and additional time statistics for this specific value. In the example, the force value of 16.8 was detected 93 times in a partial match, which results in a percentage of 28.97%. Finally, the time statistics of the sub-query are described, which are shown on the right part of Figure 6.26. They encompass additional time aspects, including the first and last occurrence timestamp, the time difference between those timestamps, as well as the mean, minimum and maximum time difference between two sub-query occurrences. Time characteristics are collected for all elements in the statistic and always describe the element that is under study in the specific statistic.

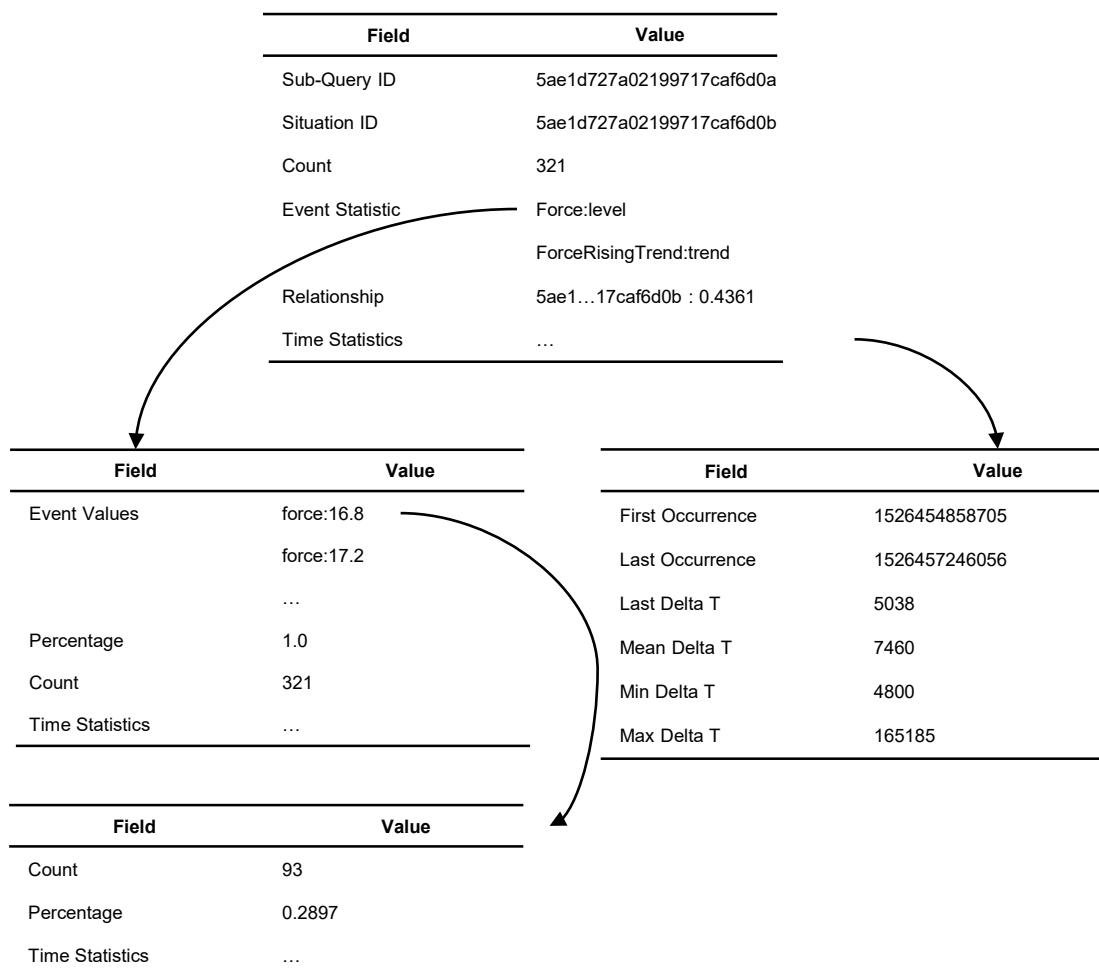


Figure 6.26: Sub-Query Statistics in Running Example

6.6 Summary

In this chapter, a concept was introduced for transparency in complex event processing systems at run-time. First, the requirements towards transparency in CEP systems were derived and a conceptual approach, namely Partial Pattern Fulfillment, was presented to tackle the issue of intransparent event processing. The derived solution makes use of the observation of partial matches of defined patterns, which correspond to relevant intermediate states of the CEP system. The solution is built on top of existing CEP systems and utilizes the underlying CEP engine and event pattern language for its purpose. Due to the broad range of available CEP engines and event pattern languages, a unifying model was introduced to represent queries in a engine-independent way for further processing. Based on this model, valid sub-queries, which are used for the observation of partial matches, are automatically derived by decomposing the primary query. In addition to the introduced solution for enhanced transparency, another solution was presented to track the current state of the system. The newly claimed transparency empowers the continuous analysis of pattern and sub-pattern matches to capture interesting aspects and offer an aggregated view on the current state of the system and the monitored situations of interest. The implementation of the new capabilities required an extension of state-of-the-art event-driven architectures. Therefore, an additional processing layer was added to the architecture realizing the new functionality. The chosen approach comes with the advantage, that the existing architecture has not to be modified in any way and expensive adoption projects become avoidable. Accordingly, all characteristics of the existing architecture, e.g. performance aspects, are kept up.

The new capabilities of transparent complex event processing allow the retrospective analysis of occurred situations and give real-time insights into the current state of the system and monitored situations. This leads to an increased comprehensibility of the mode of operation of CEP systems and might lead to an increased trustworthiness into the results. Additionally, transparent complex event processing serves as a basis for further developments. In Chapter 7, partial matches are used to introduce situation awareness in complex event processing and in Chapter 8, observed partial matches are extracted as a basis for the generation and evaluation of prediction models for predictive complex event processing. In conclusion, transparent complex event processing represents the starting point for valuable evolution steps towards a holistic situation management system.

7

Situation-Aware Complex Event Processing

Based on the introduced transparency in the preceding chapter, this chapter aims at the introduction of situation awareness into complex event processing systems. The goal is to base this conceptual extension of the CEP paradigm on a defined concept of situation awareness, as the term is overloaded and often used without a clear definition. Therefore, Section 7.1 discusses the general concept of *Situation Awareness*, its origins, its meaning as a psychological model, its role in situation management and its commonalities and differences with complex event processing. Section 7.2 describes an approach to incorporate the defined concept of situation awareness into complex event processing by introducing an explicit life-cycle for situations of interest that covers all relevant aspects. Additionally, mechanisms are developed to flexibly define transitions between the new states of a situation. In Section 7.3, a guideline is presented covering relevant steps towards an implementation of the life-cycle and describing materialization metrics to capture the evidentiary degree of fulfillment of situations required for transitions in the life-cycle. Section 7.4 presents the technical implementation and its integration into the existing architecture for transparent complex event processing, before Section 7.5 illustrates the mode of operation along the running example in this thesis. Section 7.6 summarizes in brief the contributions of this chapter.

7.1 Situation Awareness

Operating complex technical systems, like manufacturing machines, is a demanding task for human operators, as they heavily rely on up-to-date knowledge of their environment to manage the systems effectively. Therefore, operators monitor the state of all relevant elements trying to identify unusual behavior or already known patterns. Based on the goals of their tasks, operators first derive what has to be observed in the environment and then perceive the relevant aspects to understand their meaning. Known patterns help to effectively understand the implications of a situation and to allow a quick reaction. According to Endsley [Endsley, 1995], human control cannot be effective without this mechanism.

Abstracting from the specific tasks, an operator has to keep an overview of the environment and interpret all happenings. The ability to exactly observe the environment supports the building of a mental model and comprehending the occurring situations. This state of knowledge is known as *situation awareness* (SA) and is defined as "*the perception of the elements in the environment within a volume of time and space, the comprehension of their meaning, and the projection of their status in the near future*" [Endsley, 1995]. Situation awareness is an important factor in dynamic decision-making and incomplete or inaccurate situation awareness potentially leads to wrong decisions. Humans differ in their ability to acquire situation awareness, even when they receive the same input regarding a task and an environment, which results in differing decisions and reactions for similar or identical situations. Figure 7.1 shows the three hierarchical phases of situation awareness and its integration in the context of dynamic decision-making by humans according to Endsley.

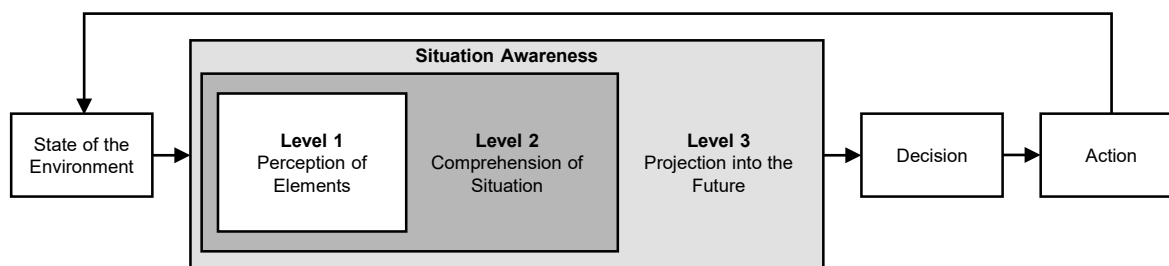


Figure 7.1: Model of Situation Awareness according to [Endsley, 1995]

The basic requirement for the situation awareness model is a human operator with a specific task that defines the context and the relevant states of the environment to be taken into consideration. The process of acquiring and maintaining situation awareness comprises three interrelated steps: perception, comprehension and projection. Based on the acquired level, operators make decisions and act accordingly, which again influences the state of the environment and serves as input for the next situation assessment cycle to maintain situation awareness. The three steps of situation awareness illustrated in Figure 7.1 also form the different levels of it.

Level 1 situation awareness comprises the perception of the status, attributes and dynamics of relevant elements in the environment. The perception of relevant elements in the environments forms the basis for situation awareness and elements are perceived from three sources: i) direct sensing by the human operator, ii) system data, iii) information communicated by others. The single elements of the environment are perceived independently of each other. **Level 2 situation awareness** is seen as the comprehension of the current situation. Perceived elements are used to form a holistic picture of the environment and comprehend the single elements and their significance for the operator goals. Operators use known patterns of elements to recognize relevant situations. This task relies on the experiences of an operator. **Level 3 situation awareness** represents the highest level of SA and encompasses the projection into the future status of the environment. It is based on the perceived level 1 elements and their comprehension in level 2. Being able to project into the future represents a desirable state, as it provides the knowledge and time to make grounded decisions and to react in the best possible way.

The introduced model by Endsley describes the psychological model of humans processing information with regard to a specific task. Nevertheless, the model is adaptable for the development of computer systems to support operators. According to Salfinger [Salfinger et al., 2013], a human's correct situation assessment is affected by information overload and time criticality. *Situation Management* systems aim to support operators with their tasks and ease the situation assessment process by better understanding occurred situations and indicating potential occurrences in the future. According to Jakobson [Jakobson et al., 2006], situation management is defined as the process of sensing and collecting information, recognizing situations, analyzing past and predicting future situations. It comprises three relevant aspects shown in Figure 7.2: the investigative, control and predictive aspect.

The *investigative aspect* is intended for the retrospective analysis of occurred situations to understand why a situation has occurred, while the *predictive aspect* is concerned with the prediction of situation occurrences. The *control aspect* deals with the controlling of situations, i.e. the detection and reaction towards it. The central element in situation management is the *Situation Model*, a mental representation of a situation, whereas a situation is considered as "part of reality that can be comprehended as a whole" [Rosemann and Recker, 2006]. Situation models are often described in a state-oriented approach, where a situation describes combinations of different states for relevant entities of the real-world and their relation among each other at a particular point in time. In this model, level 1 and level 2 situation awareness are used to link real situations to their instantiations from the situation model.

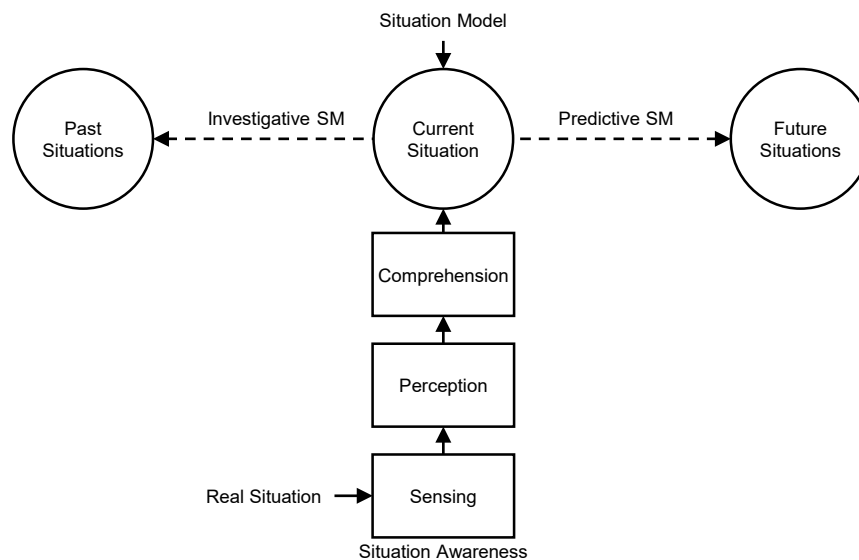


Figure 7.2: Situation Management Overview - excerpt [Jakobson et al., 2006]

Due to information overload and time criticality, operators run into danger of losing situation awareness or even being unable to establish it at all. Situation management systems support operators by recognizing situations and communicating this insight to the operator. Supporting the maintenance of situation awareness requires the system to track changes of situational states, as this represents a challenge for operators in rapidly evolving environments. In contrast to the mental capabilities of an operator, the information about the environment scales quickly, which presents a challenge to process it. From a technological point of view, monitoring systems are capable of processing huge amounts of data, but lack in supporting the user to keep up with situation awareness [Nilsson et al., 2008]. As situation awareness and its technical support with situation management systems heavily relies on real-time processing of data, complex event processing systems present a promising starting position. The suitability of complex event processing for the situation-aware processing of event streams is discussed in the next section.

7.1.1 Situation Awareness and Complex Event Processing

Existing situation-aware systems, as presented in Chapter 4, already support operators in acquiring situation awareness with regard to their tasks, but struggle with the challenges of big data (volume, velocity and variety), as the increasing amount of data has to be processed in real-time. As discussed, complex event processing successfully tackles these challenges. As described in Chapter 2, CEP systems are often used in use cases that require an automatic reaction. Contrary to that, in many use cases nowadays human operators remain in the control loop, as automatic reactions are not possible or not preferred. Due to its technological advantages and its capability to process huge amounts of data in real-time, its suitability for situation awareness and the missing links are discussed.

First, a comparison of CEP systems and their capabilities with situation management systems is conducted to evaluate their suitability as a specific implementation of a situation management system. As previously discussed, the situation model plays an important role in situation management. As stated by Jakobson [Jakobson et al., 2006], situation models comprise three components: the structural, dynamic and representational component. The *structural component* describes the entities of the environment that are related to a situation, their associated attributes and attribute values, classes of entities that share common features, and relations between those entities. The *dynamic component* describes entities, their relationship and behavior over time, whereas the *representational component* is seen orthogonal to the rest and represents a situation modeling language used to describe them.

Comparing the described situation model with situations of interest in complex event processing and their representation as queries, many common aspects are identified. First, all structural components have a corresponding element in CEP systems: entities of the environment are events, that describe relevant happenings in an observed environment. Events in complex event processing, like entities in situation management, have associated attributes with attribute values. Classes of entities that share common features are mapped to event types. The dynamic aspects are described in CEP systems by defining relations between events with time-dependent event operators. Finally, the query is described by means of an event pattern language and represents a situation modeling language. In both worlds, situations are described through a state-oriented approach by describing the conditions that have to be observed to indicate the occurrence of a situation. With regard to the basic building blocks, as illustrated in Figure 7.3, CEP systems cover all aspects of situation management. As stated earlier, situation management relies on all three levels of situation awareness. Therefore, next the capabilities of CEP systems are compared and linked to situation awareness in a way to technically replicate the model of SA.

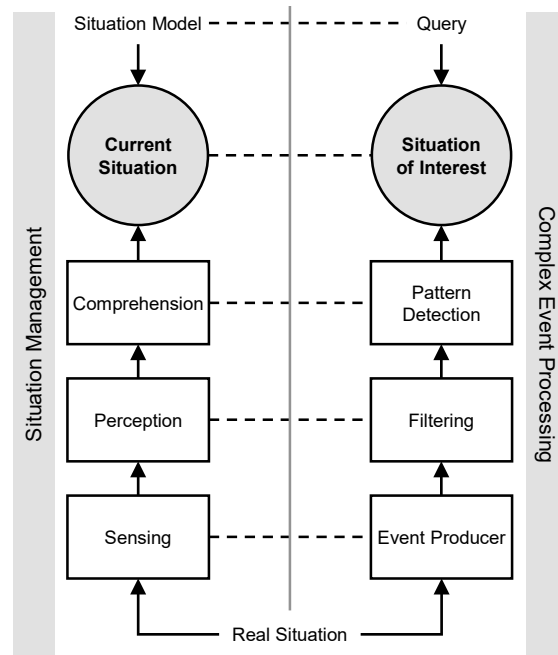


Figure 7.3: Comparison of Situation Management and Complex Event Processing

CEP systems operate on queries describing a situation of interest with its relevant events and the related pattern. With regard to the introduced situation awareness framework, the elements of a query are mapped to the fundamental elements in the framework. The system has a clear task to fulfill, the detection of occurring situations and relevant aspects are encompassed in the query. Therefore, level 1 situation awareness is implementable by means of complex event processing. Through the description of relevant events for the pattern detection, the CEP system is capable of perceiving relevant elements from the environment. The detection of patterns in event streams uses the perceived elements to identify situations. The detection goes along with the comprehension of perceived elements and the integration into a holistic picture regarding one specific situation. In contrast to human operators, who are able to comprehend perceived elements with regard to multiple situations at once, CEP systems require one query per situation to be deployed to the CEP engine. Consequently, CEP systems are considered as level 2 situation aware through their pattern detection capability. The predictive aspect of level 3 situation awareness is not encompassed in the definition of the paradigm and is therefore not covered by state-of-the-art CEP systems, as they are solely concerned with the detection of occurring situations in defined environments.

Through their ability to replicate level 1 and level 2 situation awareness, CEP systems support a human operator in becoming situation aware in terms of detecting occurring situations and informing about this condition. Thus, CEP systems already fulfill one important requirement to make use of situation awareness, the ability to recognize situations and communicate descriptions of it to others [Kokar et al., 2009]. Unfortunately, this ability only covers the situation-awareness for on-going situations. Within a user study conducted by Nilsson [Nilsson et al., 2008], human operators stated that the differentiation between emerging and already on-going situations is of utmost importance to react accordingly. This capability of maintaining situation awareness in an evolving environment requires the ability to track the evolution of situations [Salfinger et al., 2014], which corresponds to a continuous and transparent comprehension of perceived elements in an environment. As the maintenance of situation awareness in an information-rich and dynamic environment represents a highly complex task for human operators, system support has a beneficial impact in the operation of complex technical systems.

In summary, complex event processing represents a proven technology to handle huge amounts of events and is able to technically replicate two out of three levels of situation awareness. By overcoming the missing capability of predicting situations and allowing the continuous tracking of situation evolution, complex event processing is able to evolve to the next level and become fully situation-aware. Transparent complex event processing, as introduced in Chapter 6, serves as a basis, as it already allows the tracking of partial pattern matches. The missing link between both is the assignment of a specific meaning to partial observations. The goal is a situation management system based on complex event processing fully supporting situation awareness, that allows the tracking of emerging situations and supports the management of operator attention.

7.2 Situation-aware Complex Event Processing

The goal of situation-aware complex event processing is to support human-centered situation awareness and to align the way events are processed with the way humans are processing their environment. As discussed earlier, the ability to track the evolution of situations represents a key requirement for maintaining situation awareness and giving operators enough time to react prepared towards an upcoming situation. In order to be able to track situations and their evolution, first it has to be defined according to what situations should be tracked. Salfinger states in [Salfinger et al., 2014] that tracking corresponds to monitoring the elements involved in a situation. Depending on the use case, the relevant aspects that are tracked over time may vary. A general solution requires an abstract layer to be tracked. Therefore, the current life-cycle of a situation in complex event processing is further analyzed.

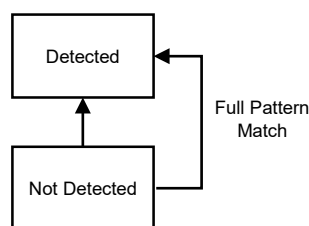


Figure 7.4: Implicit Life-cycle within state-of-the-art CEP Engines

In order to be independent of the specific use case, the tracking has to be done with regard to the abstract concept of a situation of interest. As shown in Figure 7.4, state-of-the-art CEP systems are capable to distinguish two different states of a situation: *detected* and *not detected*. The life-cycle of existing systems is derived implicitly, as it is not explicitly defined and results from the mode of operation of a CEP engine. Certainly, a CEP engine has a more detailed view on a situation, but as described in Chapter 6, the internals of the pattern matching capability are intransparent for the outer world.

With regard to the meaning of the implicit states, both have a low information content. The meaning of *detected* is clear, as it tells a user that the pattern describing a situation of interest was fully detected in the event stream. The state *not detected* only informs that the pattern describing the situation was not yet fully matched in the event stream. It represents a subsumption of all intermediate states below a situation detection and has a broad range of possible states, e.g. from not a single pattern related event was detected in the event stream up to one single event of the pattern is missing in order to detect the situation. This way too coarse discrimination offers no valuable information for a human operator as a situation might occur at any time and leaves the operator unprepared.

Establishing a situation-aware complex event processing requires the definition of a refined explicit situation life-cycle that is independent of the underlying use case. Additionally, the life-cycle has to allow the tracking of a situation's evolution and extend the previous binary view. To be adoptable to different use cases with variable information needs, it needs a mechanism to incorporate use case specifics. Consequently, this thesis introduces an extended life-cycle for situation-aware complex event processing.

7.2.1 Extended Situation Lifecycle

As stated before, supporting the acquisition and especially the maintenance of situation awareness to its full extend, requires a fine-grained situation life-cycle to differentiate between intermediate states in the situation evolution. With current CEP systems, situations start their life-cycle in the state *not detected*, which is way too coarse, as it neglects all the gradations in between. Therefore, the first step comprises the introduction of a new initial state which represents the starting point for every situation in complex event processing: the *inactive* state.

Definition (Inactive Situation). *A situation of interest s is termed inactive, if and only if no evidence is found that indicates the activation of the situation. The inactive state represents any situation's initial state.*

By introducing a new initial state and replacing the implicit existing state, an operator directly knows whether the situation under study requires attention or not, as for inactive situations no fact is found in the event stream indicating that the situation might occur. On the other end of the life-cycle, the existing *detected* state is taken over, indicating an occurred situation. As situations occur at a specific point in time, the *detected* state is considered as the final state for a situation instance, which ends the situation tracking in the life-cycle.

Definition (Detected Situation). *A situation of interest s is termed detected, if and only if the situation occurrence is evidentiary observed. The detected state represents the final state of a situation.*

Between the introduced states, intermediate states are identified that allow the tracking of situations. The differentiation between situation states has to be made on basis of its maturity, to give a clear picture of the current state and make states comparable. Another state of a situation on its way from *inactive* to its occurrence is the state *initialized*, which gives an operator the hint that situation-related indicators are found in the event stream.

Definition (Initialized Situation). *A situation of interest s is termed initialized, if and only if evidentiary facts are observed that indicate relevant happenings with regard to the situation.*

As a situation evolves further, it reaches a state that is considered more mature with regard to its evolution towards the *detected* state. The assumption behind this new state is, that mature situations have a higher probability to occur and should therefore be explicitly indicated to the operator with the goal to increase the preparation time in order to react accordingly. Therefore, the state *looming* is introduced in the extended situation life-cycle.

Definition (Looming Situation). *A situation of interest s is termed looming, if and only if significant evidentiary facts are observed that indicate the contemporary occurrence of the situation.*

By introducing the state *predicted*, CEP systems are incorporating the concept of projections to become level 3 situation aware, i.e. the extrapolation of available information to foresee possible states of the operating environment [Endsley and Connors, 2008]. It represents a mature state with one difference in comparison to the *looming* state: the predicted state contains a prediction with calculated occurrence probabilities and if suitable, an estimation of the occurrence time.

Definition (Predicted Situation). *A situation of interest s is termed predicted, if and only if the situation is predicted to occur in the near future.*

The basic idea behind the extended life-cycle is that knowledge about a situation's current state enables operators to be prepared and eventually act before the occurrence of a situation. In order to support multiple use cases with differing requirements, the transitions between situation states have to be refined and made explicit.

7.2.2 Transitions in the Situation Life-Cycle

Indicators are introduced to clearly describe the transitions between states and introduce adaptability for differing use cases. All indicators comprise a metric and a threshold to realize its transitioning functionality. First, the transition from *inactive* to *initialized* is covered by introducing an **Activation Indicator**. The metric serves as a tool to measure the degree of fulfillment and has to be defined according to the needs of the use case. The activation indicator determines the transition between the two states and relies on an *initialization threshold*. The transition from *initialized* to *looming* is handled by the **Significance Indicator**, targeting the evaluation whether a situation has reached such a mature state that it can be considered as important enough to attract an operator's attention. Since human attention is limited, only significant insights should be indicated. The indicator relies on a *significance threshold* that is defined according to the needs of the use case. The last indicator that is required for the transition to the state *predicted*, is the **Prediction Indicator**. It is based on prediction models and comes with a condition that indicates a transition: a threshold to describe a minimum probability of occurrence and/or a time horizon. Both conditions are chosen to ensure that it is worth to pay additional attention to the predicted situation. Thus, if the probability of occurrence is not sufficiently high or the time horizon is not critical, any notification towards the operator might be a distraction and consume valuable attention without any added value. Figure 7.5 illustrates the extended situation life-cycle with its transitions.

A typical path through the described life-cycle is the following example: a situation s is defined by the pattern p_s and an instance of it initially starts in the inactive state. As events occur that are part of the pattern p_s , a metric captures the evolution of situation s and moves it to the initialized state once the activation indicator is triggered. More and more events included in p_s are found in the event stream, further supporting the indication of an occurrence of situation s . As soon as the threshold for the significance indicator is reached, the mechanism determines the detection of sufficient evidence to attract the operator's attention towards this situation so the situation becomes a looming situation. In case of a deployed prediction model, the future evolution of situation s is predicted and once the probability or time horizon are within the defined constraints, the situation enters the predicted state. Lastly, if all conditions defined in pattern p_s are fulfilled, situation s is considered as occurred and enters the detected state, which also represents the end of tracking for this specific situation instance. The possible transitions of a straight forward path are also depicted in Figure 7.5.

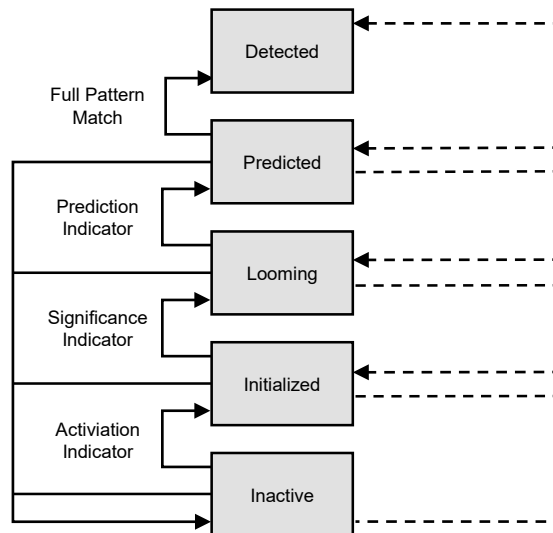


Figure 7.5: Explicit extended Life-cycle for situation-aware Complex Event Processing

Not all situations develop along this typical path, as conditions may change during the pattern detection. Situations can be discarded from the states initialized, looming and predicted before they occur and get detected. In such cases time constraints defined in the pattern become invalid. In addition, some exceptional cases in the other direction have to be considered as well (dashed lines in Figure 7.5). First, situations consisting of a single event are not trackable, as no intermediate states are distinguishable. Consequently, such a situation transitions directly from inactive to detected. Second, situations can skip one or more states if the corresponding indicator triggers a transition because of conditions that were fulfilled, e.g. the prediction of a situation satisfies the constraints defined in the prediction indicator and therefore a situation moves directly to the predicted state.

In summary, the extended life-cycle allows a fine-grained differentiation of various situation states and the tracking of a situation's evolution. Based on the life-cycle and its transitions, the following functional requirements towards a technical implementation with CEP need to be considered in the development of situation-aware CEP applications: i) defined situations of interest with respective pattern, ii) an activation indicator, iii) a significance indicator, iv) a prediction indicator and v) a metric to track a situation's progress towards full materialization.

7.3 Development of Situation-Aware CEP Applications

Implementing situation awareness in CEP systems requires the development of a component to handle the situation life-cycle, as existing systems do not support an explicit life-cycle. By following the succeeding guideline, all aspects for an implementation are defined and ready for the incorporation into a technical component. The proposed guideline for the development of situation-aware complex event processing consists of the following tasks, which are derived from the previously described requirements towards a technical implementation:

1. Definition of the Materialization Metric

The materialization metric is the core of situation-aware complex event processing, as it captures a situation's degree of fulfillment, i.e. the situation's progress towards its occurrence. It realizes the functional requirement of tracking a situation's evolution through the life-cycle, i.e. the evolution path. It allows the comparison of situations with regard to their maturity. As it measures the current materialization of a single situation, it has to be based on detected events to capture its verifiable progress. Within this thesis, the materialization metric is based on the

concept of partial pattern fulfillment as introduced in Chapter 6. The newly claimed transparency enables the observation of intermediate stages during pattern detection. Depending on the use case, different metrics can be defined with partial matches as their basis: *simple partial pattern fulfillment* (S-PPF) and *relevance-based partial pattern fulfillment* (R-PPF). The basic idea behind both is to calculate what portion of the overall pattern is fulfilled and what portion is missing. Formula 7.1 shows the calculation of the S-PPF metric for a pattern p . The degree of fulfillment is calculated by comparing the number of detected pattern-relevant events with the total number of involved events required to satisfy the pattern, which differentiates with regard to the operators used.

$$materialization_{S-PPF}(p) = \frac{|\{d \mid d \in D \cap REL_p\}|}{|\{e \mid e \in REL_p\}|} \quad (7.1)$$

D ... Set of detected events

REL_p ... Set of relevant events of pattern p

The assumption behind S-PPF is that all events are equally important and meaningful. It represents a special case of the more general relevance-based partial pattern fulfillment which incorporates the unequal likeliness of events and their unequal informativeness. The calculation is based on the concept of TF-IDF (Term Frequency Inverse Document Frequency) [Sparck Jones, 1972]. The first step comprises the calculation of the weighting (ef) for a single pattern-related event e in the corresponding pattern p and is shown in Formula 7.2.

$$ef(e, p) = \frac{f_p(e)}{\max_{e \in p} f_p(e)} \quad (7.2)$$

$f_p(e)$... Frequency of event e in pattern p

The weight is calculated by comparing the frequency of the given event with the frequency of the most frequently used event in the pattern. In simple patterns the weight of a single event is often 1, as in many cases events are only used once. Calculating the event weight relates to the term frequency of the TF-IDF concept. The calculation shown in Formula 7.3 aims at the inverse document frequency, that calculates the weight of an event (ipf) over the set of all patterns. The idea is to weight the events according to their occurrence frequency in the environment.

$$ipf(e, P) = \ln\left(\frac{|P|}{|\{p \in P : e \in p\}|}\right) \quad (7.3)$$

P ... Set of all Patterns

The overall weight (ew) of an event e for a pattern p is derived by multiplying both results, the event frequency ef and the inverse pattern frequency ipf , as shown in Formula 7.4. Afterwards, the total weight of a pattern p is calculated by summing up all weights of the related events, as shown in Formula 7.5.

$$ew(e, p, P) = ef(e, p) * ipf(e, P) \quad (7.4)$$

$$tw(p, P) = \sum_{e \in p} ew(e, p, P) \quad (7.5)$$

P ... Set of all Patterns

The total weight (tw) of a pattern p serves as the reference value for the tracking of the degree of fulfillment. As partial pattern matches are detected, their total weight is calculated and compared to the targeted value of the pattern. By using relevance-based partial pattern fulfillment, events are weighted dynamically according to their frequency. Thus, it computes the proportion of weights instead of proportion of the number of events. Consequently, rare pattern-related events are given a higher weighting, as their occurrence has more significance with regard to the possible occurrence of a pattern. The resulting materialization metric for R-PPF is shown in Formula 7.6.

$$materialization_{R-PPF}(p, P) = \frac{\sum_{e \in D} ew(e, p, P)}{tw(p, P)} \quad (7.6)$$

D ... Set of detected events

P ... Set of all Patterns

One special case that has to be considered during the calculation of a situation's materialization are hierarchical relationships between two or more situations. As described in Chapter 6, complex event processing is capable of representing hierarchies of activities, i.e. the complex event resulting from one pattern is used within a higher level query to define a new pattern. The relationship between situations is captured during the transformation and introduction of a situation and its query into the system.

$$materialization_{HIER}(p) = \sum_{e \in p} \frac{1}{|\{ev \mid ev \in REL_p\}|} * materialization_{EVENT}(e) \quad (7.7)$$

REL_p ... Set of relevant Events of Pattern p

To incorporate a hierarchical relationship in the calculation of the materialization, an adapted metric is required. Formula 7.7 shows the adopted version to incorporate hierarchies. For a pattern p , it calculates the sum of every single event materialization. First, all event occurrences, independent of being simple or complex, are equally weighted and multiplied with the materialization for the single event, which is examined with Formula 7.8. In case of an occurred event, the formula returns the value 1, signifying a full materialization. In case of a not detected complex event, its underlying pattern is analyzed and the calculation of its materialization is triggered. Both formulas are valid for S-PPF and are adaptable to R-PPF by modifying the weighting of the events in Formula 7.7 and adopting the assigned value for a detected event and the triggered materialization metric in Formula 7.8.

$$materialization_{EVENT}(e) = \begin{cases} 1 & \text{iff } e \in D \\ materialization_{S-PPF}(p_e) & \text{iff } e \in CE \cap e \notin D \end{cases} \quad (7.8)$$

D ... Set of detected Events

CE ... Set of all Complex Events

p_e ... Pattern of Complex Event e

2. Definition of the Activation Indicator

The activation indicator handles the transition from *inactive* to *initialized*. The indicator uses the defined materialization metric and checks for a minimum degree of pattern fulfillment. Therefore, a threshold representing the lower limit of the state initialized has to be defined. In general, the definition of thresholds for the situation-aware processing of events is a challenging task, as thresholds have a significant impact on the information density of the system. If the threshold is set too low, the system creates many state updates that may overwhelm an operator. If it

is set too high, the operator runs the risk of missing an important state update. Depending on the use case at hand, thresholds have to be evaluated empirically in the environment and refined if necessary.

3. Definition of the Significance Indicator

The significance indicator handles the transition from *initialized* to *looming*. The indicator is based on the defined materialization metric and checks a minimum degree of pattern fulfillment. It determines whether the evidence of a partially observed situation is sufficient to justify the state looming. A threshold has to be defined indicating the achievement of a significant degree of fulfillment and notifying the operator that it is worth to center attention on the situation. Again, the definition of a threshold has to be done by a domain expert with knowledge and experience for the monitored environment, as it has a significant impact on the information density.

4. Definition of the Prediction Indicator

The prediction indicator is required to project a situation occurrence into the future and provides a mechanism to realize the state transition to *predicted*. Predictions require a new metric that evaluates the possible evolution of a situation. Partial pattern fulfillment and the resulting statistics as described in Section 6.3.1 present one option. The probability of a sub-pattern evolving further to its full pattern is a suitable metric. Following the statistics approach represents a simple prediction model, as it is solely based on sub-pattern occurrence frequencies.

With regard to the requirements of the dedicated use case, more advanced prediction models might be employed as a metric, e.g. Neural Networks, Bayesian Networks, Hidden Markov Model, etc. Most prediction models provide a measure to quantify the reliability of predictions, which is used as the metric. Complementary, a threshold defining the minimum level to trigger a transition to the state *predicted* is needed. Depending on the prediction model, additionally the occurrence time is estimated and together with the state update communicated to the operator.

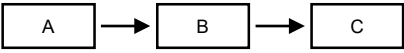


MachineOutage Situation	Explanation
	This situation describes a machine outage.
Detected Events	Remaining Events
	
Materialization Metric	Prediction Metric
Metric: S-PPF Calculation: $\frac{ Detected\ Events }{ Relevant\ Events } = 66\ \%$	Metric: Statistics Calculation: 50 %
State	
LOOMING	

Figure 7.6: Possible Visualization for an Operator

5. Definition of a Visualization

As the support of a human operator is the underlying goal of situation-aware complex event processing, insights of the system have to be communicated to the user. One possible representation is shown in Figure 7.6, which provides the operator with details to the pattern, the materialization metric, the prediction and its current state. It is a combination of static and dynamic information elements. Static information comprises a human readable representation of the pattern and the description of it. Dynamic information describes which events are already detected and which remain to be detected. Such a visualization can be implemented in different ways, e.g. as a Web or smartphone application.

7.4 Architecture

After introducing the conceptual extension of the situation life-cycle and defining all relevant aspects for an implementation, the developed component to realize the new capability is described in the following. It is based on the extended architecture of transparent complex event processing, as the materialization metric uses partial pattern fulfillment to evaluate the maturity of a situation. By leveraging the capabilities of the additional processing layer, situation awareness is introduced into complex event processing. The architecture is extended by a single component, the state handler. Figure 7.7 shows the resulting architecture with the state handler.

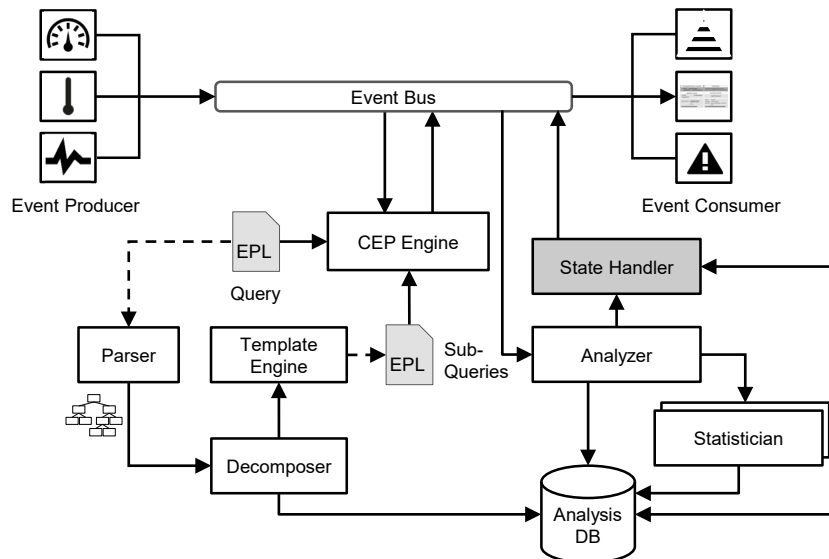


Figure 7.7: Extended Architecture with State Handler

The new component incorporates the previously defined transition conditions and determines the correct state of a situation. The settings for the implementation of the situation life-cycle and its transitions are configurable by means of a configuration file. Thus, the implementation of the component itself does not need to be modified in case another partial pattern metric has to be used or thresholds need to be adjusted. First, the state handler calculates the current materialization of a situation on basis of a partial match. Based on this result and the comparison with the defined conditions for every state, the state handler derives the current situation state. By continuously tracking the materialization and updating a situation's state, the new component implements the capability of tracking a situation through its life-cycle.

The data flow illustrated in Figure 7.8 shows the detailed mode of operation of the new component. The state handler is embedded in the workflow of the analysis part of the architecture and uses the enriched partial match events as its input. Additionally, the query of the underlying situation is derived from the database. By extracting the detected events from the partial match event and comparing it with the required events for the full pattern, it calculates the materialization metric. Depending on its configuration, S-PPF or R-PPF is used to determine a situation's maturity. Comparing the materialization value with the defined thresholds delivers the correct state for the situation at hand. By default, the state handler compares the materialization with the defined limits for the states *initialized* and *looming*. In case no further prediction model is defined by a domain expert, the state handler evaluates whether the probability of the partial match to evolve to its full match is higher than the configured threshold for the predicted state. Therefore, it accesses the database to derive the current statistics that are calculated by the sub-pattern statistician. One further task of the state handler is determining the date of expiry for the previously detected situation state. This is necessary to ensure the consistency of a situation's state in the life-cycle.

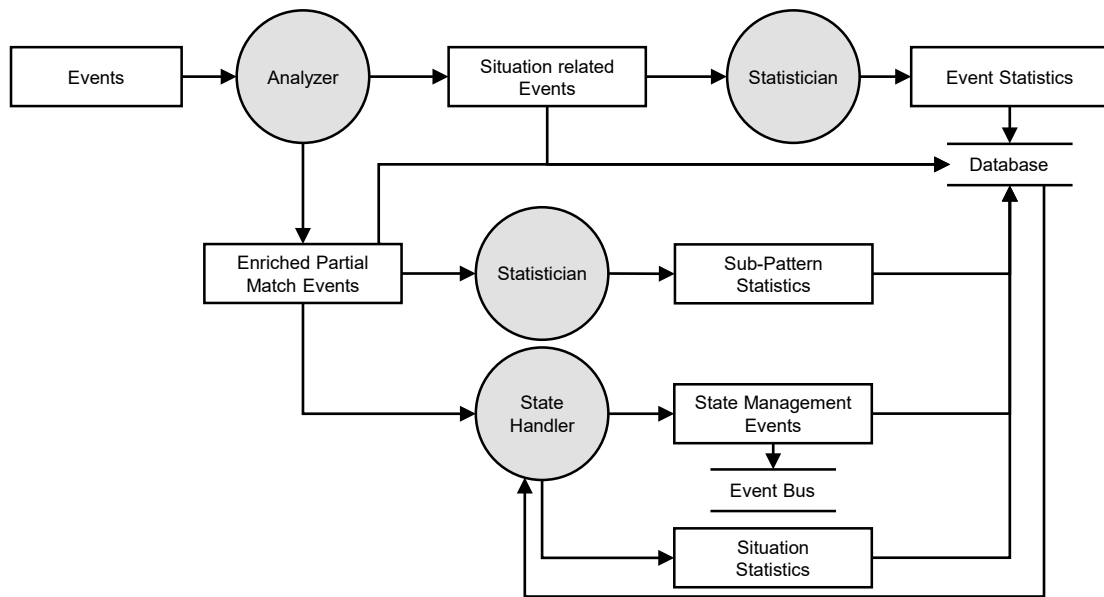


Figure 7.8: Data Flow of Extended Architecture with State Handler

The expiry date is calculated by extracting time windows from the known query of the situation and adding it to the occurrence time of the partial match. The notion behind this approach is that any further evolution of the situation has to happen within this time interval, as otherwise the temporal condition becomes invalid. After determining the correct state and its expiry date, the state handler generates a *State Management Event* that encompasses the situation ID, the detected events, the remaining events required for a full match, the expiry date and, if applicable, the probability of occurrence. Additionally, if a situation is tracked for the first time or the configuration of the state handler has changed, a *Situation Initialization Event* is sent to the event bus to inform potential event consumers of the set-up parameters of the component and the meta-data of the situation for visualization. Finally, the existing situation statistics are extended and state information is added for a better overview of historic situation occurrences. The determined data is stored in the database for potential retrospective analysis of situation occurrences and their evolution over time.

7.5 Running Example

For a better understanding, the mode of operation of the state handler, the underlying metric and life-cycle is illustrated by means of the running example introduced in Section 2.2. As presented in Section 6.5, the situation of interest, namely the dull tool, is known by the system as its query is already registered and stored in a standardized format in the database. The state handler is configured as shown in Table 7.1.

Parameter	Setting
materialization metric	S-PPF
initialization threshold	0.45
looming threshold	0.75
predicted threshold	0.90

Table 7.1: Parameter Settings for State Handler in Running Example

The state handler for the running example uses simple partial pattern fulfillment as its materialization metric and triggers a transition to the state initialized whenever a situation exceeds the threshold of 45 % materialization. In case of a materialization of 75 % or more, the situation enters the state looming. For predictions, the state handler uses the simple prediction model that is solely based on the historic comparison of occurred partial pattern matches and full pattern matches and their relative frequency. As soon as the probability for a further evolution towards the situation is higher than 90 %, the state handler triggers the transition to the state predicted. The probabilities for an evolution towards the full pattern in the running example are shown in Table 7.2 and are based on past pattern and sub-pattern occurrences of the situations *DullToolSituation* (DT) and *MaxForceSituation* (MF). The sub-pattern comprising only the *Force* event represents a valid sub-pattern for both situations (with differing event attribute values), therefore, for both situations probabilities are calculated. In contrast, the other sub-patterns are only relevant for the *DullToolSituation*, thus, probabilities are only shown for one situation.

Sub-pattern	DT	MF
<i>Force</i>	0.08	0.25
<i>Force</i> → <i>ForceRisingTrend</i>	0.43	–
<i>Force</i> → <i>MaxForce</i>	0.90	–

Table 7.2: Evolution Probabilities based on Historical Data

In the following, the behavior of the state handler is observed for the situation of a dull tool, presented in Section 2.2. As the situation has a hierarchical relationship with the MaxForce situation, it also has to be considered for an analysis. Figure 7.9 shows the excerpt from the event stream. The development of these situations and the internals of the state handler are summarized in Table 7.3.

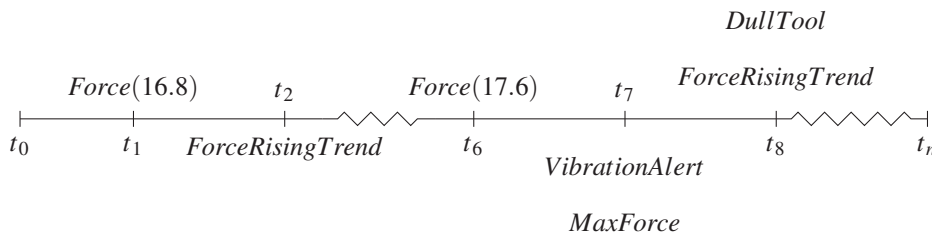


Figure 7.9: Excerpt of Event Stream

At t_0 no pattern-related event is detected in the event stream, consequently, the state handler has no information about a possible situation occurrence. Therefore, both situations remain in the state inactive. The first relevant event is detected at t_1 by means of an *Force* event with an attribute value 16.8. This specific event object is related to the DT-situation, but not to the MF-situation. Therefore, the state handler updates its internals by calculating the materialization and checking whether a transition has to be triggered. As the DT-situation pattern comprises three events and just one is detected, the materialization changes from 0 to $\frac{1}{3}$. As the threshold for a transition to the state initialized is set to 0.5, the situation's state remains unchanged. The MF-situation keeps its inactive state and its materialization of 0. Due to the filter statement in the pattern, the detected *Force* event is ignored and no partial match is detected. Also, no prediction is triggered, as the probability for an evolution to a full match is below the threshold. At t_2 , again just the DT-situation is updated, as the detected event is not encompassed in the MF-situation. With the occurrence of the *ForceRisingTrend* event, the state handler recalculates the materialization for the DT-situation. The resulting value of 0.66 lies above the defined initialization threshold and triggers a transition to the initialized state. By checking the evolution probabilities from Table 7.2, the state handler decides not to trigger a transition towards the predicted state, as the probability lies below the threshold of 0.90.

DullTool	t_0	t_1	t_2	t_6	t_7'	t_7''	t_8
detected events	0	1	2	1	2	2	3
materialization	0	0.33	0.66	0.495	0.825	0.825	1.0
state	inactive	inactive	initialized	initialized	looming	predicted	detected

MaxForce	t_0	t_1	t_2	t_6	t_7	t_8
detected events	0	0	0	1	2	2
materialization	0	0	0	0.5	1.0	1.0
state	inactive	inactive	inactive	initialized	detected	detected

Table 7.3: Development of Situations in Running Example

Between t_2 and t_6 , no further relevant events are detected, thus, no partial patterns are fulfilled. As the time window of the query describing the DT-situation expires, the state handler resets the state of the situation to inactive. At t_6 , a *Force* event occurs. The occurring event object is relevant for both situations, as its attribute value lies above the limits in both situations, respectively their queries. The state for the MF-situation is now updated to initialized, as its materialization reaches a value of 0.5. Due to the hierarchical relationship between both situations, the state handler recalculates the materialization for the DT-situation by using Formula 7.7 from Section 7.3. The calculation adds the weighted materialization of the *Force* event, which is $0.33 * 1$, and the weighted materialization of the complex event *MaxForce*, which is $0.33 * 0.5$. The resulting materialization for the DT-situation exceeds with a value of 0.495 the defined threshold. Consequently, the state handler triggers a transition to the initialized state. With the detection of the *VibrationAlert* event at t_6 , the MF-situation is fully materialized with a materialization of 1.0 and evolves to the state detected. The materialization of the DT-situation develops further to 0.825 and is calculated by using the hierarchy formula and taking into consideration the evolution of the MF-situation. Therefore, the DT-situation evolves to the looming state, as the materialization value exceeds the looming threshold. As the detected sub-pattern, *Force* \rightarrow *MaxForce*, has a probability of 0.90 to evolve further to the DT-situation, the state handler issues a prediction and triggers a state transition to predicted. Finally, at t_8 , with the occurrence of the *ForceRisingTrend* event, the DT-situation is detected.

DullToolSituation	Explanation
	This situation describes a dull tool that needs replacement.
Detected Events	Remaining Events
Materialization Metric	Prediction Metric
Metric: S-PPF Calculation: $\frac{ Detected\ Events }{ Relevant\ Events } = 66\ %$	Metric: Statistics Calculation: 43 %
State	
INITIALIZED	

Figure 7.10: Visualization at t_2

Informing potential event consumers is another task of the state handler. Therefore, it sends state management events after every update to notify about the new state, the detected and remaining events. Figure 7.10 shows a possible visualization of relevant information at t_2 to notify operators about the current state. Static information like the underlying pattern, the configuration of the state handler and an explanation of the query are communicated by the state handler at the beginning of the situation tracking by means of a situation initialization event.

7.6 Summary

This chapter presents an approach to introduce situation awareness into complex event processing extending its capabilities towards situation management. First, situation awareness as a psychological model was analyzed and its implications for a holistic situation management were examined. On basis of this analysis, the missing links between situation awareness and complex event processing were identified and a conceptual extension of the CEP paradigm was proposed. The developed solution encompasses an extended situation life-cycle, the transition logic between the life-cycle states and an implementation guideline to support the realization of situation awareness for specific use cases. By explicitly defining situation states and potential evolution paths, a situation's evolution through its life-cycle becomes trackable and supports the maintenance of situation awareness. The states of a situation were chosen to represent meaningful intermediate states of a situation on its way to its occurrence, respectively its full pattern match. They cover states representing a partial situation evolution, as well as a state indicating a prediction to incorporate level 3 situation awareness and allow projections into the future. The transitions between the states were defined dynamically in dependence of threshold values which relate to a suitable materialization metric. The purpose of the materialization metric is to capture the evidentiary degree of fulfillment of a situation and allow its tracking. Following the derived guideline, the abstract situation life-cycle becomes implementable for a specific use case. The solution in this thesis is built on top of the introduced transparent complex event processing from Chapter 6 and exploits partial pattern fulfillment within its materialization metric. Depending on the use case at hand, two different metrics were introduced, simple partial pattern fulfillment and relevance-based partial pattern fulfillment to support the emphasis of differing importance of events.

The new capabilities of situation-aware complex event processing support the implementation of a holistic situation management. By exploiting the advantages of partial pattern fulfillment and its related transparency, the controlling and predictive situation management capabilities are implemented. Through the controlling aspect, situations are tracked through their life-cycle and operators are supported in acquiring and maintaining situation awareness to be better prepared for potential consequences of occurring situations. The predictive aspect of situation management systems allows operators to stay informed about upcoming situations and proactively decide about actions to take. Finally, situation awareness and the gained insights during a situation's evolution build an important basis to support domain experts with the creation of prediction models, which is described in Chapter 8.

8

Predictive Complex Event Processing

Introducing projections into situation-aware complex event processing required the extension of the situation life-cycle by an additional state. The fundamental component for projections are prediction models. Due to the individual requirements and characteristics of various use cases it is not possible to define one prediction model that fits all problems. Instead of trying to define a general model and compromise on accuracy and flexibility, in this thesis the generation of a prediction model is supported by the system, while the task of deriving a model is left to the domain expert. By leveraging the insights from transparent and situation-aware complex event processing, the domain expert is assisted in the process of modeling. Therefore, Section 8.1 discusses *Predictive Analytics*, a suitable process model for analytics projects and its potentials to be supported by CEP systems. In Section 8.2, an approach is described to automatically label and export situation-related event traces in a structured format as input for the modeling of a prediction model. A mechanism to evaluate, monitor and deploy derived prediction models by means of complex event processing is described in Section 8.3. The introduced methods and mechanisms are illustrated along the running example in Section 8.5. The last, Section 8.6 gives a summary of the chapter.

8.1 Predictive Analytics

Looking into the future and acting proactively before a situation happens has an immense potential for social and economic factors, as prevention is often more effective than the cure [Artikis et al., 2014]. In order to become proactive, a system needs to anticipate possible happenings and predict the future. Techniques to analyze data with the goal to predict the future are summarized under the term *Predictive Analytics* (PA). It encompasses empirical methods to generate predictive models and techniques to assess their prediction power [Shmueli and Koppius, 2011]. The goal is to discover interesting and meaningful structures that are leveraged to anticipate the future development. Methods and techniques stem from several disciplines, like machine learning, data mining and statistics. Whereas this perspective covers only the methodical aspects, different solutions proposed approaches to tackle similar problems and provide a structured process model, e.g. Knowledge Discovery in Databases (KDD) [Fayyad, 1996], Cross Industry Standard Process for Data Mining (CRISP-DM) [Shearer, 2000], Team Data Science Process (TDSP) [Severtson et al., 2017]. The process models describe necessary steps in data analysis tasks in an abstract way, that allow their application in various use cases and domains. For further analysis, CRISP-DM is chosen to derive further capabilities for CEP systems to support proactivity by incorporating predictive features. CRISP-DM represents a data-centric, non-proprietary and industry-neutral process and is still the top methodology for analytics and data science projects according to [Mirski et al., 2017].

The reference model introduced by CRISP-DM describes the life-cycle of a data mining project and comprises different phases to provide a structure and guidance for the execution of it [Wirth and Hipp, 2000]. Figure 8.1 illustrates the six phases of the reference model and the most important dependencies among them. Additionally, the outer circle indicates that analytic projects are not finished with the deployment of a model, but the gained analysis insights trigger new questions and models. The single phases, tasks and goals are described in the following.

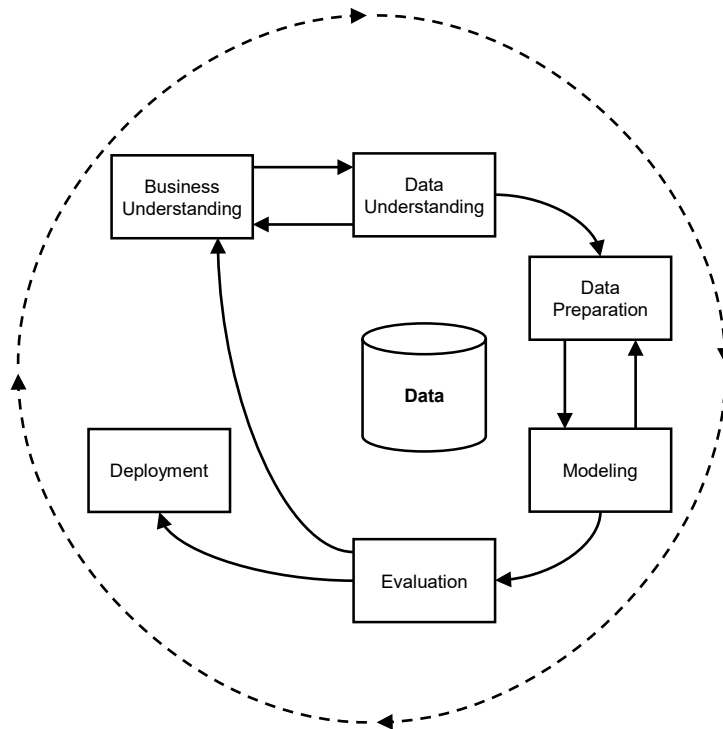


Figure 8.1: CRISP-DM Phases according [Wirth and Hipp, 2000]

- **Data:** The data is in the focus of the process model and all phases are arranged around it.
- **Business Understanding:** The first phase covers the understanding of the project objectives and requirements from the business side. The goal is to define the question that has to be answered by involving domain experts to reveal the relationship between various aspects of the domain under consideration.
- **Data Understanding:** After defining the question and understanding the problem, relevant data sources are identified and analyzed to comprehend their meaning with regard to the problem. The challenge is to identify data, that truly correlates with the problem under consideration. The phases of business and data understanding are closely related to each other, as the one requires the other.
- **Data Preparation:** The goal is the construction of the final data set that serves as input for the modeling. The preparation encompasses tasks like data selection, transformation, cleaning, dimension reduction, aggregation and enriching. Improving data quality has the highest impact on model accuracy [Rajasekharan, 2015].
- **Modeling:** The main aspect of a predictive analytics project is to determine a prediction model with a high accuracy. Within this phase different techniques, like Neural Networks or Decision Trees, are applied and checked for their suitability. The goal is to find a model with a suitable parameter set-up that answers the business question. Therefore, two steps are conducted multiple times: training of a model with a training data set and evaluation of this model with a test data set. If the results are not sufficient, the parameters of the model are adopted and another learning cycle is triggered. This phase is related to the data preparation phase, as often during modeling new requirements for data arise [Wirth and Hipp, 2000].
- **Evaluation:** The evaluation of prediction models is a central task within the process model. As more than one model might be found, it is important to evaluate the models further and not rely on the first evaluation during modeling. The goal is to check whether all business considerations are incorporated. Instead of just evaluating statistical key performance indicators, as in the previous phase, the suitability of a model for a productive environment is checked. At the end of the phase, a decision is made which model to deploy.
- **Deployment:** Finally, the evaluated model needs to be deployed in the productive environment. Depending on the needs, the model is integrated in an automatic process or is manually used.

The process model shows that although the prediction model is the central result of a predictive analytics project, many other tasks have to be conducted. According to [Rajasekharan, 2015], the preparation phases for modeling are the most time-consuming and crucial phases for predictive analytics. Supporting the phases around the model generation has the potential to reduce the development times of prediction models. For a holistic situation management system, complex event processing has to support predictions to implement predictive situation management and support level 3 situation awareness. The question that arises after the analysis of the process model is whether complex event processing is suitable to support the generation of a prediction model and how to ease the whole process for a domain expert. This question is discussed in detail in the following section.

8.1.1 Predictive Analytics and Complex Event Processing

As already identified in [Fülöp et al., 2010], predictive analytics and complex event processing are capable to support and complement each other, e.g. by using the results from predictive analytics to introduce predictions into CEP systems. With regard to the previously introduced process model for predictive analytics projects, this section analyzes whether and how CEP systems have to be extended to support predictive analytics. Situations of interest are a central element in complex event processing, as they represent a higher-level event that has significant impact on the business. Queries describing those situations encode all relevant information required to detect the occurrence of such a situation. With regard to the process model, queries and the encompassed pattern descriptions have the potential to support domain experts in the generation of prediction models. Figure 8.2 illustrates the potential application of complex event processing to support the process model for predictive analytics.

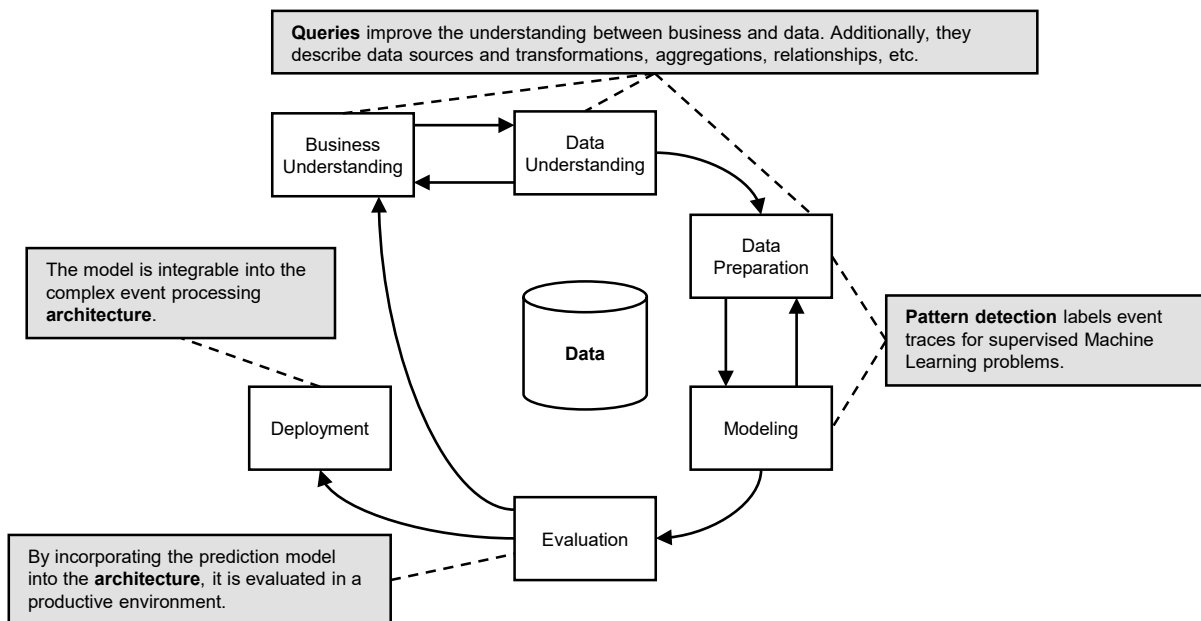


Figure 8.2: CRISP-DM Phases and Complex Event Processing

In a first step, queries improve the understanding between business and data. They describe the correlation between events to define business situations and represent an important source of information for a better understanding of the interrelationship between the first two phases. With regard to the data preparation phase, queries bridge the transition coming from the data understanding phase, as for example necessary aggregations are already considered in the pattern as part of a query. Thus, event processing is capable to serve as a preparation tool to transform the data and deliver it in a format suitable for further processing. In the modeling phase, data is required to generate a predictive model. Therefore, collected data is separated into two sets: training and testing data.

Depending on the prediction problem under consideration, supervised and unsupervised techniques are applied. Generating data represents a tedious task, especially if labeled data sets are required for supervised learning problems. Complex event processing has the capability to automate this task by analyzing event streams and deciding which set of events fulfills a defined pattern by evaluating the deployed queries. Once such an event trace matches a pattern, it is labeled with the associated situation. By drilling down from the situation occurrence, all partial pattern matches along the situation life-cycle down to the single comprised events are labeled as well. Using the newly generated data requires an extraction mechanism that provides the collected data in a suitable, structured format for the modeling phase, as the structure and format often represent a barrier for further analysis and need to be reformatted [Jagadish et al., 2014]. The modeling phase encompasses several manual steps and is not further supported by complex event processing, whereas the evaluation phase again offers the potential for an automation support. During the generation of the prediction model, the resulting model is evaluated with a limited set of test data. As the data set only represents a small excerpt of the overall data, domain experts often seek for an evaluation in a productive environment. The challenge with regard to this is that labeled data is required in the productive environment to further evaluate the model. But often this data has to be labeled manually as explained earlier. By incorporating the derived prediction model into the processing workflow of a CEP system, it is possible to automatically evaluate the model in a productive environment before deciding to deploy it for productive usage. Once the evaluation leads to the decision to deploy the model, it is integrated into the additional processing layer for situation awareness to be used automatically for situation predictions.

In summary, complex event processing has the capabilities to support different phases of the CRISP-DM process model for predictive analytics projects. While some are based on the CEP paradigm and its related concepts like queries and patterns, others require a technical extension of the introduced processing layer for situation-aware CEP systems. Based on the transparency and situation awareness presented in preceding chapters, this chapter describes the required extensions for new capabilities to support the generation and usage of prediction models.

8.2 Event Trace Labeling and Event Export Selection

As described, complex event processing systems analyze an event stream to detect predefined patterns that describe a contextual and temporal relationship between events. Therefore, events are examined whether they fulfill the defined conditions before being considered as a member of the *participating set*. As more and more participating events enter this set, the CEP system evaluates whether the relationship conditions, defined in the pattern part of a query, are fulfilled. Once a matching combination of events is found, the *matching event set* is forwarded to the derivation step of the pattern detection. Matching sets are also called *Event Traces* and represent a documentation of a situation occurrence that precisely describes the state of the environment at that point in time.

The goal of the modeling phase is the creation of a prediction model to project the future development of a situation and whether it will occur or not. Relevant input encompasses all aspects that are related to the situation of interest. Identifying the relevant sources of data and its correlation is a fundamental prerequisite for a successful modeling phase. Due to the descriptive and informative nature of queries, relevant sources of data and their relationship are already available. By applying the queries on the event stream, data sets are generated for the modeling phase. The resulting event traces, encompassing the matching set of events, represent the basis for the analysis. Therefore, the processing logic of situation aware complex event processing is extended to support the automatic labeling of event traces. The general sequence of activities is illustrated in Figure 8.3. The questions to be answered are how the event traces are labeled during the processing of event streams and how to derive labels for various classes of event traces. The labels have to refer to the situation of interest to offer one identifier throughout the whole analysis.

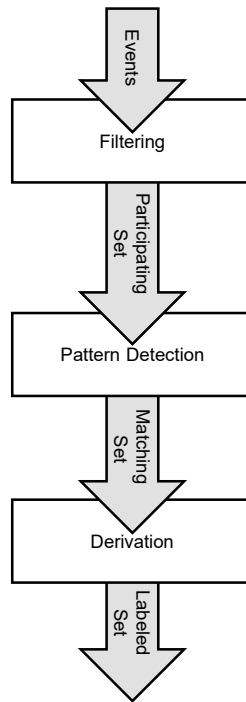


Figure 8.3: Event Trace Labeling

Elements like partial pattern matches or single events objects, that are related to a situation of interest are already captured and stored in the database. As presented in Chapter 6, on basis of a query, sub-queries are derived representing the partial pattern matches. Signifying a partial pattern fulfillment is done by generating a complex event comprising the unique identifier of the sub-pattern and the related situation. Along the way of a situation's evolution through its life-cycle, all relevant event and sub-pattern occurrences are already stored to enable the investigative situation management and allow the retrospective analysis of situation occurrences. To enable the automatic labeling, the existing processing logic is extended by recursively adding the situation label to all related elements. Thus, once the situation is detected, the already stored elements are modified and the situation ID is added as a label to indicate, that this specific element was part of an occurred situation. The result is a steadily growing data set of event objects, (sub-)pattern matches and related event traces with the information whether this set of events has led to a situation occurrence. This allows the continuous generation of new training data as input for succeeding model updates, as the behavior of the environment might change over time.

Situation-related data elements are collected on multiple levels to capture different perspectives on a situation. The illustration in Figure 8.4 shows various event traces describing different constellations during the collection of situation-related data. The first example, shown in the left part of the illustration, presents the collected data objects for an occurred dull tool situation, known from the running example in this thesis. It comprises all complex events indicating a partial pattern match for situation the of a dull tool and its related max force situation, as both situations have a hierarchical relationship. The dull tool situation has three sub-patterns (DT1, DT2, DT3) that are derived during the transparent processing, whereas the max force situation has only one sub-pattern (MF1). Each stored data element that relates to a complex event indicating a partial pattern match, describes one specific occurrence of a partial pattern and is related to the specific event objects that led to its detection. As the full patterns (DT, MF) were also detected in the event stream, the event objects and partial match events are labeled with the respective situation (*DullToolSituation*, *MaxForceSituation*). The first example also illustrates how one single event object, like the *Force* event, is labeled by two situations, as it is encompassed in both event traces.

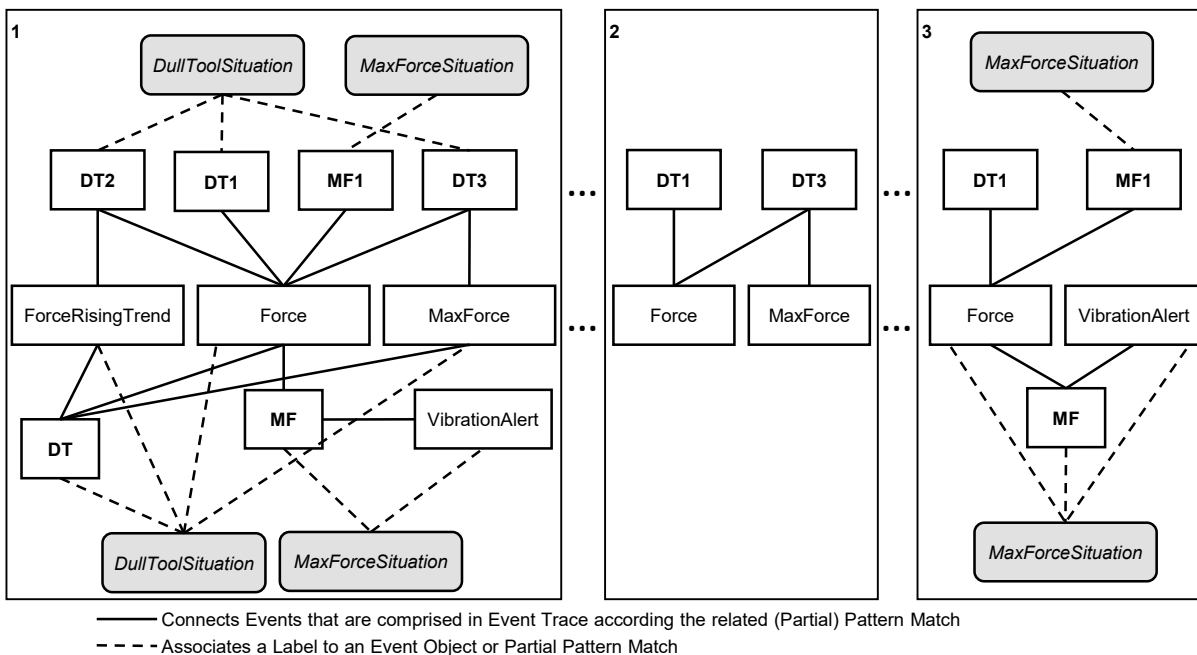


Figure 8.4: Event Trace Extraction Levels for Running Example

The second event trace, in the middle of the illustration, shows a partial detection of the dull tool situation that did not further evolve to a full pattern match. Therefore, event objects and partial match events are not labeled with the situation. Nevertheless, the data set has the potential to encompass important information for the derivation of a prediction model, which has to be assessed by the domain expert. Finally, the third data set, shown in the right part of the illustration, describes a mixed situation, where one event object is part of multiple partial pattern matches, but only one has further developed to a full pattern match. Thus, the event object is labeled by only one situation. All three examples illustrate the result of the retrospective, recursive labeling of event traces, which have to be exported to serve as a basis for further analysis steps.

```
dt3_ts , dt3_id , force_ts , force_id , force_force , mf_ts , mf_id , mf_force , mf_vibration , label1
1527520899 , 5afb...97d , 1527520897 , 5afb...85b , 17.6 , 1527520898 , 5afb...904 , 17.6 , 3.0 , DullToolSituation
```

Listing 8.1: Structured Export for DT3

To support domain experts in the modeling phase, the collected data has to be extracted in a structured format. Structured data formats are supported by many analysis libraries in various programming languages, like R and Python. Consequently, by supporting the data extraction for structured formats, like Comma-Separated Values (CSV) files or JSON, a broad range of analysis environments are supported. The collected data is extracted in multiple files, each offering a different view on the data. Listing 8.1 shows a structured data export for the previously shown example in Figure 8.4 for all occurrences of the partial match event DT3. It shows the timestamp of the partial match, the related ID in the system referencing the complex event and all information of the involved event objects, in this case the *Force* and *MaxForce* event. The last column indicates whether the set of events, respectively partial pattern match, was involved in a full pattern materialization. Additionally, all other partial match events are extracted in a similar way, always encompassing the collected data from its perspective. The lowest level of information is an extract for every single event type, showing the captured attribute values and situations the specific event object was involved in. By extracting all collected data elements from different perspectives, the system offers a high flexibility for the domain expert and the further analysis.

The presented approach leverages the existing information about situations of interest to extract potentially relevant features for the modeling phase from the available queries. By embedding the event trace labeling into the processing logic of transparent and situation-aware complex event processing, data sets are created and exported automatically to serve as input for the model generation.

8.3 Continuous Precision Monitoring

After the modeling phase, which is done manually by the domain expert, the resulting prediction model has to be embedded in the situation-aware complex event processing system to be used for projections into the future. By extending the capabilities of CEP architectures to incorporate sophisticated prediction models, the projections of situations can be further improved compared to predictions based on pure statistics. Additionally, the deployment and evaluation phase of the CRISP-DM process model are supportable. Although the evaluation phase precedes the deployment phase, first deployment strategies are discussed, as the deployment builds the basis for an automatic evaluation of prediction models.

In dependence of the underlying use case, developed prediction models are executed in various ways. If single predictions are required occasionally, the model is executed manually within its development environment whenever needed. A second approach, suitable for use cases with a continuous model execution, are scoring engines supporting different formats to deploy a prediction model, like PMML (Predictive Model Markup Language). The advantage is, that models can be trained in any environment, exported into a standardized format and transferred to other environments without having to adopt the model. Several scoring engines exist, varying from free to commercial products, like JPMML¹ and Zementis Predictive Analytics². Scoring engines are embedded in the productive environment and integrated into the data workflow to continuously generate predictions that are further processed in a succeeding workflow step. The most flexible approach, often also supported by scoring engines, follows the idea of encapsulating the predictive functionality behind a service and calling the prediction model, e.g. by means of HTTP requests and a REST-API (Representational State Transfer Application Programming Interface), where the input parameters are handed over as part of the request. For an automatic model evaluation, the prediction model has to be made available through a service or by integrating a scoring engine directly into the processing. The decision has to be made under consideration of the use case requirements, e.g. whether the existing scoring engine is capable of executing the required model, as not every engine supports all prediction models. Thus, it might be necessary to implement it specifically for a use case and encapsulate the logic as a service.

Enabling the automatic evaluation of models, requires a definition of the frame conditions. The following restrictions are defined with regard to the prediction models: i) the models predict the probability of occurrence for defined situations in the CEP environment, ii) per situation multiple models might exist that are based on different feature sets (like single events or various partial matches) and iii) the models take the respective feature set as input. In context of the running example and the previous illustration, prediction models are generated for the partial matches of the dull tool situation, leading to three models, one per partial pattern (DT1, DT2, DT3). Figure 8.5 illustrates the approach to incorporate the prediction model: for every detection of an involved pattern or partial pattern match, the feature set is forwarded to the prediction model and a predicted situation with its probability is returned as a result. Embedding prediction models into the regular processing logic to derive projections into the future requires an extension of the capabilities to incorporate models.

¹ <https://openscoring.io/>

² https://www.softwareag.com/corporate/products/apama_webmethods/zementis/default.html

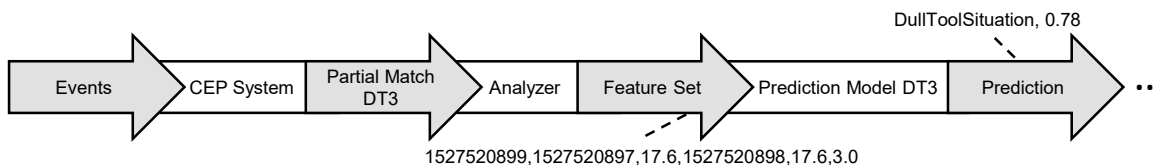


Figure 8.5: Situation Prediction based on Feature Sets

A prediction model has to be evaluated before being deployed for productive usage. Therefore, the results of the embedded model from the previous step are used to continuously monitor performance metrics – like precision and recall – during the processing of productive data. Once the performance of the model is successfully evaluated, it is activated for productive usage and is used to project a situation’s future development. Therefore, an extension is required that allows the continuous monitoring of a model’s performance and a mechanism to activate it once it is considered suitable. The technical details of the presented approach are described in the following section by extending the architecture for situation-aware complex event processing.

8.4 Architecture

After introducing the conceptual approach to support the generation and evaluation of prediction models, the architectural extension is described in the following. It is based on the architecture of transparent and situation-aware complex event processing, as the data is already collected during the transparent processing of events for a retrospective analysis and is further enriched during the tracking of a situation’s evolution through the introduced life-cycle. The architecture is extended by three components introducing the described new capabilities. Figure 8.6 illustrates the resulting architecture with the new components, namely *Prediction Handler*, *Performance Monitor* and *Data Labeler & Exporter*. For the sake of completeness, the manual task of modeling a prediction model is also shown in the illustration, as it consumes data and delivers the resulting model as input for the new components.

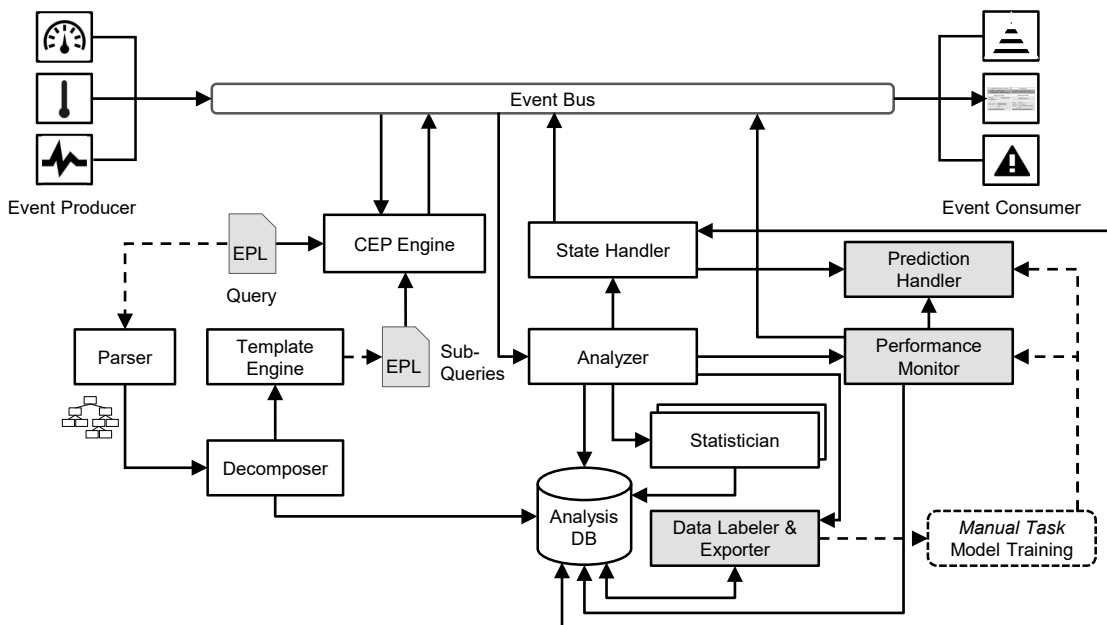


Figure 8.6: Situation Prediction based on Feature Sets

The first step to support the CRISP-DM process model encompasses the data collection and preparation. Therefore, the data labeler & exporter component is triggered whenever a full pattern match is detected to retrospectively label all involved events and partial pattern matches in the database. The data extraction is done continuously and is stored in a structured format. The exported files are afterwards used in a manual step to derive a prediction model. Embedding the resulting model is done by means of the prediction handler, which is capable of executing prediction models directly through an integrated scoring engine or calling an external prediction service through a request. Before activating the productive projection into the future, the incorporated models have to be tested and evaluated. This task is taken over by the performance monitor, that triggers predictions by means of the prediction handler and checks whether a prediction becomes true or not. After a successful evaluation, the model is used in the productive environment by the state handler to derive projections to realize level 3 situation awareness.

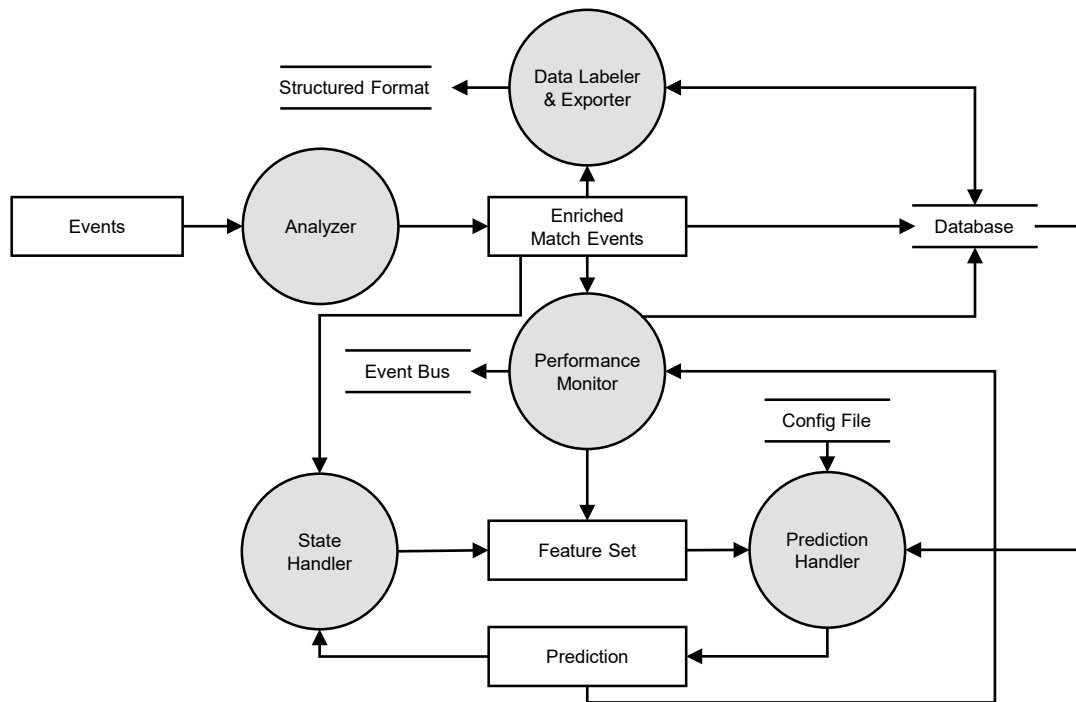


Figure 8.7: Prediction Support Data Flow

The data flow, illustrated in Figure 8.7, shows the detailed integration of the new components into the existing architecture. For reasons of clarity, the previously described data flows for situation-aware complex event processing are not shown in the illustration. The analyzer is extended and integrates two of the new components by handing over the enriched match events, that incorporate the patterns, respectively sub-patterns and relevant events that participated in their occurrence. The data labeler & exporter is triggered by full pattern matches, as its task is to retrospectively label historic sub-pattern and event occurrences. Therefore, based on the encompassed event information in the enriched match event, it finds all relevant data elements in the database and adds the label of the situation. The labeled data is continuously exported as a file in a structured format as new full pattern matches are detected. The data export serves as input for the manual prediction model generation.

The prediction handler offers two options to integrate prediction models: either by importing them into a scoring engine in the architecture to directly derive predictions or by providing a URL to access a prediction service. As described earlier, related to one situation, multiple prediction models might be generated in dependence of the input, as single event objects or partial pattern matches are taken into consideration. The assignment of prediction models to input feature sets is done via a configuration file, listing the covered input and its related model.

In case a URL is assigned, the prediction handler calls the defined endpoint and hands over the input feature set, that is forwarded by the state handler. In any other case, it attempts to import the defined model from the database and to load it in the scoring engine. In both cases, the result is a projection encompassing the predicted situation and the related probability of occurrence. The result is further processed via the state handler to check whether it is above the defined threshold for the predicted state or not. If this is the case, the situation moves on to the predicted state. The second component that receives results from the prediction handler is the performance monitor.

The performance monitor processes the enriched match events from the analyzer and analogously to the state handler extracts feature sets to trigger a prediction by the prediction handler. Afterwards, it tracks the development of the prediction and whether the projected situation occurs or not. Based on this observation, the performance monitor calculates the defined performance metrics. The performance of the monitored models is stored in the database and additionally sent to the event bus to inform the domain expert about the current performance of the models, e.g. by visualizing the results. Once a prediction model is successfully evaluated, the domain expert activates it by flagging it as an active model in the configuration file of the prediction handler. From this point on, the prediction model is actively used in the productive environment for level 3 situation awareness.

8.5 Running Example

The mode of operation of the new components is illustrated by means of the running example introduced in Section 2.2. As stated earlier, the situation of a dull tool, its related sub-queries and events are already stored in the database. The behavior of the components is illustrated along the example event stream, shown in Figure 8.8.

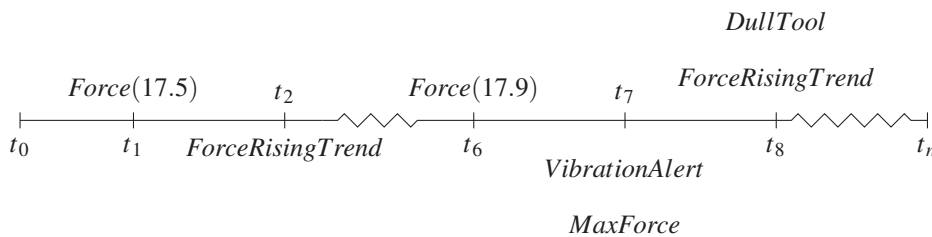


Figure 8.8: Excerpt of Event Stream

The occurring *Force* event at t_1 fulfills all requirements to be considered as a participant event in both relevant situations, namely *DullToolSituation* and *MaxForceSituation*. Thus, the detection of the *Force* event leads to partial pattern matches (DT1, MF1) for both situations that are stored in the database. With the occurrence of the *Force Rising Trend* event at t_2 , another partial pattern (DT2) is identified and stored in the database. Between t_2 and t_6 , no further situation-relevant events are detected, thus the evolution of the situation stops before reaching the detected state. The resulting set of events that is stored in the database is shown in Figure 8.9 on the left side. As no situation is detected, the state labeler component is not triggered to retrospectively label the persisted data.

This behavior changes with the next incoming events, beginning with a *Force* event at t_6 that again fulfills the conditions for both situations to become a participant event. As a *Vibration Alert* event is detected at t_7 , the *MaxForceSituation* is materialized. This full pattern match triggers the data labeler to enter the database and modify the label of the involved events. As shown in Figure 8.9, the full pattern match (MF), its related partial match (MF1) and the involved events are labeled with the situation. Additionally, the partial pattern DT3 is detected. With the occurrence of the *Force Rising Trend* event at t_8 , the full pattern of the dull tool situation is detected and the data labeler adds the correct labels again to all involved data elements. The resulting set of data elements and their labels are shown in the right part of the illustration in Figure 8.9.

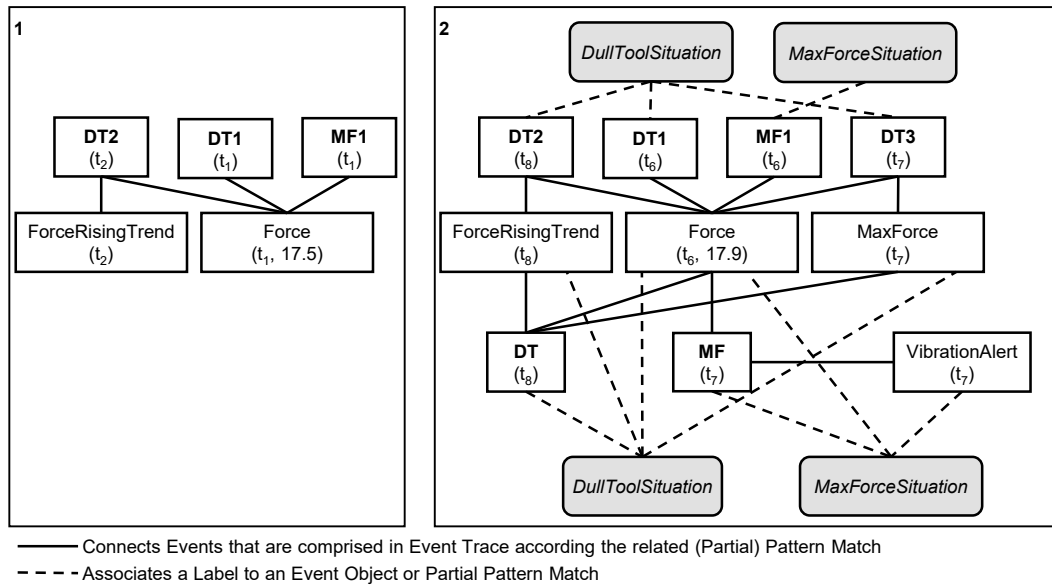


Figure 8.9: Running Example Event Traces

The growing information base is continuously exported by the data labeler & exporter. As it might be interesting for a domain expert to analyze the data from various perspectives, multiple files are exported, always encompassing the same features and structure. For every event type that is involved in a situation of interest, a file is exported showing all occurred event objects of this type and in which detected situations it was comprised. This point of view is shown in tabular form in Table 8.1: two *Force* event objects were recorded during the processing of the event stream, whereas the first has no label, as it was not involved in any situation occurrence and the second object contains two labels, as it was part of *DullToolSituation* and *MaxForceSituation*. The data is exported in a structured format to offer easy import possibilities, as many analysis tools, libraries and platforms support a wide range of structured formats. The resulting content of the file is shown in Listing 8.2.

Force ts	Force ID	Force force	label1	label2
t_1	5afb...846	17.5	-	-
t_6	5afb...125	17.9	MF	DT

Table 8.1: Collected Data for Event Type *Force*

```
force_ts,force_id,force_force,label1,label2
t1,5afb...846,17.5,-,-
t6,5afb...125,17.9,MF,DT
```

Listing 8.2: Structured Format Export for Force Events

Two other perspectives are shown in Table 8.2, where exemplary the data extracts are shown for MF1, a sub-pattern of *MaxForceSituation* (MF), and DT2, a sub-pattern of *DullToolSituation* (DT). Both tables comprise the sets of events that led to a partial pattern match and whether the partial match is part of a situation occurrence or not. As all identifiers for involved events and partial matches are captured in addition, domain experts always have the option to merge multiple perspectives, respectively files together for an analysis.

The extracted elements are used as features in a manual step to generate prediction models for situations. Once a model is derived, it has to be embedded in the system. As described, various options are supported within the approach in this thesis. In the running example, the scoring is implemented by triggering an external service.

MF1 ts	MF1 ID	Force ts	Force ID	Force force	label1
t_1	5afb...97d	t_1	5afb...846	17.5	-
t_6	5afb...a7x	t_6	5afb...125	17.9	MF

DT2 ts	DT2 ID	FRT ts	FRT ID	FRT inc	Force ts	Force ID	Force force	label1
t_2	5afb...x7e	t_2	5afb...t9e	0.3	t_1	5afb...846	17.5	-
t_8	5afb...x23	t_8	5afb...t34	0.4	t_6	5afb...125	17.9	DT

Table 8.2: Collected Data for Partial Matches MF1 and DT2

The related data element, like the partial match DT2, serves as an input for this request. Therefore, the configuration file has to be adopted. As shown in Chapter 6, every complex event indicating a partial pattern match has a unique identifier which is automatically assigned by the system. This ID is used in the configuration file to map a prediction service to a partial pattern. Listing 8.3 shows an excerpt of an exemplary configuration file for the running example, assigning the URL of a REST-API for predictions based on the partial pattern match DT2 and the *Force* event.

```

predictions:
  PartialMatch_5ae1d727a02199717caf6d0a: http://192.168.12.122/predictions/dulltool/dt2
  Force: http://192.168.12.122/predictions/dulltool/force
activated:
  - PartialMatch_5ae1d727a02199717caf6d0a

```

Listing 8.3: Prediction Configuration

The default behavior of the prediction handler is to request a prediction and hand over the feature set as input. The feature set is generated according to the mapped data element, i.e. it always contains all features in the same structure as it was used for the training of the model with the structured data extract. Listing 8.4 shows an example input used to trigger a prediction for an occurred partial match (DT2). In this example, it is handed over in the standardized JSON format. The result of the prediction is as well returned in the JSON format and comprises the predicted situation and a probability of occurrence.

```

{
  "dt2_ts": "t10",
  "frt_ts": "t10",
  "frt_inc": 0.4,
  "force_ts": "t9",
  "force_force": 18.1
}

```

Listing 8.4: DT2 Input for Prediction

The defined prediction models are used for evaluation purposes to measure the performance of a model. Therefore, the performance monitor uses the defined prediction models by triggering the prediction handler and keeps track of the development and whether the predicted situation occurs within a valid time window or not. Consequently, the performance monitor is able to capture the model performance and inform the domain expert about the performance metrics of a model. Once the domain expert decides that the model performance is sufficient for a productive usage, the prediction model is enabled by activating it in the configuration file (see Listing 8.3).

8.6 Summary

In this chapter, the CEP technology was leveraged to support the introduction of predictive capabilities into complex event processing. First, potentials were examined by analyzing the field of predictive analytics and suitable process models that describe a structured approach for the realization of predictive functionalities. Within the CRISP-DM process model, various phases were identified for a support by extended CEP capabilities with the goal to assist domain experts in the modeling, evaluation and execution of predictive models. By leveraging the captured insights from transparent and situation-aware complex event processing, a conceptual extension of the processing layer was described to automatically label and extract situation-related data from multiple perspectives in a structured format for further analysis. The resulting processing logic and its results support domain experts with the manual generation of predictive models. Additionally, the extension also tackles the challenge of embedding predictive models in situation-aware complex event processing to project a situation's future evolution and automatically evaluate deployed models, as new data sets are continuously created by the underlying CEP engine. Consequently, models are evaluated in a productive environment before being used in a productive setup.

The new capabilities of predictive complex event processing support the proactive situation management. Through the integrative design of the extended processing logic, additional prediction models might be integrated to further enhance the predictive functionality, e.g. by adding a model to predict the time of occurrence for a situation of interest. The predictive capabilities complement the reactive nature of complex event processing. In case a prediction is wrong, the underlying CEP system identifies the occurrence of a situation and reacts accordingly. The specific data set for this false prediction is stored and is available for a deeper analysis to improve the prediction model. The enhanced capabilities allow the extraction of relevant features on basis of defined queries. By highly depending on the queries, one risk of the presented approach is to introduce a high bias into the prediction model, as all features and labeled data sets are derived from the information in the query. This aspect also puts queries and the encompassed pattern into focus, as a wrong or inaccurately modeled pattern leads to an inaccurate prediction model. Therefore, it is necessary that a domain expert always stays in the control loop during the creation of the model and actively decides that the model performance is sufficient for productive usage.

Part IV

Finale

9

Evaluation

Within this chapter, the introduced approach for situation management on basis of complex event processing is evaluated. The evaluations are based on developed software artifacts that implement the presented concepts, methods and models in this thesis. In Section 9.1, the evaluation strategy and the underlying implementation are described. Section 9.2 presents a conceptual investigation to evaluate the fulfillment of requirements towards an approach to answer the stated research questions in this thesis. In Section 9.3, the overall situation management system is evaluated by means of a user study to assess the usability and perceived usefulness of the offered capabilities. Performance aspects of the introduced approach are assessed in Section 9.4.

9.1 Evaluation Strategy

In course of this thesis, on basis of complex event processing, a holistic situation management system was stepwise introduced. As all three conceptual extensions presented in Chapter 6, 7 and 8 build upon each other, the evaluation is also done in a combined evaluation. The goal of the evaluation is to assess whether the identified gaps and problems are solved by the contributions of this thesis. As the thesis follows the design science paradigm for information systems research, an evaluation of the derived artifacts is required. Therefore, the evaluation follows a three step approach: First, a conceptual investigation analyzes the fulfillment degree of the presented approach and its resulting artifacts compared to the elicited requirements in Chapter 5. The primary evaluation metric for a situation management system is its capability to support domain experts within monitoring scenarios by offering three required functionalities of situation management as presented in Chapter 4: the investigative, controlling and predictive management of situations of interest. Therefore, as a second step in the evaluation, an expert user survey was conducted to assess the offered functionality and whether it is considered as useful for situation management or not. Finally, performance experiments were conducted to obtain insights on the run-time performance for the presented running example and to evaluate the impact of the additional processing layer on the underlying CEP architecture. Additionally, the suitability of the introduced approach for high-throughput use cases was examined to assess its technical limitations.

9.1.1 Software Artifacts

The introduced concepts, methods and models for transparent, situation-aware and predictive complex event processing are not bound to a specific programming language, framework or tool. The resulting situation management system demands only one prerequisite to be fulfilled: the underlying architecture has to follow the CEP paradigm. In the course of this thesis, a reference implementation was developed, which is also used for the evaluation in this chapter. The implementation follows the presented architectural design for a situation management system (see final design in Section 8.4) and offers all required capabilities. The developed modules are shown in Figure 9.1 and briefly described in the following.

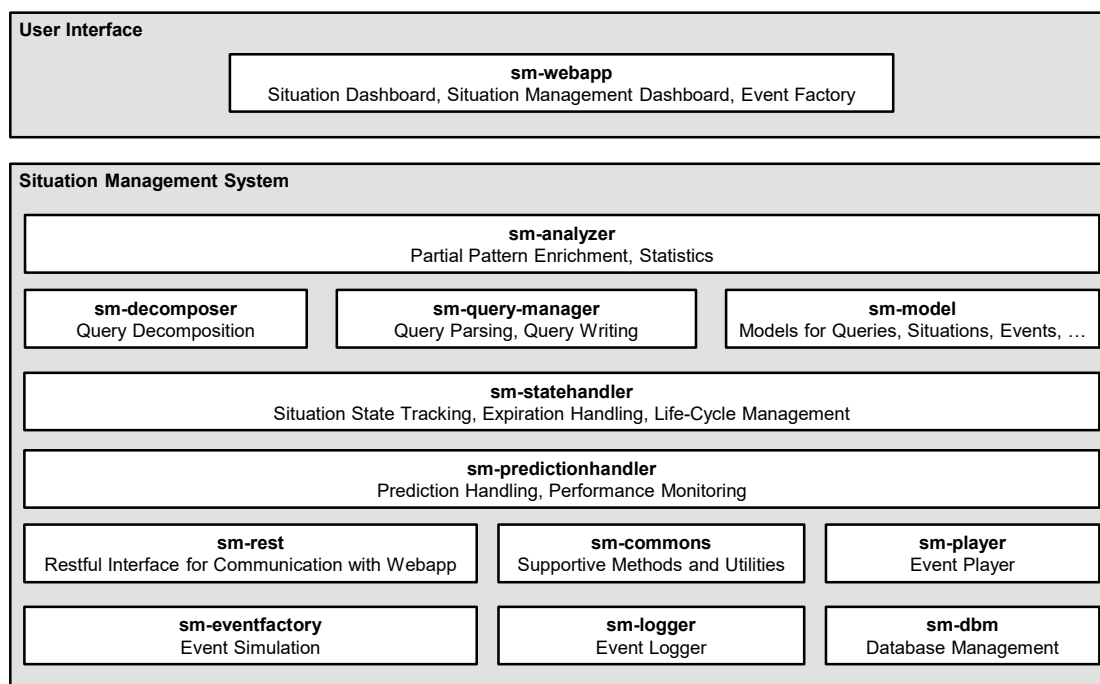


Figure 9.1: Implementation Overview

The graphical user interface (*sm-webapp*) encompasses the visualization of a situation dashboard, representing simple monitoring capabilities following the reactive nature of a basic CEP architecture, a situation management dashboard incorporating the enhanced capabilities of a situation management system and an event factory. The event factory allows tests with synthetically generated data by defining a set of events and patterns to be generated automatically. The graphical user interface is implemented with the Angular¹ web application platform.

The first two layers of the situation management system comprise modules involved in the transparent processing of event streams. The *sm-analyzer* module represents the central component that analyzes occurring situations, their related partial patterns and enriches the data to calculate various statistics. The remaining three modules – *sm-decomposer*, *sm-query-manager* and *sm-model* – are related to the management of engine-specific queries, their transformation into the introduced common model for situations of interest and their decomposition into valid sub-queries. The third layer (*sm-statehandler*) implements the concepts and methods behind situation-aware complex event processing by tracking a situations development through its life-cycle and handling expiring states. The prediction support targeted at in research question 3 to establish predictive situation management is implemented by the module *sm-predictionhandler* that is capable of triggering predictions, monitoring the performance of a prediction model and automatically labeling and extracting situation-related data in a structured format. The remaining two layers consist of auxiliary modules, e.g. the REST-API for communication between the backend and the user interface or the database management component for accessing the underlying database. All backend modules are developed with Java 8², whereas the *sm-analyzer*, *sm-statehandler* and *sm-predictionhandler* are implemented as Apache Flink³ programs in Java 8 to leverage its stream processing capabilities.

¹ <https://angular.io/>

² <https://www.java.com/>

³ <https://flink.apache.org/>

9.2 Conceptual Investigation

The requirements towards a situation management system, derived in Chapter 5, encompass capability- and system-related requirements. In the following, their fulfillment is discussed by using a conceptual investigation method and mapping developed concepts, methods and models to it. Figure 9.2 shows the three research questions of this thesis and how the different requirements are related to each question.

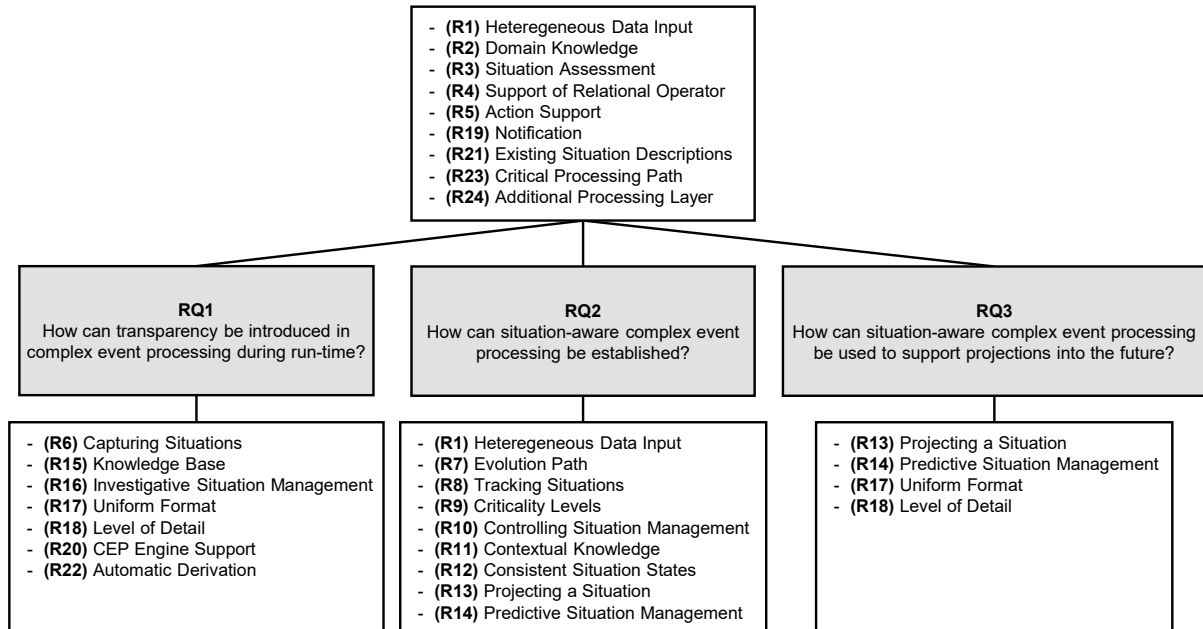


Figure 9.2: Research Questions and related Requirements

9.2.1 Fulfillment of Requirements

The first set of requirements **(R1-R19)** relates to needed functionalities to implement a holistic situation management. After reviewing the the capability requirements, the system-related requirements are examined and discussed. Whereas the capabilities have an influence on the question *what* a situation management system has to offer, the technical requirements aim at the question *how* to realize these functionalities and has an impact on the architectural design and implementation of the solution.

(R1) Heterogeneous Data Input, (R2) Domain Knowledge, (R3) Situation Assessment, (R4) Support of Relational Operator and (R5) Action Support

In contrast to other solutions discussed in Chapter 4, the approach in this thesis aims at establishing a dynamic situation management suitable for various fields of application. Therefore, the situation management is based on complex event processing, as various requirements are fulfilled by leveraging its capabilities. Technologies relying on event-driven architectures, like complex event processing, support the most common data types and allow arbitrary combinations for more complex data structures, i.e. composite data elements. Domain knowledge is incorporated into the system through the definition of queries describing the business-related questions by modeling the relationship between events that correlate to happenings in the domain. For this task, domain experts draw on typical relational operators, e.g. sequences, disjunctions, conjunctions and combinations of them. This enables the definition of structural and time-related relationships. Detecting situations is accomplished by evaluating defined

queries in near real-time. Additionally, action support is given, as reactions are triggered automatically or recommendations for a reaction are given. Therefore, the requirements **R1**, **R2**, **R3**, **R4** and **R5** are fulfilled by building on top of CEP and further enhancing it. All requirements serve as basis for the situation management system and relate to all three research questions, as all enhanced capabilities are based on these fundamental requirements.

(R6) Capturing Situations, (R7) Evolution Path, (R8) Tracking Situations, (R9) Criticality Levels and (R10) Controlling Situation Management

The introduction of transparency in complex event processing by observing the intermediate development of situations during the processing of event streams allows to capture current situation information and its current state. Thus, by the incorporation of partial pattern fulfillment in the processing logic requirement **R6** is fulfilled. With the introduction of an explicit situation life-cycle for situation-aware complex event processing and by assigning a meaning to each state in the life-cycle, requirements **R7** and **R9** are fulfilled. By exploiting the transparency achieved with the realization of requirement **R6** and introducing materialization metrics based on this information, requirement **R8** is fulfilled and situation-aware complex event processing allows the tracking of a situation's state through the defined evolution path. All those requirements together lead to the fulfillment of **R10** and allow the purposeful direction of attention with the goal of controlling a situation as part of a holistic situation management system. The aspect of controlling situation management is further evaluated during the user study in Section 9.3.

(R11) Contextual Knowledge

Allowing the incorporation of contextual knowledge that is related to the characteristics of the use case improves the generalizability of a solution. With the introduction of an adoptable transition mechanism for situation-aware complex event processing, the criticality levels of the situation life-cycle can be fine-tuned with regard to the needs of the underlying use case. The transition mechanism, introduced in Chapter 7, fulfills requirement **R11** and allows the embedding of contextual knowledge.

(R12) Consistent Situation States

Supporting an operator requires the situation-aware complex event processing to ensure a consistent situation state as it is crucial for the direction of an operator's attention. Therefore, requirement **R12** is fulfilled by the introduction of a consistency tracking capability in the state handler required for situation awareness. The state handler tracks the evolution of a situation through its life-cycle and ensures that states are only valid as long as their expiration date and time are not yet reached.

(R13) Projecting a Situation and (R14) Predictive Situation Management

Fulfilling the requirement of a predictive situation management first necessitates the incorporation of the concept of predictions/projections into the CEP paradigm. Therefore, the situation life-cycle is extended by the state predicted to incorporate the concept of predictions into a situation's possible evolution path. Additionally, the predictive support of the extended architecture for predictive complex event processing allows the embedding of prediction models into the CEP architecture and supports the generation and evaluation of prediction models. Thus, requirements **R13** and **R14** are implemented by situation-aware and predictive complex event processing developed throughout in this thesis. The fulfillment of predictive situation management is further evaluated during the user study presented later in Section 9.3.

(R15) Knowledge Base, (R16) Investigative Situation Management, (R17) Uniform Format and (R18) Level of Detail

The investigative situation management plays an important role, as many further capabilities are based on it, e.g. prediction support for predictive complex event processing. Establishing the option to retrospectively analyze situation-related information requires the system to collect all relevant data and offer a knowledge base for further functionalities. With the introduction of transparent complex event processing, requirement **R15** is fulfilled. The extraction of situation relevant data from various perspectives within Chapter 8 further supports the fulfillment of requirement **R16** by offering a uniform format (**R17**) for analysis and extracting the data with different levels of detail. Additionally, the knowledge base allows to drill down into the data and analyze it on different abstraction levels, which fulfills requirement **R18**. The investigative aspect of the solution and its related capabilities are further evaluated during the user study in Section 9.3.

(R19) Notification

Improving trust and understanding requires the communication of situation-related observations during the processing of the event stream. Therefore, the system notifies all interested consumers about a situation's current state, the detected partial pattern matches and potential upcoming situations. The notification feature implements requirement **R19** and can be used for the visualization of relevant data or further processing in other components.

(R20) CEP Engine Support

All capabilities of the situation management system developed in this thesis rely on the underlying CEP engine. Therefore, the requirement was derived to introduce a mechanism to support arbitrary existing CEP engines. First, for transparent complex event processing, an abstract unifying model is introduced to allow an engine-independent representation of situations of interest. The developed mechanism requires solely a description of the grammar of the underlying event pattern language and adoptions towards the mapping between language constructs and their representation in the model. This approach fulfills requirement **R20** and is implemented for two state-of-the-art CEP engines, namely Apama Streaming Analytics⁴ and Esper⁵.

(R21) Existing Situation Descriptions and (R22) Automatic Derivation

To ease the usage of the extended capabilities of the developed solution in this thesis, requirements were derived to minimize the effort needed to make use of the enhanced functionalities. Therefore, existing situation descriptions serve as the basis for transparent complex event processing and are imported into the situation management system in their abstract representation, as presented in Chapter 6. The internal model of situations is used for situation-aware and predictive complex event processing. All further required aspects of a situation of interest are automatically derived by the system. Therefore, requirements **R21** and **R22** are fulfilled.

⁴ https://www.softwareag.com/corporate/products/apama_webmethods/analytics/default.html

⁵ <http://www.espertech.com/esper/>

(R23) Critical Processing Path and (R24) Additional Processing Layer

The central functionality of an existing CEP architecture is the detection of occurring situations of interest. This critical processing path serves as the baseline for the monitoring system and should not be negatively affected by the extended architecture for transparent, situation-aware and predictive complex event processing. Fulfilling **R23** and **R24** leads to an architectural design comprising an additional processing layer to minimize the effect of the additional workload on the basic system, decouple the functionalities of both systems and allow the flexible extension with further enhanced capabilities. The impact of the extended architecture on the critical path is further evaluated in Section 9.4.

9.2.2 Discussion

As the conceptual investigation shows, all derived requirements are fulfilled and covered by the various concepts, methods and models of the presented approach in this thesis. Additionally, the fulfillment of all requirements led to an architectural design that allows the further expansion of the approach. Situation-related data is processed in an additional processing layer that allows the further enhancement of CEP systems by adding new components and extending its capabilities. The investigation additionally shows that many requirements are highly interrelated with each other, which is also reflected in the various approaches presented in Chapter 6, 7 and 8. All approaches build one upon another, with transparent complex event processing representing the fundament of the newly introduced situation management system. With every additional approach, the basic functionalities of a situation management system are introduced to cover the whole range of capabilities.

9.3 User Study

One further element of the evaluation in this thesis is the conduction of a user study with experts. The participants came from relevant domains, namely monitoring applications, event processing and prediction modeling. One additional selection criterion was that participants at least needed to have experience with CEP technology and monitoring applications. The study was performed to gain insights on how experts assess the new capabilities on basis of the contributions of this thesis. Therefore, a situation management application is compared with the functionality of a simple monitoring application on basis of the insights that a state-of-the-art CEP engine gives.

9.3.1 Design

The goal of the user study is to receive feedback from experts to evaluate different aspects of the contributions of this thesis. The basis for this survey is a questionnaire comprising four sections. First, an introduction into the running example of this thesis is given, that describes the setup of a punching environment and the problem of a dull tool. The introduction also encompasses the introduced pattern to detect a dull tool and an example event stream that represents the data set for this survey. The introduction is completed by three questions to assess the experience level of the participants within the previous mentioned relevant fields. Based on the introduced example, the following sections of the questionnaire describe three monitoring applications by showing screenshots from a running system capturing the different states of the application during the processing of the introduced example event stream and explaining the elements on the screen. The first monitoring application is solely based on state-of-the-art complex event processing systems and the information they offer. The second application follows the concepts and methods of situation-aware complex event processing introduced in this thesis. Finally, the third monitoring

application leverages the concepts and methods of transparent complex event processing and presents the collected data in addition to the situation-aware monitoring application, that allows a drill-down into the data. After each introduction of a monitoring application, a short questionnaire follows to assess multiple statements about the shown application. In total, 23 statements are presented to the participating experts to be assessed using a five point Likert scale ranging from 1 (strongly disagree) to 5 (strongly agree). Multiple identical statements are presented to the participants after each introduction of a new monitoring application. Consequently, the questionnaire comprises ten unique statements. The complete questionnaire and its results are shown in Appendix A.

Statement	Simple Situation-Aware Transparent		
The state of the system during the processing of the event stream is transparent.	S1	S8	S15
The Application encompasses interesting situation-related details.	S2	S9	S16
The Application directs the attention of an operator towards a situation at an early stage.	S3	S10	
The operator is consequently kept up-to-date of the exact state of the situation of interest.	S4	S11	S17
The development of the emerging situation of interest is comprehensible.	S5	S12	S18
The Application supports a reactive mode of operation for the operator.	S6	S13	S19
The Application supports a proactive mode of operation for the operator.	S7	S14	S20
The Application Dashboard allows the retrospective analysis of situations of interest.			S21
The automatic data extraction is suitable for further analysis.			S22
The automatic data extraction is suitable to support the generation of prediction models.			S23

Table 9.1: Mapping between Statements and Monitoring Scenarios

Table 9.1 shows the ten unique statements and for which scenarios they are considered as relevant. All statements belong to one of the following three categories: transparency, situation awareness and prediction/analysis support. The last statement of the survey is reserved for participants with experience in the generation of prediction models. All statements relate to the contributions of this thesis, that are derived as answers to the presented research questions in Chapter 1. Figure 9.3 illustrates the mapping between a statement and its related research question.

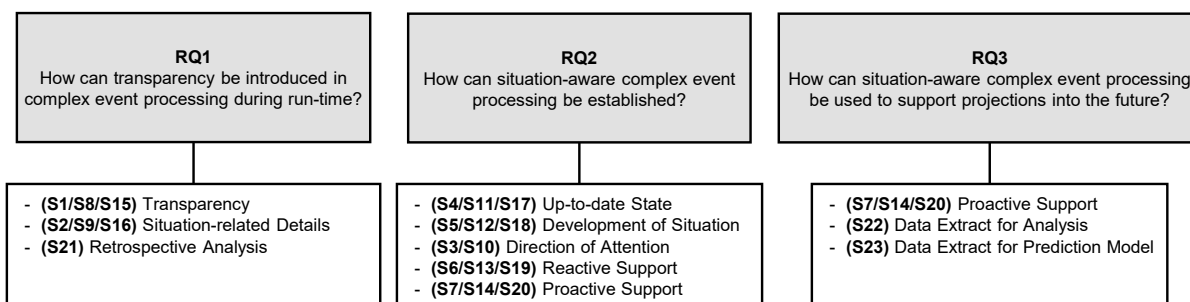


Figure 9.3: Research Questions and related Statements in the User Study

The user study targets expert users with technical skills and experience in the relevant fields, who are aware of the mode of operation of complex event processing systems, as all evaluated monitoring applications are based on the CEP paradigm. Nine individual participants were chosen from academia and industry to assess the introduced statements. As shown in Figure 9.4, all participants are experienced with CEP technology and monitoring applications with rank 4 or higher. Additionally, 7 out of 9 participants also have experience with the generation of prediction models, rated with rank 4 or higher.

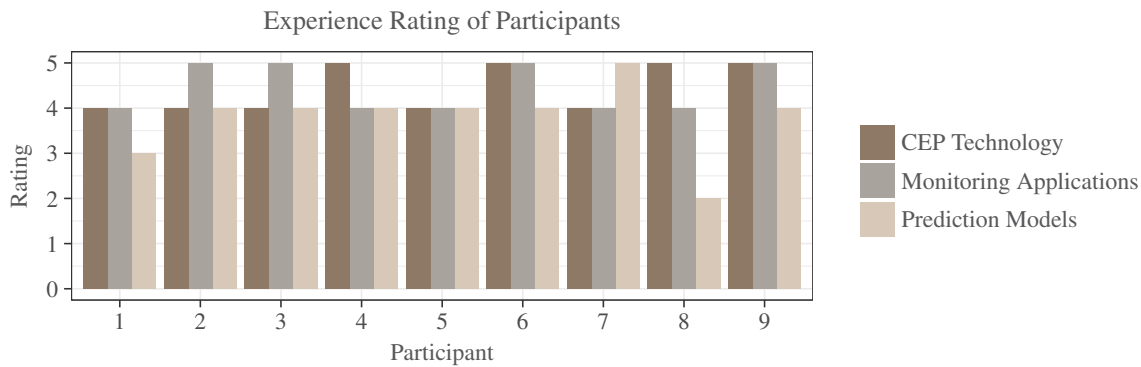


Figure 9.4: Survey Answers related to Experience

9.3.2 Results

This section presents the results of the user study. With regard to the first research question, the participants evaluated whether the approach presented in this thesis successfully introduces transparency in the processing of event streams (S1, S8 and S15). The black box character of state-of-the-art CEP systems was perceived by the experts, as 7 out of 9 evaluated the state of the simple monitoring application as non-transparent with a rating between 1 and 2. Two participants ranked it neutral (rank 3), leading to an average rank of 1.78. In contrast, the transparency of situation-aware and transparent monitoring applications was rated higher with an average rank of 4.22, respectively an even higher average rank of 4.67 for the very detailed transparent monitoring application. Concluding, the enhanced capabilities of the applications on basis of the introduced concepts and methods in this thesis are perceived as more transparent than classical monitoring applications on basis of state-of-the-art CEP systems. The individual and summarized results are shown in Figure 9.5.

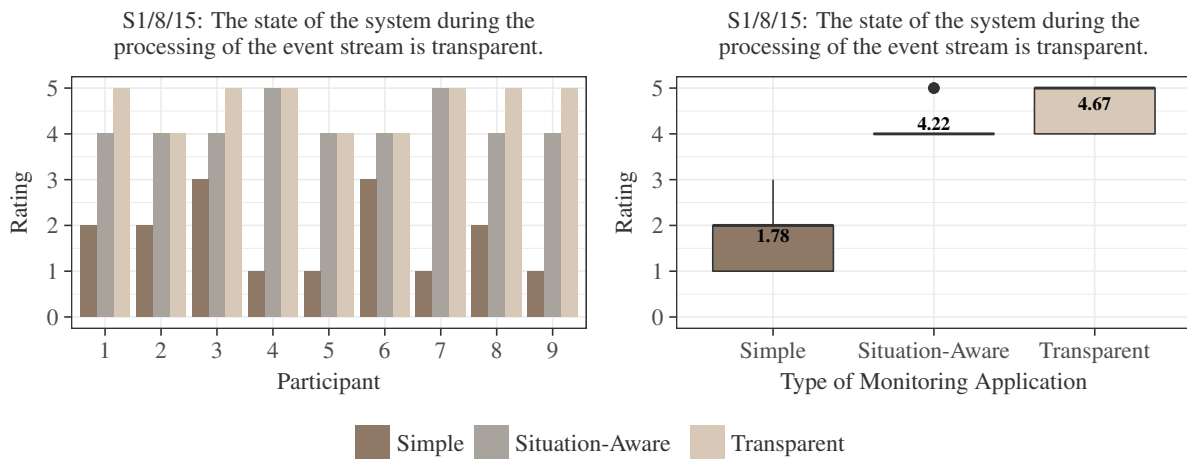


Figure 9.5: Survey Answers related to Transparency

Additional to transparency, the collection of situation-related details is one aspect that is taken into consideration in answering the first research question. Therefore, participants were asked to assess whether interesting situation-related details were encompassed in the monitoring applications (S2, S9 and S16). As expected after analyzing the first statement, the simple monitoring application was ranked low with an average rank of 1.44 and individual ratings between 1 and 3. Again, situation-aware and transparent monitoring were ranked high with an average of 4.56, respectively 4.67 for transparent monitoring. Both are highly interrelated with each other, as the gained insights during the transparent processing of event streams is used to establish situation awareness. Thus it is reasonable that both applications were rated similarly high. Finally, the experts were asked to evaluate the suitability

of the collected data for the retrospective analysis of situations of interest. This capability represents one of the three fundamental functionalities of a situation management system as introduced in this thesis. 7 out of 9 experts rated the analysis capabilities with rank 5, while one participant assigned a rank of 4. One remarkable rating was given by participant 8 with rank 3. An assumption that explains this rating is that due to the representation of the collected information, the retrospective analysis capabilities are not intuitively perceived as given. Further implications are discussed after the presentation of the results. In total, the analysis capability was rated high with an average rank of 4.67. The individual and summarized results are shown in Figure 9.6.

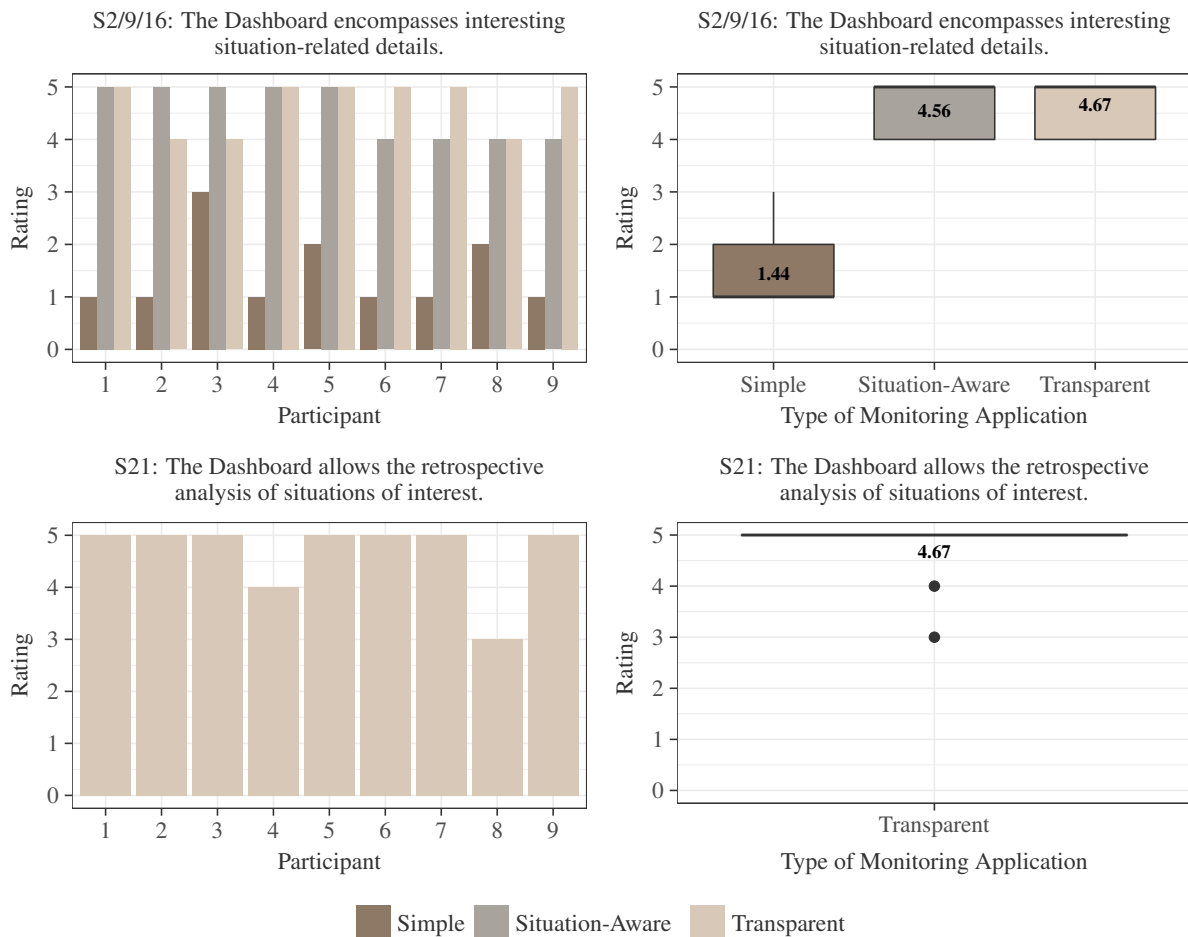


Figure 9.6: Survey Answers related to Level of Details and Retrospective Analysis

The next set of statements is formulated to capture the suitability of the various underlying concepts behind the monitoring applications to establish situation awareness, which relates to the second research question in this thesis. The first of four statements in this direction, asks whether the operator in the introduced running example is consequently kept up-to-date about the current state of the situation of interest (**S4**, **S11** and **S17**). Against expectation, the simple monitoring application was rated unequally with ranks between 1 and 4, resulting in an average rank of 2. This result is ascribed to the fact that the statement might have been misinterpreted in the way, that as soon as the situation occurs, an operator is kept up-to-date and the continuous aspect was disregarded. Nevertheless, the remaining monitoring applications were perceived better and rated high with average ranks of 4.44 for situation-aware and 4.22 for transparent monitoring. As all experts agreed to the continuous aspect for both applications, ratings were more balanced with ranks between 4 and 5, respectively 3 and 5 for transparent monitoring. The below-average ratings are ascribed to the fact, that transparent monitoring applications follow a pull-based approach, whereas situation-aware applications actively inform the operator about the current state.

The next statement with focus on situation awareness, asks the experts to evaluate the capability of the applications to comprehensibly track the development of an emerging situation of interest (**S5**, **S12** and **S18**). Due to the fact, that simple monitoring applications are lacking of transparency, they are not capable to track the evolution of a situation. This speculation is also represented in the results with a low average rating of 1.33. Comparing the results for situation-aware and transparent monitoring one noticeable observation was made: whereas both are rated high with average ranks of 4.22 and 4.56, not all experts equally agreed in their individual rankings. While all experts rated transparent monitoring with ranks between 4 and 5, two participants rated situation-aware monitoring with rank 3. This observation leads to the conclusion, that the representation of situations in situation-aware monitoring applications have to be revised and expanded with additional information. The individual and summarized results are shown in Figure 9.7.

The last specific statement related to situation awareness targets the timely direction of an operator's attention towards an upcoming situation (**S3** and **S10**). As previously mentioned, transparent monitoring does not incorporate a push based notification mechanism, like simple and situation-aware monitoring applications. Therefore, participants assessed the associated statement only for the first two scenarios. Due to the reactive nature of state-of-the-art CEP systems, the timely direction of attention with simple monitoring was rated low with an average of 1.22, with two participants rating a rank of two. This ranking is retraced to the capability of CEP systems to timely inform about an *occurred* situation, which does not take into consideration further evolution states. As situation-aware monitoring applications are designed to actively direct the attention of an operator at an early stage, their evaluation turned out high with an average rank of 4.56 and individual ratings between 4 and 5. The individual and summarized results are shown in the last two plots in Figure 9.7.

The next set of statements, represents the crossing from the second to the third research question. Whereas one statement assess the support for a reactive mode of operation, the second aims at the proactive mode of operation (**S6**, **S13**, **S19** for reactivity and **S7**, **S14**, **S20** for proactivity). As stated earlier, state-of-the-art complex event processing follows a reactive approach by indicating the occurrence of a situation to trigger a reaction. This characteristic is represented in the evaluation results by a high rating with an average of 4.67. Similarly, the two remaining monitoring applications are rated high with regard to reactivity. Situation-aware monitoring achieves an average rank of 4.56 with one participant rating it with rank 3, which is not comprehensible, as the reactive nature is kept during the extension of the situation life-cycle by integrating the detected state. Transparent monitoring achieves a slightly lower average rank of 4.22 with two below-average ratings that are ascribed to its pull-oriented information strategy. The reactive nature of simple monitoring applications leads to a low rating with regard to supporting proactivity, resulting in an average rank of 1.22. The remaining two monitoring applications are rated high with average ranks of 4.67 for situation-aware monitoring and 4.78 for transparent monitoring, which is accredited to the sheer amount of available information. Thus, the integration of predictions into the processing of events is perceived as useful to establish a predictive situation management and allow proactivity. The individual and summarized results are shown in Figure 9.8.

The final two statements **S22** and **S23** are related to the concepts introduced to answer research question 3 to support domain experts with the generation of prediction models and further analysis. Therefore, the participants were first asked to assess whether the automatic, structured extraction of situation-related information fulfills its target to support further analysis on basis of the collected data. The results indicate, that the structured extraction of data is perceived as helpful with a high average rating of 4.44 and individual ratings between 3 and 5. With regard to its suitability to support the generation of prediction models, only 7 out of 9 participants assessed the last statement, leading to an average rank of 4.29, with individual ranks between 3 and 5. Despite one participant, who rated rank 3, all other experts perceived the extracted data as useful for the generation of prediction models. The individual and summarized results are shown in Figure 9.9.

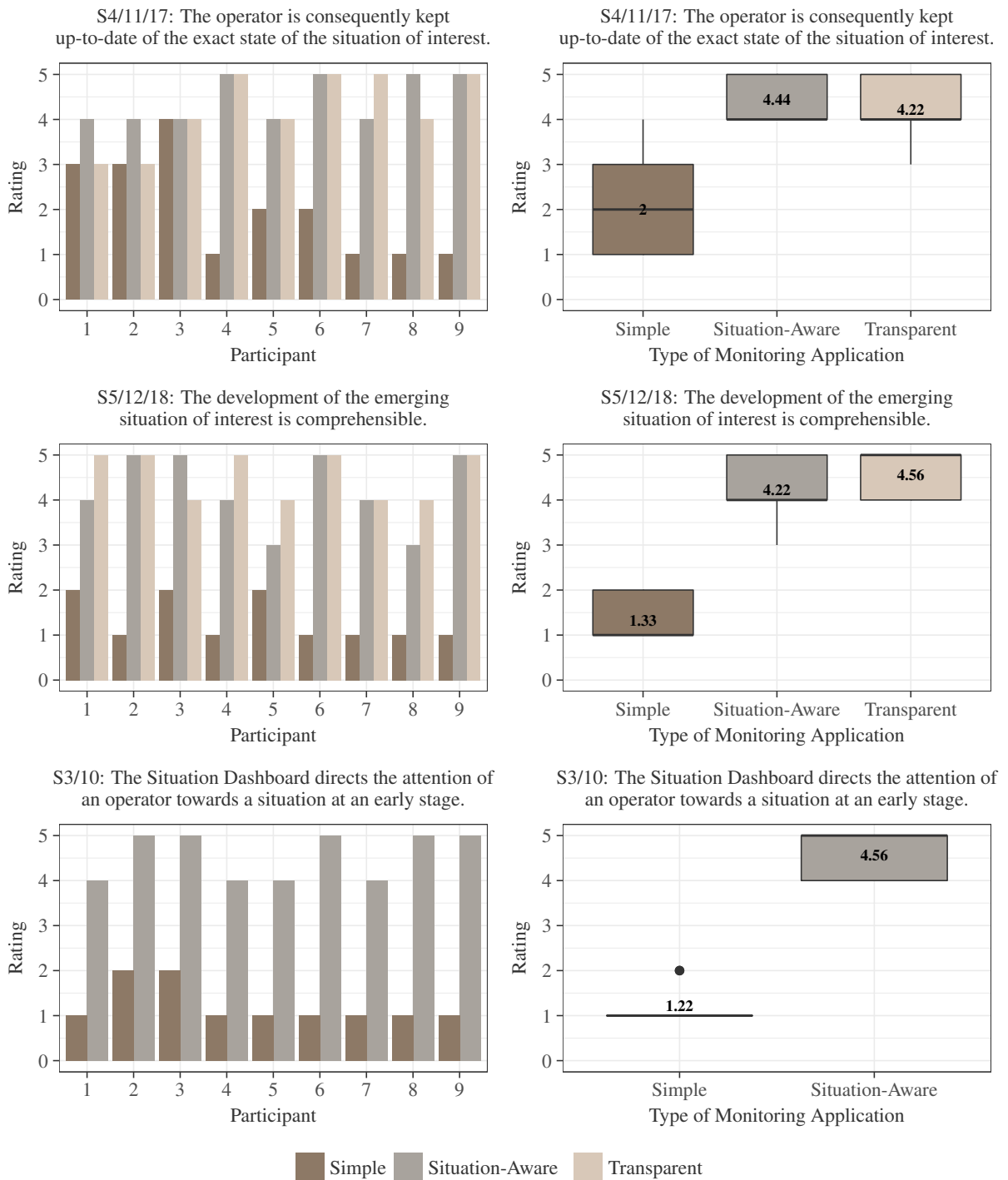


Figure 9.7: Survey Answers related to Situation Awareness

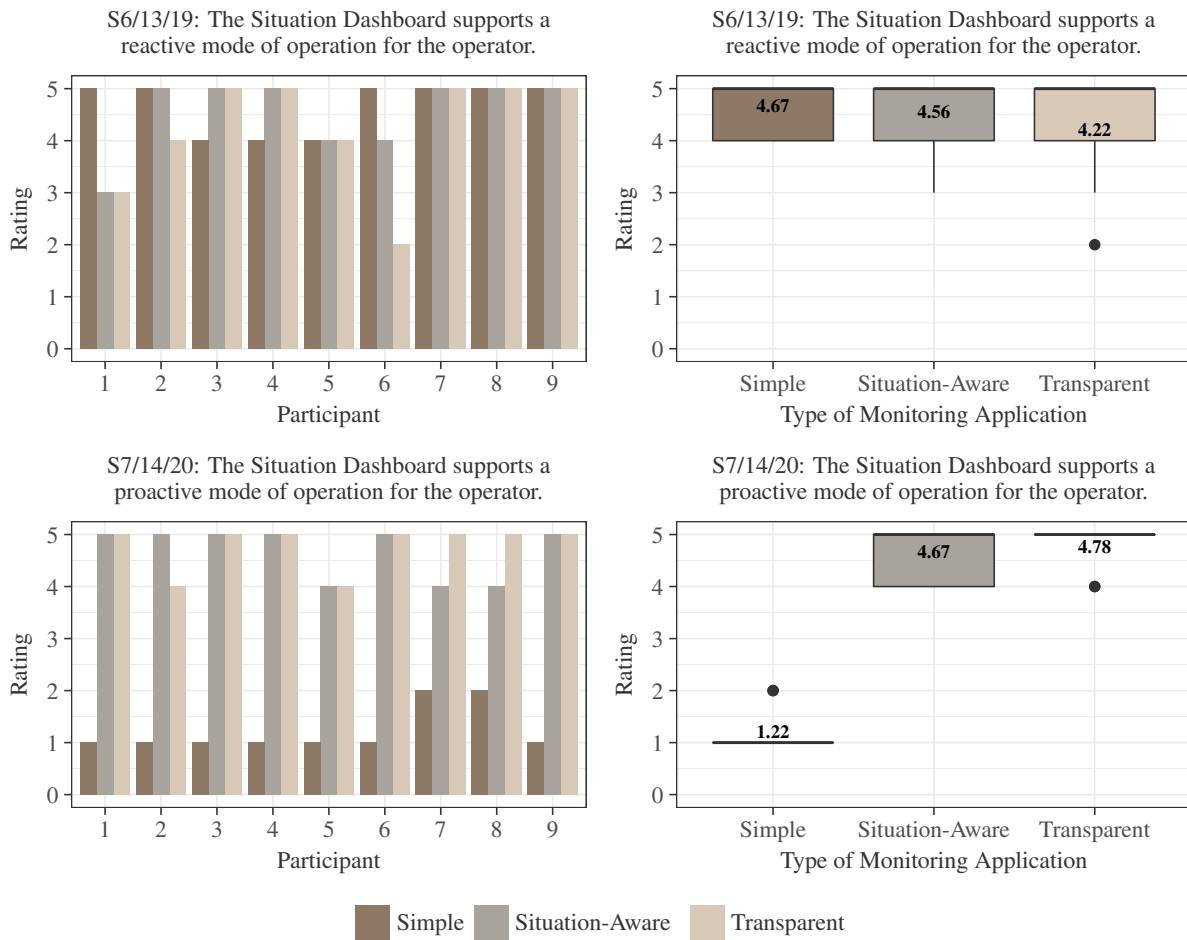


Figure 9.8: Survey Answers related to Reactivity/Proactivity

9.3.3 Discussion

Within this section, the results from the conducted user survey are briefly discussed. The goal of the survey is to evaluate the introduced concepts and methods of this thesis with regard to their contribution in answering the presented research questions. Although a survey with nine experts provides limited insights, the questioning of experienced participants shows that the introduced concepts and methods that build the basis for the assessed monitoring applications is perceived as useful. Additionally, the survey shows, that the goal of introducing situation management on basis of complex event processing was fulfilled. All three capabilities of a situation management system are perceived as available (retrospective, controlling and predictive management). An overall comparison of the expert ratings shows that situation-aware and transparent monitoring is rated higher than simple monitoring applications with average ranks of 1.95 vs. 4.50. As already mentioned in the results section, some aspects were rated below-average, which is ascribed to the representation of relevant information and collected data. As the graphical user interface is implemented as a prototype, improvements are necessary to further increase the perceived value of situation-aware and transparent monitoring applications.

The different capabilities of situation-aware and transparent complex event processing were assessed similarly, but with differences in some aspects, e.g. their suitability for proactivity. By combining the advantages of both and offering a monitoring application that allows the tracking of a situation’s evolution through its life-cycle and the detailed drill-down into the collected data, a holistic situation management system is established that further supports operators and their task to monitor a complex technical environment, like a production facility.

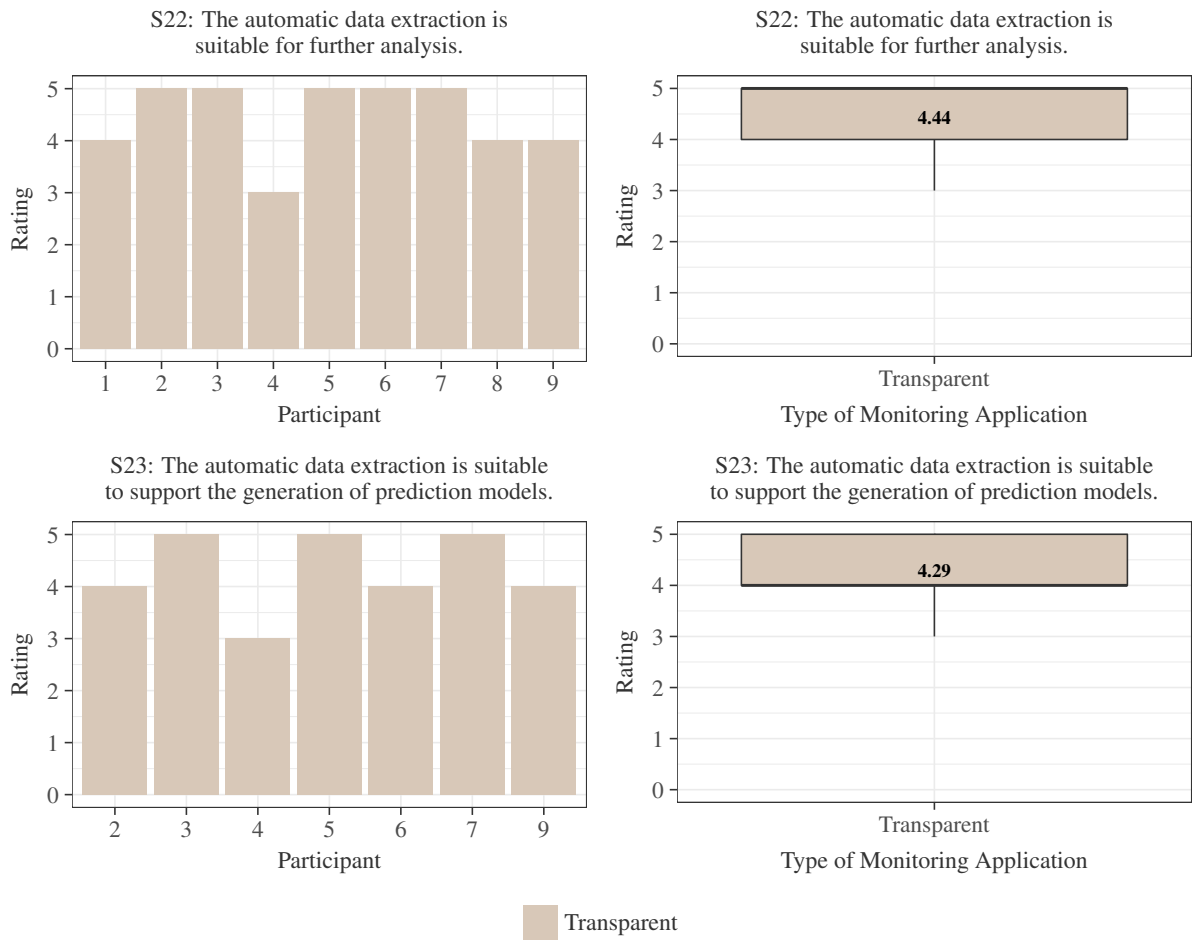


Figure 9.9: Survey Answers related to Predictive Support

9.4 Performance Experiments

This section evaluates different performance aspects of the presented approach for situation management on basis of CEP. The enhanced capabilities of transparent, situation-aware and predictive complex event processing rely on a properly working basic architecture. Therefore, one examined aspect is the impact of the additional processing layer on the underlying basic CEP architecture. Additionally, the limitations of the system in terms of throughput are tested by setting up different environments to examine the system behavior for varying event streams.

9.4.1 Setup

All experiments are performed on a single physical system equipped with an Intel® i7-7500U CPU and 24 GB (Gigabytes) of memory (see Table 9.2). With regard to the basic CEP architecture, the state-of-the-art CEP engine *Apama Streaming Analytics* is used to process the event stream and detect relevant patterns. The communication of the various components is realized by means of the *Universal Messaging*⁶ event bus. Both components are deployed with their default settings. For the experiments, MongoDB is used as a document-oriented, NoSQL database to store all relevant information required to realize the enhanced capabilities. All other components are used as described in Section 9.1 and deployed on the same physical system.

⁶ https://www.softwareag.com/corporate/products/az/universal_messaging/default

Feature	Configuration
Memory	24 GB
CPU	Intel® i7-7500U (4x2.70 Ghz)
Operating System	Windows 10 Pro
CEP Engine	Apama Streaming Analytics 10.1
Event Bus	Universal Messaging 10.1
Database	MongoDB 3.4.10 (disk based)

Table 9.2: Setup for Experiments

The running example, introduced in Section 2.2, serves as a use case for the experiments. Therefore, a working day, following the characteristics of the introduced production process, is simulated to generate an event stream consisting of 23.200 events. The resulting event stream is recorded as a logfile to allow the flexible replay of the collected data. The situation of interest *DullToolSituation* occurs 109 times within this data set. The state handler is configured as previously described in Section 7.5 (see Table 9.3). The materialization is measured with the help of simple partial pattern fulfillment (S-PPF) and predictions are done solely on basis of the collected statistics.

Parameter	Setting
materialization metric	S-PPF
initialization threshold	0.45
looming threshold	0.75
predicted threshold	0.90

Table 9.3: Parameter Settings for State Handler in Running Example

Based on this setup, multiple experiments were conducted. The first set of experiments aimed at the analysis of the generated overhead of the additional processing and its impact on the basic system and the critical path in processing. By comparing the resource utilization of the basic architecture with and without the enhanced capabilities for situation management, it is examined to which extent the additional processing effort influences the underlying CEP architecture. The second set of experiments was conducted to evaluate the maximum throughput of the situation management system, under which still correct results are derived. Considering all experiments, the generated event stream was used in different variations, where the full data set was taken to analyze the generated overhead and excerpts and variations of it were used for more specific experiments.

The goal of the conducted performance experiments was to evaluate from which order of magnitude on the additional overhead might influence the performance of the underlying state-of-the-art components and affect the critical processing path. The goal was not to test the general performance of the underlying components, i.e. event bus and CEP engine. Additionally, the experiments assessed for which type of scenarios the approach in this thesis is suitable and whether further performance aspects have to be considered in the future.

9.4.2 Results

Following the structure of the conducted experiments, the results are presented in the following. First, results regarding the overhead experiments are shown, before the impact of the overhead on the system is examined. Lastly, the results of the throughput experiments are reported.

Overhead

During the processing of the previously described event stream for the running example, additional events were generated by the various components involved in the realization of the situation management capabilities. The goal was to capture the total overhead generated along the processing of the full data set, comprising 23.200 events, and to map the additionally generated events to their sources. Therefore, the event stream was processed and all generated events, that were sent over the event bus, were logged. The overhead encompasses all events that are not generated during the processing with the basic CEP architecture, but are required for the introduction of transparent, situation-aware and predictive complex event processing as introduced in Chapter 6, 7 and 8.

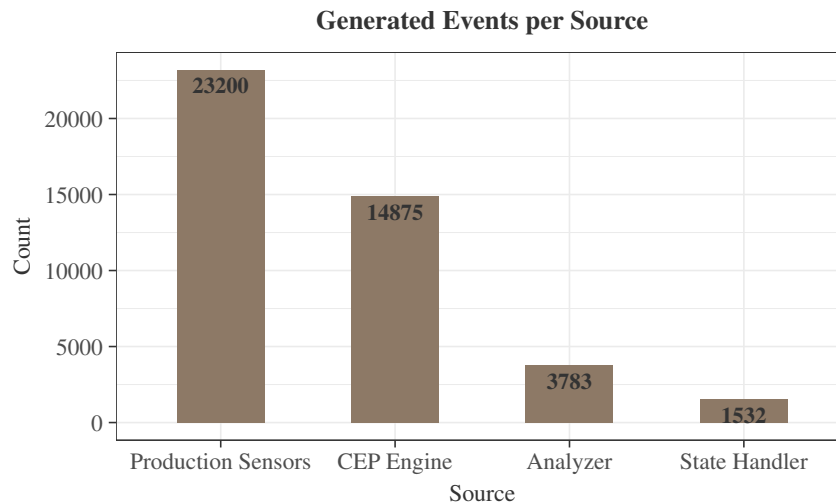


Figure 9.10: Overview of generated Events in the Running Example

Figure 9.10 shows the resulting distribution of events per source. The production sensors generated 23.200 events during the production day. Those raw events were processed by the CEP engine to detect the defined patterns (*MaxForceSituation*, *DullToolSituation*). In order to evaluate all defined patterns, the engine calculated various aggregations of the raw event data, e.g. the average force value. All this processing of the CEP engine resulted in additional 14.875 events. In total 38.075 events were generated to capture and process the production-related data with a state-of-the-art CEP architecture. As shown in Figure 9.10, a total of 5.315 events was additionally generated through the additional processing layer. The majority of this overhead was generated by the transparent processing of events, as all intermediate steps of a pattern detection are observed and indicated by new events. The additional events represent the complex events indicating the partial pattern fulfillment of deployed queries. On basis of partial pattern matches, the state handler keeps track of the situation state during its evolution through the life-cycle. This results in additional 1.532 events used to inform about the current state of a situation.

Besides the overhead generated on the event bus, also the CEP engine had to handle extra workload, as the observation of intermediate states requires the engine to evaluate further queries. As described in Chapter 6, the introduction of transparency requires the system to observe all sub-patterns of a defined pattern in a query. Therefore, the system automatically derives all relevant sub-queries and deploys them in the CEP engine. Within the running example, three additionally derived queries had to be evaluated during the processing of the event stream.

The resulting overhead leads to an additional strain on the basic architecture, i.e. on the event bus and the CEP engine. With regard to requirement R23 (Critical Processing Path), the impact of the additional processing should be kept to a minimum to avoid negative effects on the central functionality of CEP architectures, the detection of occurring situations. Therefore, the next section examines the impact of the overhead on the basic system.

Impact on Basic System

To measure the impact on the underlying CEP architecture, the resource utilization of the central components, i.e. the CEP engine and event bus, were recorded during the processing of an excerpt of the data set. The larger part of the overhead is generated during the evolution of a situation of interest, as the intermediate states of its pattern are observed. Therefore, an excerpt of the data was chosen, that encompassed multiple situation occurrences and partial pattern matches. This 50 minute extract from the full data set was injected into the event bus and processed in two settings: i) the basic CEP architecture and ii) with enhanced capabilities for situation management. During the processing in both environments, the CPU usage and memory consumption were recorded with the help of Java VisualVM⁷. To smooth potential outliers, the experiments were executed five times per environment and the results averaged afterwards.

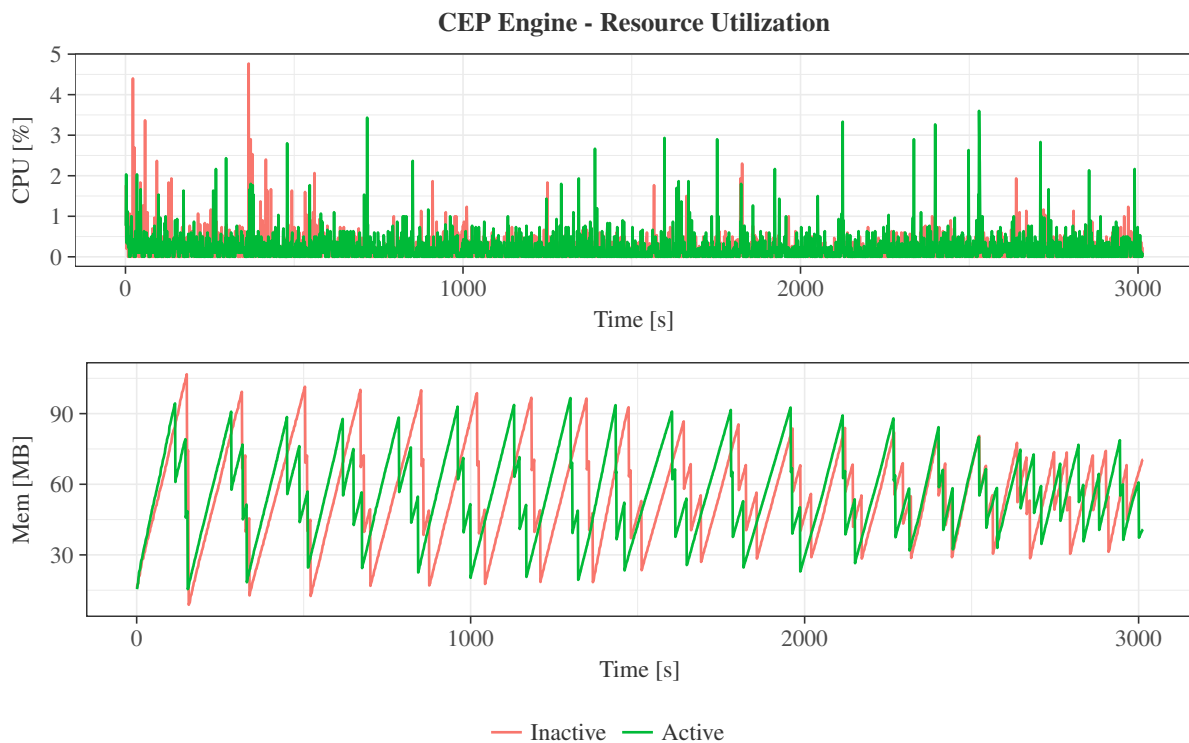


Figure 9.11: Comparison of Resource Utilization of CEP Engine *Apama*

Figure 9.11 presents two charts illustrating the resource utilization of the CEP engine used in the experiments. The upper chart shows the CPU usage, whereas the lower chart depicts the memory consumption during the processing. Both charts show the behavior for both settings, the basic architecture without activated transparency and situation awareness and the enhanced architecture with the new capabilities. Considering the CPU usage, both settings behave very similar with a median of 0.1 % in both cases and average values of 0.19 % for the basic architecture and 0.21 % for the enhanced architecture. Peaks in both settings are the result of internal tasks of the CEP engine, e.g. writing a logfile for debugging purposes. With regard to the memory consumption, a similar behavior is observed. In both settings, the memory usage follows a similar pattern with an average of 55.85 MB, respectively 55.97 MB and a median of 55 MB for both settings. Additionally, the cycles of the automatic memory management (garbage collection) follow the same pattern and vary only slightly, with a time difference between two runs of 50.17 seconds in average and a median of 25 seconds, respectively an average of 51.74 seconds and a median of 29.5 seconds for the extended architecture.

⁷ <https://docs.oracle.com/javase/8/docs/technotes/guides/visualvm/>

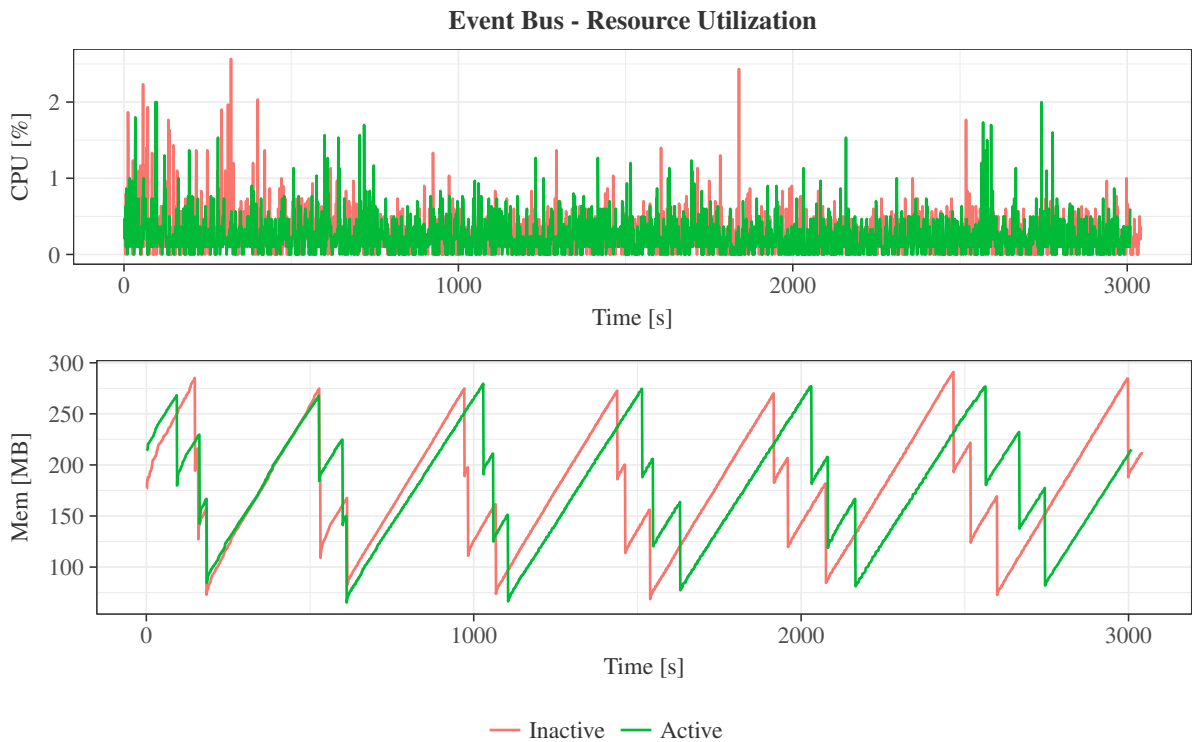


Figure 9.12: Comparison of Resource Utilization of Event Bus *Universal Messaging*

After analyzing the impact of the additional queries on the CEP engine, Figure 9.12 shows the resource utilization of the event bus during the processing of the event stream excerpt. The upper chart shows the CPU usage for both settings. Analogously to the behavior of the CEP engine, the effect of the generated overhead on the event bus is negligible. The CPU usage shows a similar behavior for both settings with an average of 0.25 % and a median of 0.2 % for the basic architecture and an average of 0.23 % and median of 0.2 % for the enhanced capabilities. With regard to the memory consumption, the lower chart shows a slight difference with an average value of 177.19 MB, respectively 178.09 MB, whereas the median shows a slightly higher difference with 173.27 MB compared to 178.20 MB for the enhanced architecture. Another difference is found in the time between two garbage collection runs, where the basic architecture has an average of 152.5 seconds and a median of 82 seconds compared to 157.8 seconds in average and a median of 80.5 seconds.

As the experiments show, the generated overhead in the running example does not have a significant effect on the resource utilization. The impact on the performance of the basic architecture highly depends on the underlying use case. The behavior of the system might change significantly in a high-throughput scenario with more events being generated and/or more situations being detected and tracked. To better understand the limits of the basic architecture and system setup, further experiments were conducted to analyze the resource utilization in high-throughput scenarios. This information serves as an indicator, at which scale the generated overhead might lead to a decreased performance, i.e. higher resource utilization and shorter cycles for the garbage collection. Therefore, the resource consumption is tracked during the processing of three different event streams, with differing throughput (1.000, 10.000 and 50.000 events per second). The event streams were synthetically generated on basis of the event objects of the running example. Figure 9.13 shows the results of the experiments.

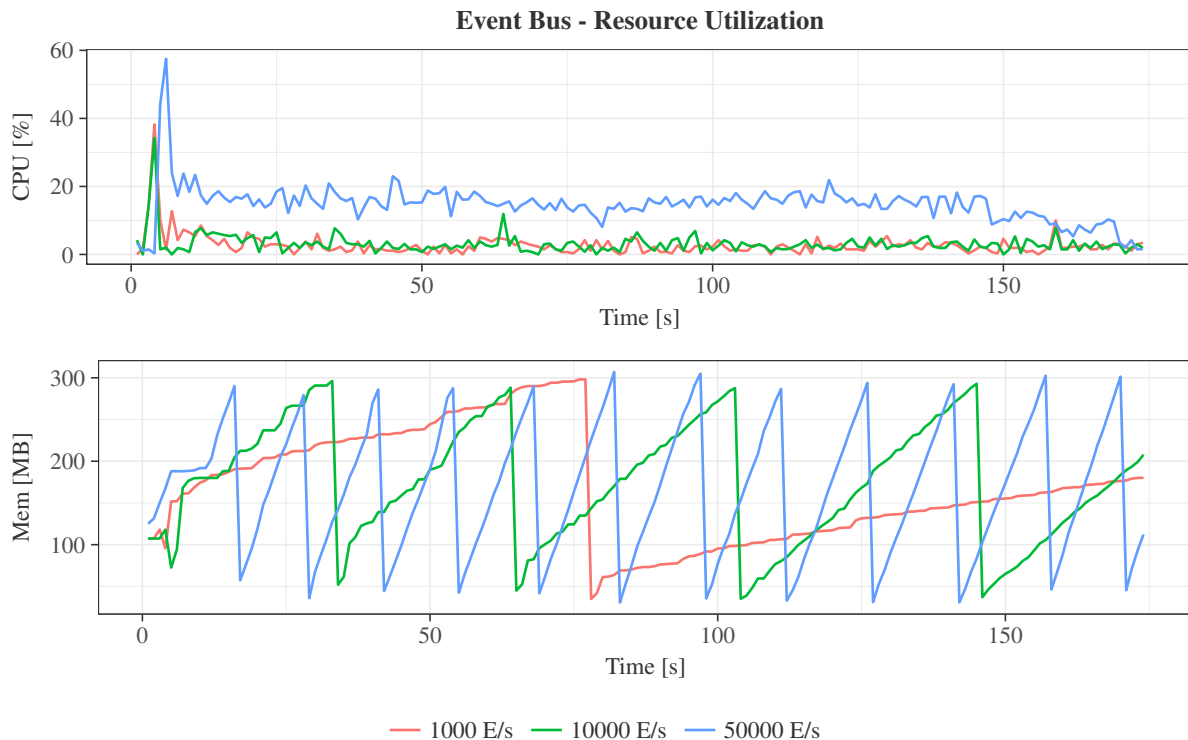


Figure 9.13: Resource Utilization of Event Bus *Universal Messaging*

The upper chart shows the CPU usage for all three event streams that are generated synthetically by using the generated event stream from the running example. The CPU usage differs between all three event streams, where 1.000 events per second led to an average CPU usage of 2.7 % (median 2.3 %), respectively an average of 3.11 % (median 2.6 %) for 10.000 events per second. Significantly more CPU time was used for the processing of 50.000 events per second, resulting in an average CPU usage of 14.37 % with a median of 15.15 %. The impact of the higher throughput is also represented in the memory consumption. Whereas the average and the median for all three event streams is similar (169.98/162.38, 172.71/175.44 and 170.84/173.48 MB), the time difference between two garbage collection runs varies significantly. With 74 seconds, the time between two runs is the highest for 1.000 event per second event, whereas the average time for 10.000 events per second lies at 35.25 seconds. The shortest cycle time are 14 seconds in average for 50.000 events per second. As anticipated, the workload of the system increases with the increasing throughput.

Lastly, the effect of additional queries on the CEP engine was examined by deploying multiple queries (3, 7 and 15 queries) that are based on the queries used in the running example and have the same complexity, i.e. number of event types, time windows and aggregations. In addition, the throughput of the system was also increased, as a low-throughput event stream was not sufficient to analyze the impact of additional queries on the system. Therefore, the experiment was conducted with 500 events per second originating from the generated event stream. Figure 9.14 shows the results of the experiments.

The CPU usage for all three settings is shown in the upper chart. As depicted, the CPU usage differs between all three scenarios. With three queries deployed, the engine had an average CPU usage of 7.3 % (median 6.1 %), whereas those values increased to 10.35 % (median 9.2 %) for seven queries and to 13.30 % (median 11.9 %) for 15 queries. The effects of additional queries were also represented in the memory consumption. Whereas the average memory usage was almost the same for all settings (45 MB), their median differs slightly with 48.58 MB for three queries being the highest. This effect results from the longer cycle times of the garbage collection, as the memory

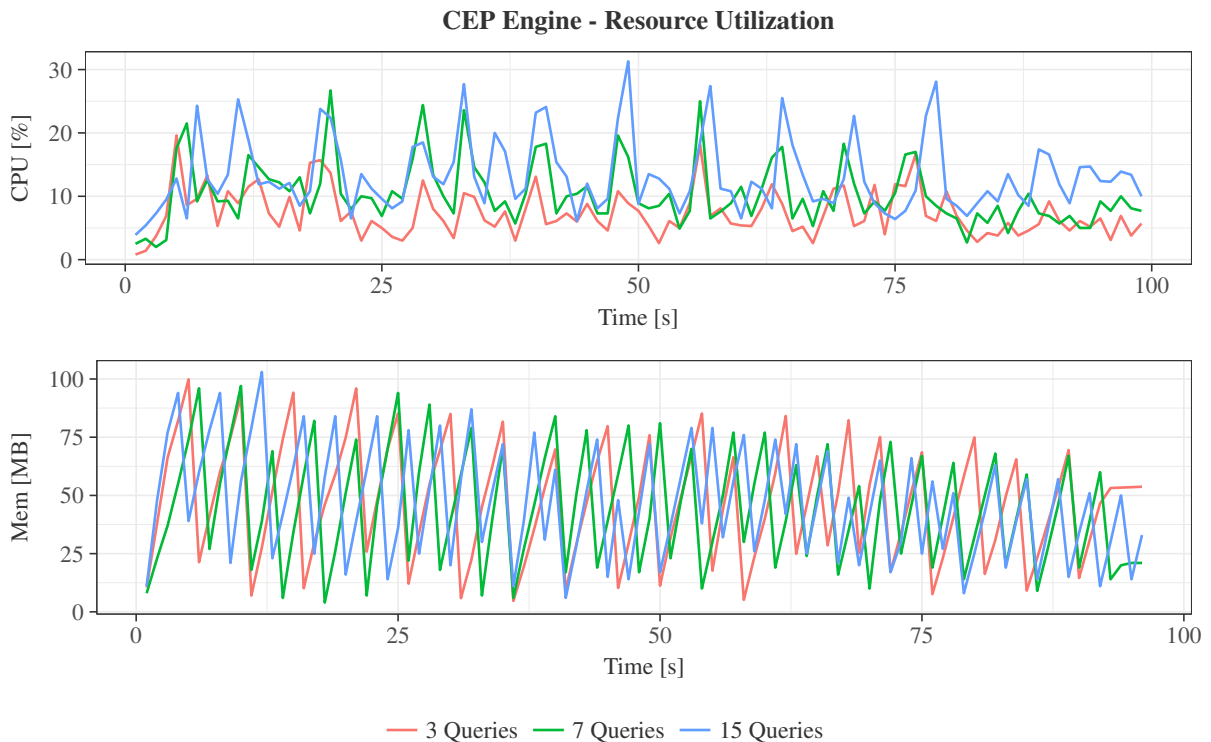


Figure 9.14: Resource Utilization of CEP Engine with multiple Queries

is used longer before getting freed. This aspect was also reflected in the cycle times, with an average cycle time of 4.5 seconds for three queries, 3.45 seconds for seven queries and 2.97 seconds for 15 queries. Similar to the behavior of the event bus, the workload of the CEP engine increased with additional queries.

The goal of the experiments was not to test the performance of the involved basic components (event bus and CEP engine), but to show from which order of magnitude on the overhead might lead to a significant impact on the performance of current state-of-the-art architectures. Further performance aspects are briefly discussed in the following subsection which are focused on the throughput of the situation management system.

Throughput

The last set of experiments was conducted to evaluate the limitations of the implementation with regard to the maximum throughput that is still processed correctly in the previously described setup. The throughput test was designed as follows: An event excerpt was constructed comprising 20 situation occurrences of *DullToolSituation*, which had to be detected by the enhanced architecture in multiple settings. First, the event excerpt was replayed with one event per second as its throughput. The resulting outcome of the processing was logged and used as a baseline for comparison. Next, the replay speed was increased to simulate a throughput of 20 events per second. This modification also required the adoption of the time window in the query by dividing it through the speed up factor. Again, the outcome was captured for analysis. The speed up factor was further increased and the outcome compared until incorrect results were detected. The correctness of the resulting outcomes of processing were compared and are shown for various settings in Table 9.4.

The upper table shows the amount of logged events for all involved event bus channels and for the differing experiment settings. As shown in the table, for all three runs the amount of events received on the channel *ProductionChannel* was equal. This channel comprised all raw events coming from the production sensors, as well

Channel	1 Event/Second	20 Events/Second	50 Events/Second
ProductionChannel	664	664	664
AnalysisChannel	202	202	202
StateHandlerChannel	115	115	93

State	1 Event/Second	20 Events/Second	50 Events/Second
Inactive	15	15	22
Initialized	45	45	32
Looming	31	31	18
Predicted	4	4	1
Detected	20	20	20

Table 9.4: Comparison of Results for varying Throughputs

as all aggregated events, e.g. a rising force trend. The second channel, *AnalysisChannel*, encompassed all partial pattern matches. As the transparent processing of event streams is based on the underlying CEP architecture by automatically adding derived sub-queries, all three experiments led to the same results. This observation changes for the last observed channel, *StateHandlerChannel*. For one and 20 events per second, the same state transitions were logged, whereas for 50 events per second fewer transitions were recorded on the channel. This observation required further analysis by drilling down and considering the single event types and event objects received on the channel to identify the bottleneck.

As shown in the second table, all three settings correctly detected 20 situation occurrences and the first two settings followed the identical situation evolution path through the life-cycle. The last experiment with 50 events per second led to fewer state transitions. Considering the architectural design and its implementation, this behavior is ascribed to the database, as the state handler heavily relies on the data stored in the database and a timely read/write access. By increasing the throughput, the database became a bottleneck and important elements, e.g. event objects or partial pattern matches, were not yet available for the state handler, although the partial pattern was already detected. As the detection of a full pattern match does not rely on the collected information in the database, the final transition to the state detected was always correct, as long as the underlying CEP engine is able to handle the event stream. During the experiments, correct results were examined for up to 49 events per second for the described system hardware/software setup.

9.4.3 Discussion

The conducted performance experiments show that the generated overhead of the extended architecture might have an impact on the performance of the critical path for high-throughput use cases. Additionally, the correctness of the processing results for the extended architecture can only be guaranteed up to a limited amount of events per second, depending on the underlying hardware/software and complexity of the queries. As stated earlier, the main goal of this thesis was further enhancement of complex event processing towards a holistic situation management system. Nevertheless, in the following, possible design changes are discussed that might beneficially affect the performance of the system.

First, due to its current architectural design, the additional processing effort can easily be outsourced to a second instance of a basic CEP architecture. Thus, the generated overhead in terms of additional events and queries, has not to be processed by the existing architecture, but can be deployed in the new parallel architecture. By adjusting

the processing logic in this way, the only overlap remains the reading access to receive relevant events from the involved event producers. This modification eliminates any negative effect on the basic architecture, but comes at higher costs, as an additional instance of a CEP engine and event bus has to be set up. The second aspect to be considered in course of this discussion, is a modification to improve the throughput of the additional processing layer. As previously described, the software artifacts are implemented on basis of the Apache Flink stream processing framework, which allows the parallel and distributed execution of the analysis logic. By adjusting the processing workflow of the components, a parallel (stream partitioning) and distributed execution of the event stream can be implemented. Another boundary of the system is represented by the complexity of the pattern of a situation of interest. The design of the underlying pattern has a significant impact on the run-time aspects of the basic CEP architecture and accordingly also affects the situation management system, e.g. a very long time-window leads to performance challenges, as all incoming event objects have to be temporarily stored by the CEP engine. This requires the CEP architecture to be sufficiently dimensioned for the use case.

10

Conclusion

In the preceding chapters, concepts, methods and models were introduced to establish a holistic situation management on basis of complex event processing. The presented approach extends the capabilities of state-of-the-art CEP systems in three steps: First, by establishing transparency for the processing of CEP, second, by adopting concepts from the psychological model of situation awareness and third, by supporting the generation and embedding of prediction models. Additionally, a reference implementation of the concepts is developed to demonstrate and evaluate the application of the situation management capabilities. In Section 10.1, the main research contributions are recapitulated along the identified research questions. Section 10.2 discusses the significance of the results and its suitability for emerging application areas in the IoT field. Finally, Section 10.3 elaborates on future work to further improve the support of human operators with their task of monitoring complex technical environments.

10.1 Summary

Based on the increasing digitization in various fields with humans staying in the control loop, e.g. manufacturing, more and more generated data has to be comprehensibly processed in near real-time. Under these circumstances, human operators have to be continuously kept up-to-date with regard to potential situations of interest and given the reliable foundation for profound decision-making. Due to its maturity and near real-time processing paradigm, complex event processing is considered as the technology to tackle the arising challenges of situation assessment in (Industrial) Internet of Things use cases. Nevertheless, becoming a holistic situation management system requires the further evolution of the complex event processing paradigm towards situation awareness and transparency to support the enhanced capabilities of situation management. Therefore, the principal research questions of this thesis aims at the expansion of complex event processing:

How can complex event processing be further evolved to a holistic situation management system?

The establishment of a holistic situation management system by means of complex event processing requires the introduction of additional capabilities to support the investigative, controlling and predictive aspect of situation management. The vision of a holistic situation management system is split up into three research questions that are the fundament of this thesis. In the following, the answers to these questions are summarized.

Research Question 1 (Transparency). *How can transparency be introduced in complex event processing during run-time?*

The first research question, relating to the missing transparency in complex event processing systems, is answered in Chapter 6. Establishing situation management and its related capabilities requires deep knowledge about the situations of interest. Enabling complex event processing systems to capture situation-related details requires the introduction of a mechanism to observe the intermediate steps of the situation detection.

In the first step, a language-independent model for situations of interest and their related queries is introduced, transforming engine-specific representations of queries into an abstract model with the goal to establish a generalizable approach for arbitrary CEP engines. The intermediate steps of situation detection are made observable by the concept of partial pattern fulfillment. For this purpose, a method is designed on basis of the abstract model to decompose a transformed pattern into valid sub-structures, leading to sub-queries representing the partial pattern matches. By redeploying the derived sub-queries into the CEP engine, its intermediate processing steps become observable and usable for the collection and calculation of situation-related data. Consequently, the investigative situation management is enabled as all situation-related data is captured during the processing.

The main advantage of this approach is, that on basis of existing situation descriptions, relevant intermediate processing steps are automatically derived by a mechanism that additionally supports arbitrary CEP engines. The resulting capability to capture situation-related data in multiple levels of detail increases transparency of the black-box mode of operation of state-of-the-art CEP systems. The additional insights into the processing serve as a fundament for further enhancements of CEP required for the enhanced situation management capabilities.

The implemented reference system provides support for two state-of-the-art CEP engines to introduce transparency by evaluating the partial pattern fulfillment. During the processing of event streams situation-related details are collected and stored to allow the retrospective analysis of situation occurrences referring to the investigative aspect of a holistic situation management. The approach is evaluated by conceptually investigating whether all requirements towards the research question are fulfilled. Additionally, a user study is conducted to give experts the possibility to assess the newly introduced transparency and usefulness of the collected information during run-time.

Research Question 2 (Situation Awareness). *How can situation-aware complex event processing be established?*

The second research question, relating to the inefficient management of an operator's attention, is answered in Chapter 7. State-of-the-art CEP systems have a limited view on situations by differentiating only two situation states, namely *detected* and *not detected*. Neglecting the gradations between those two states hampers the process of becoming and especially staying situation-aware for a human operator. By extending the situation life-cycle with additional states (*inactive*, *initialized*, *looming*, *predicted* and *detected*) and making it explicit, a mechanism is introduced supporting all three levels of the psychological model of situation awareness and allowing the tracking of a situation's evolution through its life-cycle. By leveraging the insights from the previously achieved transparency, methods are presented to derive a materialization metric on basis of partial pattern fulfillment to evaluate a situation's completion degree. Elaborating further on the introduced metrics, a mechanism is presented mapping fulfillment degrees to situation states allowing the assignment of criticality levels in dependence of the underlying use case. Following this approach, the whole life-cycle and its transitions between the introduced states becomes adjustable by incorporating use case specifics and contextual knowledge of the domain expert. Through the introduction of a state *predicted*, the integration of prediction models, to establish level 3 situation awareness, is already designated. Introducing the full scope of situation awareness forms an important prerequisite to establish a holistic situation management. Besides the prediction of situations, also their management is supported covering the controlling aspect of situation management.

By introducing a general life-cycle, the states of a situation of interest are abstracted from the underlying use case and mapped towards criticality levels suitable for arbitrary use cases relying on the detection of situations occurrences. Through the design of the transition mechanism to allow the incorporation of use case characteristics and contextual knowledge of the domain expert, the life-cycle and its tracking mechanism support the establishment of situation awareness for various use cases. Consequently, this results in the advantage that the complex event processing paradigm is extended to support the acquisition and maintenance of situation awareness in general and to improve the direction of an operator's attention towards an evolving situation.

Based on the implemented transparency in the reference system, the concepts and methods for situation awareness in complex event processing are implemented. The involved components leverage the insights gained during the processing of event streams to assess the current state of a situation in the life-cycle. The determined state is communicated to inform potential consumers of the information. Within the reference implementation a graphical user interface informs the operator about the current state to focus the operator's attention towards an upcoming situation. The approach was evaluated along the requirements elicited in Chapter 5 to conceptually investigate their fulfillment. The conducted user study evaluated the usability of the new capability to comprehensibly inform about the development of a situation, the timely direction of an operator's attention towards a situation and whether the operator is consequently kept up-to-date of the exact state of a situation.

Research Question 3 (Prediction Support). *How can situation-aware complex event processing be used to support projections into the future?*

The third research question, relating to the missing proactivity of state-of-the-art CEP systems, is answered in Chapter 8. Based on the underlying functionality of CEP systems to detect situation occurrences, concepts are introduced supporting the generation and succeeding integration of prediction models to establish level 3 situation awareness. First, data collected during the transparent processing of event streams is enriched by a labeling mechanism that associates captured event traces with situation labels in case of an occurrence. The pre-processed data is continuously extracted in a structured format to support the generation and initial testing of prediction models for projecting a situation's future development. Mechanisms are introduced to embed the manually generated models in the processing workflow of event streams. By executing and triggering predictions, the state *predicted* of the situation life-cycle is further enhanced and allows the integration of complex prediction models. Through the integration of models into the processing workflow, additional automatic evaluation potentials are exploited, as the CEP system continuously generates new samples to evaluate the performance of the prediction model during its run-time. Once the domain expert decides that the achieved performance metrics are sufficient, the prediction model is integrated into the productive workflow and used continuously for predictions. By supporting the generation of prediction models and their embedding in the processing workflow, the evolution of CEP towards a holistic situation management system is completed with the covering of the predictive situation management.

As the introduction of one single prediction model for situations in general poses a challenge that leads to limitations in terms of flexibility and mightiness, the approach in this thesis follows the idea to support the different phases of the CRISP-DM process model for predictive analytics. The advantage is, that the introduced concepts and methods ease the various tasks to be conducted in the phases of the process model, but do not represent any limitation. In addition, the generated extract in a structured format serves as a basis for further analysis tasks and is not restricted to prediction-related problems.

The presented concepts and methods are implemented as part of the reference system in this thesis. Collected and labeled data is continuously extracted in a structured format during the run-time and the system allows the integration of prediction models by two different mechanisms, namely PMMLs and REST-APIs. In the evaluation, the fulfillment of requirements related to research question 3 was conceptually investigated and the supportive functionality and usefulness of the extracted data was assessed by experts as part of the user study. The complete situation management system, encompassing the transparent, situation-aware and predictive processing of event streams, was additionally evaluated to analyze performance aspects, e.g. generated overhead.

10.2 Significance of the Results

The established approach to introduce transparency into the situation detection of complex event processing allows further evolution of the CEP paradigm towards a holistic situation management. The proposed concepts, methods and models take up the arising challenge of comprehensibly processing the increasing amount of data and supporting human operators with their data-driven tasks, like monitoring. In addition, the operator's attention is purposefully directed towards upcoming situations. At the same time, the approach offers a wide range of new insights during the run-time allowing the further enhancement of CEP capabilities, e.g. for optimization tasks.

With regard to visionary concepts like the Internet of Things and its industrial application in Industrial Internet of Things, many obstacles have to be overcome to realize the potential benefits of interconnectedness. Despite the technical barriers, one major challenge is the missing acceptance of the involved parties, like operators and domain experts. Increasing the comprehensibility of systems that tackle the technical challenges, improves the probability of success for strategic initiatives like Industrie 4.0. The presented concepts for a holistic situation management system are suitable to take over this task and increase the transparency and trust of data processing systems. On one side, situation management helps to gain a deeper understanding in use cases from the machine-to-machine application field, where transparency leads to trust in the processing of data and automatic decision-making. On the other side, use cases with interfaces to human operators that rely on human-machine-interaction, also benefit from a holistic situation management system. The improved transparency enables deeper insights allowing human operators to build trust in the system and to increase its acceptance. In addition, the investigative, controlling and predictive nature of situation management comprehensibly supports humans in cognitively demanding tasks. Consequently, the introduced approach reduces the action distance by directing the attention of an operator towards a situation at an early stage allowing timely reactions to minimize the consequences of delays. Finally, the proposed approach pushes the field of complex event processing into a new direction by shifting its focus from technical challenges like throughput to taking human aspects into consideration.

10.3 Outlook

This thesis represents the first step towards a holistic situation management. The derived concepts and methods provide the fundament for supporting human operators in complex technical environments. Potential directions of future research for further improvements are sketched in the following.

Semantic Description Within the field of complex event processing, several vendors provide multiple engines with different advantages, e.g. with special focus on visual modeling or throughput. This work introduced a unifying model to abstract from engine-specific representations of situations of interest, to overcome the challenge of limiting the support of the presented situation management approach to a single CEP engine. Consequently, a grammar is required for each CEP engine to be supported by the situation management. Based on the assumption, that the semantic description of situations of interest is more intuitive than the technically inspired description with a declarative, engine-specific language, semantic modeling represents a beneficial future research direction. Introducing semantic modeling capabilities and decoupling the description of situations of interest completely from the specifics of the used engine, has the potential to additionally support users. In addition, CEP engines, their capabilities and event operators can also be described semantically to allow the automatic transformation of abstract situation descriptions into CEP-specific queries. Consequently, this extension also facilitates the replacement of the underlying CEP engine, as situations are not bound to one specific system.

Automatic Self-Tuning As every use case has its own specifics and needs, the concepts and methods for situation awareness and proactivity in complex event processing allow the dynamic customization of their mode of operation, e.g. by adjusting thresholds to vary the criticality levels behind a situation's state in the life-cycle. The definition of thresholds and decision about a materialization metric to be used, requires empirical experience with the system and the situations of interest. Introducing a self-tuning mechanism into the situation management system has the potential to enhance the benefits of the system, support the domain expert with the set-up of the system and additionally reduce the time to turn it into production. This implies that according to a classification of situations with regard to their criticality, the system analyzes existing historic data and automatically derives thresholds to allow a timely reaction. Therefore, the system is able to analyze the probability of occurrence from different perspectives, e.g. the different partial pattern matches, and determine the remaining reaction time if the attention of the operator is directed towards the evolving situation. This extension requires the definition of additional meta-information for a situation, like which actions are to be taken for the single situations, how long their implementation might take and how critical a situation is considered. Consequently, the situation management also has to allow the flexible definition of thresholds on basis of single situations and not just use cases.

The contributions of this thesis led to a holistic situation management system on basis of the CEP paradigm and build the first step towards a human-centered processing of events. The newly claimed capabilities serve as a fundament for further enhancements of complex event processing systems.



Appendix A

A.1 Survey Results

ID	Statement	#1	#2	#3	#4	#5	#6	#7	#8	#9
1	The state of the system during the processing of the event stream is transparent.	2	2	3	1	1	3	1	2	1
2	The Application encompasses interesting situation-related details.	1	1	3	1	2	1	1	2	1
3	The Application directs the attention of an operator towards a situation at an early stage.	1	2	2	1	1	1	1	1	1
4	The operator is consequently kept up-to-date of the exact state of the situation of interest.	3	3	4	1	2	2	1	1	1
5	The development of the emerging situation of interest is comprehensible.	2	1	2	1	2	1	1	1	1
6	The Application supports a reactive mode of operation for the operator.	5	5	4	4	4	5	5	5	5
7	The Application supports a proactive mode of operation for the operator.	1	1	1	1	1	1	2	2	1
8	The state of the system during the processing of the event stream is transparent.	4	4	4	5	4	4	5	4	4
9	The Application encompasses interesting situation-related details.	5	5	5	5	5	4	4	4	4
10	The Application directs the attention of an operator towards a situation at an early stage.	4	5	5	4	4	5	4	5	5
11	The operator is consequently kept up-to-date of the exact state of the situation of interest.	4	4	4	5	4	5	4	5	5
12	The development of the emerging situation of interest is comprehensible.	4	5	5	4	3	5	4	3	5
13	The Application supports a reactive mode of operation for the operator.	5	5	5	5	4	4	5	5	5
14	The Application supports a proactive mode of operation for the operator.	5	5	5	5	4	5	4	4	5
15	The state of the system during the processing of the event stream is transparent.	5	4	5	5	4	4	5	5	5
16	The Application encompasses interesting situation-related details.	5	4	4	5	5	5	5	4	5
17	The operator is consequently kept up-to-date of the exact state of the situation of interest.	3	3	4	5	4	5	5	4	5
18	The development of the emerging situation of interest is comprehensible.	5	5	4	5	4	5	4	4	5
19	The Application supports a reactive mode of operation for the operator.	3	4	5	5	4	2	5	5	5
20	The Application supports a proactive mode of operation for the operator.	5	4	5	5	4	5	5	5	5
21	The Application Dashboard allows the retrospective analysis of situations of interest.	5	5	5	4	5	5	5	3	5
22	The automatic data extraction is suitable for further analysis.	4	5	5	3	5	5	5	4	4
23	The automatic data extraction is suitable to support the generation of prediction models.	-	4	5	3	5	4	5	-	4

Table A.1: Survey Answers

A.2 Questionnaire

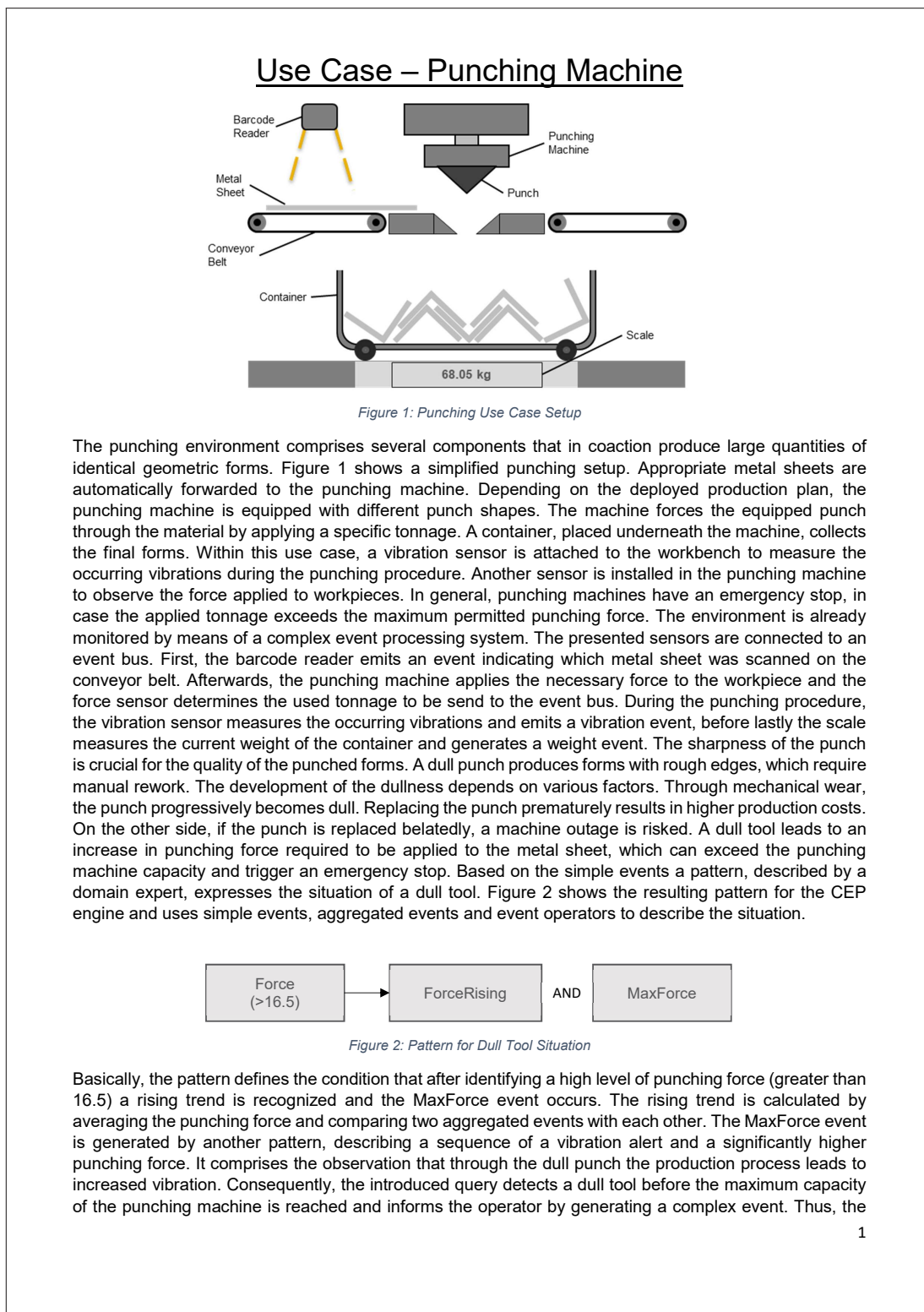
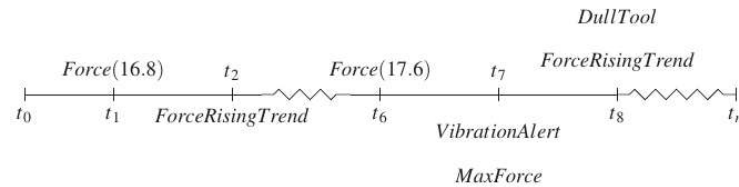


Figure A.1: Questionnaire Introduction - Page 1

punch is replaced before the machine triggers an emergency stop and leads to an unplanned maintenance, which results in higher maintenance costs. This gives a machine operator the possibility to replace the dull tool and restart the production process. Therefore, spare punches have to be hold available for replacement.

On the upcoming pages, the visualization of different monitoring applications is shown as the following event stream is getting processed by the underlying CEP engine:

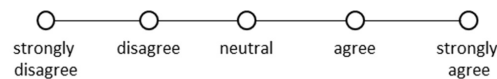


The illustrated event stream encompasses events that are captured in the production environment at different timestamps (t_x). The result of the processing of the event stream is the detection of the previously described situation at t_8 . All monitoring applications are based on Complex Event Processing and on the introduced pattern. The underlying CEP engine and event bus remain the same for all examples.

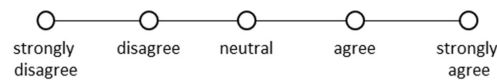
The survey consists of various statements regarding the presented monitoring applications and a scale to assess whether you agree or disagree to that statement.

Before beginning with the survey, please rate the following two statements with regard to your experiences.

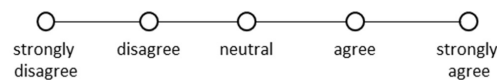
- I have experience with Complex Event Processing technologies.



- I have experience with monitoring applications.



- I have experience with the generation of prediction models.



1. State of the Art Monitoring

The first monitoring application, the **Situation Dashboard**, is based on a state-of-the-art CEP architecture comprising a CEP engine and an event bus. The left screenshot in Figure 3 shows a **Situation Dashboard** between t_0 and t_7 . With the occurrence of the situation at t_8 , the dashboard shows a graphical element representing the situation and its new state "Detected". State-of-the-art CEP monitoring applications differentiate between two situation states: Not Detected and Detected.

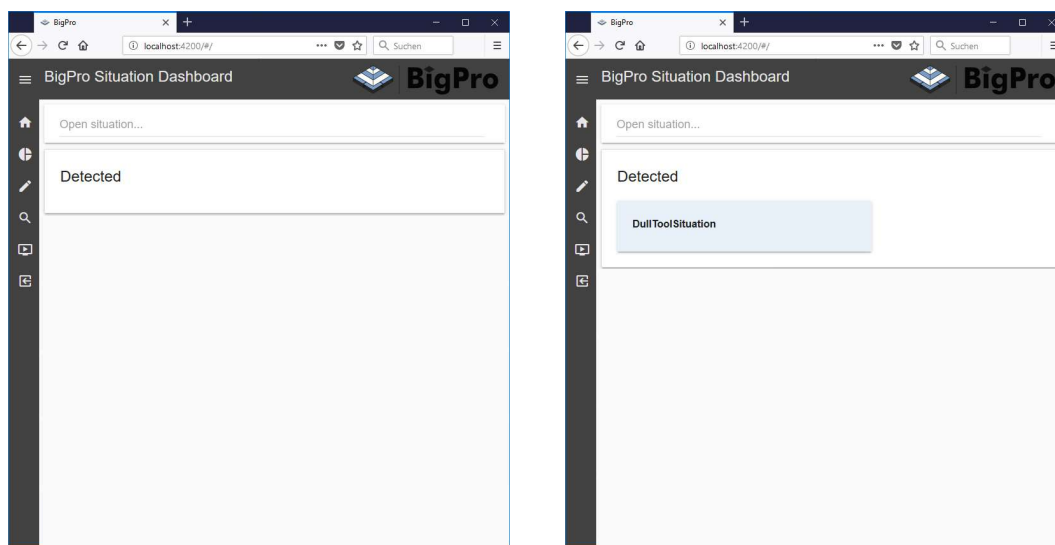
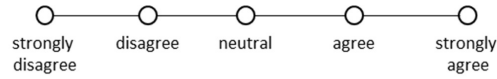


Figure 3: Situation Dashboard

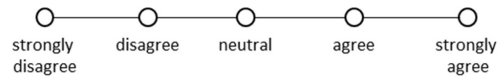
Please assess the following statements (on page 4) by taking into consideration the given information.

State of the Art Monitoring - Statements

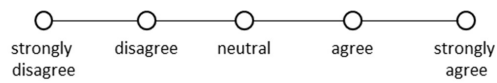
- **S1:** The state of the system during the processing of the event stream is transparent.



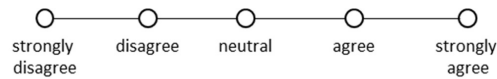
- **S2:** The Situation Dashboard encompasses interesting situation-related details.



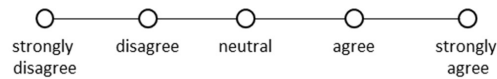
- **S3:** The Situation Dashboard directs the attention of an operator towards a situation at an early stage.



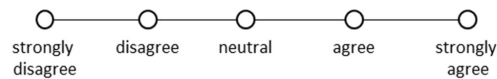
- **S4:** The operator is consequently kept up-to-date of the exact state of the situation of interest.



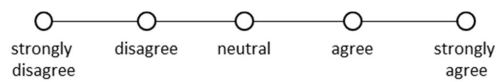
- **S5:** The development of the emerging situation of interest is comprehensible.



- **S6:** The Situation Dashboard supports a reactive mode of operation for the operator.



- **S7:** The Situation Dashboard supports a proactive mode of operation for the operator.



2. Situation-Aware Monitoring

The second monitoring application, the **BigPro Situation Management Dashboard**, follows a situation-aware approach and differentiates between five situation states. A situation starts in the state “Inactive”, as shown in Figure 4, and keeps this state until enough information about a potential situation occurrence is captured to justify a transition to the state “Initialized”. The card representing a situation known from the previous monitoring application is extended with additional information regarding a situation, like the current materialization, which events of the underlying pattern are already detected and which are still missing for the occurrence of the situation.

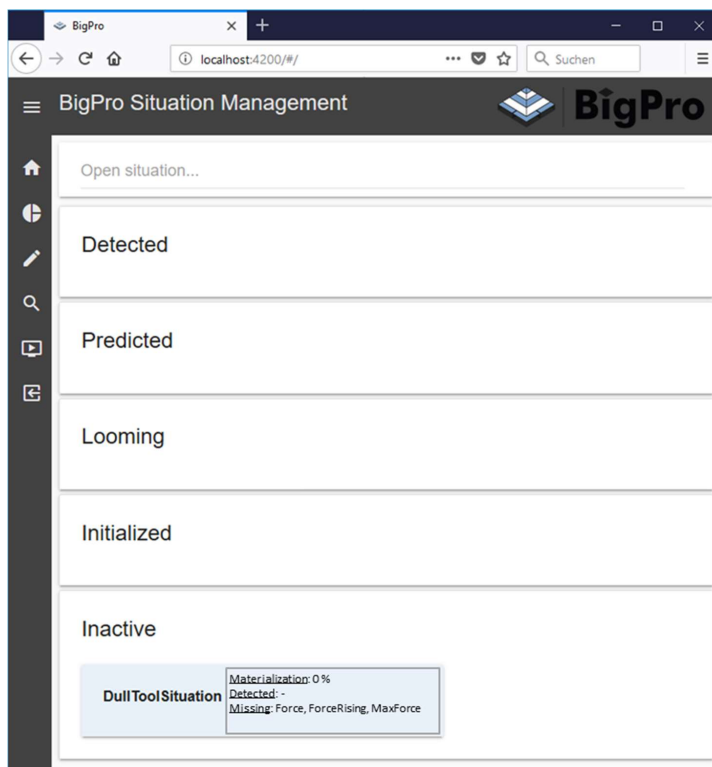


Figure 4: Situation Management Dashboard at t_0

The monitoring application tracks the situation during its development through the different states and updates the related information. Figure 5 shows the Situation Management Dashboard at t_2 with a current materialization of 66 %, as two out of three events of the pattern were already detected leading to the state “Initialized”.

This behavior continues, as shown in Figure 6, with the state transition to the state “Looming”, as a significant portion of the pattern has been detected at t_7 to justify a more mature state. As soon as a prediction becomes available for a situation, the state further evolves to the state “Predicted”, which also leads to an additional information about the probability of occurrence. The tracking of a situation ends with its detection and transition to the state “Detected”, which is shown in Figure 7.

Please assess the statements on page 8 after reviewing the visualization during the processing of the introduced event stream.

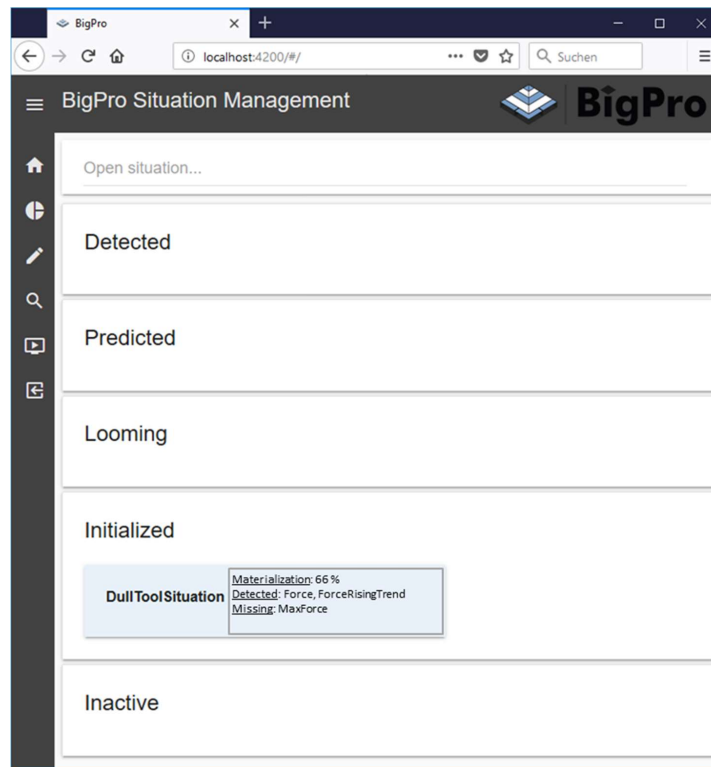


Figure 5: Situation Management Dashboard at t_2

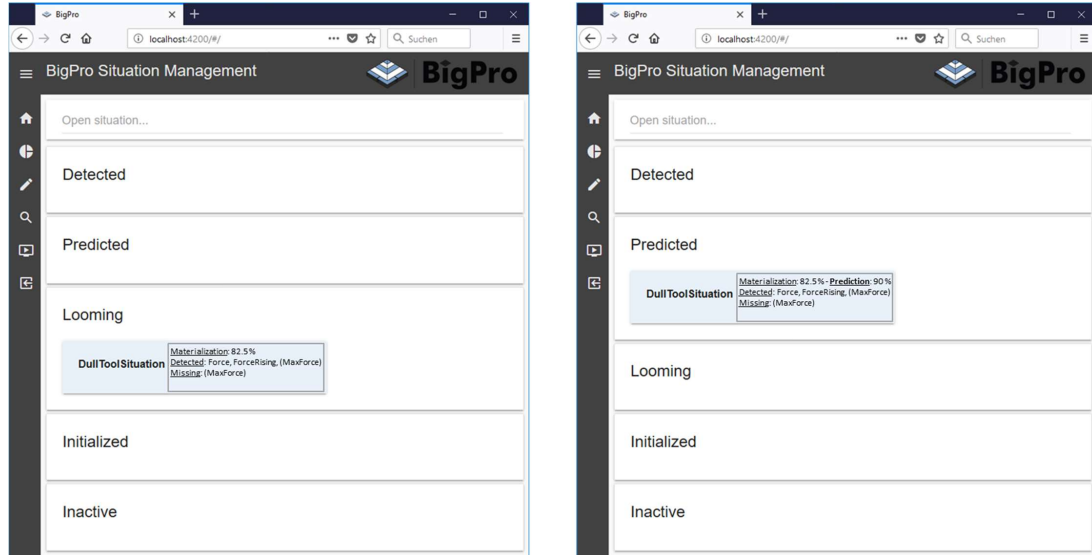


Figure 6: Situation Management Overview at t_3

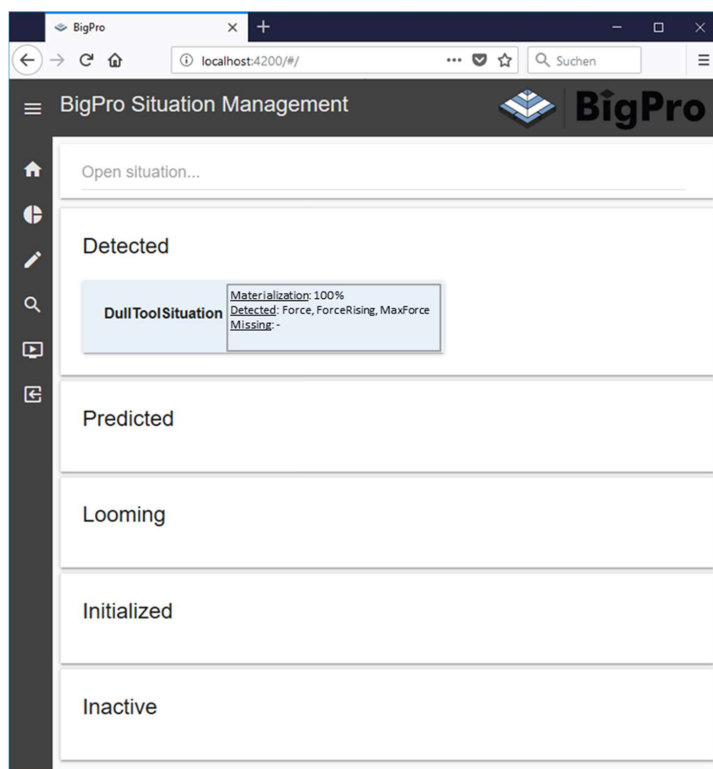
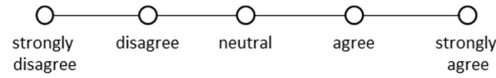


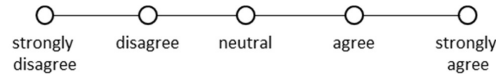
Figure 7: Situation Management Overview at t_8

Situation-Aware Monitoring - Statements

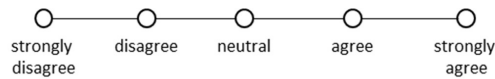
- **S8:** The state of the system during the processing of the event stream is transparent.



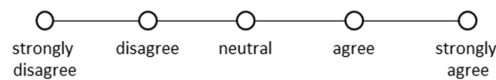
- **S9:** The Situation Management Dashboard encompasses interesting situation-related details.



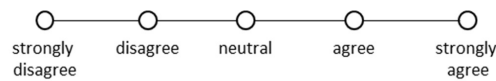
- **S10:** The Situation Management Dashboard directs the attention of an operator towards a situation at an early stage.



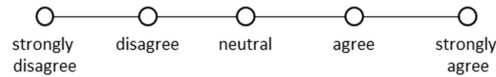
- **S11:** The operator is consequently kept up-to-date of the exact state of the situation of interest.



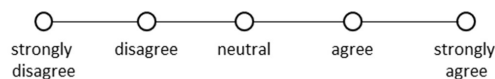
- **S12:** The development of the emerging situation of interest is comprehensible.



- **S13:** The Situation Management Dashboard supports a reactive mode of operation for the operator.



- **S14:** The Situation Management Dashboard supports a proactive mode of operation for the operator.



3. Transparent Monitoring

The third monitoring application shows an extended version of the previously presented situation-aware monitoring by additionally allowing drilling down into a situation's information by clicking on the specific situation. The following screenshots show different views on situation-related information. Figure 8 shows the overview for the situation of interest and a full pattern match. Firstly, the pattern-related data is shown, like the used time window (2 seconds) and the structure of the pattern. Additionally, basic data is shown related to the occurrence of the situation, like the count of occurrences up to now, the last materialization date and time, the last probability of occurrence that was calculated before this occurrence and the current state of the situation (in this case "Detected").

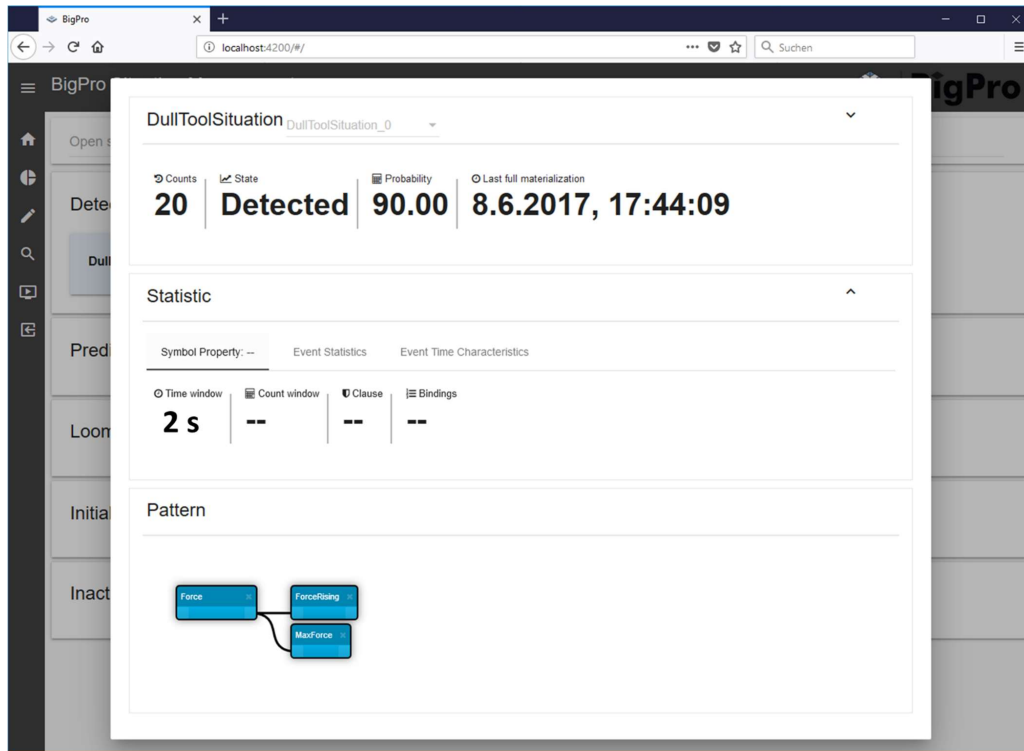


Figure 8 Details for Situation of Interest (Full Pattern)

As the collected information allow a drill down, it is possible to review the involved events retrospectively for current and past situation occurrences. Figure 9 shows a screenshot of this view. For every situation occurrence the involved event types, their time of occurrence and their value are shown. Additionally, it is possible to see statistics for every involved event type by selecting the event type in the pattern representation, shown in the "Pattern" section of the monitoring application.

Figure 10 depicts information that are shown for partial matches of a pattern. The information is accessible whenever needed by selecting the partial pattern in the drop-down at the top of the window ("DullToolSituation_2"). The screenshot shows details for the sub-pattern consisting of two events (Force and ForceRising). As shown, the partial pattern has occurred 22 times, with its last materialization as part of the full pattern match. Additional, to all information that are also available for full pattern matches, sub-patterns also contain data about their potential evolution paths and probabilities. Please note, that all this information are available independent of the fact whether a situation was already detected (full pattern match). The data is continuously available as the monitoring applications processes the production-related data and can be accessed any time.

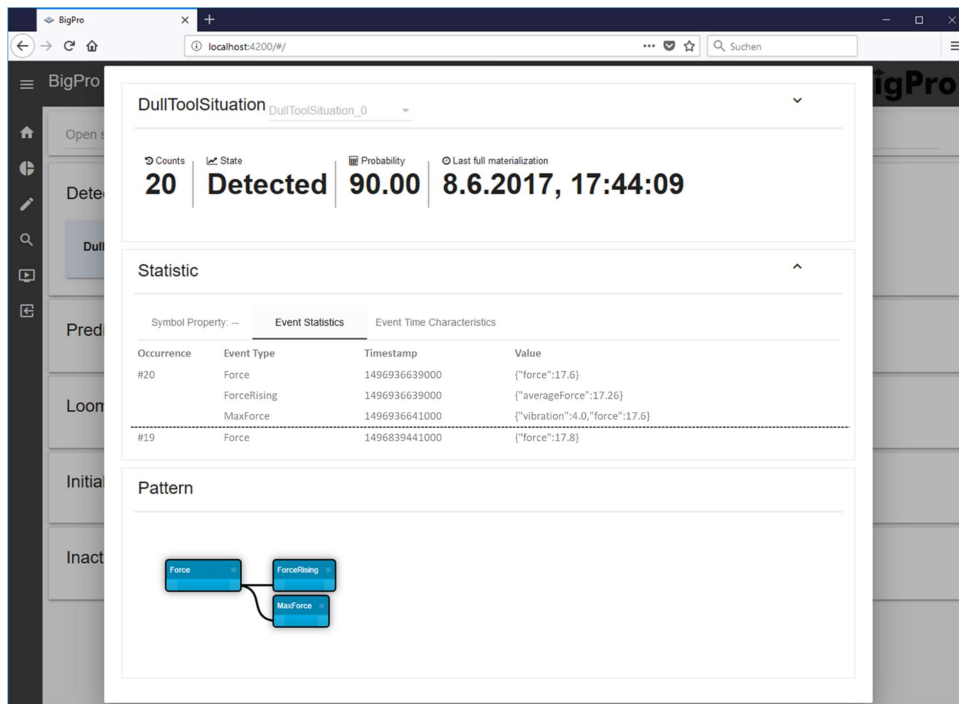


Figure 9 Event Details for occurred Situation

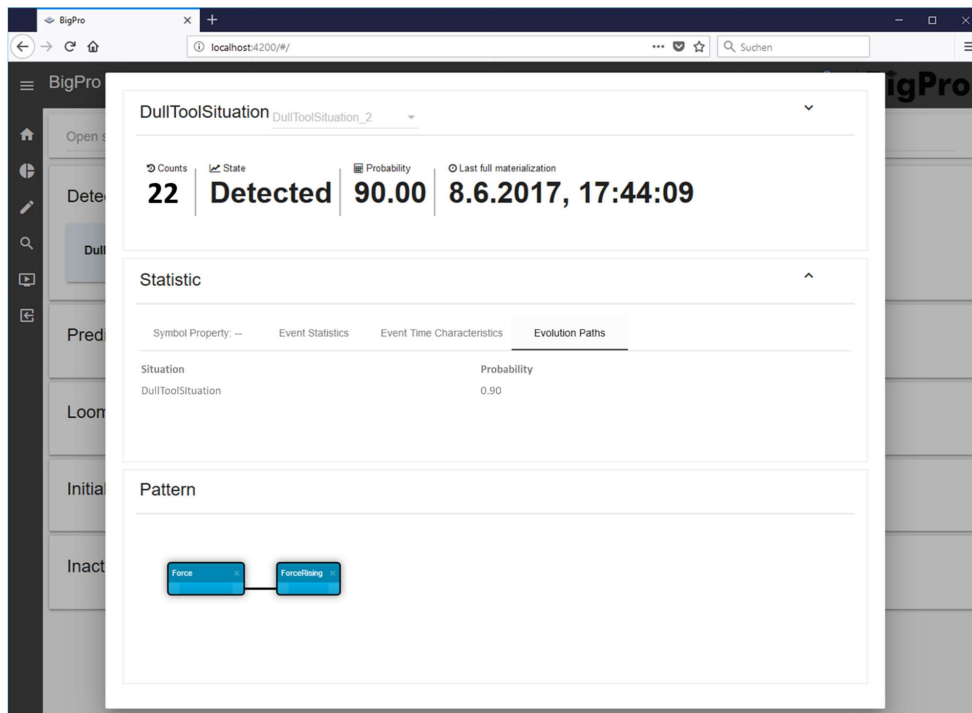


Figure 10 Information collected for Sub-Pattern DullToolSituation_2

In addition to collecting all situation-related information and representing it in the monitoring application, the system also extracts the data continuously in a structured format to support further analysis and processing of it, e.g. for the generation of prediction models for proactive situation management. The data is collected and labeled whether it was involved in a situation occurrence or not. Data is extracted from different perspectives for a situation, e.g. an extract is generated for every sub-pattern of the situation and whether the detected sub-pattern led to a situation or not. An example is shown in Figure 11. As all data is extracted, all IDs are kept in the files to allow cross-referencing between the different extracts. The last column shows the label of the specific row.

```

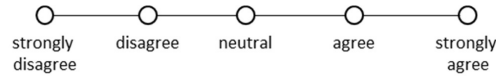
DTS2_TimeStamp,DTS2_EventID,DTS2_Force_Timestamp,DTS2_Force_EventID,DTS2_Force_force_Name,DTS2_Force_force_Value,DTS2_ForceRising_Timestamp, DTS2_ForceRising_EventID,...
1497093949720,SblD0ab4d8c9847a03369e4d,1497093949720,SblD0ab4d8c9847a033695f,force,17.3,1497093949720,SblD0ab4d8c9847a03369e0,averageForce,16.975,DullToolSituation
1497093959762,SblD0ac74d8c9847a0336b1d,1497093959762,SblD0ac74d8c9847a0336ae4,force,17.5,1497093959762,SblD0ac74d8c9847a0336ae6,averageForce,16.994444444444444,DullToolSituation
1497093964741,SblD0acc4d8c9847a0336f5c,1497093964741,SblD0acc4d8c9847a0336d0d,force,17.4,1497093964741,SblD0acc4d8c9847a0336d0e,averageForce,17.010526315789473,DullToolSituation
1497093969818,SblD0ad14d8c9847a0336f5e,1497093969806,SblD0ad14d8c9847a0336f5e,force,17.4,1497093969806,SblD0ad14d8c9847a0336f5f,averageForce,17.03,DullToolSituation
1497093974728,SblD0ae4d8c9847a033720a,1497093974728,SblD0ae4d8c9847a03371d0,force,17.4,1497093974728,SblD0ae4d8c9847a03371d1,averageForce,17.047619047619047,DullToolSituation
1497093979759,SblD0aeb4d8c9847a03374e3,1497093979754,SblD0aeb4d8c9847a033746a,force,17.4,1497093979754,SblD0aeb4d8c9847a033746b,averageForce,17.063636363636363,DullToolSituation
1497093984829,SblD0aef4d8c9847a0337796,1497093984806,SblD0aef4d8c9847a033773e,force,17.3,1497093984806,SblD0aef4d8c9847a033773e,averageForce,17.073913043487826,DullToolSituation
1497093989789,SblD0aee4d8c9847a0337a90,1497093989789,SblD0aee4d8c9847a0337a44,force,17.5,1497093989789,SblD0aee4d8c9847a0337a45,averageForce,17.091666666666666,DullToolSituation
1497093994765,SblD0aef4d8c9847a0337d6f,1497093994765,SblD0aef4d8c9847a0337d6e,force,17.5,1497093994765,SblD0aef4d8c9847a0337d6f,averageForce,17.099999999999999,DullToolSituation
1497093999787,SblD0aef4d8c9847a033810a,1497093999787,SblD0aef4d8c9847a03380cc,force,17.5,1497093999787,SblD0aef4d8c9847a03380cd,averageForce,17.115384615384613,DullToolSituation
1497094005120,SblD0af54d8c9847a03384e4,1497094005120,SblD0af54d8c9847a0338457,force,17.4,1497094005120,SblD0af54d8c9847a0338459,averageForce,17.125925925925923,DullToolSituation
1497094010141,SblD0af4d8c9847a033895c,1497094010141,SblD0af4d8c9847a033890b,force,17.3,1497094010141,SblD0af4d8c9847a033890c,averageForce,17.132142857142859,DullToolSituation
1497094015159,SblD0aff4d8c9847a0338c3c,1497094015159,SblD0aff4d8c9847a0338bf5,force,17.3,1497094015159,SblD0aff4d8c9847a0338bfa,averageForce,17.137531034482754,DullToolSituation
1497094020182,SblD0b04d8c9847a03390de,1497094020182,SblD0b04d8c9847a0339017,force,17.4,1497094020182,SblD0b04d8c9847a0339018,averageForce,17.146666666666666,DullToolSituation
1497094025119,SblD0b094d8c9847a033949f,1497094025119,SblD0b094d8c9847a0339457,force,17.5,1497094025119,SblD0b094d8c9847a0339458,averageForce,17.153333333333333,DullToolSituation
1497094030158,SblD0b0e4d8c9847a03398fe,1497094030152,SblD0b0e4d8c9847a03398b1,force,17.5,1497094030158,SblD0b0e4d8c9847a03398b2,averageForce,17.176666666666666,DullToolSituation
1497094035099,SblD0b134d8c9847a0339d9a,1497094035099,SblD0b134d8c9847a0339d4b,force,17.4,1497094035099,SblD0b134d8c9847a0339d4c,averageForce,17.193333333333332,DullToolSituation
1497094040162,SblD0b184d8c9847a033a2e6,1497094040162,SblD0b184d8c9847a033a211,force,17.3,1497094040162,SblD0b184d8c9847a033a212,averageForce,17.203333333333333,DullToolSituation
1497094045189,SblD0bd14d8c9847a033a763,1497094045188,SblD0bd14d8c9847a033a715,force,17.4,1497094045188,SblD0bd14d8c9847a033a717,averageForce,17.209999999999999,DullToolSituation
1496839441000,SblD0b224d8c9847a033ac90,1496839441000,SblD0b224d8c9847a033ac3e,force,17.8,1496839441000,SblD0b224d8c9847a033ac3f,averageForce,17.236666666666666,DullToolSituation
1496936469000,SblD0b274d8c9847a033b1d0,1496936469000,SblD0b274d8c9847a033b191,force,17.6,1496936469000,SblD0b274d8c9847a033b192,averageForce,17.263333333333332,DullToolSituation
    
```

Figure 11: Data Extract for DullToolSituation_2

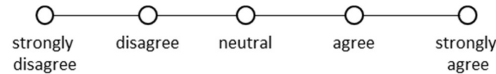
Please assess the following statements.

Transparent Monitoring - Statements

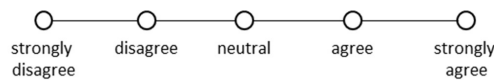
- **S15:** The state of the system during the processing of the event stream is transparent.



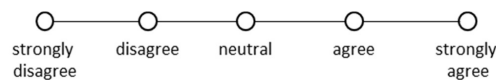
- **S16:** The Situation Management Dashboard encompasses interesting situation-related details.



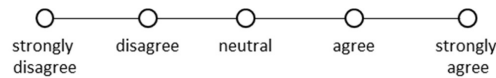
- **S17:** The operator is consequently kept up-to-date of the exact state of the situation of interest.



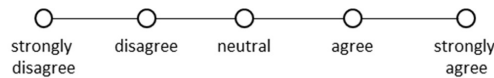
- **S18:** The development of the emerging situation of interest is comprehensible.



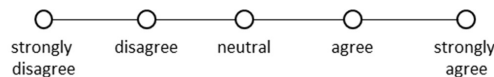
- **S19:** The Situation Management Dashboard supports a reactive mode of operation for the operator.



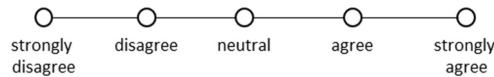
- **S20:** The Situation Management Dashboard supports a proactive mode of operation for the operator.



- **S21:** The Situation Management Dashboard allows the retrospective analysis of situations of interest.



- **S22:** The automatic data extraction is suitable for further analysis.



If you have experience with the generation of prediction models, please assess the last statement.

- **S23:** The automatic data extraction is suitable to support the generation of prediction models.

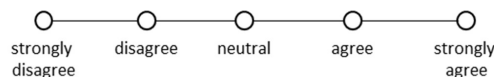


Figure A.12: Transparent Monitoring Scenario Statements - Page 12

List of Figures

1.1	Structure of the Thesis	9
2.1	Punching Example	13
2.2	Development of Punching Force vs. Punch Sharpness	13
2.3	Excerpt of Event Stream	16
2.4	Development of the Value of an Information according to [Fülöp et al., 2012]	17
3.1	Event Definition Example	24
3.2	Event Processing Network Example	25
3.3	Hierarchy of Event Processing Agents	27
3.4	Basic Complex Event Processing Architecture	30
3.5	Comparison of Programming Models	31
4.1	Relationship between Fields of Interest and Goals of Thesis	33
5.1	Research Questions and related Requirements	44
6.1	Levels of Transparency	50
6.2	Transparency Concept in Complex Event Processing	51
6.3	Model Overview	53
6.4	Model of an Situation of Interest	53
6.5	Example for a Hierarchical Relationship	54
6.6	Example for a Situation Hierarchy in a Query	54
6.7	Internals of a generic Event Processing Agent according to [Etzion and Niblett, 2011]	55
6.8	Model of an Input Stream	55
6.9	Example for an Input Stream in a Query	56
6.10	Model of a Pattern Detector	57
6.11	Example for a Pattern in a Query	58
6.12	Model of an Output	58
6.13	Example for an Action in a Query	59
6.14	Example for a Sequence Pattern Decomposition	60
6.15	Example for an AND-Pattern Decomposition	60
6.16	Example for an OR-Pattern Decomposition	60
6.17	Levels of Information	63
6.18	The General Architecture	66
6.19	Transformation Process of a Query-File into a Query-Model	67
6.20	Data Flow for Decomposition of Queries	67
6.21	Data Flow for Analysis of Queries	69
6.22	Example Query with Annotations	69
6.23	Example Situation Model	70
6.24	Example Pattern Model	71
6.25	Decomposition in Running Example	71
6.26	Sub-Query Statistics in Running Example	73
7.1	Model of Situation Awareness according to [Endsley, 1995]	76

7.2	Situation Management Overview - excerpt [Jakobson et al., 2006]	77
7.3	Comparison of Situation Management and Complex Event Processing	78
7.4	Implicit Life-cycle within state-of-the-art CEP Engines	79
7.5	Explicit extended Life-cycle for situation-aware Complex Event Processing	82
7.6	Possible Visualization for an Operator	85
7.7	Extended Architecture with State Handler	86
7.8	Data Flow of Extended Architecture with State Handler	87
7.9	Excerpt of Event Stream	88
7.10	Visualization at t_2	89
8.1	CRISP-DM Phases according [Wirth and Hipp, 2000]	92
8.2	CRISP-DM Phases and Complex Event Processing	93
8.3	Event Trace Labeling	95
8.4	Event Trace Extraction Levels for Running Example	96
8.5	Situation Prediction based on Feature Sets	98
8.6	Situation Prediction based on Feature Sets	98
8.7	Prediction Support Data Flow	99
8.8	Excerpt of Event Stream	100
8.9	Running Example Event Traces	101
9.1	Implementation Overview	108
9.2	Research Questions and related Requirements	109
9.3	Research Questions and related Statements in the User Study	113
9.4	Survey Answers related to Experience	114
9.5	Survey Answers related to Transparency	114
9.6	Survey Answers related to Level of Details and Retrospective Analysis	115
9.7	Survey Answers related to Situation Awareness	117
9.8	Survey Answers related to Reactivity/Proactivity	118
9.9	Survey Answers related to Predictive Support	119
9.10	Overview of generated Events in the Running Example	121
9.11	Comparison of Resource Utilization of CEP Engine <i>Apama</i>	122
9.12	Comparison of Resource Utilization of Event Bus <i>Universal Messaging</i>	123
9.13	Resource Utilization of Event Bus <i>Universal Messaging</i>	124
9.14	Resource Utilization of CEP Engine with multiple Queries	125
A.1	Questionnaire Introduction - Page 1	136
A.2	Questionnaire Introduction - Page 2	137
A.3	Simple Monitoring Scenario Description - Page 3	138
A.4	Simple Monitoring Scenario Statements - Page 4	139
A.5	Situation-Aware Monitoring Scenario Description - Page 5	140
A.6	Situation-Aware Monitoring Scenario Description - Page 6	141
A.7	Situation-Aware Monitoring Scenario Description - Page 7	142
A.8	Situation-Aware Monitoring Scenario Statements - Page 8	143
A.9	Transparent Monitoring Scenario Description - Page 9	144
A.10	Transparent Monitoring Scenario Description - Page 10	145
A.11	Transparent Monitoring Scenario Description - Page 11	146
A.12	Transparent Monitoring Scenario Statements - Page 12	147

List of Tables

1.1	Guidelines in Design Science for Information Systems Research	6
7.1	Parameter Settings for State Handler in Running Example	87
7.2	Evolution Probabilities based on Historical Data	88
7.3	Development of Situations in Running Example	89
8.1	Collected Data for Event Type <i>Force</i>	101
8.2	Collected Data for Partial Matches MF1 and DT2	102
9.1	Mapping between Statements and Monitoring Scenarios	113
9.2	Setup for Experiments	120
9.3	Parameter Settings for State Handler in Running Example	120
9.4	Comparison of Results for varying Throughputs	126
A.1	Survey Answers	135

Listings

2.1	Apama Query for Detection of Dull Tool	14
2.2	Apama Query for Detection of Max Force	14
6.1	Derived Sub-Query in AQL	72
8.1	Structured Export for DT3	96
8.2	Structured Format Export for Force Events	101
8.3	Prediction Configuration	102
8.4	DT2 Input for Prediction	102

Bibliography

- [Abadi et al., 2003] Abadi, D. J., Carney, D., Çetintemel, U., Cherniack, M., Convey, C., Lee, S., Stonebraker, M., Tatbul, N., and Zdonik, S. (2003). Aurora: a new model and architecture for data stream management. *the VLDB Journal*, 12(2):120–139.
- [Agrawal et al., 2008] Agrawal, J., Diao, Y., Gyllstrom, D., and Immerman, N. (2008). Efficient pattern matching over event streams. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 147–160. ACM.
- [Artikis et al., 2014] Artikis, A., Baber, C., Bizarro, P., Canudas-de Wit, C., Etzion, O., Fournier, F., Goulart, P., Howes, A., Lygeros, J., Paliouras, G., Schuster, A., and Sharfman, I. (2014). Scalable proactive event-driven decision making. *Technology and Society Magazine, IEEE*, 33(3).
- [Atzori et al., 2010] Atzori, L., Iera, A., and Morabito, G. (2010). The internet of things: A survey. *Computer networks*, 54(15):2787–2805.
- [Babcock et al., 2002] Babcock, B., Babu, S., Datar, M., Motwani, R., and Widom, J. (2002). Models and issues in data stream systems. In *Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 1–16. ACM.
- [Balboni and Cook, 2012] Balboni, F. and Cook, S. (2012). Analytics in the boardroom. accelerating competitive advantage. *IBM Global Business Services Executive Report*, pages 1–16.
- [Bassi and Horn, 2008] Bassi, A. and Horn, G. (2008). Internet of things in 2020: A roadmap for the future. *European Commission: Information Society and Media*, 22:97–114.
- [Baumgrass et al., 2014] Baumgrass, A., Dijkman, R., Grefen, P., Pourmirza, S., Völzer, H., and Weske, M. (2014). A software architecture for a transportation control tower. *Technische Universiteit Eindhoven, Tech. Rep. Beta Working Paper series 461*.
- [Bell et al., 2009] Bell, G., Hey, T., and Szalay, A. (2009). Beyond the data deluge. *Science*, 323(5919):1297–1298.
- [Boyd, 1996] Boyd, J. R. (1996). The essence of winning and losing. *Unpublished lecture notes*, 12(23):123–125.
- [Bruns and Dunkel, 2015] Bruns, R. and Dunkel, J. (2015). *Complex Event Processing: komplexe analyse von massiven datenströmen mit CEP*. Springer-Verlag.
- [Chandy and Schulte, 2009] Chandy, K. and Schulte, W. (2009). *Event processing: designing IT systems for agile companies*. McGraw-Hill, Inc.
- [Chen and Zhang, 2014] Chen, C. P. and Zhang, C.-Y. (2014). Data-intensive applications, challenges, techniques and technologies: A survey on big data. *Information Sciences*, 275:314–347.
- [Corritore et al., 2003] Corritore, C. L., Kracher, B., and Wiedenbeck, S. (2003). On-line trust: concepts, evolving themes, a model. *International journal of human-computer studies*, 58(6):737–758.
- [Costa et al., 2016] Costa, P. D., Almeida, J. P. A., Pereira, I. S., van Sinderen, M., and Pires, L. F. (2016). Rule-based support for situation management. In *Fusion Methodologies in Crisis Management*, pages 341–364. Springer.
- [Cugola and Margara, 2012] Cugola, G. and Margara, A. (2012). Processing flows of information: From data stream to complex event processing. *ACM Computing Surveys (CSUR)*, 44(3):15.

- [Dayal et al., 1988] Dayal, U., Buchmann, A. P., and McCarthy, D. R. (1988). Rules are objects too: a knowledge model for an active, object-oriented database system. In *International Workshop on Object-Oriented Database Systems*, pages 129–143. Springer.
- [Endsley, 1995] Endsley, M. R. (1995). Toward a theory of situation awareness in dynamic systems. *Human Factors: The Journal of the Human Factors and Ergonomics Society*, 37(1).
- [Endsley and Connors, 2008] Endsley, M. R. and Connors, E. S. (2008). Situation awareness: State of the art. In *Energy Society General Meeting*.
- [Engel and Etzion, 2011] Engel, Y. and Etzion, O. (2011). Towards proactive event-driven computing. In *Proceedings of the 5th ACM International Conference on Distributed Event-based System*, DEBS '11, pages 125–136. ACM.
- [Engel et al., 2012] Engel, Y., Etzion, O., and Feldman, Z. (2012). A basic model for proactive event-driven computing. In *Proceedings of the 6th ACM International Conference on Distributed Event-Based Systems*, DEBS '12, New York, NY, USA. ACM.
- [Etzion and Niblett, 2011] Etzion, O. and Niblett, P. (2011). *Event processing in action*. Manning Greenwich.
- [Eugster et al., 2003] Eugster, P. T., Felber, P. A., Guerraoui, R., and Kermarrec, A.-M. (2003). The many faces of publish/subscribe. *ACM computing surveys (CSUR)*, 35(2):114–131.
- [Fayyad, 1996] Fayyad, U. M. (1996). Data mining and knowledge discovery: Making sense out of data. *IEEE Expert: Intelligent Systems and Their Applications*, 11(5):20–25.
- [Feldman et al., 2013] Feldman, Z., Fournier, F., Franklin, R., and Metzger, A. (2013). Proactive event processing in action: A case study on the proactive management of transport processes (industry article). In *Proceedings of the 7th ACM International Conference on Distributed Event-based Systems*, DEBS '13, New York, NY, USA. ACM.
- [Flouris et al., 2017] Flouris, I., Giatrakos, N., Deligiannakis, A., Garofalakis, M., Kamp, M., and Mock, M. (2017). Issues in complex event processing: Status and prospects in the big data era. *Journal of Systems and Software*, 127:217–236.
- [Fournier et al., 2015] Fournier, F., Kofman, A., Skarbovsky, I., and Skarlatidis, A. (2015). Extending event-driven architecture for proactive systems. In *EDBT/ICDT Workshops*.
- [Fülöp et al., 2012] Fülöp, L. J., Beszédes, Á., Tóth, G., Demeter, H., Vidács, L., and Farkas, L. (2012). Predictive complex event processing: a conceptual framework for combining complex event processing and predictive analytics. In *Proceedings of the Fifth Balkan Conference in Informatics*, pages 26–31. ACM.
- [Fülöp et al., 2010] Fülöp, L. J., Tóth, G., Rácz, R., Pánczél, J., Gergely, T., Beszédes, A., and Farkas, L. (2010). Survey on complex event processing and predictive analytics. Technical report, University of Szeged.
- [Glass et al., 2008] Glass, A., McGuinness, D. L., and Wolverson, M. (2008). Toward establishing trust in adaptive agents. In *Proceedings of the 13th international conference on Intelligent user interfaces*, pages 227–236. ACM.
- [Golab and Özsu, 2005] Golab, L. and Özsu, M. T. (2005). Update-pattern-aware modeling and processing of continuous queries. In *Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, pages 658–669. ACM.
- [Hang and Fong, 2010] Hang, Y. and Fong, S. (2010). The impacts of data stream mining on real-time business intelligence. In *Proceedings of the 2nd International Conference on IT and Business Intelligence (ITBI 2010)*.

- [Hevner et al., 2004] Hevner, A. R., March, S. T., Park, J., and Ram, S. (2004). Design science in information systems research. *MIS quarterly*, 28(1):75–105.
- [IEEE, 1990] IEEE (1990). Ieee standard glossary of software engineering terminology (ieee std 610.12-1990). los alamos. CA: *IEEE Computer Society*, 169.
- [Jagadish et al., 2014] Jagadish, H., Gehrke, J., Labrinidis, A., Papakonstantinou, Y., Patel, J. M., Ramakrishnan, R., and Shahabi, C. (2014). Big data and its technical challenges. *Communications of the ACM*, 57(7):86–94.
- [Jakobson et al., 2006] Jakobson, G., Buford, J., and Lewis, L. (2006). A framework of cognitive situation modeling and recognition. In *Military Communications Conference, 2006. MILCOM 2006. IEEE*, pages 1–7. IEEE.
- [Jeschke et al., 2017] Jeschke, S., Brecher, C., Meisen, T., Özdemir, D., and Eschert, T. (2017). Industrial internet of things and cyber manufacturing systems. In *Industrial Internet of Things*, pages 3–19. Springer.
- [Johnson and Johnson, 1993] Johnson, H. and Johnson, P. (1993). Explanation facilities and interactive systems. In *Proceedings of the 1st international conference on Intelligent user interfaces*, pages 159–166. ACM.
- [Kagermann et al., 2013] Kagermann, H., Helbig, J., Hellinger, A., and Wahlster, W. (2013). *Recommendations for implementing the strategic initiative INDUSTRIE 4.0: Securing the future of German manufacturing industry; final report of the Industrie 4.0 Working Group*. Forschungsunion.
- [Kalmykov and Kalmykov, 2015] Kalmykov, L. V. and Kalmykov, V. L. (2015). A white-box model of s-shaped and double s-shaped single-species population growth. *PeerJ*, 3:e948.
- [Kantowitz et al., 1997] Kantowitz, B. H., Hanowski, R. J., and Kantowitz, S. C. (1997). Driver acceptance of unreliable traffic information in familiar and unfamiliar settings. *Human Factors*, 39(2):164–176.
- [Kibangou et al., 2017] Kibangou, A., Artikis, A., Michelioudakis, E., Paliouras, G., Schmitt, M., Lygeros, J., Baber, C., Morar, N., Fournier, F., and Skarbovsky, I. (2017). An integrated and scalable platform for proactive event-driven traffic management. *arXiv preprint arXiv:1703.02810*.
- [Kizilcec, 2016] Kizilcec, R. F. (2016). How much information?: Effects of transparency on trust in an algorithmic interface. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, pages 2390–2395. ACM.
- [Kokar et al., 2009] Kokar, M. M., Matheus, C. J., and Baclawski, K. (2009). Ontology-based situation awareness. *Information fusion*, 10(1):83–98.
- [Laney, 2001] Laney, D. (2001). 3d data management: Controlling data volume, velocity and variety. *META Group Research Note*, 6(70).
- [Luckham, 2002] Luckham, D. (2002). *The power of events*, volume 204. Addison-Wesley Reading.
- [Luckham, 2011] Luckham, D. C. (2011). *Event processing for business: organizing the real-time enterprise*. John Wiley & Sons.
- [Ma et al., 2015] Ma, M., Wang, P., Chu, C.-H., and Liu, L. (2015). Efficient multipattern event processing over high-speed train data streams. *IEEE Internet of Things Journal*, 2(4):295–309.
- [Madia, 2014] Madia, K. (2014). The evolution of complex event processing. <http://www.ibmbigdatahub.com/blog/evolution-complex-event-processing/>. [Online; accessed 03-May-2018].
- [McCarthy and Dayal, 1989] McCarthy, D. and Dayal, U. (1989). The architecture of an active database management system. In *ACM Sigmod Record*, volume 18, pages 215–224. ACM.

- [Metzger et al., 2012] Metzger, A., Franklin, R., and Engel, Y. (2012). Predictive monitoring of heterogeneous service-oriented business networks: The transport and logistics case. In *SRII Global Conference (SRII), 2012 Annual*.
- [Mirski et al., 2017] Mirski, P., Bernsteiner, R., and Radi, D. (2017). Analytics in human resource management the openskimr approach. *Procedia Computer Science*, 122:727–734.
- [Mousheimish et al., 2017] Mousheimish, R., Taher, Y., and Zeitouni, K. (2017). Automatic learning of predictive cep rules: bridging the gap between data mining and complex event processing. In *Proceedings of the 11th ACM International Conference on Distributed and Event-based Systems*, pages 158–169. ACM.
- [Nagel, 2016] Nagel, J. (2016). Electronic trading in fixed income markets.
- [Nilsson et al., 2008] Nilsson, M., Van Laere, J., Ziemke, T., and Edlund, J. (2008). Extracting rules from expert operators to support situation awareness in maritime surveillance. In *Information Fusion, 2008 11th International Conference on*, pages 1–8. IEEE.
- [Rajasekharan, 2015] Rajasekharan, A. (2015). Data preparation strategies for advanced predictive analytics. <http://www.ibmbigdatahub.com/blog/data-preparation-strategies-advanced-predictive-analytics>. [Online; accessed 25-May-2018].
- [Rosemann and Recker, 2006] Rosemann, M. and Recker, J. C. (2006). Context-aware process design: Exploring the extrinsic drivers for process flexibility. In *The 18th International Conference on Advanced Information Systems Engineering. Proceedings of Workshops and Doctoral Consortium*, pages 149–158. Namur University Press.
- [Salfinger et al., 2013] Salfinger, A., Retschitzegger, W., and Schwinger, W. (2013). Maintaining situation awareness over time - a survey on the evolution support of situation awareness systems. In *Technologies and Applications of Artificial Intelligence (TAAI), 2013 Conference on*, pages 274–281. IEEE.
- [Salfinger et al., 2014] Salfinger, A., Retschitzegger, W., and Schwinger, W. (2014). Staying aware in an evolving world - specifying and tracking evolving situations. In *Cognitive Methods in Situation Awareness and Decision Support (CogSIMA), 2014 IEEE International Inter-Disciplinary Conference on*, pages 195–201. IEEE.
- [Schroeck et al., 2012] Schroeck, M., Shockley, R., Smart, J., Romero-Morales, D., and Tufano, P. (2012). Analytics: Big data in der praxis. wie innovative unternehmen ihre datenbestände effektiv nutzen. *IBM Global Business Services Executive Report*.
- [Schwegmann et al., 2013] Schwegmann, B., Matzner, M., and Janiesch, C. (2013). precep: facilitating predictive event-driven process analytics. In *International Conference on Design Science Research in Information Systems*, pages 448–455. Springer.
- [Seidewitz, 2003] Seidewitz, E. (2003). What models mean. *IEEE software*, 20(5):26–32.
- [Sejdovic, 2016] Sejdovic, S. (2016). Partial pattern fulfillment and its application in event processing. In *Proceedings of the 10th ACM International Conference on Distributed and Event-based Systems*, pages 358–361. ACM.
- [Sejdovic et al., 2017] Sejdovic, S., Euting, S., Riemer, D., and Sure-Vetter, Y. (2017). Considering human factors in the development of situation-aware cep applications. In *Proceedings of the 11th ACM International Conference on Distributed and Event-based Systems*, pages 24–33. ACM.
- [Sejdovic et al., 2016] Sejdovic, S., Hegenbarth, Y., Ristow, G. H., and Schmidt, R. (2016). Proactive disruption management system: how not to be surprised by upcoming situations. In *Proceedings of the 10th ACM International Conference on Distributed and Event-based Systems*, pages 281–288. ACM.

- [Sejdovic and Kleiner, 2016] Sejdovic, S. and Kleiner, N. (2016). Proactive and dynamic event-driven disruption management in the manufacturing domain. In *Proceedings of the IEEE 14th International Conference on Industrial Informatics (INDIN)*, pages 1320–1325. IEEE.
- [Sen, 2013] Sen, S. (2013). *Efficient and effective event pattern management*. PhD thesis, Karlsruhe Institute of Technology.
- [Sen and Stojanovic, 2010] Sen, S. and Stojanovic, N. (2010). Gruve: a methodology for complex event pattern life cycle management. In *International Conference on Advanced Information Systems Engineering*, pages 209–223. Springer.
- [Sen et al., 2010] Sen, S., Stojanovic, N., and Stojanovic, L. (2010). An approach for iterative event pattern recommendation. In *Proceedings of the Fourth ACM International Conference on Distributed Event-Based Systems*, pages 196–205. ACM.
- [Severtson et al., 2017] Severtson, B., Rohm, W. A., Martens, J., Ericson, G., and GuhaThakurta, D. (2017). What is the team data science process? <https://docs.microsoft.com/en-us/azure/machine-learning/team-data-science-process/overview>. [Online; accessed 25-May-2018].
- [Sharon and Etzion, 2007] Sharon, G. and Etzion, O. (2007). *Event processing network - A conceptual model*. Technion - Israel Institute of Technology, Faculty of Industrial and Management Engineering.
- [Shearer, 2000] Shearer, C. (2000). The crisp-dm model: the new blueprint for data mining. *Journal of data warehousing*, 5(4):13–22.
- [Shmueli and Koppius, 2011] Shmueli, G. and Koppius, O. R. (2011). Predictive analytics in information systems research. *Mis Quarterly*, pages 553–572.
- [Simon, 1971] Simon, H. A. (1971). *Designing Organizations for an Information-Rich World*.
- [Sparck Jones, 1972] Sparck Jones, K. (1972). A statistical interpretation of term specificity and its application in retrieval. *Journal of documentation*, 28(1):11–21.
- [Stich et al., 2015] Stich, V., Jordan, F., Birkmeier, M., Oflazgil, K., Reschke, J., and Diews, A. (2015). Big data technology for resilient failure management in production systems. In *IFIP International Conference on Advances in Production Management Systems*, pages 447–454. Springer.
- [Stojanovic and Artikis, 2011] Stojanovic, N. and Artikis, A. (2011). On complex event processing for real-time situational awareness. In *International Workshop on Rules and Rule Markup Languages for the Semantic Web*, pages 114–121. Springer.
- [Stonebraker et al., 2005] Stonebraker, M., Çetintemel, U., and Zdonik, S. (2005). The 8 requirements of real-time stream processing. *ACM Sigmod Record*, 34(4):42–47.
- [Terroso-Sáenz et al., 2016] Terroso-Sáenz, F., González-Vidal, A., and Skarmeta, A. F. (2016). Towards anticipate detection of complex event processing rules with probabilistic modelling. *International Journal of Design & Nature and Ecodynamics*, 11(3):275–283.
- [Tóth et al., 2010] Tóth, G., Fülöp, L. J., Vidács, L., Beszédes, A., Demeter, H., and Farkas, L. (2010). Complex event processing synergies with predictive analytics. In *Proceedings of the Fourth ACM International Conference on Distributed Event-Based Systems, DEBS '10*, New York, NY, USA. ACM.
- [Van der Aalst et al., 2011] Van der Aalst, W. M., Schonenberg, M. H., and Song, M. (2011). Time prediction based on process mining. *Information systems*, 36(2):450–475.

- [van Dongen et al., 2008] van Dongen, B. F., Crooy, R. A., and van der Aalst, W. M. (2008). Cycle time prediction: When will this case finally be finished? In *OTM Confederated International Conferences "On the Move to Meaningful Internet Systems"*, pages 319–336. Springer.
- [Wang and Cao, 2014] Wang, Y. and Cao, K. (2014). A proactive complex event processing method for large-scale transportation internet of things. *International Journal of Distributed Sensor Networks*, 10(3):159052.
- [Wirth and Hipp, 2000] Wirth, R. and Hipp, J. (2000). Crisp-dm: Towards a standard process model for data mining. In *Proceedings of the 4th international conference on the practical applications of knowledge discovery and data mining*, pages 29–39. Citeseer.
- [Wortmann and Flüchter, 2015] Wortmann, F. and Flüchter, K. (2015). Internet of things. *Business & Information Systems Engineering*, 57(3):221–224.
- [Wu et al., 2006] Wu, E., Diao, Y., and Rizvi, S. (2006). High-performance complex event processing over streams. In *Proceedings of the 2006 ACM SIGMOD international conference on Management of data*, pages 407–418. ACM.
- [Ye and Keogh, 2009] Ye, L. and Keogh, E. (2009). Time series shapelets: a new primitive for data mining. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 947–956. ACM.
- [Zhang et al., 2017] Zhang, S., Vo, H. T., Dahlmeier, D., and He, B. (2017). Multi-query optimization for complex event processing in sap esp. In *Data Engineering (ICDE), 2017 IEEE 33rd International Conference on*, pages 1213–1224. IEEE.
- [Zowghi and Coulin, 2005] Zowghi, D. and Coulin, C. (2005). Requirements elicitation: A survey of techniques, approaches, and tools. In *Engineering and managing software requirements*, pages 19–46. Springer.