

# Efficient and Effective Event Pattern Management

Dissertation  
von

**M.Sc. Sinan Şen**

Zur Erlangung des akademischen Grades eines Doktors der  
Wirtschaftswissenschaften (Dr. rer. pol.) von der Fakultät für  
Wirtschaftswissenschaften des KIT genehmigte Dissertation

Hauptreferent:	Prof. Dr. Rudi Studer
Korreferent:	Prof. Dr. York Sure-Vetter
Tag der mündlichen Prüfung	08.02.2013



*to Shoguf & Sara Elif*



# Abstract

Complex Event Processing (CEP) provides an effective and efficient infrastructure for an agile decision support. The strength of CEP resides in the declarative and centralized representation of relevant business situations in form of event patterns, the real-time capability to detect relevant business situations and the scalable architecture to deal with a high throughput of events. In recent years CEP has become the most important enabling technology to provide a real-time situational awareness in several business domains like algorithmic trading, business activity monitoring or fraud detection to name a few.

However, while the real-time capability is in the core of CEP systems and well supported, the issue of event pattern management, including the modelling, deployment and continuous evolution of event patterns, is still in its infancy. First, existing tools to model event patterns are restricted on manual event pattern generation without any systematic modelling guidance. Moreover they neglect the reuse of existing pattern knowledge, that encode the most valuable asset in CEP systems. Second, the evolution of these patterns, which is a crucial point of nowadays IT systems, is left to the user (pattern engineer) of the CEP system. Third, the life cycle of an event pattern is not well investigated in order to provide adequate methods and tools at different life cycle stages of an event pattern. In a nutshell the lack of methods, tools and methodologies in event pattern management decreases the efficiency and effectiveness of the pattern engineer.

The goal of this thesis is to reduce the barriers stopping more enterprises from accessing CEP technology by providing additional support in managing relevant business situations. Therefore we outline the role of event pattern management and present a methodology, methods and tools aiming at an efficient and effective event pattern management. We provide a meta model for event patterns, an event pattern life cycle methodology, methods for guidance, refinement and evolution. The life cycle methodology aims at covering both design time aspects and run time aspects of an event pattern. The meta model enables the detection of event pattern relationships on different event pattern structure levels that will be used to refine and adapt an event pattern. The methodology and methods are incorporated by the PANTEON tool that provides a graphical user interface for modelling, searching, refining, deploying and evolving event patterns.



# Contents

List of Figures	xi
List of Tables	xv
<b>I Foundation</b>	<b>1</b>
<b>1 Introduction</b>	<b>3</b>
1.1 Motivation . . . . .	3
1.2 Tackling event patterns . . . . .	5
1.3 Research contribution . . . . .	7
1.4 Project context and publications . . . . .	8
1.5 Reader's guide . . . . .	11
<b>2 Complex Event Processing</b>	<b>13</b>
2.1 Event processing architecture . . . . .	13
2.1.1 Events . . . . .	14
2.1.2 Event producer . . . . .	16
2.1.3 Event processing agent . . . . .	16
2.1.4 Event consumer . . . . .	17
2.2 Event patterns . . . . .	18
2.2.1 Basic patterns . . . . .	21
2.2.1.1 Logical operator pattern . . . . .	21
2.2.1.2 Threshold operator pattern . . . . .	22
2.2.1.3 Subset operator pattern . . . . .	22
2.2.2 Dimensional pattern . . . . .	22
<b>II Efficient and Effective Event Pattern Management</b>	<b>25</b>
<b>3 The Need for Event Pattern Management</b>	<b>27</b>
3.1 Why event pattern management matters? . . . . .	27
3.1.1 Academic research on event pattern management . . . . .	29
3.1.2 Event pattern management in current CEP solutions . . . . .	30

3.2	Related research areas . . . . .	31
3.2.1	Business rule management . . . . .	31
3.2.2	Active database management . . . . .	32
3.3	Efficiency and effectiveness aspects in event pattern management .	32
3.4	Conclusion . . . . .	34
<b>4</b>	<b>Event Pattern Life Cycle Methodology</b>	<b>37</b>
4.1	Introduction . . . . .	37
4.2	Generation phase . . . . .	38
4.3	Execution phase . . . . .	41
4.4	Evolution phase . . . . .	42
4.5	Related work . . . . .	43
4.6	Summary . . . . .	44
<b>5</b>	<b>Event Pattern Meta Model</b>	<b>47</b>
5.1	Introduction . . . . .	47
5.2	Event layer . . . . .	48
5.3	Event pattern layer . . . . .	54
5.4	Implementation . . . . .	56
5.5	Related work . . . . .	58
5.6	Summary . . . . .	61
<b>6</b>	<b>Relationships between Event Patterns</b>	<b>63</b>
6.1	Introduction . . . . .	63
6.2	Conceptual model . . . . .	64
6.3	Graph theory . . . . .	66
6.4	Graph relationships between event patterns . . . . .	66
6.4.1	Event pattern subsumption and overlapping . . . . .	68
6.4.2	Best practice event pattern detection . . . . .	71
6.5	Event pattern similarity . . . . .	72
6.5.1	Taxonomy-based event pattern similarity . . . . .	73
6.5.2	Feature-based event pattern similarity . . . . .	75
6.6	Implementation . . . . .	75
6.6.1	User interface . . . . .	76
6.6.2	User interaction . . . . .	77
6.6.3	Architecture . . . . .	78
6.7	Related work . . . . .	79
6.8	Summary . . . . .	79
<b>7</b>	<b>Event Pattern Evolution</b>	<b>81</b>
7.1	Introduction . . . . .	81
7.2	Transparent event pattern matching . . . . .	82
7.3	The process of execution-driven evolution . . . . .	84
7.3.1	Constructing the pattern execution statistics . . . . .	85
7.3.2	Pattern frequency distribution and frequency clustering . .	86



---

7.4	Detection of evolution candidates . . . . .	87
7.4.1	Preparation for the evolution . . . . .	89
7.4.2	Evolution adaptation . . . . .	91
7.5	Implementation . . . . .	92
7.6	Related Work . . . . .	93
7.7	Summary . . . . .	93
<b>III</b>	<b>Finale</b>	<b>95</b>
<b>8</b>	<b>Evaluation</b>	<b>97</b>
8.1	Usability evaluation . . . . .	97
8.1.1	System Usability Scale (SUS) . . . . .	99
8.2	Evaluation in the context of the ALERT project . . . . .	99
8.3	Final evaluation based on SUS . . . . .	102
8.3.1	Evaluation methodology . . . . .	102
8.3.2	Scenario definition . . . . .	104
8.3.3	Task completion procedure . . . . .	106
8.3.4	Evaluation results . . . . .	106
8.3.5	Discussion of the results . . . . .	109
8.4	Performance tests . . . . .	109
8.4.1	Modelling expressivity . . . . .	110
8.4.2	Event pattern relation detection . . . . .	110
8.4.3	Performance of the event pattern evolution component . . . . .	112
8.5	Conclusion . . . . .	112
<b>9</b>	<b>Conclusion</b>	<b>113</b>
9.1	Summary . . . . .	113
9.2	Future direction . . . . .	115
9.3	Conclusion . . . . .	116
<b>A</b>	<b>Appendix</b>	<b>117</b>
A.1	ALERT Questionnaire . . . . .	117
A.2	ALERT Questionnaire Results . . . . .	124
A.3	EPTS event processing use case template . . . . .	136
	<b>Bibliography</b>	<b>139</b>



# List of Figures

1.1	The building blocks of knowledge management according to [Prob98].	6
2.1	The structure of an event processing application according to [EtNi10].	14
2.2	Different type of event processing agents according to [EtNi10].	17
3.1	Customer survey taken from ebizQ.	28
3.2	Most important event processing features.	30
3.3	Dimensions of rule management in active databases [PaDi99].	32
4.1	Systems/Software Development Life Cycle (SDLC).	38
4.2	The event pattern life cycle management methodology.	38
4.3	Pattern modelling activities.	39
4.4	Transformation of an event pattern in native CEP language.	42
4.5	Methodology according to [ViKD10].	44
4.6	Comparison of event pattern rule-management in existing systems (a) and the proposed approach (b) by [OSSK <sup>+</sup> 11].	44
5.1	Event meta model.	49
5.2	Event meta model extended with event operator in order to define complex events.	52
5.3	Event pattern meta model.	54
5.4	The PANTEON event editor.	57
5.5	Definition of the event types and attributes in PANTEON event editor.	57
5.6	Definition of the event sources and attributes in PANTEON event editor.	58
5.7	Definition of the final event template in PANTEON event editor.	58
6.1	The process of new event pattern (EPAT) development.	64

6.2	Conceptual model of pattern suggestion. . . . .	65
6.3	A simple example of graph $G$ . . . . .	67
6.4	A directed graph and an undirected graph . . . . .	68
6.5	Example: Overlapping and subsumption relation on the type level (SE = simple event, EO = event operator). . . . .	69
6.6	Example: Overlapping and subsumption on the instance level. . .	70
6.7	Example: Overlapping on the the property level. In this case the overlapping is a subsumption. . . . .	70
6.8	Best practice pattern on the instance level. . . . .	71
6.9	Example: Modelling context as input for the taxonomy-based simi- larity. . . . .	74
6.10	Example: Taxonomy of event type, event source and event operator ( $H_{et}, H_{es}, H_{eo}$ ). . . . .	74
6.11	Event pattern, $ep \in EPS$ , which will be used for the taxonomy- based similarity calculation. . . . .	74
6.12	The pattern modelling environment in PANTEON. . . . .	76
6.13	Graphical presentation of events, event operators and complex events.	76
6.14	User interaction in PANTEON. . . . .	78
6.15	Definition of the modelling context and the relationship selection.	78
6.16	Presentation of the results as pattern icons. . . . .	78
6.17	Selection of the relevant event pattern from the result list. . . . .	78
6.18	Extension of the initial modelling context with additional event nodes.	79
6.19	The architecture of pattern relation detection. . . . .	79
7.1	Black-box vs. white-box event pattern matching. . . . .	83
7.2	Extension of the event pattern graph with additional analyzer nodes in order to receive pattern execution data. . . . .	84
7.3	Overview of the evolution approach. . . . .	84
7.4	Example: Pattern session. . . . .	85
7.5	Example: Monitoring window. . . . .	86
7.6	Example: Evolution time window. . . . .	86
7.7	Pattern frequency cluster with centroid and radius. . . . .	87
7.8	Extended usual pattern frequency cluster. . . . .	88

---

7.9	Absolute usual pattern frequency cluster. . . . .	88
7.10	Unusual pattern execution detection. . . . .	89
7.11	The process of evolution adaptation. . . . .	91
7.12	Overview of the detected evolution candidates with highlighting (red bordered event) the events that might lead to the detection. .	92
7.13	Detailed view of the detected evolution candidate. . . . .	92
7.14	Additional evolution information in order to adapt the event pattern.	93
8.1	Overview of the SUS evaluation [TuSt04]. . . . .	98
8.2	SUS questionnaire [Broo96]. . . . .	99
8.3	Selected results regarding the user interface. . . . .	101
8.4	The process of the final evaluation. . . . .	106
8.5	SU results for the three settings and tasks. . . . .	106
8.6	An example of an event patten including 20 nodes. . . . .	111



# List of Tables

8.1	Task completion time for task 1. . . . .	107
8.2	Failure rate for task 1. . . . .	107
8.3	Task completion time for task 2. . . . .	108
8.4	Failure rate for task 2. . . . .	108
8.5	Task completion time for task 3. . . . .	108
8.6	Transformation and storage of event patterns. . . . .	110
8.7	Response time for the event pattern graph relationship detection (in milliseconds). . . . .	111









# **Part I**

## **Foundation**



# 1

## Introduction

In this chapter we describe the main motivation and the original research contribution of this thesis. Starting with the necessity for the real-time<sup>1</sup> data processing we outline the importance of the event pattern management. We describe aspects that may raise problems when using a Complex Event Processing system without an adequate event pattern management. Further we briefly describe projects that accompanied the research presented in this thesis. We conclude this chapter with the list of publications where the main contribution of this thesis has been published.

### 1.1 Motivation

The speed at which data is being produced in and around enterprises has been increased tremendously in recent years [ChSc09]. Business processes, customer opinion, press coverage, to mention a few areas, produce a large amount of data. Without effective techniques for collecting, analysing, correlating the data the enterprises will end up in an information overload where relevant and irrelevant data are mixed together [Lund06]. Especially in order to guarantee the situational awareness and an effective operational intelligence it is necessary to have mechanisms to detect relevant business situations [Luck12]. Hence, ensuring effective

---

<sup>1</sup>In Complex Event Processing real-time is equivalent to low-latency. Near real-time or business real-time are also used synonyms.

responses to critical business situations decrease the gap between knowing the state of internal operations and the enterprise environment.

The need for the real-time data processing is increasing, since not only the content of the data but also its real-time context<sup>2</sup>, is determining the value of data in order to gain immediate insights [TSSG12]. The real-time context demands an increase in the responsiveness of a system meaning that a system must react to relevant business situations faster [ChEA11], [ChSc09], [Luck01], [EtNi10]. For instance algorithmic trading systems act within milliseconds without human intervention. Also, since enterprises nowadays are influenced by many different and dynamically changing factors, like multiple business processes, customer requests, competitors and shorter product releases, data in isolation has little value. The value of data derives from a more intelligent combination to relevant business situations and their detection in real-time. However, since not every combination is a useful one, the data combination process must be designed and performed in an effective and efficient way. With this rising importance of timely operational intelligence, enterprises have adopted Complex Event Processing (CEP) as their backbone [ChSc09]. Ultimately, event processing<sup>3</sup> is the key for a real-time operational intelligence.

Complex Event Processing is the analysis of events from different event sources in real-time to generate immediate insights and responses to changing business conditions [Luck01]. The main strength of CEP is to derive high level events from a set of input events. Event pattern matching is the most powerful capability of CEP [EtNi10]. Event patterns encode the knowledge for a relevant business situation that is of interest. A relevant business situation can be the detection of bottlenecks, process failure or customer dissatisfaction. Luckham [Luck01] considers the event patterns as the foundation of CEP systems. From the business perspective event patterns represent the most valuable asset in CEP. Event patterns are used in order to derive high level complex events describing a relevant business situation.

While nowadays CEP systems provide an efficient and effective infrastructure to detect event patterns, the management of event patterns is still in its infancy. Compared to other IT-systems, CEP systems still lack in support of tools and methods allowing users to configure a system easily or to refactor services and components [RoSS07]. Popular CEP vendors like StreamBase<sup>4</sup>, Tibco<sup>5</sup>, SAP<sup>6</sup>, Microsoft<sup>7</sup>, or Progress<sup>8</sup> and academic CEP research (see [PiBa02], [ASSA<sup>+</sup>99], [AdEt02], [ACcC<sup>+</sup>03], [BDGH<sup>+</sup>07] and [ABBC<sup>+</sup>04], [AFRS11]) are mainly focused on the efficient matching of a high volume of events in a short time period and disregard the role of supporting software tools for event pattern management.

---

<sup>2</sup>Context in this case describes the cooccurrence of data.

<sup>3</sup>In this thesis Complex Event Processing and event processing are used synonymously.

<sup>4</sup><http://www.streambase.com>

<sup>5</sup><http://www.tibco.com/products/event-processing/complex-event-processing/default.jsp>

<sup>6</sup><http://www.sap.com/solutions/technology/database/complex-event-processing/index.epx>

<sup>7</sup><http://msdn.microsoft.com/en-us/library/ee362541.aspx>

<sup>8</sup><http://www.progress.com/en/Product-Capabilities/complex-event-processing.html>

Although most of the commercial solutions provide a graphical environment to create and modify event patterns they are not based on a fundamental methodology, methods and techniques to cover the life cycle of an event pattern. A methodology in general is a set of complementary methods and a clear description how to apply them [KlHi01], [ArNH86]. It is a guideline for achieving a global objective using components such as phases, tasks, methods and tools. Having a methodology for the event pattern life cycle can increase the efficiency and effectiveness of the event pattern management especially since the need for the real-time situation detection is increasing. In the software development domain methodologies have been used to increase the productivity and quality of software development and decrease the time and effort [HaKS00], [HCRS<sup>+</sup>94], [DyMA05], [RiHD02]. In [OSSK<sup>+</sup>11], [Kulk11], [ChEA11], [EtNi10], [Luck12] the authors stress the importance of methodologies and tool support for improving the usability of CEP systems. The authors further consider the role of user interfaces and abstraction levels as an important factor to remove the barriers from accessing a CEP system and to increase the efficiency and the effectiveness of the pattern engineer<sup>9 10</sup>. An efficient event pattern management in general should include a set of methods for dealing with event pattern modelling, formalization, refinement, deployment and adaptation to changing business situations. For example during the modelling of a new pattern the pattern engineer can be provided with the information that there exist similar patterns or an equal event pattern that could be reused. Another example could be that the pattern engineer is notified about an unusual behaviour of an event pattern based on the pattern execution history. These kinds of support would increase the efficiency and effectiveness of the pattern engineer and help to prevent an incorrect or delayed modelling or detection of relevant business situations.

In the next section we describe several aspects that cause problems when a pattern engineer currently manages event patterns.

## 1.2 Tackling event patterns

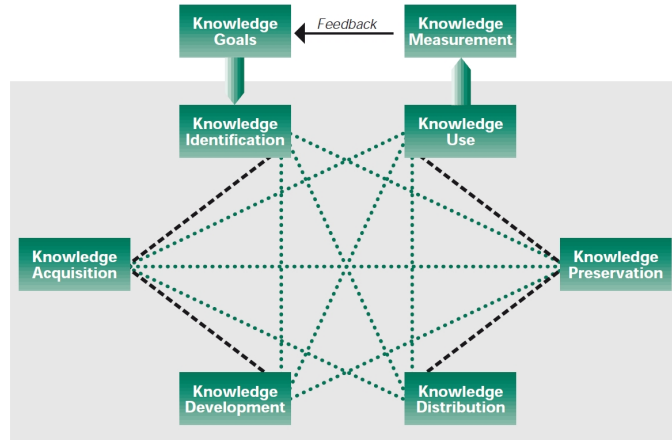
The value of event patterns arises from their natural occurrence and their real need for the daily business. A pattern encompasses the codified information within an enterprise and hence forms a knowledge artefact (see also [Snow04]). A knowledge artefact is created for a practical purpose. A knowledge artefact might be a document, process, source code, an engineering schematic or a template for a proposal and so forth [Snow04]. Figure 1.1 displays the building blocks of knowledge management as described in [Prob98]. The cycle consists of knowledge identification, acquisition, development, distribution, preservation, use and measurement.

---

<sup>9</sup>In Chapter 3 we describe the need for the event pattern management considering different research and application areas in more detail.

<sup>10</sup>A pattern engineer is a person who is responsible for the modelling, deployment and evolution of event patterns. It can be a business user but also a technician.

The feedback cycle clarifies the importance of the knowledge measuring to ensure that the focus is on achieving the knowledge goal. The knowledge goal is the transformation of the accumulated knowledge into a corporate asset.



**Figure 1.1:** The building blocks of knowledge management according to [Prob98].

Considering the current event pattern management from the knowledge management perspective presented in Figure 1.1 we present below issues decreasing the efficiency and effectiveness of a pattern engineer. The overall goal of having an efficient and effective event pattern management is to model the most suitable event patterns for detecting the relevant business situation by keeping the time period that is needed for the modelling of the relevant business situations short.

- Event pattern identification and acquisition** Most of today's knowledge needed to model an event pattern is distributed across the enterprise. It is either part of existing event patterns, hidden in the business workflows or occurred data or just in peoples' heads. The identification of this knowledge and the formalization of the resulting event patterns is a challenge. There exist neither methodologies nor tools supporting the event pattern identification and acquisition. The non-existence of methodologies and tools lead to the event pattern acquisition problem similar to the knowledge acquisition problem in the knowledge management (see also [Wagn07], [Wagn00], [Wate86]). In order to define a new event pattern it is indispensable to have an approach to consider systematically different sources in order to identify and acquire event patterns for a given business situation. Possible sources could be for example the knowledge in people's head, historical event data, historical event pattern data and their execution, existing workflows or business processes. The partially acquired event pattern knowledge needs to be combined and refined which should be done while modelling the pattern.
- Event pattern modelling and distribution:** Like in knowledge management it is important to transfer the event pattern knowledge between those who have it and those who do not. Only explicit event pattern knowledge<sup>11</sup>

<sup>11</sup>Event pattern knowledge that has been acquired and formalized.



can be stored and distributed across enterprises and individuals for a later reuse (see also [Mark01]). Reusing knowledge includes both recall and recognition. While recall describes the modelling and storage of the information the aim of the recognition is whether the information meets users' needs. Therefore, the formalized event patterns should be shared in order to speed up the modelling and refinement of event patterns by reusing the existing event patterns. Moreover, since event patterns are the most valuable asset in CEP systems their shareability could support the efficiency and the effectiveness aspect of the event pattern management.

- **Event pattern evolution and adaptation** The knowledge that has been defined can be either inaccurate or change over time. It is a known issue that pattern engineers make mistakes [Wagn07]. Furthermore, maintenance can introduce inaccuracies or inconsistencies into previously correct event patterns. While it is reasonable to expect that a pattern engineer will be able to provide an event pattern specification partially, providing all the required details is a hard task, even for pattern engineers who are domain experts [TuGW09]. Moreover, in many active systems, event patterns may change over time, due to the dynamic nature of the domain. Such changes complicate even further the specification task, as the pattern engineer must constantly update the patterns. Since CEP systems are designed for real-time environments, outdated patterns or a delay in a pattern update may lead to an increased downtime and possible ignorance of real problems.

## 1.3 Research contribution

CEP is rather a young research area and most of the research activities are focused on the matching algorithms and their optimization. However, in order to cope with the evolving nature of business environments having methodologies, tools and methods for event pattern management are identified as being promising [EtNi10],[Luck12]. It is known that the management of event patterns is not an easy task and that pattern engineers seek additional support to the definition of event patterns, beyond expert opinion [TuGW09].

Below we present the contribution of this thesis and refer to the chapters where this contribution is described in more detail. The main contribution of this thesis centers on the idea to leverage pattern engineer's strengths by providing her/him additional support during the whole event pattern management process.

- **Event pattern life cycle** The first step towards an efficient and effective pattern management is having the understanding of the event pattern life cycle. Similar to the knowledge management building blocks that form a cycle we define the event pattern life cycle methodology as a process covering the modelling, refinement, execution and evolution of event patterns. The

life cycle methodology consists of three main phases that cover both design-time issues and run-time issues. Each phase again contains a set of tasks. These phases form a feedback loop enabling continual flow of information collected in the entire life cycle of an event pattern. This contribution is described in Chapter 4.

- **Event pattern modelling and distribution** In order to model event patterns, we provide a high level graphical user interface to hide the underlying complexity and enable also pattern engineers that are non-technicians to model, deploy and evolve event patterns. The modelling of event patterns is based on refinement and reuse of event patterns. Refining and reusing existing knowledge is an essential part of knowledge management. In order to enable a pattern engineer to reuse and refine event patterns we introduce two approaches for detecting pattern relations. The graph-based approach aims at finding event pattern overlapping and event pattern extensions. Additionally the similarity-based approach delivers the most similar event pattern taking into account the concept hierarchy and feature definition of event patterns. These approaches are semi-automatic meaning that the results will be revised by the pattern engineer. Additionally, in order to support the modelling and distribution of event patterns we introduce a meta model describing the concepts of an event pattern. These contributions are described in Chapter 5 and Chapter 6.

- **Event pattern evolution and adaptation**

Current CEP engines are designed as a black-box that match a pattern against incoming events. The engine creates a complex event only in cases in which the whole pattern expression is evaluated true. We consider the execution of an event pattern as an essential part of the event pattern life cycle. Therefore we describe a white-box approach where the execution of an event pattern is monitored. We introduce a set of requirements and define a lightweight approach how an existing CEP engine could be extended with the monitoring feature. In this thesis the white-box CEP provides the data that will be used to evolve event patterns. This contribution is described in Chapter 7.

Event patterns are subject to change. Nowadays, the evolution of patterns is done by business experts fully manually which is the usual case when business conditions change or new requirements arise. In this thesis we extend the evolution of event patterns with the concepts of execution-based evolution. Based on pattern execution monitoring we determine the execution statistics and compare current pattern execution statistics with past execution statistics. To goal is to detect deviations in the event pattern execution. This approach supports the pattern engineer in keeping the event pattern repository up-to-date (continued relevance of an event pattern). This contribution is also described in Chapter 7.

## 1.4 Project context and publications

The methodology, methods and tools described in this thesis have been developed mostly in different projects during the time period of August 2008 until May 2012. Below we describe briefly the projects that accompanied the development of the approaches described in this thesis.

- **STIBA (Semantic Technologies in Business Applications), 11-2007 till 12-2008** The main goal of the STIBA project (in cooperation with living-e AG that has been acquired by Empolis GmbH in 2010) was to apply semantic technologies to real world scenarios. In the context of the STIBA project we developed the initial idea for event pattern similarity. We implemented and tested in the context of the STIBA project the similarity metrics which will be presented in this thesis.

### Publications

- **Sinan Sen, Slavko Tomcic. An Enterprise Knowledge Management Platform on top of the Ontology-based Similarity.** *LWA, Band 448 der Technical Report. Department of Computer Science, University of Würzburg, Germany, 2008.* (see [SeTo08])
- **Sinan Sen and Jun Ma. Contextualised Event-driven Prediction with Ontology-based Similarity.** *AAAI Spring Symposium: Intelligent Event Processing, March 23-25, 2009, Stanford, California, USA.* (see [SeMa09])
- **IDEO, 09-2008 till 04-2009** The goal of the IDEO project (in cooperation with Union Investment IT in Frankfurt) was to analyse the CEP landscape and to implement a prototype in order to connect different events from the IT infrastructure and to detect relevant business situations. In this project we made the main experience with business users that the management of event patterns is of paramount importance. Additionally we found out that existing solutions and approaches do not satisfy their needs since these approaches are not considered as being usable. In this project we defined the main pillars of this research based on the tool evaluation results, discussion with the business users and a workshop we organized at the Union Investment IT in February 2009.

### Publications

- **Sinan Sen, Nenad Stojanovic, Ruofeng Lin. A Gaphical Editor For Complex Event Pattern Generation.** *The 3rd ACM International Conference on Distributed Event-Based System (DEBS), July 6-9, 2009, Nashville, Tennessee, USA.* (see [SeSL09])

- **Sinan Sen**, Nenad Stojanovic. **GRUVe - A Methodology for Complex Event Pattern Life Cycle Management**. *22nd International Conference on Advanced information Systems Engineering (CAiSE)*, June 09-11, 2010, Hammamet, Tunisia. (see [SeSt10])
- **VIDI (Visualising the Impact of the legislation by analysing public DIscussions using statistical means), 01-2009 till 12-2010** The VIDI project (funded by the European Commission) had the goal to provide a set of visualisation techniques in order to understand public discussions for the policy development and to receive real-time notifications about certain word patterns in public discussions. In this project we mainly developed the modelling of event patterns and the extension of the CEP engine in order to monitor pattern execution statistics.

### Publications

- **Sinan Sen**, Nenad Stojanovic, Ljiljana Stojanovic. **An Approach for Iterative Complex Event Pattern Recommendation**. *The 4th ACM International Conference on Distributed Event-Based System (DEBS)*, June 20-24, 2010, Cambridge, UK. (see [SeSS10b])
- **Sinan Sen**, Nenad Stojanovic, Bijan Fahimi Shemrani. **EchoPAT: A System for Real-time Complex Event Pattern Monitoring**. *The 4th ACM International Conference on Distributed Event-Based System (DEBS)*, June 20-24, 2010, Cambridge, UK. (see [SeSS10a])
- Mitja Trampus, Marco Grobelnik, **Sinan Sen**, Nenad Stojanovic. **Visualisation of Online Discussion Forums**. In *Yannis Charalabidis and Sotirios Koussouris (eds.), Empowering Open and Collaborative Governance*, Springer-Verlag Berlin Heidelberg, 2012. (see [TSSG12])
- **ALERT (Active support and reaL-time coordination based on Event pRocessing in FLOSS development, 10-2010 till 04-2013)** The ALERT project (funded by the European Commission) has the goal to improve the bug resolution process in Open Source developers' collaborative environments. In ALERT we refined our event pattern life cycle methodology. The main work done in ALERT was related to the evolution of event patterns based on event pattern execution statistics. Additionally we evaluated within the ALERT setting parts of our system with business users.

### Publications

- Nenad Stojanovic, Ljiljana Stojanovic, Darko Anicic, Jun Ma, **Sinan Sen**, Roland Stühmer. **Semantic Complex Event Reasoning - Beyond Complex Event Processing**. In *Dieter Fensel (ed.), Foun-*

*dations for the Web of Information and Services. Springer, Berlin-Heidelberg, 2011. (see [SSAM<sup>+</sup>11])*

- **Sinan Sen**, Ruofeng Lin, Bijan Fahimi Shemrani. **Complex Event Pattern Evolution based on Real-Time Statistics.** *The 5th ACM International Conference on Distributed Event-Based System (DEBS), July 11-15, 2011, New York, NY, USA. (see [SeLS11])*

## 1.5 Reader's guide

The thesis is organized into three parts.

In *Part I* we introduce the main motivation behind this thesis and give an overview of event processing. We present the main building blocks and present the concept of event patterns.

In *Part II* we describe the main contribution of this thesis related to the event pattern management. In *Chapter 3* we present the need for event pattern management. In *Chapter 4* we describe the event pattern management life cycle. In *Chapter 5* we introduce the meta model for the event pattern representation. In *Chapter 6* we specify the refinement and reuse of event patterns during the generation phase. In *Chapter 7* we describe our approach for evolving event patterns based on pattern execution statistics.

In *Part III* we present the evaluation results and give a conclusion of the work presented in this thesis which is followed by an outlook for future development.



# 2

## Complex Event Processing

In this chapter we describe the fundamental definitions and terminologies behind Complex Event Processing (CEP). We explain the main building blocks of event processing <sup>1</sup> and describe the concept of event patterns in more detail.

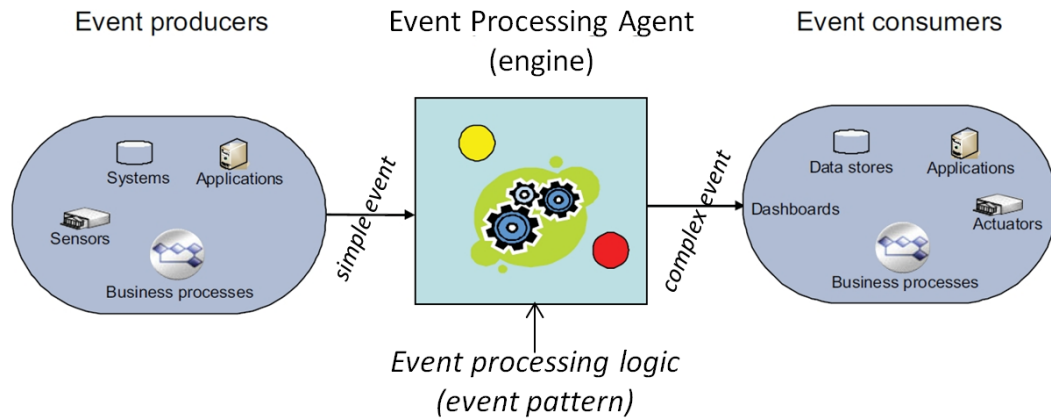
### 2.1 Event processing architecture

Complex Event Processing is about performing operations on events, including reading, creating, transforming, abstracting or discarding them [EtNi10], [Luck01]. It is based on the analysis of events from different event sources in real-time in order to generate immediate insight and enable immediate response to changing business conditions. As a research topic, CEP emerged in the late 90's and has already become the foundation of modern real-time information technology systems. Its origin is located in the area of discrete event simulation, weather simulation, networks and internet. CEP provides a set of techniques and methods to analyse and understand event-driven systems [Luck01]. The key idea is to explore temporal, causal, and aggregation relationships among events to make sense of them in a timely fashion [ChEA11]. Since the occurrence of events is often not foreseeable, it is necessary to ensure that the response latency is low [ChSc09].

In contrast to the traditional information systems, that work in a synchronous request and response manner, event processing is built on top of the asynchronous

---

<sup>1</sup>In this thesis event processing and CEP are used synonymously.



**Figure 2.1:** The structure of an event processing application according to [EtNi10].

communication based on events. An asynchronous communication enables a data-driven and real-time reaction to occurring events instead of posing endless requests. Messages are sent by the event producer without waiting for the response. CEP can be used for observations, dynamic operational behaviour or information dissemination to name a few areas. Figure 2.1 displays the separation of event processing logic from the event producers and event consumers that will be described below. Before going into details and explaining these building blocks, we describe the concept of events in the context of CEP.

### 2.1.1 Events

In event processing an event is an occurrence of something within a particular domain or system [EtNi10]. It is something that has happened, or is contemplated as having happened in that domain. Chandy et. al. [ChCC07] define an event as a significant change in the state of universe. In this sense every piece of new data can be considered as an event since it changes the state of universe. Events which require a reaction are forming a situation [AdEt02]. Not only the occurrence of something denotes an event but also the absence of expected events conveys information [ChSc09].

The concept of an event is also used in a broader sense to describe a programming entity (for example a class in a programming language) that represents the occurrence or non-occurrence of an activity in a system. Luckham [Luck01] defines an event as an object that can be processed by computers and should satisfy the following three aspects:

- **Form** Every occurred event is represented as an object with particular attributes. A form can be a string or a tuple of data components including, the time stamp, the source and additional attributes.
- **Significance** Every event signifies an activity. The activity is called as the significance of the event. The form of the event contains the attributes that describe the activity it signifies.



- **Relativity** Events are related to other events by time, causality and aggregation. The set of relationships between an event and other events are called as the relativity of the event.

**Example** *An input event signifies the activity of a NewMail message. The message would be in some format such as XML, containing data fields for the subject, sender, content, and so on. The event's form would be similar to the message but would contain extra data fields, for example giving its time of generation, time of arrival, and relation to other events. The form of an input event would be an object of a Java class called MailEvent as shown below:*

```
Class MailEvent{Name NewMail;
String id;
Person sender;
Person receiver;
String content;
Time T;
Causality (Id1, Id2, ...);
...
}
```

Every event belongs to a class of events, which specifies a common structure for all members of this class. This specification is called event type. Every event with the same event type has the same structure and semantics. The value of its attributes will be used for the filtering, aggregation or pattern detection to name a few operations. Further an event can be either simple or complex [ChEA11]. A simple event is atomic and does not contain further events. In comparison to a simple event, a complex event is composed of other simple or complex events. In [ChEA11] complex events are also described as summary-level facts.

Luckham [Luck01] describes time, causality and aggregation as the most important common relationships between events. These relationships are called partial ordering rather than total ordering since there are events where none of the relationships can be applied. The time aspect is used to order events. This type of relationship depends upon a system clock. A timestamp is assigned to each event when it is created. The order of the events defines their relationship to each other. An event can have more than one timestamp if it is a complex event. For example if the event *A* always happens before the event *B* then there is a causal relationship between *A* and *B* which is *A* caused *B*. In this scope causality is defined as a dependence relationship between events in a system. If there is no causal relationship then the events are independent.

The aggregation is an abstraction relationship. An event *A* is created when a set of other events *B<sub>i</sub>* happens. The event *A* is a complex event since it occurs over a time interval where a set of other events occurred before.

### 2.1.2 Event producer

In event-driven environments there is at least one event producer that creates and publishes events. Often event producers are called sensors which might be hardware or software sensor. An event producer creates events according to certain event reporting logic which is embedded in the event producer. An event producer can produce at least one type of events. [EtNi10] summarizes event producers into following categories:

- **Hardware as an event producer** This kind of event producers are sensors that generate events that indicate a certain aspect of the physical environment where it is integrated. Motion or temperature sensors are an instance therefore.
- **Software as an event producer** Every application can be designed as part of an event-driven system such that it continuously produces events. Beside applications which produce events there are instrumentation techniques where software agents observe applications and report the state of the application.
- **Human interaction as an event producer** In several situations a human interaction generates directly an event. For example to post a tweet or to make a phone call triggers the creation of an event by an underlying software component.

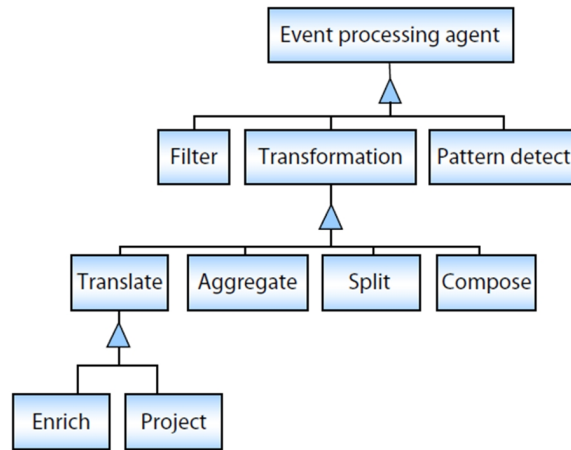
Luckham [Luck01] describes event sources on a more abstract level and classifies into:

- **IT Layer** Events from this source are based on the observation of the communication between software components in the IT landscape. The IT layer contains several components such as the middleware, database, web service calls etc.
- **Instrumentation** Components of the target system are equipped with sensors in order to create events as a side effect of the normal behaviour.
- **CEP** Complex Event Processing engine serves also as an event source. It creates new complex events based on the provided event processing logic.

The next step in the processing of events takes place within the event processing agent.

### 2.1.3 Event processing agent

The detection of a situation is done in the event processing agent (EPA) by applying the event processing logic. The EPA is the place where the CEP engine is



**Figure 2.2:** Different type of event processing agents according to [EtNi10].

included. Figure 2.2 shows the hierarchy of the various EPA building block types described in [EtNi10].

- *Filter agent* filters out uninteresting events by applying a filtering operation. The operation is stateless meaning that the filtering is done solely based on the content of the event instance. An example would be to discard a transaction event if its value is greater than 50.
- *Pattern detection agent* detects the occurrence of a pattern over a set of events. The definition of a pattern is based on event relations like temporal, spatial or semantic relations. Pattern detection is the most powerful capability of an event processing system. An example would be to detect a credit card fraud based on plenty low level events.
- *Transformation agent* modifies the content of the received event object.

Transformation agents are further classified based on the cardinality of their inputs and outputs.

- *Translate agent* translates an incoming event to an output event based on a translation operation.
- *Aggregate agent* takes a set of incoming events from an event stream and produces an event by applying an aggregation function.
- *Split agent* takes a single incoming event and emits at least two or more event objects.
- *Compose agent* is similar to the join operation in relational algebra. It takes two streams of events and produce a single stream of output events.

The translation agent is further classified in one of the following agents.

- *Enrich agent* extends an incoming event with additional information.
- *Project agent* deletes information from an incoming event.

### 2.1.4 Event consumer

In every event-driven application there are one or more event consumers. The consumer receives the events from the EPA and triggers certain type of actions. The event consumer encodes the reactive behaviour of the system. An event consumer can have plenty of different functionalities like storing, visualising or forwarding of events. An event consumer can consume either simple events that are atomic or complex events, that are generated as a result of the event processing that takes place in the EPA. Etzion and Niblet [EtNi10] classify event consumer into three categories:

- **Hardware event consumer** Hardware that consumes events is generally called actuator and is the counterpart of the hardware sensor. An actuator performs a physical action based on events like the locking or unlocking of a door.
- **Human interaction** Events can be consumed also by humans. For example someone could be alerted about a critical situation in order to react on it. Usually there are also user interfaces that visualize the simple or complex events in different formats.
- **Software event consumer** This kind of applications do not provide an explicit user interface for the humans. Rather events trigger certain workflows or create new instances of certain business processes.

## 2.2 Event patterns

In event processing event patterns are abstracted as event processing logic. In event processing the most powerful mechanism is the detection of patterns over events [EtNi10], [Luck01], [ChEA11]. An event pattern is a template that matches a set of events (simple or complex). Each match of a pattern is a partially ordered set constructed by replacing the variables in the event pattern with actual values [Luck01]. Chandy and Schulte [ChSc09] describe the role of event patterns as "connecting the dots". Event patterns are defined in a declarative way. Declarative means that the language describes what needs to be done and not how it is actually done. Event patterns are used to create complex events that signify a set of events. Further, event patterns provide an abstraction of relevant business situations. The same event pattern can trigger continuously complex events of the same type.

A pattern describes the occurrence context of a set of events. This context contains the event parameter each event have and the relationship between the set of events using event operators. Event patterns are described in languages called event pattern language (EPL).

The pattern below is presented in the tabulator format for the sake of readability.

**Example** (taken from [Luck01])

Variables: Subject S, Message M, String ID, Time T, Time T1, Time T2  
 Event Types: Publish(Subject S, String Id, Message M, Time T)  
 Receive(Subject S, String Id, Message M, Time T)  
 Event Operators: and  
 Pattern: Publish (S, Id, M, T1) and Receive(S,Id,M,T2)  
 Context test: T2-T1<35 mins and S=''StockTrade''

The pattern of type *Publish (S,Id,M,T1) and Receive(S,Id,M,T2)* above matches any pair of events that have the same *Id*, message *M*, subject *S* which is *StockTrade* and the time difference between publishing and receiving is less than 35 minutes.

An event pattern enables to pick out the interesting events from a large number of events. This is fundamental to viewing and controlling an event-driven system [Luck01]. Backtracking from a complex event to its members in CEP is called drill down [Luck01].

[Luck01] defines the following requirements for EPLs.

- **Power of expression** The expressivity must be powerful enough in order to cover different application domains. The expressivity of an EPL is mostly dominated by the event operators it supports. Chakravarthy and Mishra [ChMi94], Anicic et. al. [AFRS11] and others introduced a set of event operators including logical and temporal operator for event processing purposes.
- **Notational simplicity** The definition of event patterns must be easy and succinct.
- **Precise semantics** The language must be a mathematically precise concept of match meaning that the set of events that can match a pattern are known.
- **Scalable pattern matching** The design of the underlying language should not have negative effects on the performance of the pattern matching process itself meaning that the matching should be scalable enough to match a large number of event patterns over high volumes of events in real time.

These requirements are somehow concurring. For example the expressivity of a pattern language is concurring with the ease of use, simplicity of the pattern language and efficiency of pattern matching. Further a scalable pattern matching inevitably influence the language design. Luckham [Luck01] describes this as follows:

If an EPL is simple and easy to use, we won't be able to specify some kinds of complex patterns in it. On the other hand, if it is powerful

and let us specify complex patterns, it will contain "advanced" features or options that take time to learn how to use. And pattern matching for complex patterns is computationally demanding and difficult to implement efficiently.

In order to ease the definition of event patterns [Luck01] defines pattern macros (PM) as an abstraction feature for event patterns. These macros are used in order to abstract commonly used patterns and to name them. The definition of a PM can be for example (taken from [Luck01])

pattern PM (*parameter-list*) (*p-pattern*)

where PM is the name of the macro and it names the *p-pattern* which is also called the body of the macro. The way macros intended to be used is that they are called in various patterns with a *parameter-list* like *...PM(actual parameter list)...* Abstraction features like the pattern macros are useful in order to abstract commonly used patterns, to build up libraries of patterns for each application [Luck01].

Event patterns that trigger an action are called event pattern rules [Luck01]. In this sense the event pattern rule defines a causal relationship between the events that match the pattern and the events that are created as the result of the matching. In STRAW-EPL<sup>2</sup> [Luck01] for example a rule consists of a trigger, which is an event pattern, and an action, which is an event that is created whenever the pattern matching is true.

In the example below the event pattern from the previous example is extended with an action that creates a complex event to inform the sender that the message has been received successfully. The event which will be created is called *MessageReceived* and has two parameters, namely the *Id* of the message and the timestamp *T2* when the message was received.

### Example

```
Variables: Subject S, Message M, String ID, Time T, Time T1, Time T2
Event Types: Publish(Subject S, String Id, Message M, Time T)
Receive(Subject S, String Id, Message M, Time T)
Event Operators: and
Pattern: Publish (S, Id, M, T1) and Receive(S,Id,M,T2)
Context test: T2-T1<35 mins and S='StockTrade'
Action: create MessageReceived(Id,T2)
```

Additional meta information for already defined event patterns provides an additional abstraction level and could be useful in various ways, like for the ease of readability, clustering of pattern, or the definition of pattern libraries.

The classification of event patterns below is taken from [EtNi10].

---

<sup>2</sup>Strawman pattern language

## 2.2.1 Basic patterns

Basic event patterns are frequently used in event processing applications and they don't depend on the timing or ordering of the incoming event.

### 2.2.1.1 Logical operator pattern

In event processing a logical operator (sometimes also called logical connective) is a symbol used to connect two or more events. Event patterns that use logical operators belong to the most frequent used patterns in CEP.

**Conjunction operator pattern (AND)** The conjunction operator is used to look for subsets of incoming events containing all events that are defined in the conjunction pattern. The pattern is detected when the incoming events contain at least one instance that satisfies the events in the event list of the conjunction pattern. The order of the incoming events is immaterial.

#### Example

(EventA AND EventB)

In the example above the pattern is fulfilled whenever *EventA* and *EventB* are in the matching set.

**Disjunction operator pattern (OR)** The disjunction operator looks for occurrences of any of the relevant events containing just one member. The pattern is satisfied if the incoming events contain an instance of any of the events in the event list of the disjunction pattern.

#### Example

(EventA OR EventB)

In the example above the pattern is fulfilled whenever *EventA* or *EventB* are in the matching set.

**Negation operator pattern (NOT)** The negation pattern (NOT) detects the absence of any events with certain specified characteristics. The matching set is in this case empty. This pattern can be used with temporal aspects to detect time outs.

#### Example

(EventA AND (NOT EventB))

In the example above the pattern is fulfilled whenever *EventA* is in the matching set and *EventB* not.

### 2.2.1.2 Threshold operator pattern

The threshold operator is based on an aggregation operation which is performed against the set of participant events. The aggregation value is compared against a threshold value. Threshold operator can be classified into following types:

- *Count* - The count operator counts the number of incoming event instances and checks this number using a threshold assertion. The assertion can be one of the relations  $=, \leq, \geq, <, >, \neq$
- *Average* - The average operator is satisfied when the average value of a specific attribute satisfies the value average threshold assertion. The assertion can be one of the relations  $=, \leq, \geq, <, >, \neq$
- *Minimum and maximum* - This type of operator is satisfied when the maximal or minimal value of a specific attribute over all incoming events satisfies the max or min threshold assertion. The assertion can be one of the relations  $=, \leq, \geq, <, >, \neq$

### 2.2.1.3 Subset operator pattern

Event patterns with the subset selection criteria are concerned with selecting a subset of events from a set of incoming events. [EtNi10] introduces the *n-highest* and *n-lowest*

- *n-highest* - Patterns using the *n-highest* operator deliver a set of output events that have the highest value for a specific attribute. If there are more than n events in the matching set then the result set will contain n events. If there are less than n events in the matching set then the result set will contain all these events.
- *n-lowest* - Patterns using the *n-lowest* operator deliver a set of output events that have the lowest value for a specific attribute. If there are more than n events in the matching set then the result set will contain n events. If there are less than n events in the matching set then the result set will contain all these events.

## 2.2.2 Dimensional pattern

Dimensional patterns are patterns that consider the time and space dimension. Patterns related to the time dimension are very often used while spatial patterns are not well researched in CEP [EtNi10].

In the context of this thesis we use the temporal order pattern. The *followed by* pattern or *sequence* pattern (denoted as SEQ) is a temporal order pattern in



which time plays the major role. The basic underlying assumption is that an event happens at a single point in time. The sequence pattern is similar to the conjunction pattern but is also considering the order of event occurrences. The pattern is satisfied whenever the incoming events occur in the same order like described in the event list. It is an ordered list of events. The temporal order is part of the order policy which should be supported by the underlying CEP engine (for more details about the pattern and order policies see [EtNi10])

### Example

EventA SEQ EventB.

In the example above the pattern is fulfilled whenever *EventB* occurs after *EventA* has occurred.



## **Part II**

# **Efficient and Effective Event Pattern Management**



# 3

## The Need for Event Pattern Management

In this chapter we describe why having an efficient and effective event pattern management is essential for the enterprises. We deduce the necessity of the event pattern management based on the analysis of academic research, by considering existing CEP solutions and by analysing the most related research areas like active databases and business rules.

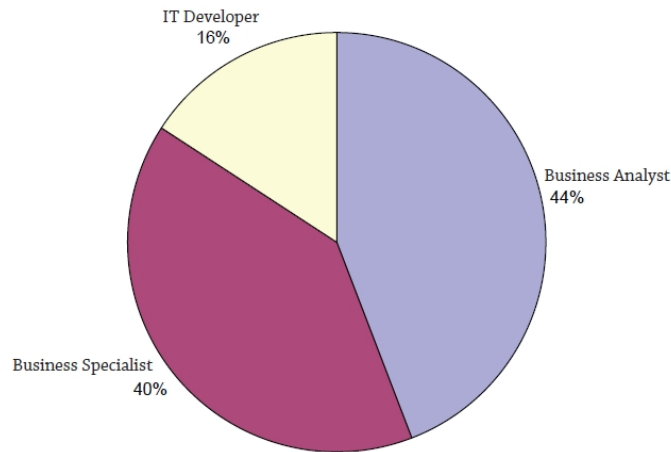
### 3.1 Why event pattern management matters?

The current generation of event processing tools is rather programming oriented where fundamental programming skills are needed to develop event patterns [EtNi10]. However, there is a demand towards easy to use tools allowing users, who might not have deep programming skills, to define, deploy and maintain event patterns. Figure 3.1 displays a customer survey<sup>1</sup> from 2007 conducted by ebizQ<sup>2</sup> that indicates that most of the customers surveyed would like to have event pattern defined by business analysts or business specialists. Although the survey might be outdated its main findings are still valid. Business analysts and business specialists who will be the pattern engineers need proper tool support including the recognition of duplicate event patterns, recommendation for event patterns and support during the pattern modelling and pattern evolution.

---

<sup>1</sup><http://www.complexevents.com/2007/10/30/event-processing-market-pulse-2007/> - The survey was based on 400 responses to the survey from 33 industries.

<sup>2</sup><http://www.ebizq.net/>



**Figure 3.1:** Customer survey taken from ebizQ.

Luckham [Luck01] outlines the importance of supporting tools and define the need for event pattern management. The management procedure in this case must detect relationships between the new event pattern and the existing one in order to detect duplicates, to decide which ones are sent to the CEP engine and which ones are deleted. Further he describes the role of analysis tools for the CEP infrastructure. These tools are consumers of information provided by the CEP infrastructure and produce human-readable information. The purpose is to use all the information a CEP system can deliver to help to find out what is happening in the system and why. In 2007 Luckham described<sup>3</sup> the problem with event patterns in CEP systems. This time the focus was on pattern engineers. Especially when the number of event patterns increases their management is a challenge independently from whether the pattern engineer is a business user or a technician. The problem arises since the languages in which an event pattern is written are incomprehensible. The result is that it is hard to read and understand the event patterns. Therefore he proposes a high-level pattern language that can be used to express rules succinctly in a way that makes them understandable by also non-technician pattern engineers.

The following example demonstrates the complexity of an event pattern. This example joins 2 event streams using the Esper EPL language<sup>4</sup>. The first event stream consists of fraud warning events for which we keep the last 30 minutes (1800 seconds). The second stream is withdrawal events for which we consider the last 30 seconds. The streams are joined on the account number.

#### Example:

```
select fraud.accountNumber as acctNum, fraud.warning as warn,
withdraw.amount as amount, MAX(fraud.timestamp,
withdraw.timestamp) as timestamp, 'withdrawlFraud' as desc
```

<sup>3</sup>What is the difference between ESP and CEP?

<sup>4</sup>Esper open source event engine -<http://esper.codehaus.org>

```
from FraudWarningEvent.win:time(30 min) as fraud,  
    WithdrawalEvent.win:time(30 sec) as withdraw  
where fraud.accountNumber = withdraw.accountNumber
```

The syntax of the Esper EPL is based on SQL and without fundamental knowledge in SQL the event pattern is not easy to read and to understand. Nowadays enterprises that have event processing systems have plenty of event patterns in order to detect the relevant business situations in real-time. It is quite obvious that the definition, deployment and maintenance of these event patterns can be challenging, time consuming and error-prone if the management of event patterns is done on the event pattern language level. Etzion and Niblet [EtNi10] consider the management issues in event processing rather from a software engineering point of view. Etzion and Niblet describe the need for more software engineering methodologies and tool support. Further Etzion and Niblet outline the definition of design patterns and collection of best practices for implementing an event processing system.

In the event processing manifesto (see [ChEA11]) written by the participants of the 2010 Dagstuhl Seminar on event processing the following non-functional requirements of an EP system are described:

- **Usability** - The tools must provide support for several kinds of users (IT experts, domain experts) who will be defining and maintaining the event patterns. There are IT professional, engineers, medical staff, business managers to name a few.
- **Versatility** - The tools must allow contributions from multiple stakeholders that may have different backgrounds.
- **Expressiveness** - The language that the tools provide must allow the development of complex applications.
- **Maintainability** - Event processing platforms must be manageable meaning that new event patterns can be added or removed. The running system with its processing logic must be monitored to help diagnose and fix problems.

These requirements reveals the importance usable tools that allow pattern engineer with different background to maintain event patterns. As next we consider two related research areas in order to support the need for the event pattern management.

### 3.1.1 Academic research on event pattern management

Although there are many successful event processing applications, most event processing approaches are focused on the extreme processing of a large number

of events and neglect the event pattern management. Below we present current approaches dealing with some aspects of the event pattern management.

Turchin et. al [TuGW09] describe the limitations of pattern development by pattern engineers. Since they have limited time, knowledge and access to the existing pattern knowledge it is a hard task to let them define all relevant event patterns. Therefore they propose a framework for automating the task of specifying event patterns by combining the knowledge possessed by pattern engineers with automatic techniques for specification of pattern parameters.

The EPTS<sup>5</sup> recognized the importance of event management and defines it as a discipline that encompasses among other things, the design, the development and the maintenance of events. It is even more complicated in the case of event patterns, as an event pattern consists of several events connected by event operators.

Kulkarni [Kulk11] describes that current CEP solutions are less usable by business users. The author describes that the reuse of existing event scenarios and the creation of similar event scenarios is a frequent task. In contrast to that the definition of completely new event scenarios is rather rare. The existing CEP solutions are rather usable by technically trained personnel and require significant effort to achieve the modelling goal. Therefore the aim should be to close the gap between the IT developer and the business user.

In [OSSK<sup>+</sup>11] the authors describe the need for a framework for technical-versed power users as well as business users. In their approach technical experts model event patterns in parallel that are fully integrated in the CEP application. A business user can assemble the event pattern from prepared and easy-to-use building blocks using a wizard-based user interface. The goal is to provide business users an abstraction layer that hides the underlying complexity.

### 3.1.2 Event pattern management in current CEP solutions

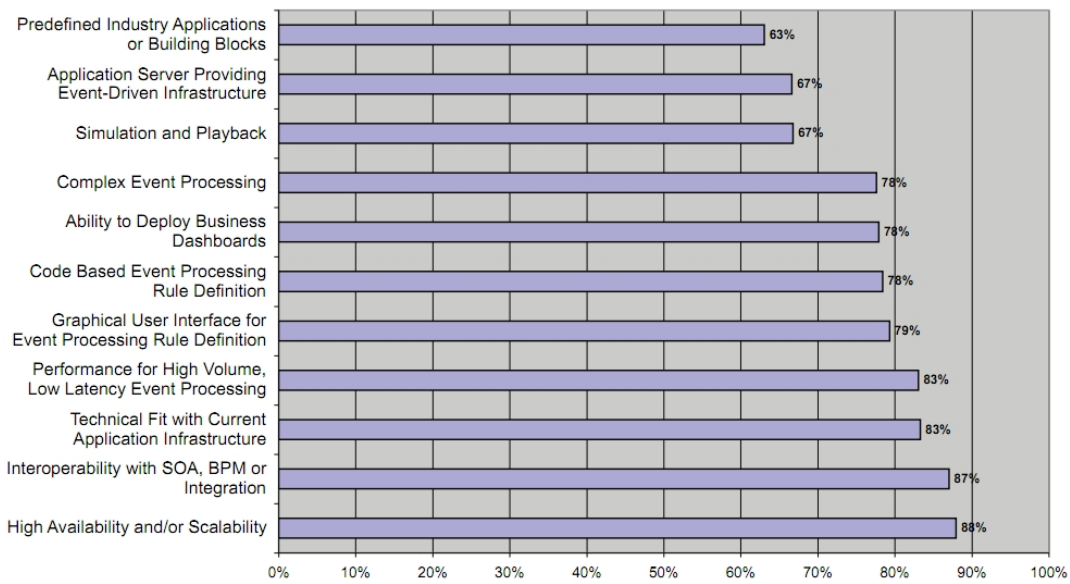
The list of commercial CEP applications is rather long. However, also in industry the management of event patterns is in its infancy. The following chart in Figure 3.2 taken from the ebizQ survey shows that 77-78% of respondents require beside the code-based event pattern<sup>6</sup> definition also a graphical pattern definition. What the survey does not show is what exactly the graphical user interface should support. Nowadays most of the commercial tools provide an user interface in order to configure the CEP system to develop and to deploy event patterns. Although these environments are a first step towards a better event pattern management and seem to be promising they are rather designed for IT experts. There is no methodology the pattern modelling tools are based on. The refinement, reuse and evolution of event pattern is not tackled by these solutions. All the tasks related to

---

<sup>5</sup>The Event Processing Technical Society is a diverse community (research+industry) interesting in Event Processing. <http://www.ep-ts.com>

<sup>6</sup>The survey uses the terminology of event processing rules for event patterns.





**Figure 3.2:** Most important event processing features.

refinement, reuse and evolution are done manually without any systematic system support during the life cycle of an event pattern.

## 3.2 Related research areas

The most closest research areas for this thesis are business rule management and active database that are described in this section.

### 3.2.1 Business rule management

A business rule management system (BRMS) is a software system designed to define, deploy, execute and maintain the decision logic. The main strength of the business rules approach is the externalization of the decision logic from the application logic. The externalization supports the maintainability of the system since rules are described in a declarative way and not in a procedural one. The business rules are executed in the business rule engine (BRE) which is a part of the BRMS.

A BRMS is based on the assumption that the rules change more often than the application logic. The separation of business logic from the application logic simplifies the change of rules. Whenever the business conditions change the rules are modified without recompiling the whole application.

The role of business rule management and evolution have been identified as a crucial point in using business rules. Rosca et. al [RoFW97] presents a methodology to cope with the maintenance and evolution of business rules. They present a

methodology covering the business rule acquisition, the business rule deployment and the business rules evolution. The most relevant aspect for our work is that they consider the data obtained through the monitoring of the underlying operational system in order to decide how well the business rules achieve the enterprise objectives. Taking this information into account they update the business rules. Lin et. al. [LiEW05] present an approach to support the evolution of business rules that are coded as part of the source code. Lin et al. [LiEW03] describe the importance of business rule evolution and identify the unforeseen changes to business rules as the main challenge for business rule evolution. It is important that the changes are implemented quickly, economically and reliably. In reality the evolution of business rules is time consuming and error-prone [EmSh03].

Although business rules and event patterns are rather similar there are some fundamental differences that makes it hard to use business rule management systems for event pattern management [EtNi10]. The first difference is that BREs are request-driven while CEP engines are event-driven. The processing of data is in BREs independent (state-less) while CEP engines operate on set of events. In order to derive high-level knowledge it is essential to aggregate events from different event sources in real-time. Another difference is the usage of temporal aspects in CEP. It is possible to extend the BREs in order to cover temporal aspects. However, this creates typically more overhead in the BRE (higher latency) for handling incoming events [ChSc09].

### 3.2.2 Active database management

Active database systems respond to events that are occurring either inside or outside the database itself [PaDi99]. However, the focus of active database systems is on specification and processing of triggers within a database management system (DBMS).

Paton [PaDi99] presents several possible dimensions (see Figure 3.3) for the rule management in active databases. The management of rules contains a set of facilities provided by the underlying system for rule representation and programming support for rules.

The description of rules is about the database language used to express the rules. There are several operations on rules in order to activate or deactivate a rule. The signal represents an external event which is used to notify the rule system about external occurrences. The adaptability is about changing rules either at run time or design time. The dimension of data model is considered as having a significant influence on the designers of the rule system. The last dimension, programming support, is considered as being the most important dimension if rules are used as a mainstream technology in business environments. Therefore methodologies for rule analysis, debugging, explanation and querying are of paramount importance. Dittrich et. al. [DiGG95] describe the programming environment

<b>Description</b> $\subset$ {Programming Language, Query Language, Objects}
<b>Operations</b> $\subset$ {Activate, Deactivate, Signal}
<b>Adaptability</b> $\in$ {Compile Time, Run Time}
<b>Data Model</b> $\in$ {Relational, Extended Relational, Deductive, Object-Oriented}
<b>Programmer Support</b> $\subset$ {Query, Trace}

**Figure 3.3:** Dimensions of rule management in active databases [PaDi99].

in active database systems as an essential operational feature. They describe a list of tools that an active database management system should provide including maintenance tools to support the user in defining and evolving the rule base.

### 3.3 Efficiency and effectiveness aspects in event pattern management

Increasing the efficiency and effectiveness of nowadays business are the main driver of event processing [EtNi10]. While efficiency describes the comparison between inputs used in a certain activity and produced outputs the effectiveness describes the production of right results. The aim of efficiency and effectiveness is to improve the speed of detection of relevant business situations and the reaction to these situations. CEP already provides capabilities that support the efficiency and effectiveness of the core processing of events. Chandy and Schulte [ChSc09] describe these capabilities as follows:

- *C1 - Real-time behaviour* Event processing is about event occurrence and proper reactions to occurring events.
- *C2 - Abstraction layer* Event processing is based on event patterns that encode the knowledge about a certain situation. This logic can be separated from the operational logic allowing to manage the event pattern without changing the operational logic.
- *C3 - Decoupling* The same events can be consumed by different parties that are not aware of each other. The event consumer and the event producer are not forced to know each other or to define routing between producer and consumer. The decoupling allows to deploy new services with low effort.

While the feature *C2* supports mainly the effectiveness, *C1* and *C3* support the efficiency of the event processing system. The efficiency is based on the fact that event processing is used in domains where business agility is an important issue. It may be necessary to change the event pattern quickly in order to observe new or arising situations. This could increase the speed of business agility. The efficiency issue in event processing considers the amount of events processed in

a given time period [EtNi10]. However, these aspects are rather restricted on the pattern matching itself. The application of effectiveness and efficiency to the relevant event pattern management aspects is also necessary and should be part of the overall efficiency and effectiveness issue in CEP.

Every event pattern is designed to detect a certain business situation. The definition of event patterns can be simple in cases where the all the needed events for a given business situation are known. If the relevant business situations are known but not all the events in order to model an event pattern the definition of event pattern is a hard task. In this case event patterns can be defined as an approximation to a relevant business situation and adapted over the time.

The issue of effectiveness in pattern management should be based on the degree of suitability of an event pattern to a relevant business situation. The most effective event pattern is the pattern which is able to cover the relevant business situations every time they may occur. A pattern is considered as less effective if some of the relevant business situations are not detected because of the structural combination of events and event operator and their configuration respectively. An incremental approach which will enable refinements of event patterns to sustain the pattern relevancy could increase the effectiveness of an event pattern.

Since patterns will change in time having methods of continually updating the patterns in order to ensure their relevance for new situations is also supporting the effectiveness. A continual improvement of event patterns enables an adaptation of patterns to the situations that need to be detected. Taking into account the assumption that an event pattern is not perfect at the beginning makes it necessary to bridge the gap between the functionality offered by an event pattern management system and the needs of a pattern engineer who models, deploys and modifies event patterns.

The efficiency in pattern management covers aspects that are both related to design time and run time. The goal is to decrease the time needed and hence the needed effort to model, deploy and evolve a pattern. In order to support the efficiency, it is important to provide an environment that enables a pattern engineer to operate on event patterns in a consistent and easy way. Graphical user interfaces that present the functions of the underlying system in a natural way could also have a positive influence on the efficient management of event patterns.

In the nutshell the efficiency is focused on achieving the definition and evolution of event patterns timely while the effectiveness is focused on having the right event patterns for the relevant business situations. We note that in pure end-user driven systems the productivity of each end-user in defining new rules and maintaining them is limited [Svio90]. Additionally, systems driven by users are often poorly structured, incomplete, highly coupled, and thus, difficult to maintain [Wagn00]. Therefore, there is more than a need to empower the pattern engineer with additional system support to be efficient and effective in managing event patterns.

## 3.4 Conclusion

Based on the ebizQ survey and [Luck01] we derived that it is necessary to provide abstract user interfaces for defining and maintaining event patterns. Since there are also pattern engineers that are non-technicians the event patterns needs to be understandable and shareable between pattern engineers. Further it is essential to provide them support during the definition and evolution of event patterns.

Taken into account [EtNi10], [ChEA11] and [Prob98] we identified that event pattern management starts with introducing a methodology dealing with the life cycle of an event pattern. The methodology should provide several phases consisting of multiple tasks in order to support the user to define the most suitable event pattern for a given situation. Both software engineering methodologies and knowledge management should be considered by defining the event pattern life cycle methodology.

Considering the requirements in [ChEA11] we identified the trade off between the expressiveness and the maintainability of event patterns. Expressive event patterns are not easy to manage and maintainable event patterns are not expressive. However, we will focus on the maintainability since most of the provided event operators that increase the expressiveness of an event pattern are not that frequently used in real world applications (see also [EtNi10]).

From the academic research on event pattern management, see [TuGW09], [Kulk11] and [OSSK<sup>+</sup>11], we concluded that there is a need for a holistic view on event patterns in order to support the pattern engineer during different life cycle stages of an event pattern.

Considering the current CEP solutions we found out that most of the systems provide a graphical user interface but neglect the additional tool support for pattern engineers. These tools are also rather designed for pattern engineers that have a fundamental programming background.

Last but not least from the related research areas, [PaDi99], [DiGG95], [RoFW97], [LiEW03], [EmSh03] and [LiEW05] we learned that the issue of evolution and tool support enables to decrease the barriers for using a system.



# 4

## Event Pattern Life Cycle Methodology

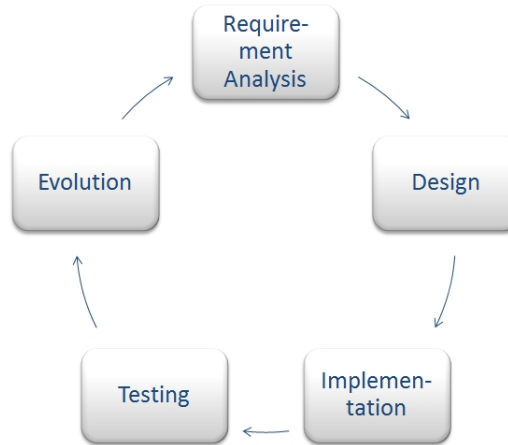
In this chapter we describe the event pattern life cycle methodology that covers both the design and run-time aspects of event patterns. While design-time aspects deal with the modelling and evolution of event patterns, run-time aspects deal with the execution of an event pattern in the CEP engine. We describe why designing and maintaining event patterns are more than just writing the pattern code and its deployment in the CEP engine. Especially in dynamic and rapidly changing enterprise environments methodological approaches should support the development of proper tools and methods as a reaction to the unforeseen changes. We further describe several tasks in each life cycle stage in order to support the pattern modelling, deployment and evolution.

### 4.1 Introduction

A methodology in general is a set of complementary methods and a clear description how to apply them [KlHi01], [ArNH86]. It is a guideline for achieving a global objective using components such as phases, tasks, methods and tools. The global objective in the case of this thesis is to increase the efficiency and effectiveness of a pattern engineer.

In contrast to the methodology a method tackles the accomplishment of a given task by providing a proper tool or software solution. A method specifies what needs to be done and in what order in order to achieve a given goal.

Methodologies have been used in software development to increase the productivity and quality of software development and decrease the time and effort [HaKS00], [HCRS<sup>+</sup>94], [DyMA05], [RiHD02]. The first formalization effort of the software development process has led to the Systems Development Life Cycle (SDLC).



**Figure 4.1:** Systems/Software Development Life Cycle (SDLC).

SDLC like shown in Figure 4.1, originated in the 1960s, is a series of software development phases observed by software developer. The aim was to deliver the right software that satisfies defined specifications [Elli04]. Since then plenty of software development methodologies have been introduced in order to control the process of information system development. Each methodology has its own recognized strengths and weaknesses (see [Gera06] for an analysis of SDLC).

The *ISO/IEC 12207* established a common framework for software life cycle processes that can be referenced by the software industry. It consists of process, activities and tasks that can be applied during the acquisition, supply, development, operation, maintenance and disposal of software products.

Taking into account the methodological approaches in software development and knowledge management, the observations we made in the research and industrial projects, described in Section 1.4, and the need for an event pattern management, described in Chapter 3, we derive the event pattern life cycle management methodology shown in Figure 4.2. The methodology constitutes of three phases forming a feedback loop. This feedback loop enables a continual flow of information collected in the entire life cycle of an event pattern. This information will be used for the continual improvement of event patterns.

The event pattern life cycle management methodology covers the phases Generation, Execution and Evolution of event patterns. Each phase contains a set of tasks that need to be realized in order to enable a pattern engineer to model and evolve event patterns efficiently and effectively. Below, we describe the phases of the methodology.



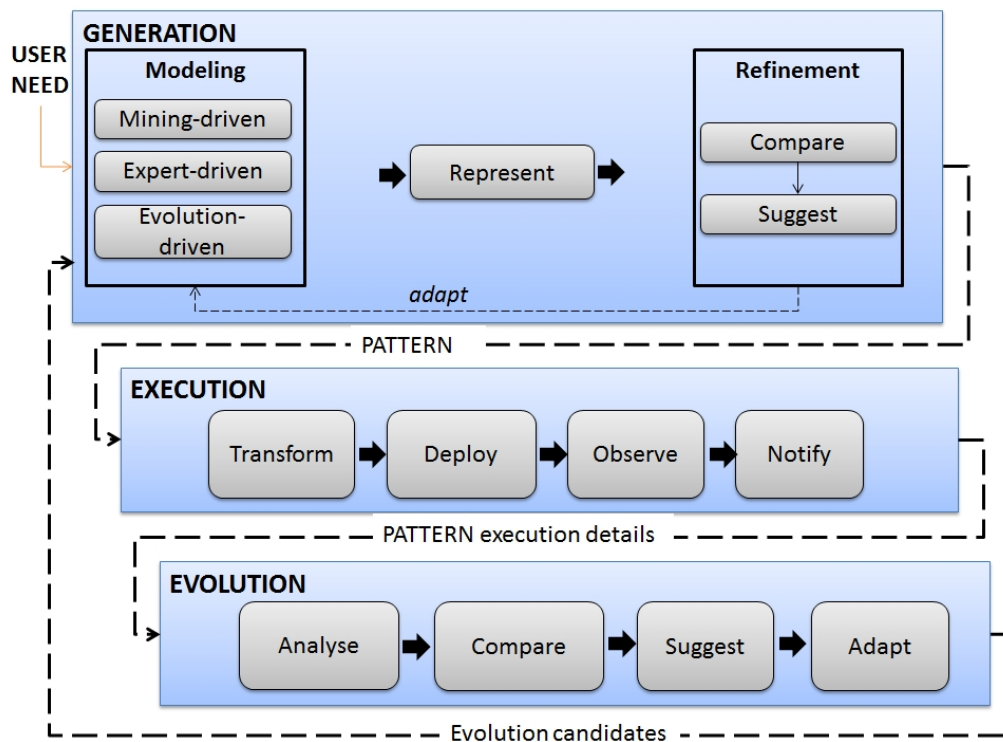


Figure 4.2: The event pattern life cycle management methodology.

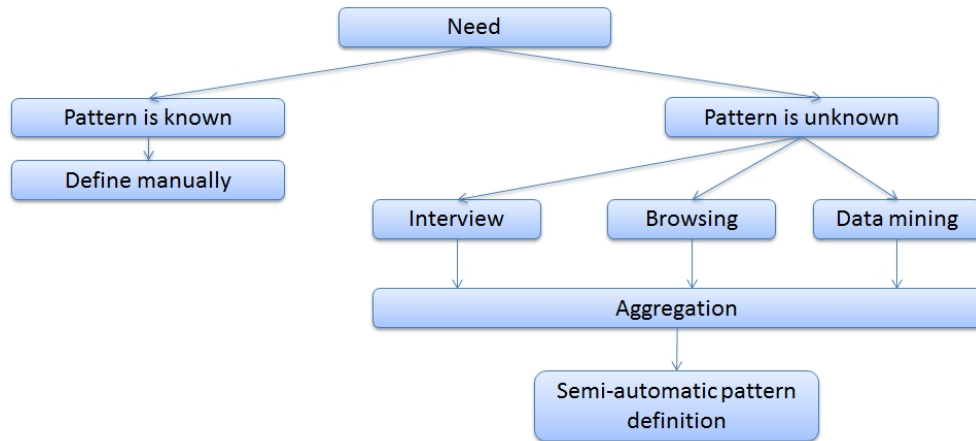
## 4.2 Generation phase

The life cycle of an event pattern starts with an initial need for a pattern. In enterprises the need is driven by new regulatories, business environment, products, processes etc. Once the need is identified the next step is to select the right strategy for the development of the event pattern. We classify the modelling of event patterns into three strategies which are expert-driven, mining-driven and the evolution-driven pattern development (Modelling in Figure 4.2). Starting from the initial need the important issue is whether the set of patterns that is needed to cover the relevant business situations is known or not. This initial modelling process is depicted in Figure 4.3.

**Example:** The pattern for detecting the relevant business situation is known.

*Whenever there are two transaction events following each other where the transaction value is greater 500*

In order to detect the situation in the example above we need to define a temporal order pattern (sequence) including a threshold pattern that filters all incoming transaction events with the value greater 500. In such a case the definition of the pattern is rather obvious and can be done manually. This belongs to the category of *expert-driven modelling* where the pattern engineer exactly knows what to model as a pattern. It is the externalization of the enterprise knowledge. The manual modelling of pattern works for domains where the pattern engineer



**Figure 4.3:** Pattern modelling activities.

is experienced and has solid knowledge about the domain. The challenge here is to find the right types of events and event operators, to combine and order them in the right way and foresee and solve the intermediate conflicts in the selected events. Therefore a pattern engineer should be provided with an overview of existing events to browse in the event repository, select events and combine them by using event operators.

On the other hand we are surrounded with critical situations that are much more complex. In this case the knowledge of the pattern engineer is not sufficient to model all event patterns that cover the relevant business situations effectively.

**Example:** The set of patterns is not known or partially known.

*whenever there is a credit card fraud*

To provide a set of patterns to cover a credit card fraud is rather hard since there exist plenty of ways for a fraud and additionally there is always space for new fraudulent patterns. We need a different approach in order to define the set of patterns for situations where the patterns are not known. For such cases we propose the following procedure to model event patterns:

- **Interview** Business experts will be interviewed in order to collect a list of patterns that might be of interest. Once these event patterns are identified they can be modelled as event patterns for the CEP engine. This can be done either by using a graphical user interface or the event processing language of the underlying CEP engine directly. The EPTS provides an initial document in order to support the identification of event patterns. However, this document should be considered as an initial draft (see appendix A3). Taking into account that document, our experiences in the area of CEP and the discussions with the end user, especially in the ALERT project<sup>1</sup>, we

<sup>1</sup><http://www.alert-project.eu>

propose the following classification of event patterns based on the complexity of the patterns. These classes allow the collection of event patterns for different business situations.

- **Low Complexity** This class of event patterns considers only single events. All event patterns that are members of this class define certain filters on event properties or check whether a certain event has occurred or not.
- **Medium Complexity** Event patterns of this category extend the low complexity event patterns with the capability of combining multiple events. Events can have different types and sources. In order to combine multiple events medium complexity patterns include conjunction (AND), disjunction (OR) and further logical operators without time and aggregation constraints.
- **High Complexity** High complexity patterns extend medium complexity patterns with the capability of temporal aspects and aggregation functionalities. These patterns include followed-by operator (SEQ), time windows and count windows.

The focus of this classification is on the maintainability of event patterns as described in Section 3.1. The event operators that are covered by this classification are threshold patterns, conjunction patterns, disjunction patterns, negation patterns, subset patterns and temporal order patterns as presented in Section 2.2.

- **Browsing the Pattern Repository** Browsing takes into account any navigation queries that are related to search. This method is a more pattern engineer-oriented approach enabling to find equal patterns and hence avoid duplicates, to find patterns that are more general or specific, to find similar patterns or to find frequently used event patterns that could be relevant. By analysing the past event pattern execution statistics a pattern engineer can obtain new insights and define new event patterns through the analysis and interpretation of event pattern executions. The generation of new patterns based on existing pattern execution statistics can be considered as a process. We present in Chapter 6 and Chapter 7 several search methods in order to derive new pattern knowledge from existing patterns or from pattern executions.
- **Event Pattern Mining** Beside event patterns developed by experts explicitly, there are also implicit patterns in the domain, reflected in the behaviour of the environment. They can be discovered through the analysis of the event repository (event logs). The analysis is driven by tools analysing log files and attempt to detect patterns.

In the research community there are initial thoughts about mining event patterns from historical data to support the pattern engineer in defining new

event patterns (see [MKFB<sup>+</sup>12]). The task is to discover new event patterns that might be of interest. The main challenge here is that existing data mining technologies deliver frequent patterns meaning events that occur together quite often. However, these patterns may not be relevant since in CEP the aim is to define patterns of interest that are essential for the business and require some kind of reaction. The occurrence of such event patterns could be also rather rare meaning that current data mining algorithms were not able to detect the relevant event patterns as being relevant.

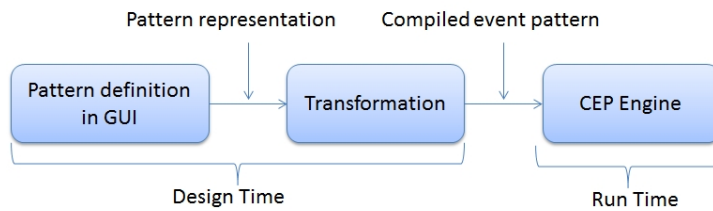
As these strategies complement each other, it is important that a combination of them should be targeted. During the aggregation there should be a manual review and refinement by the pattern engineer that finalizes the pattern definition. Using such a strategy may decrease the number of false positives generated for example by the underlying data mining technique.

Regardless of how a pattern is generated it should be represented (Represent in Figure 4.2) according to an event pattern model. Rozsnyai et al. [RoSS07] describe the importance of a well defined event model and its impact on the flexibility and usability of CEP tools. Adi et al. [AdBE00] point out the importance of a proper event representation in event processing systems. Contemporary event representations lack the capability to express much of the event semantics and relationships to other entities. In order to support the management of event patterns we present in Chapter 5 a meta model for event patterns that will be used for modelling and representation of event patterns. The aim of the meta model is to support the modelling and the evolution of event patterns through the event pattern relationship detection.

Since the event pattern modelling is not a very precise formulation of a pattern in the first place but an iterative refinement (Refinement in Figure 4.2) of the event patterns the refinement of patterns should be offered to the pattern engineer by taking into account the current modelling context and comparing this context with the existing event patterns in the pattern repository. The refinement which is done through the comparison of the modelling context and suggestion of relevant event patterns is described in Chapter 6.

### 4.3 Execution phase

The execution phase of an event pattern starts with its deployment and ends with its undeployment. The first task of this phase is the transformation (Transformation in Figure 4.2) of the event pattern modelled by the pattern engineer into its CEP engine specific representation as presented in Figure 4.4. The transformation of an event pattern should cover the transformation of events, event operators and complex events respectively. The transformed event pattern will be registered



**Figure 4.4:** Transformation of an event pattern in native CEP language.

(Deployment in Figure 4.2) in the CEP engine for matching to continue with the life cycle of the event pattern.

However, nowadays matching process in the CEP engine is designed as a black-box. The pattern is evaluated true if and only if the whole pattern is evaluated true. As a matter of fact the black-box approach introduces a chasm into the event pattern life cycle. The data regarding the matching is not feed into the pattern life cycle. However, the matching details can be used to detect possible errors in the event pattern definition.

In order to collect the data during the pattern execution we propose a white-box approach. The goal of the white-box approach is to track as much as possible data about the execution of event patterns in the CEP engine (Obverse in Figure 4.2). The purpose of this tracking is to send the data (Notify in Figure 4.2) in order to improve the event patterns in the evolution phase. The white-box approach is described as part of the evolution approach in Chapter 7.

## 4.4 Evolution phase

The Evolution phase is responsible for coping with changes that may affect an event pattern. In a more open and dynamic business environment the domain knowledge evolves continually. A pattern that must not become rapidly obsolete must change and adapt to the changes in its environments [TuGW09]. Therefore, if a pattern aims at remaining useful it is essential that it is able to accommodate the changes that will inevitably occur. Facilitating those changes is complicated especially if large quantities of events and event patterns are involved. Developing patterns and their applications are expensive but evolving them is even more expensive.

We classify the changes that affect an event pattern into two categories, namely external changes and changes triggered by the pattern execution. In this thesis we focus on the changes that are triggered by pattern executions which will be described in Chapter 7.

**Evolution driven by external changes** On the one hand the environment in which systems operate can change. For example new event sources make it necessary to

change the pattern logic. Events created by these new sources could be used either to create new patterns or update existing patterns by considering the information flow from these new sources. On the other hand users' requirements often change after the system has been built, warranting system adaptation. For example the user can change an existing pattern which was modelled to detect a certain business situation.

**Evolution driven by pattern execution statistics** While a good design may prevent many pattern errors some problems will not be detected before patterns are in use, like:

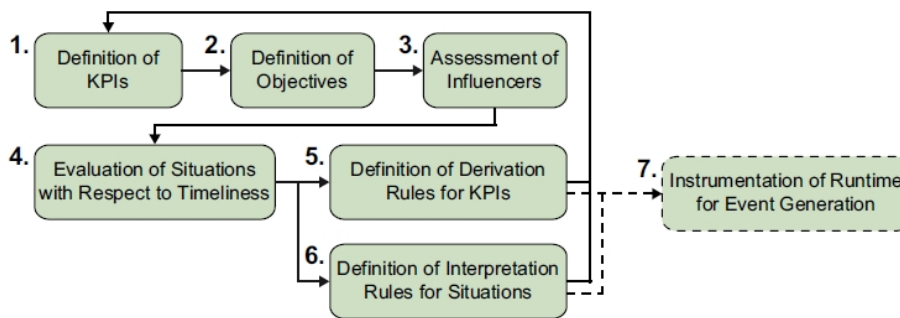
- The events are not received by the CEP engine because of some defective behaviour of the network or information sources.
- The pattern is not relevant any more after a certain time period.
- The pattern is defined imprecisely meaning it is either too specific, it is detected very seldom or it is designed too general meaning it is detected very often which leads again to an information overload.

Therefore the analysis of the execution data of an event pattern is of paramount importance in the process of event pattern generation and should not be neglected. For example, if an event pattern is never executed because a particular condition was never satisfied, the reason could be that the event pattern is defined very specific and need to be relaxed. This bottom-up pattern development enables a continual improvement by adapting execution-mining techniques. Furthermore the quotient between partially fulfilment and total fulfilment can provide knowledge for the evolution. The partially fulfilment describes the state of an event pattern where parts of the pattern have been evaluated true. In order to justify whether there exists a problem it is necessary to compare the current execution data with historical execution data. This will be described in Chapter 7.

## 4.5 Related work

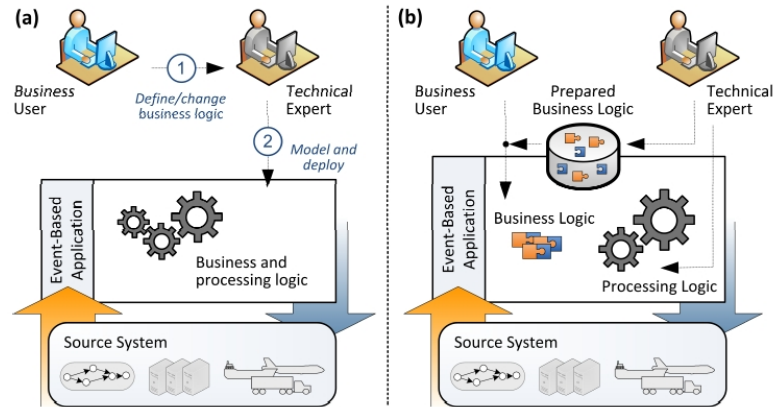
Vidackovic et. al [ViKD10] present a CEP development methodology (see Figure 4.5) that utilizes the Zachman framework [Zach87] by building a monitoring model in a top-down approach. The stepwise development methodology starts with the specification of business goals and the definition of key performance indicators (KPIs) and business objectives. The next step is the assessment of influencers' impact on goals and the evaluation of situations where low-latency can increase the business value. The definition of event pattern rules without technical details is the last step before the event pattern rules are specified using a concrete EPL. Compared to our methodology the development methodology is more on an

abstract strategic and conceptual level. Although the methodology can be useful from a business-oriented perspective, it does not consider the event patterns from a more technical perspective like how to model, execute and evolve them. We see the role of the methodology presented by [ViKD10] as part of the initial process where the relevant business situations are identified. However, both the methodology presented in [ViKD10] and the methodology presented in this thesis can be connected in series. Once the initial need for a relevant business situation is defined through the definition of the KPIs (Key Performance Indicators) the *Instrumentation of Runtime for Event Generation* (step 7 in Figure 4.5) can be done using the methodology presented in this thesis.



**Figure 4.5:** CEP development methodology according to Vidackovic et al. [ViKD10].

Obweger et al. [OSSK<sup>+</sup>11] describe that the practical relevance of CEP rises and falls with the manageability of the underlying event-pattern rules. The authors describe that event patterns (they use the terminology of business logic) are associated with different user groups in an enterprise. Each user group has different technical skills and competences. The approach aims at solving the issue that business users need technical experts in order to model and deploy event patterns (see (a) in Figure 4.6) by creating a pattern library where technical experts prepare the patterns which are used by the business user (see (b) in Figure 4.6). While creating an event pattern by combining existing business logic is a valid approach, there are no details about how the technical experts know what the business users expect. To offer a management environment that supports both the needs of technical-versed power users as well as business users seems promising. However, the main focus of Obweger et al. [OSSK<sup>+</sup>11] is not the life cycle management of event patterns. They neither describe the modelling nor the refinement and evolution of event patterns in general. We further describe for each phase several tasks that support a pattern engineer at different stages. While our approach is more designed for pattern engineers that do not have deep CEP knowledge the approach presented by [OSSK<sup>+</sup>11] combines the expertise of a technical experts and business user in one management environment.



**Figure 4.6:** Comparison of event pattern rule-management in existing systems (a) and the proposed approach (b) by [OSSK<sup>+</sup>11].

## 4.6 Summary

In this chapter we presented the event pattern management life cycle covering the modelling, the deployment and the evolution of event patterns. In each phase we defined multiple tasks in order to accomplish the modelling, the deployment and the evolution of event patterns. We described several strategies to model new event patterns and outline the importance of partially fulfilled event patterns in order to support the evolution of event patterns. The evolution of event patterns is needed in order to guarantee the constant relevance of an event pattern. In the next chapters we will describe the phases of the methodology by providing the details of our approaches and tools.



# 5

## Event Pattern Meta Model

In this chapter we present the meta model for describing the elements of an event pattern from the event pattern management point of view. The role of the event pattern meta model is to support the event pattern modelling and evolution. Events and event patterns are handled as a fundamental information unit to be stored and queried in order to support pattern engineers during the pattern modelling and evolution. The meta model is an explicit description of how a domain specific event pattern model should be built. Since the way data modellers and knowledge representation researchers view events is diversified (see [Amar11]) it is necessary to provide a meta model considering the management of event patterns.

### 5.1 Introduction

According to [Pidd00] a model is an external and implicit representation of a part of reality as seen by people who wish to use that model to understand, change, manage and control that part of reality. A meta model is the construction of a collection of concepts within a certain domain. The meta model is another abstraction layer for a model describing the properties of the model itself [SAJP<sup>+</sup>02]. Like ontologies meta models are used to describe and analyze the relations between concepts [SAJP<sup>+</sup>02]. In this chapter we analyze the concepts and their relationships to each other that are needed to describe an event pattern. The current strategy

to provide a meta model through a top level ontology<sup>1</sup> for events is based on the identification of a common set of attributes like timestamp, type, place references and so forth (see for example [KhSt09]). Other approaches (see [GGMO<sup>+</sup>02], [HMSH<sup>+</sup>11] [SFSS09]) use the event relevant elements in existing upper level ontologies like DOLCE<sup>2</sup> (Descriptive Ontology for Linguistic and Cognitive Engineering) to define event ontologies. These strategies lead to an ontology model that will not be an efficient one since it include all possible event attributes like *What happened?*, *When it happened?*, *Who is involved?* and so forth [TePa09]. The resultant meta models are not designed from the event management point of view where the focus should be on supporting the pattern engineer during the pattern life cycle.

In order to represent event patterns based on the meta model we use a graph-based approach. A graph-based approach provides the flexibility to define inter-event relationships [Amar11]. RDFS<sup>3</sup> (Resource Description Framework Schema) is a proper candidate in order to define events and event patterns as graphs. Further RDFS has the advantage that it offers formal and explicit definitions of concepts and relations between concepts. It provides a set of primitives to describe lightweight ontologies in Resource Description Framework<sup>45</sup> (RDF) using the RDF model and syntax.

Below we present the meta model consisting of the event layer and the event pattern layer. The examples are serialized in RDF Turtle<sup>6</sup> syntax.

## 5.2 Event layer

As described earlier an event is an occurrence of something within a particular domain or system [EtNi10]. It is something that has happened or is contemplated as having happened in that domain. In its very minimal form an event is a signal containing only a time stamp of the event occurrence. Contemporary event formats are proprietary composed of attribute values [Luck01] like shown in the example below. However, this proprietary format is not well suited for the event pattern management.

### Example

*tweet(authorName, timeOfCreation, tweetText)*

<sup>1</sup>A top-level ontology is an ontology which contains very general and knowledge domain independent concepts.

<sup>2</sup><http://www.loa.istc.cnr.it/DOLCE.html>

<sup>3</sup><http://www.w3.org/TR/rdf-schema/>

<sup>4</sup><http://www.w3.org/RDF/>

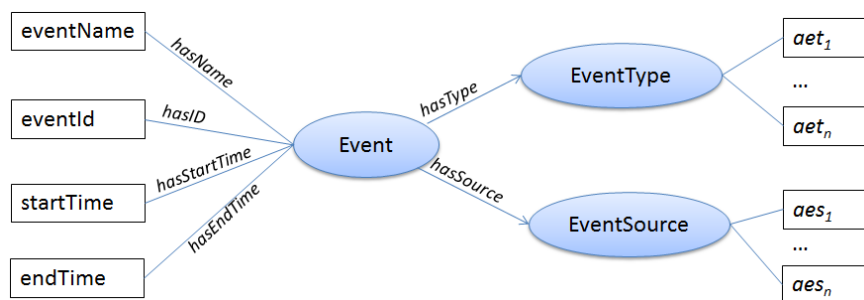
<sup>5</sup>Please notice that our focus is on the meta model and not providing an ontology in RDFS. RDFS is used for specifying a concrete event pattern. One can also use other graph based representation languages that provide type inheritance and property specifications (with domain and range

<sup>6</sup><http://www.w3.org/2007/02/turtle/primer/>

The event above which is an example for a tweet<sup>7</sup> in Twitter consists of a set of attributes which are *authorName*, *timeOfCreation* and *tweetText*. All attributes are handled as strings. Additional information regarding the type and source of the event are also embedded in the proprietary format.

In order to construct the event pattern meta model we start with the meta model describing the nature of events and their relationship to other event related concepts. We propose an event meta model where an event is a construct consisting of the following three concepts.

- **Event** This concept contains the minimal set of properties an event should have. This concept denotes an untyped event describing the occurrence of something. In order to specify the source and type details we introduce the concepts *EventType* and *EventSource*, an event is linked to. These concepts support the browsing and the relationship detection as it will be described in Chapter 6.
- **Event Type** Events can be classified into event types according to their attributes. All events with the same event type share the same set of attributes. For example events that carry the temperature information would belong to the same event type class. In order to classify events there exist several approaches in linguistics, artificial intelligence and temporal databases [Amar11]. However, a comprehensive inclusion of such classification into an event model is missing and therefore needed [Amar11].
- **Event Source** Every event has an event source as described in Subsection 2.1.2. All events created from the same event source belong to the same event source concept. For example events created from a tweet would belong to the same event source class.



**Figure 5.1:** Event meta model.

Like shown in Figure 5.1 an *Event* is linked to the concepts *EventType* and *EventSource* by using the relationships *hasSource* and *hasType*. Below we introduce a more formal definition of an event and its related concepts described in Figure 5.1.

<sup>7</sup>A tweet contains more attributes than displayed above. For more details regarding a tweet see the project <http://twitter4j.org/en/index.html>

**Definition:** A simple event is defined as a 6-tuple  $SE := (I, N, S, E, ES, ET)$  where:

- I is the numeric event id.
- N is the alphanumeric event name.
- S is the numeric start time of an event.
- E is the numeric end time of an event.
- ES is a reference to a single event source concept in the ontology.
- ET is a reference to a single event type concept in the ontology.

A concrete occurrence of a simple event has an unique id and time interval consisting of start and end time of an event. Since simple events are atomic the start and end time are set to the same value. The time interval is needed in order to define complex events that are composed of other events. Further an event has an event type and an event source that are used in order to classify events.

The event type characterizes a class of event objects according to the attributes of an event object. For example every event that carries a tweet has the the same event type (see also [AdBE00]).

**Definition:** The event type is defined as a tuple  $ET := (H_{et}, A_{et})$  where:

- $H_{et}$  is an acyclic event type hierarchy.
- $A_{et}$  is a set of event type specific properties  $\langle aet_1, \dots, aet_n \rangle$  with range constraints.

An event source is an entity that indicates where an event occurred, e.g. a software module, a sensor, a web service etc. The event source characterizes a class of event objects according to the attributes that are only valid for the source.

**Definition:** The event source is defined as a tuple  $ES := (H_{es}, A_{es})$  where:

- $H_{es}$  is an acyclic event source hierarchy
- $A_{es}$  is a set of event source specific properties  $\langle aes_1, \dots, aes_n \rangle$  with range constraints.

Every event consists of an instance of the concept *Event* that is connected to an instance of the *EventType* and *EventSource*. Below we present a scenario for web data monitoring that will be used to define events, event operators and the final event pattern throughout this chapter.

**Example:**

Whenever there is a news article about **Greece** followed by a **comment referring to the article** from the same **online source** create a new complex event (the complex event is used for the notification about the occurred situation) with the reference to the web page. The source of the information should be located in **Germany**.

In order to cope with this scenario we start with a possible specification of events, event types and event sources based on the meta model<sup>8</sup>. The concept *EventSource* has the concept *Country* as its subclass denoting the source of the event. The event type is more distinctive. It has the concept *Web* as its direct subclass. The concept *Web* again has two concepts namely *Comment* and *News* as its direct subclasses. The overview of the concepts *Event*, *EventSource* and *EventType* would look like as follows:

```
:Event a rdfs:Class;
    rdfs:subClassOf owl:Thing.
:EventSource a rdfs:Class;
    rdfs:subClassOf owl:Thing.
    :Country a rdfs:Class;
        rdfs:subClassOf :EventSource.
:EventType a rdfs:Class;
    rdfs:subClassOf owl:Thing.
:Web a rdfs:Class;
    rdfs:subClassOf :EventType.
    :Comment a rdfs:Class;
        rdfs:subClassOf :Web.
    :News a rdfs:Class;
        rdfs:subClassOf :Web.
```

In order to detect the situation in the above scenario we define two events<sup>9</sup> whereby one represents a news article and the other a comment. Both events have a location which is the geographic location of the publishing source (country). Based on the meta model the two events would look like as follows

```
:NewsEvent a :Event;
    :hasSource :Country;
    :hasType :News.
:CommentEvent a :Event;
    :hasSource :Country;
    :hasType :Comment.
```

<sup>8</sup>The subclass definition and type hierarchy is only used to exemplify the use of the meta model and is not a complete specification of the domain.

<sup>9</sup>For the sake of simplicity the remaining event attributes are not used.

where both events are linked to the proper event type and the event source. In order to detect the relevant situation we further define attributes for the event types ( $aet_1, \dots, aet_n$ ) and event sources ( $aes_1, \dots, aes_n$ ) like described in the event type and event source definition.

```

:countryName a owl:DatatypeProperty;
    rdfs:domain :Country;
    rdfs:range xsd:string.
:articleId a owl:DatatypeProperty;
    rdfs:domain :Web;
    rdfs:range xsd:string.
:url a owl:DatatypeProperty;
    rdfs:domain :Web;
    rdfs:range xsd:string.
:subject a owl:DatatypeProperty;
    rdfs:domain :Web;
    rdfs:range xsd:string.
:description a owl:DatatypeProperty;
    rdfs:domain :Web;
    rdfs:range xsd:string.

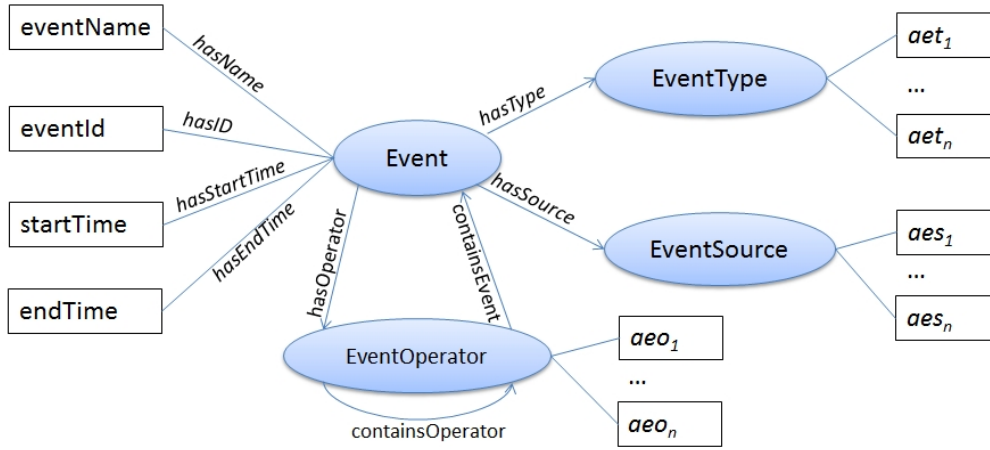
```

The concept *country* has only one attribute called *countryName* that denotes the name of the country. The concepts *Comment* and *News* inherit four attributes from the super concept *Web*. These attributes are *articleId*, *url*, *subject* and *description*. At run-time a concrete event occurrence represented based on the above event definition would contain the values for the event type and event source attributes.

So far we only defined the simple events *NewsEvent* and *CommentEvent* based on the meta model. Since a complex event is the result of an event processing logic it has a reference to the event operator concept. Like mentioned above a complex event may have different values for start time and end time. The start time indicates when the first event arrives that was part of the processing. The end time indicates when a complex event was finally generated.

**Definition:** A complex event is defined as a 7-tuple  $CE := (I, S, N, E, ES, ET, EO)$  where:

- I is a numeric event id.
- N is a alphanumeric event name.
- S is a numeric start time of an event.
- E is a numeric end time of an event.
- ES is a reference to the event source concept.
- ET is a reference to the event type concept.



**Figure 5.2:** Event meta model extended with event operator in order to define complex events.

- EO is a reference to the event operator concept.

Event operators (described in Section 2.2) can be aggregation operators (AGG,AVG, COUNT,...), window operators (WITHIN,...), logical operators (AND, OR,...), temporal operators (SEQ, ...) or geospatial operators (near,...)(see [EtNi10] for a detailed overview of possible event operators).

**Definition:** An event operator is defined as a tuple  $EO := (H_{eo}, EV, OV, A_{eo})$  where:

- $H_{eo}$  is an acyclic event operator hierarchy
- $EV$  is a set of events (either simple or complex) that an event operator operates on
- $OV$  is a set of event operators it is connected to
- $A_{eo}$  is a set of event operator specific properties  $\langle aeo_1, \dots, aeo_n \rangle$  with range constraints.

In order to detect the pattern presented in the previous example we need to define the resultant complex event. We call this event *InfoEvent*. Beside the link to the event type and event source a complex event has an additional link to the event operator. The source of the complex event would be a specific CEP engine where the matching takes place. Additionally we have to introduce a new event type concept representing the attributes of the complex event and a concept for the event operator the complex event is linked to. Since in our scenario we have a temporal order between the two events we use the sequence operator (SEQ) in order to define the complex event. The complex event and its event type, event source and event operator definition would look like as follows

```
:InfoEvent a :Event;
```

```

    :hasSource :CepEngine;
    :hasType :NewsAggregation;
    :hasOperator :SEQ.
:CepEngine a rdfs:Class;
  rdfs:SubClassOf :EventSource.
:NewsAggregation a rdfs:Class;
  rdfs:subClassOf :Web
:EventOperator a rdfs:Class;
  rdfs:subClassOf owl:Thing.
:SEQ a rdfs:Class;
  rdfs:subClassOf owl:EventOperator.

```

where the concept *CepEngine* is a direct of *EventSource*, *NewsAggregation* is a direct subclass of *EventType*, *EventOperator* is a direct subclass of *owl:Thing* and the event operator *SEQ* is a direct subclass of *EventOperator*. The new introduced concepts have additional attributes which are

```

:engineName a owl:DatatypeProperty;
  rdfs:domain :CepEngine;
  rdfs:range xsd:string.
:containsEvent a owl:DatatypeProperty;
  rdfs:domain :EventOperator;
  rdfs:range :Event.
:operatorConstraints a owl:DatatypeProperty;
  rdfs:domain :EventOperator;
  rdfs:range xsd:String.
:eventOrder a owl:DatatypeProperty;
  rdfs:domain :SEQ;
  rdfs:range :Event.

```

where *engineName* contains the value of the CEP engine, *containsEvent* denotes the events that the event operator operates on, *operatorConstraint* denotes the restrictions for the linked events and *eventOrder* which describes the temporal order of the events that are linked to the sequence operator. The attributes *operatorConstraint* and *eventOrder* are operator specific attributes ( $\langle aeo_1, \dots, aeo_n \rangle$ ). An example will be presented in the next section as part of the final event pattern.

### 5.3 Event pattern layer

The event pattern layer shown in Figure 5.3 extends the event layer with a set of additional concepts in order to specify the final event pattern. The additional concepts are *PatternDomain* and *EPAT*. Further it defines additional properties for these two concepts. The properties are *refersTo*, *belongsTo*, *hasId*, *hasName*, *hasTimeOfCreation* and *hasDescription*.



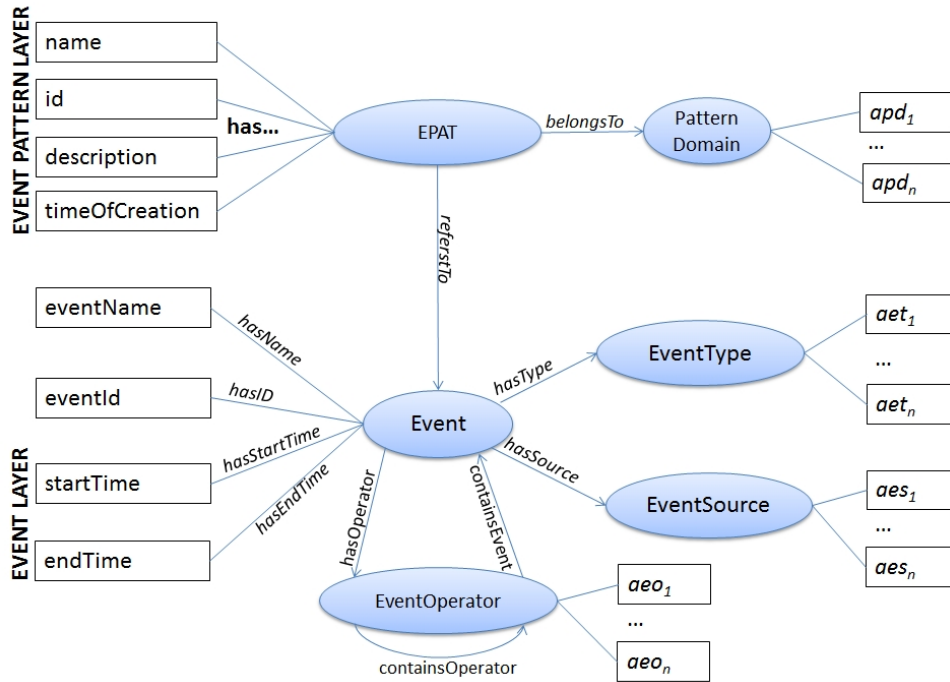


Figure 5.3: Event pattern meta model.

**Definition:** An event pattern is defined as a 6-tuple  $EPAT := (PI, PS, PN, PC, PD, PE)$  where:

- PI is a numeric pattern id.
- PS is a alphanumeric description of the pattern.
- PN is a alphanumeric pattern name.
- PC is a numeric time of creation.
- PD is a reference to the pattern domain.
- PE is a reference to the event concept.

Every event pattern must belong to a pattern domain. That allows to categorize event patterns regarding the target domain where the event pattern will be used for. For example all event patterns modelled for the observation of social media content in Twitter, Facebook, Blogs etc. would belong to the pattern domain *SocialMediaMonitoring*.

**Definition:** The pattern domain is defined as a tuple  $PD := (H_{pd}, A_{pd})$  where:

- $H_{pd}$  is an acyclic pattern domain hierarchy.
- $A_{pd}$  is a set of pattern domain specific properties  $\langle apd_1, \dots, apd_n \rangle$  with domain and range constraints.

In order to finalize the event pattern definition the last step is to use the complex event *InfoEvent* and to define the final event pattern. We introduce the pattern domain *NewsMonitoring* which is an instantiation of the *PatternDomain*.

The final event pattern *epat* has an *id*, *name*, *description*, *timeOfCreation*, *belongsTo* and *refersTo*.

```
:PatternDomain a rdfs:Class;
  rdfs:subClassOf owl:Thing.
:NewsMonitoring a :PatternDomain .
:EPAT a rdfs:Class;
  rdfs:subClassOf owl:Thing.
```

Taking into account the event layer and event pattern layer we can define the final event pattern (this is a concrete instantiation of an event pattern including all described concepts) that represents the scenario introduced in Section 5.2. Let's start with the definition of the event pattern which looks like as follows:

```
:epat a : EPAT;
  :Id "wq778981"^^xsd:String;
  :Name "MonitoringPattern"^^xsd:String;
  :description "Greek news"^^xsd:String;
  :timeOfCreation "04.03.2012,12:34"^^xsd:String;
  :belongsTo :NewsMonitoring;
  :refersTo :InfoEvent.
```

The concrete event pattern *epat* which is an instantiation of *EPAT* has an *id*, *name*, *description*, *timeOfCreation* the pattern domain it belongs to and the complex event it refers to.

The complex event instantiation based on the definition presented in Section 5.2 is defined as follows:

```
:infoevent a :Event;
  :hasSource :cepEngine;
  :hasType :newsAggregation;
  :hasOperator :seq.
```

The *infoEvent* which is an instantiation of *Event* has a link to a specific event source, type and operator as defined below:

```
:cepEngine a :CepEngine;
  :engineName "ESPER"^^xsd:string.
:newsAggregation a :NewsAggregation;
  :articleId "ft5703"^^xsd:string;
```

```

:url "http://www.ftd.de/thema/griechenland"^^xsd:string;
:subject
  "Geldgeber streiten \"uber Griechenland"^^xsd:string;
:description
  "Ausnahmsweise sind es nicht die Griechen..."^^xsd:string;
:seq a : SEQ;
:containsEvent :news, :comment;
:operatorConstraints
  "news.articleId=comment.articleId"^^xsd:string;
:operatorConstraints
  "news.description contains Griechenland^^xsd:string;
:operatorConstraints
  "news.subject contains Griechenland^^xsd:string;
:eventOrder :news, :comment.

```

The *cepEngine* which is an instantiation of *CepEngine* has the name *ESPER*. The final complex event has the event type *NewsAggregation* carrying the information about the relevant article about Greece which was detected as part of the matching process. The operator which was involved in the matching process is a concrete instantiation of the sequence operator (SEQ). The operator is linked to specific news and comment events. In order to detect the relevant articles from the mentioned scenario it includes a constraint value where the article ids of the news and comment event should be the same. Further the news article needs to contain at least the word "Griechenland" in its subject or description. In order to guarantee the correct matching where a news article needs to be published before a comment event is received is expressed in the *eventOrder* where the event *news* needs to happen before the event *comment*.

## 5.4 Implementation

As we described in Chapter 3 for removing the barriers from accessing a CEP system it is important to provide easy to use graphical user interfaces to the underlying event processing components. In this section we describe the event editor as part of the PANTEON tool that allows the pattern engineer to design new events<sup>10</sup> based on the meta model. These events will be used to model new event patterns. The modelling of new event patterns will be described in Chapter 6.

Although the meta model does not contain many concepts, its instantiation to a concrete domain is for those that are not familiar with RDFS and ontology editors

<sup>10</sup>The event is used as an template or schema. It will be used for the pattern generation where the attribute values are set.

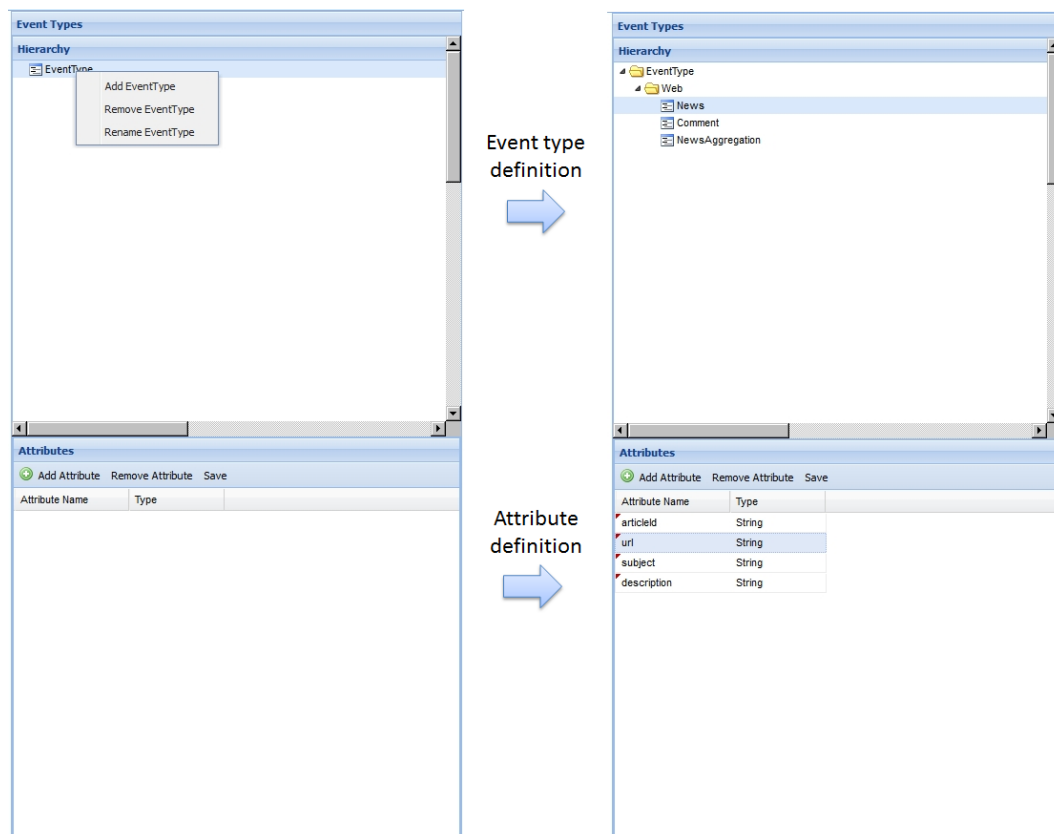


Figure 5.4: The PANTEON event editor.

like, Protege<sup>11</sup>, TopBraid Composer<sup>12</sup> not easy. Furthermore the usage of different tools for event and event pattern definition might decrease the efficiency of the pattern engineer.

The *PANTEON event editor* allows the pattern engineer to define a concept hierarchy for event types and event sources with additional attributes for each concept. Attributes can be inherited by subconcepts. Every attribute has a type which is the *range* of the attribute. The *domain* of the attribute is the concept for which it is defined. Figure 5.4 displays the event type and event source hierarchy for the scenario we modelled in the previous section. The definition of a new event includes three steps described below in the next paragraphs.

**Definition of the event types and event type attributes** The list of initial event types contains only the concept *EventType* which does not provide any attributes and subclasses (left part in Figure 5.5). It is the strict implementation of the meta model where only the concept name is provided. In order to modify the event



**Figure 5.5:** Definition of the event types and attributes in PANTEON event editor.

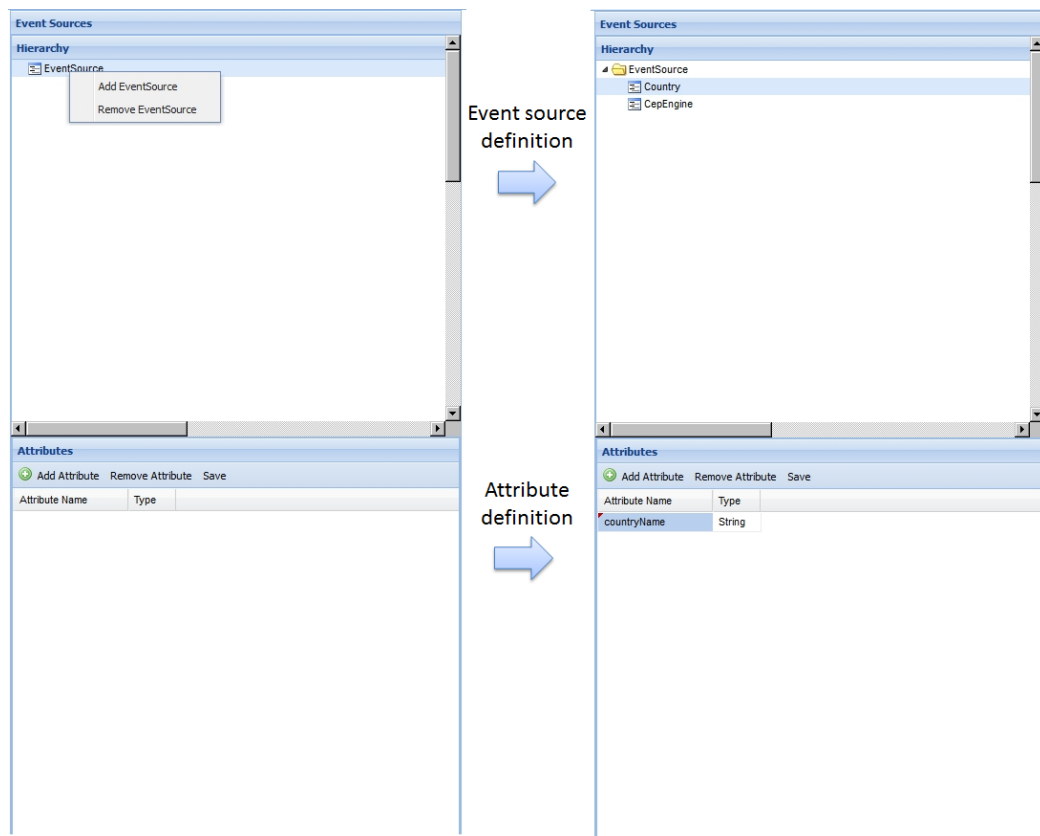
type hierarchy the event editor provides the operations *Add EventType*, *Remove EventType* and *Modify EventType*. Each new type is defined as a sub concept of the previously selected event type. In Figure 5.5 we defined all the event types defined in the context of the previous scenario. The attributes can be defined

<sup>11</sup><http://protege.stanford.edu/>

<sup>12</sup><http://www.topquadrant.com>

for each concept. Attributes defined for super concepts are inherited by the sub concepts. Each attribute has a data type which is the range of the attribute. This type can be a reference to simple data types like string, integer or complex like ontological concepts. The reference to a concepts is defined through the URI (Uniform Resource Identifier).

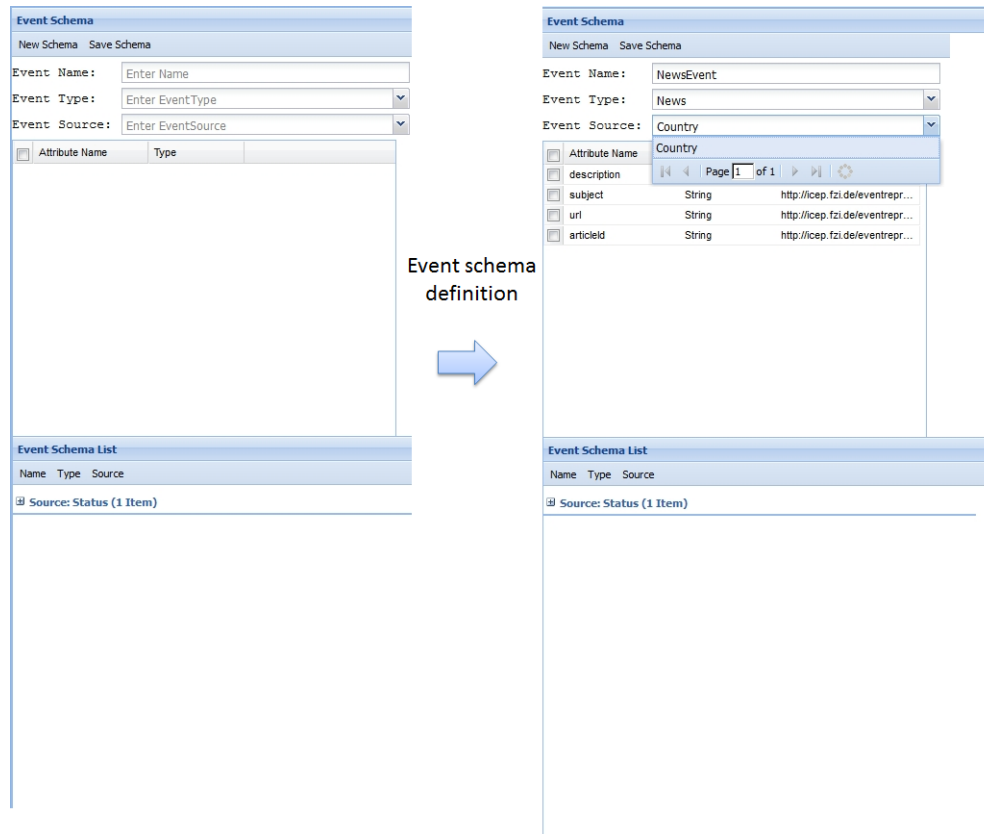
**Definition of the event sources and attributes** The definition of the event sources is similar to the definition of event types like shown in Figure 5.6. Like the definition of event types first the hierarchy of event sources is defined followed by the definition of event source attributes. Figure 5.6 displays the event sources defined in the web data monitoring scenario in Section 5.2.



**Figure 5.6:** Definition of the event sources and attributes in PANTEON event editor.

**Definition of events** The final step of the event (event schema or event template) definition is the creation of the link between the event, event type and event source. The initial state of the event editor is shown in the left part of Figure 5.7. In the right part in Figure 5.7 a new template for the event *NewsEvent* is defined where the event type is *News* and the event source *Country*. The event type and event source are selected either from the type and source hierarchy or typed in the proper fields. In order to ease the selection of event types and event source the proper fields support auto complete functionality like shown for the event source in Figure 5.7. The attribute list displays all the attributes available for the

event type and event source. Every event template is stored in a event template repository. The event template list displays all event templates that have been defined so far. These templates can be browsed by name, event type or event source. These events will be later used by the pattern engineer in order to define a new event pattern.



**Figure 5.7:** Definition of the final event template in PANTEON event editor.

## 5.5 Related work

Most of the work regarding the modelling aspects of events and event patterns is related to the run-time aspects and not to the management aspects of event patterns. Event models exist in different domains like in *Active Databases*, *Event-oriented Spatiotemporal Databases*, *Sensor Networks*, *Multimedia Information Systems*, *Video and Audio Analysis* and *Surveillance Systems* to name a few. There exist also ontologies which include the concept of events, like:

- **Event Ontology** <sup>13</sup>
- **DOLCE+DnS Ultralite** <sup>14</sup>

<sup>13</sup><http://motools.sourceforge.net/event.html>

<sup>14</sup>[http://ontologydesignpatterns.org/wiki/Ontology:DOLCE+DnS Ultralite](http://ontologydesignpatterns.org/wiki/Ontology:DOLCE+DnS_Ultralite)

- GEM [WoHo04]
- LODE <sup>15</sup>

However, the difference between our meta model and these ontologies is that none of these ontologies provide a representation of event patterns which is the crucial point of an event pattern management. In these ontologies events are mainly used in order to define real world events. Additionally the relation between events, complex events, event operators and event patterns is not part of these ontologies which is of course the main purpose of our meta model.

Therefore we describe in this section only event and pattern models from the area of event-based systems that are the most relevant ones for this thesis. Additional event formats from different domains can be found in [Amar11].

The SARI event model [Rozs08] is centred around the concept of event types. Event types are hierarchical structures containing an arbitrary number of event attributes. Each event attribute has an attribute type. Possible attribute types are collection types, dictionary types and single-value types. Single-value types are the basic types such as string or integer. The event types in SARI are managed as libraries which should be valid in the given event processing system. The intention of having event type libraries is to share events and not event patterns among a set of CEP related applications. In contrast to this approach our goal with the meta model is to enable the sharing and reuse of event pattern through event types, event sources and event operators that are hierarchical concepts. These concepts and the proper hierarchy enable several relationship detections between event patterns as it will be described in Chapter 6. Our meta model provides an abstract view on event patterns and the main relationships between the concepts involved in a pattern definition. This abstract view can help pattern engineers to browse, share and evolve event pattern easily which is not in the focus of the approach presented in [Rozs08].

Kharbili [KhSt09] presents a core ontology for modelling events for policies and rules for the compliance management. An *Event* can be either an *Input Event* or an *Output Event*. Each *Event* can be itself implemented as either *Complex Event* or as *Event Stream*. An event can be an event expression which combines events using event operators. Complex events can be a combination of event occurrences or *Event Patterns*. This ontology is the most closest to our meta model. Both our meta model and the ontology presented in [KhSt09] share the concepts *event*, *event operator*, *complex event* and *event pattern*. However, while the ontology presented in [KhSt09] is designed to describe events in the context of business policies and rules for the compliance management our model is independent from any domain. Our model provides additional concepts like the event type and event source hierarchies in order to support the pattern engineer in defining and evolving event patterns (will be described in Chapter 6 and Chapter 7). We

---

<sup>15</sup><http://linkedevents.org/ontology/>



further extended the notion of event patterns with attributes and pattern domain in order to cluster patterns according to their target domain. The clustering will be used in order to detect frequently used event patterns which will be described in Chapter 6.

Scherp et al. [SFSS09] present an event model, called F, based on a set of patterns. The model specializes a version of the DOLCE-Lite and the DNS (Description and Situations) ontologies. The F model, which is derived from the *E* model of Westermann and Jain [WeJa06], supports the following aspects:

- **Participation of objects in events** Both living and non-living objects such as people, animals, and other material objects should be represented by the event model.
- **Temporal duration of events** As events unfold over time, their temporal duration need to be modelled.
- **Spatial extension of objects** Events also unfold over space and hence their spatial extension needs to be modelled too.
- **Structural relationships between events** There exist three kinds of structural relationships between events. These relationships are mereological, causal and correlation relationships. The mereological relationships describe how events are made up of other events (e.g. subevent-of). Causal relationships require the modelling of causes and effects and should support the integration and use of different causal theories. The correlation relationship refers to two events that may or may not have a common cause.
- **Annotability of events** The annotability refers to the ability to associate an arbitrary number of additional information to any event.
- **Event interpretation** Since relationships between events can be a matter of subjectivity and interpretation the model should provide the ability to associate relationships with further attributes.

For each aspect there is an ontology pattern describing the aspect in more detail. Model F covers an event in its broadest sense (living and non-living objects) and hence contains dozens of concepts. The difference to our meta model is that the event model in [SFSS09] is designed to ease the interchange of event information between different event-based components and not for management aspects. It is a heavyweight and expressive ontology that require ontology experts to maintain event patterns based on the model F. Model F is an example for an expressive ontology to represent patterns but not maintainable. This trade-off has been already described in Section 3.1. Compared to the model F the aim of our meta model is maintainability and to support the modelling of event patterns using the frequently used event operators.

## 5.6 Summary

Contemporary event pattern representations are not designed for design-time event pattern management issues. In order to support the event pattern management we presented in this section an event pattern meta model. The meta model serves as the foundation of the graph-based event pattern representation which is extensible and flexible. It is important to note that a proper representation should not only define event pattern related information but also add reasoning capabilities like the subsumption reasoning that will be used for the similarity calculation in Chapter 6. We will use subsumption reasoning to refine and reuse event patterns. We presented a graphical environment to define domain specific events (as a template) based on our meta model. These event templates will be used as an input in order to model event patterns. In the next chapter we describe the usage of this model for the detection of different event pattern relationships to access existing event pattern knowledge and to make use of them for refining new event patterns.

# 6

## Relationships between Event Patterns

In this chapter we present the user-driven pattern modelling in more detail. The main goal is to support the pattern engineer to access existing pattern knowledge, to reuse this knowledge or to derive new knowledge by analysing these patterns. We describe a set of pattern relationships based on the graph representation of an event pattern. Further we present the concept of best practice patterns where the aim is to build a library of patterns that are frequently used for a domain of interest. These concepts are relevant especially for domains where plenty of users interact with the CEP systems managing a large number of event patterns.

### 6.1 Introduction

An increased complexity and size of event patterns make the maintenance of event patterns a difficult task. Hence it is important to detect patterns that may be redundant, disjoint or subsumed by other patterns with respect to other event patterns. Similar problems exist in the rule based community where the generation and maintenance of rule bases are a challenging task (see [MaML89], [NPLP87]) or in ontology pattern design where the aim is to describe a generic recurring construct in ontologies (see [BISa05]).

As we described in Section 4.2 there are different strategies to model a new event pattern. In this thesis we follow a semi-automatic pattern modelling strategy. The semi-automatic strategy extends the pure manual modelling by using techniques from semantic web, graph theory and statistics. These techniques are adopted to

discover relevant pattern candidates and support the pattern engineer during the event pattern modelling process.

## 6.2 Conceptual model

The knowledge engineering process can be categorized into knowledge creation and knowledge reuse [DaJB96]. Although knowledge creation is considered as more important and more difficult to manage, the reuse of knowledge can increase the organizational effectiveness [Dixo00],[Mark01]. Further the storage and access to knowledge is essential for the operational efficiency of an organization [Mark01].

As we described in Chapter 4 modelling event patterns is rather an iterative process with the objective of developing the most suitable event pattern for a given business situation. Within these iterative process steps it is important to enable the pattern engineer to access the pattern repository for event patterns that could be relevant for the modelling of a new business situation. This can be considered as a recommendation during the pattern modelling process taking into account the modelling context of the pattern engineer. Recommender systems are mostly known from E-commerce systems where products are recommended to the user for buying (for example Amazon). Generally, recommendations are based on top overall sellers, on demographics or on the past buying behaviour of the customers [ScKR99].

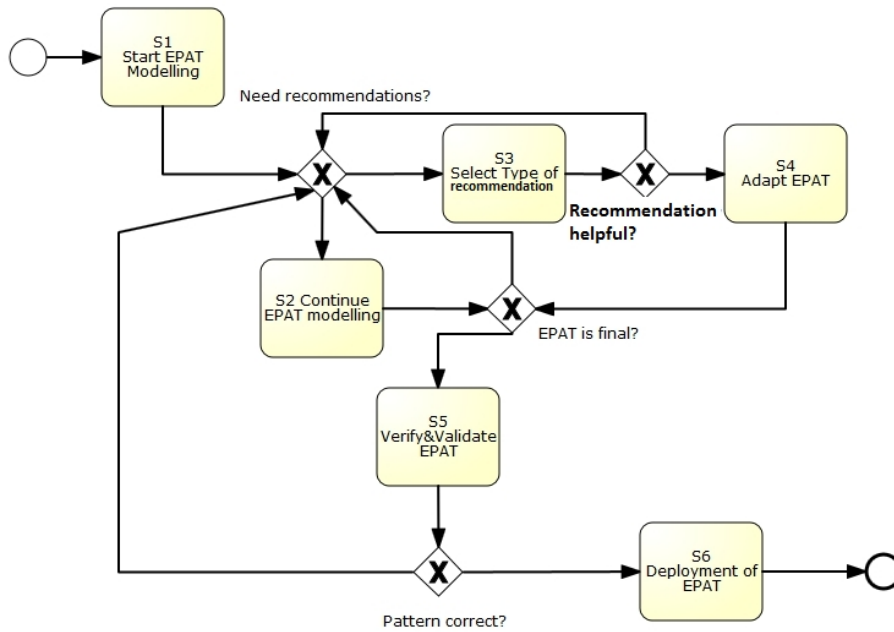
In event processing the recommendation and the reuse of existing pattern knowledge can increase the speed of pattern modelling. The process diagram<sup>1</sup> in Figure 6.1 displays the overview of our event pattern modelling process.

Starting from the need to develop a new event pattern (S1) the goal is to develop the most suitable event pattern using existing pattern knowledge. The initial step of modelling a new event pattern is continued either by selecting the type of the recommendation (S3) or a pure manual definition (S2). Whenever the recommendation is helpful the pattern engineer can continue by adapting the initial pattern. Otherwise the system is requested for additional recommendations. If the pattern is the final one the next step is to verify and validate the pattern (S5) followed by the deployment of the event pattern (S6).

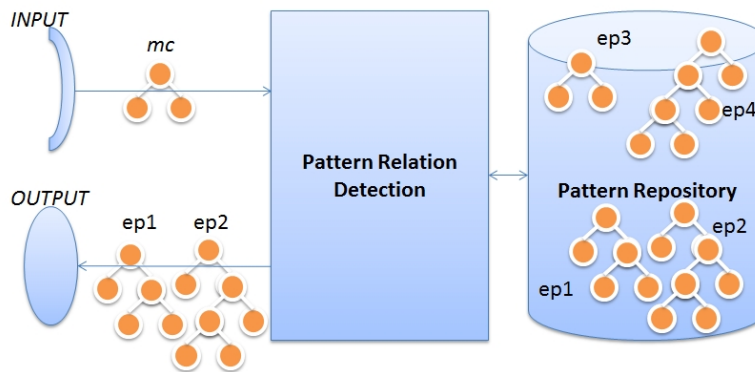
Figure 6.2 displays the conceptual model of event pattern recommendation. The recommendation of event patterns is done through the detection of different type of event pattern relationships. The detection of pattern relationships has the modelling context, *mc*, as input. Taking into account the modelling context and the pattern repository, the result of the relationship detection contains the most relevant event patterns based on different types of relationship detection approaches which will be described further in this chapter.

---

<sup>1</sup>in BPMN 2.0 notation



**Figure 6.1:** The process of new event pattern (EPAT) development.



**Figure 6.2:** Conceptual model of pattern suggestion.

**Definition: Modelling context** A modelling context, denoted as  $mc$ , is a partially defined event pattern that represents the current event pattern development progress containing all elements of the meta model (pattern domain, events, the event operators, property value definitions and the links between events and event operators).

The initial modelling context is empty. The smallest number of elements in the modelling context that will be considered for the recommendation generation is 1. It can be either an event or an event operator. The modelling context is the reference object that is compared to the elements of the event pattern repository.

**Definition: Event pattern repository** Event pattern repository, denoted as  $eps$ , is a container where every event pattern  $ep$  is serialized as a directed acyclic graph.

**Definition: Event pattern relation** The relation between the modelling context  $mc$  and an event pattern  $ep$  is defined through the graph intersection of the modelling context  $mc$  and  $ep$ . If the intersection is empty, the modelling context  $mc$  and the  $ep$  are said to be disjoint, denoted as  $mc \cap ep = \emptyset$ . Otherwise if  $mc \cap ep = mc = ep$  then  $mc$  and  $ep$  are said to be equal.

Based on these definitions the algorithm below describes the high level pattern relation detection.

- 1: given a modelling context  $mc$  and an event pattern repository  $eps$
- 2: **for all**  $ep \in eps$  **do**
- 3:   **if**  $relDetected(mc, ep)$  **then**
- 4:      $\delta \leftarrow dist(mc, ep)$
- 5:      $resList \leftarrow ep, \delta$
- 6:   **end if**
- 7: **end for**

The function  $relDetected()$  delivers *four* types of relations namely, *subpattern*, *overlapping*, *similarity* and *best practices* which will be described in Section 6.4 and Section 6.5. Two patterns that do not have a subpattern or overlapping relation are disjoint. The  $dist()$  function delivers the distance between the event pattern and the modelling context in order to rank the results according to the distance value between an event pattern  $ep$  and the modelling context  $mc$ . This value is used for the ranking of the results. In our approach we used for calculated the ranking by subtracting the number of nodes in  $ep \in eps$  from the number of nodes in  $mc$ ,  $|ep| - |mc|$ . If there exists a relationship between an  $ep$  and the  $mc$  and  $\delta = 0$  then  $ep$  and  $mc$  are said to be equal. Otherwise if  $\delta$  is smaller than zero the  $ep$  is said to be more general and otherwise if  $\delta$  is greater than zero.

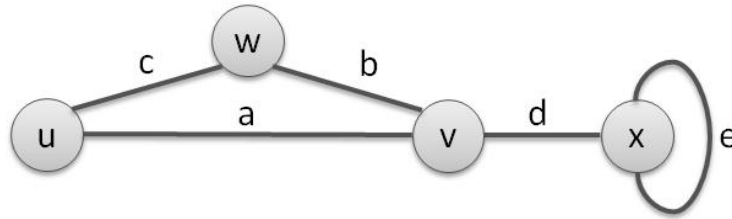
In order to detect the relationships between event patterns we use the graph traversation. The event pattern graph is a connected graph without any circles and hence forms a tree. Before describing the different approaches we will describe some graph theoretical aspects that are relevant for this chapter.

### 6.3 Graph theory

Graph theory is a model used for the description of relationships among a collection of items. Many real-world problems can be modeled as graphs. We introduce in this section relevant fundamentals of graph theory from [Bond76], [Wils96].

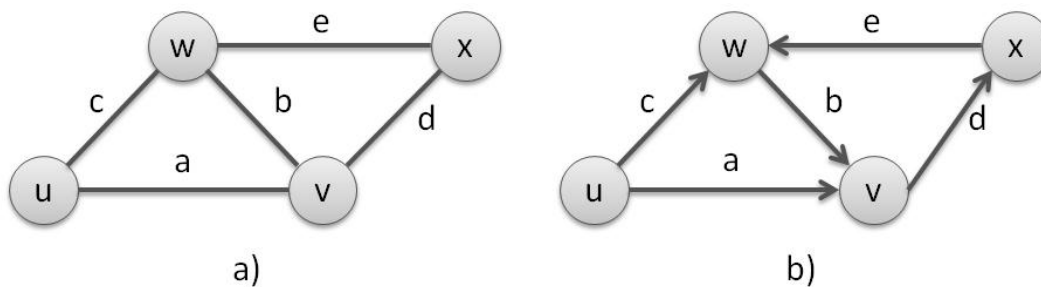
A graph  $G = (V, E)$  consists of a set of elements  $V := \{v | v \in V\}$ , called *nodes* (or *vertices*), and a set of elements  $E := \{e | e \in E\}$ , which connects pair of elements in  $V$ , called *edges* (or *arcs*). If an edge  $e$  connects two nodes  $u$  and  $v$ , it is said to *join* the nodes  $u$  and  $v$ ,  $u$  and  $v$  are called *ends* of the edge  $e$ .  $V(G)$ ,  $V_G$ , and

$E(G)$ ,  $E_G$ , are used as notations of nodes and edges for a certain graph  $G$ , and they are simplified as  $V$  and  $E$  [Bond76]. Figure 6.3 displays a simple example of a graph. The circles present the nodes and the lines between them present the edges.  $V(G) = \{u, v, w, x\}$ ,  $E(G) = \{a, b, c, d, e\}$ .



**Figure 6.3:** A simple example of graph  $G$  with nodes  $\{u, v, w, x\}$  and edges  $\{a, b, c, d, e\}$ .

A *subgraph* of graph  $G$  is a graph which has a subset of  $G$ 's nodes as its nodes and a subset of  $G$ 's edges as its edges. If a graph  $H$  is a subgraph of graph  $G$ ,  $G$  is called *supergraph* of  $H$ . For example, in Figure 6.3, nodes  $u, v, w$  and edges  $a, b, c$  can build up a subgraph of graph  $G$ . Graphs with directed edges are called *directed graphs*, vice versa *undirected graphs*. Figure 6.4 is a comparison of an undirected graph in a) and a directed graph in b).



**Figure 6.4:** A directed graph and an undirected graph: a) undirected graph  $G$ ; b) directed graph  $H$ .

If there is a  $(u, v)$ -path for nodes  $u$  and  $v$ , these two nodes are said to be *connected*. A graph is *connected*, if there is always a path for every two nodes of the graph. Otherwise this graph is *disconnected*. The connected subgraphs of a graph are *components* of the graph.

A graph containing no circles is called *forest*, and if a forest is connected, it is a *tree*. A tree is a simple type of graphs with additional properties.

## 6.4 Graph relationships between event patterns

We introduce the following three event pattern relationship levels considering:

- **Relationship on the node type level** The relationship on the type level considers only the type of the elements in the modelling context which are

*Simple Event (SE), Complex Event<sup>2</sup> (CE), Event Operator (EO), Event Type (ET), Event Source (ES).*

- **Relationship on the node instance level** The relationship on the instance level extends the relationship on the node type level to that effect that the concept instantiations are considered. Concepts that can be instantiated are *ET*, *ES*, *EO*, *SE* and *CE*.
- **Relationship on the node property instantiation level** The property level relationship extends the instance level relationship to that effect that also the attribute values of each instance *ET*, *ES*, *EO*, *SE* and *CE* are considered for the relationship detection.

The event pattern relationship levels enable a flexible event pattern recommendation in the sense that the pattern engineer is able to select the concrete relationship (described below) and the level of this relationship (described above) in order to adjust the recommendation results.

In the following subsections we present the specific event pattern relationships including the overlapping and subsumption relationship. Further we describe the detection of domain-dependent best practice event patterns. The role of best practice patterns is to build automatically pattern libraries which can be delivered to the pattern engineer. The relationship levels presented above are applied to the subsumption, overlapping and best practice pattern detection. Finally we introduce a similarity metric for detecting similar event patterns through the taxonomy and the features of an event pattern.

### 6.4.1 Event pattern subsumption and overlapping

The assumption behind the event graph relationships is that the event pattern modelling starts with a more general pattern and is specialized in order to describe a relevant business situation. In order to incorporate something specific under a more general category we introduce the *overlapping* and *subpattern* relationships.

The event pattern *overlapping* is derived from the subgraph definition. From the graph theory we know that a graph  $G' = (V(H), E(H))$  is called a subgraph of graph  $G = (V(G), E(G))$  if  $V(G') \subseteq V(G)$  and  $E(G') \subseteq E(G)$ .

**Definition: Event pattern overlapping** A pattern  $ep'$  has an overlapping with another pattern  $ep$ , denoted as  $ovl(ep', ep)$  if  $ep'$  and  $ep$  have a common subgraph denoted as  $S = (E, V)$ . Further  $V(S) \subset V(ep)$  and  $V(S) \subset V(ep')$  and  $E(S) \subset E(ep)$  and  $E(S) \subset E(ep')$ .

<sup>2</sup>In the meta model complex events are events that are connected to an event operator by the attribute *hasOperator*. See definition in Section 5.2



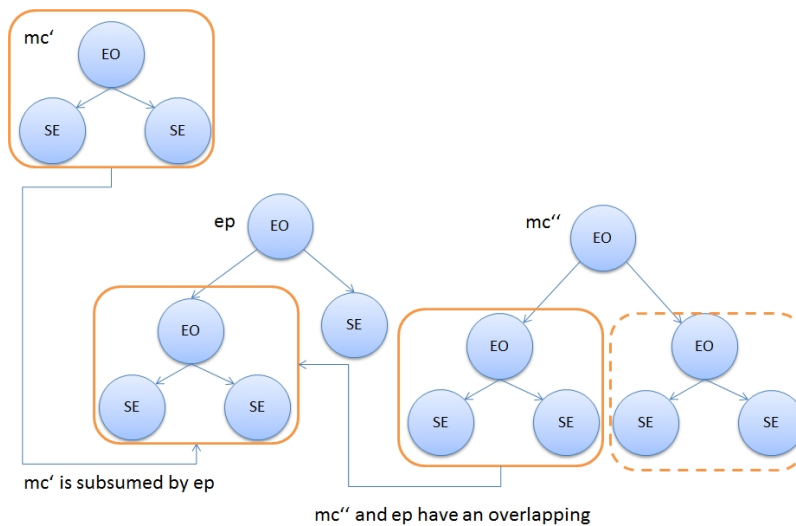
The minimal number of nodes in a common overlapping subgraph is one. On the other hand if the number of nodes in  $S$  is equal to the number of nodes in  $ep$  we have a subpattern relation between  $ep$  and  $ep'$ .

**Definition: Event pattern subpattern** A pattern  $ep'$  is a subpattern of another pattern  $ep$ , denoted as  $sub(ep', ep)$  if  $ep'$  is a subgraph of  $ep$ .  $ep$  is said to be a superpattern of  $ep'$ , denoted as  $sup(ep, ep')$ .

The subpattern relation is a special form of the overlapping relationship where the modelling context is a complete subgraph of another event pattern. That means that every subpattern is an overlapping but not vice versa.

**Relationship on the node type level** The overlapping and subsumption relationship on the node type level aims at delivering patterns that strictly consider the types of the nodes in the event pattern graph without considering the specific event type and event source of the events. Therefore the concepts that are considered are event (SE) and event operator (EO).

The detection of relationship on the type level delivers only patterns that share the same graph structure without taking into account the concrete instantiation of the event type, source, operator and their values. In this sense the pattern relationship on the node type level is the most abstract relationship between two event patterns.

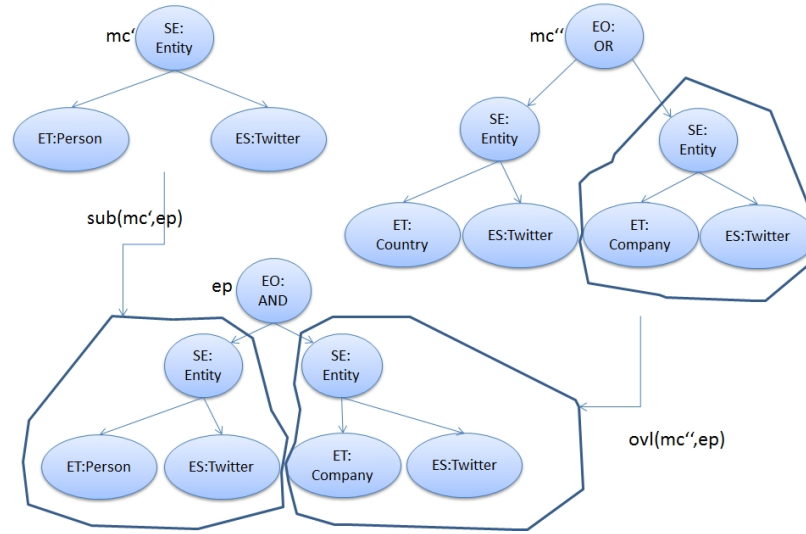


**Figure 6.5:** Example: Overlapping and subsumption relation on the type level (SE = simple event, EO = event operator).

In the example in Figure 6.5 we have two modelling contexts  $mc'$ ,  $mc''$  and the event pattern  $ep$ . Between  $mc'$  and  $ep$  there exists a subsumption relation where  $mc'$  is subsumed by  $ep$ . The distance  $\delta$  between  $mc'$  and  $ep$  is 2 since  $|mc'| = 3$  and  $|ep| = 5$ . Between  $mc''$  and  $ep$  we have an overlapping relation ( $SE EO SE$ ). The distance  $\delta$  between  $mc''$  and  $ep$  is -2 since  $|mc''| = 7$  and  $|ep| = 5$  which

means that  $mc''$  is more specific than  $ep$ . However, between  $mc''$  and  $ep$  there exist another overlapping relationship displayed in dotted lines in Figure 6.5.

**Relationship on the node instance level** The next type is the relationship between two patterns on the instance level. A pattern that satisfies the overlapping and subsumption relationship on the instance level satisfies also the relationship on the node type level. The relationship on the node instance level considers the concrete instantiation of the concepts that are part of the event pattern which are  $EO$ ,  $SE$ ,  $ET$  and  $ES$ .



**Figure 6.6:** Example: Overlapping and subsumption on the instance level.

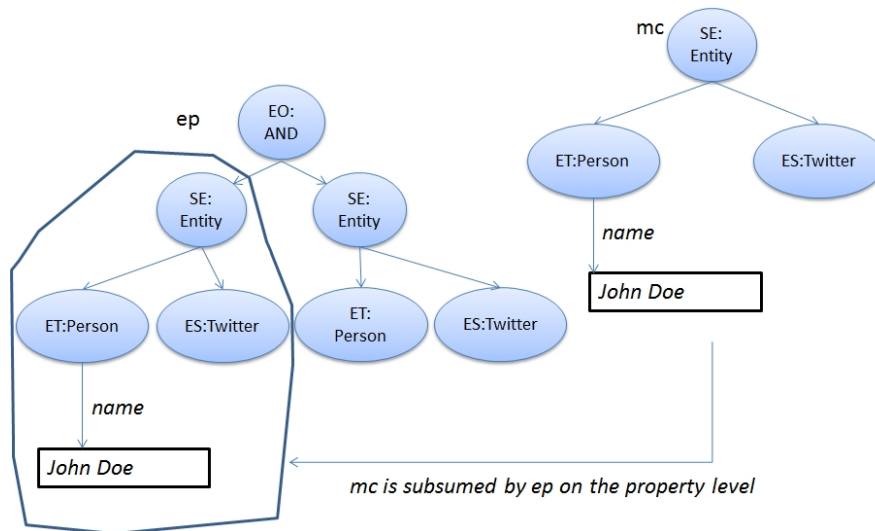
In the example in Figure 6.6 we have two modelling contexts  $mc'$  and  $mc''$  and an event pattern  $ep$ .  $mc'$  denotes a single event where the event denotes the publishing of a tweet containing information about a person.  $mc''$  denotes an event pattern describing a situation where either a tweet containing a country or a company is published.

While  $mc'$  is a subpattern of  $ep$  on the instance level,  $mc''$  and  $ep$  have a common overlapping on the instance level.

The overlapping and subpattern relationships on the instance level are true since the following conditions are true for  $mc'$ ,  $mc''$  and  $ep$ :

- The event shared between  $mc'$ ,  $mc''$  and  $ep$  is the simple event  $SE:Entity$ .
- The event source of the event  $SE:Entity$  in  $mc'$ ,  $mc''$  and  $ep$  is the event source  $ES:Twitter$ .
- The event type of  $SE:Entity$  in  $mc'$  and  $ep$  is  $ET:Person$ . The event type of  $SE:Entity$  in  $mc''$  and  $ep$  is  $ET:Company$ .

**Relationship on the node property level** The relationship on the node property level is based on the structure equality of the node. This relationship extends the relationship on the instance level by taking into account the properties and their values. A node  $n$  is detected structural equal in two event patterns  $ep', ep''$  if  $n \in V(ep')$  and  $n \in V(ep'')$ . Further the node can be interchanged between these event patterns without changing the semantics of the pattern. That means that the situation that has been covered by the event pattern stays as it was previously defined. In the example in Figure 6.7 the modelling context  $mc$  is subsumed by



**Figure 6.7:** Example: Overlapping on the the property level. In this case the overlapping is a subsumption.

the event pattern  $ep$ . In this example the event type  $ET:Person$  has a property node  $name$  with "John Doe" as its value.

## 6.4.2 Best practice event pattern detection

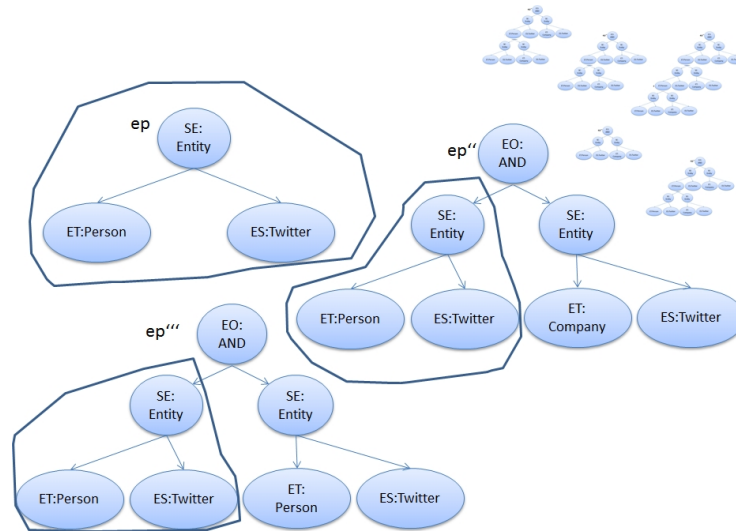
Taking into account the graph relationships between two event patterns we introduce in this section the concept of best practice event patterns. The concept is derived from the frequent item set discovery which has been used in data mining to discover association rules (see [AgIS93]).

**Definition: Best Practice Event Pattern** An event pattern subgraph,  $sg$ , is detected as a best practice event pattern, denoted  $bpp$ , if the quotient of the number of event patterns containing  $|sg|$  and the total number of event patterns in an event pattern repository  $eps$  is greater than a given threshold  $\tau$ ,  $\frac{|sg|}{|eps|} \geq \tau$ .

A subgraph  $sg$  is detected as  $bpp$  if the quotient is greater or equal than the threshold  $\tau$ . The value of  $\tau$  indicates the minimum coverage percentage of the  $bpp$  in an event pattern repository. The value of  $\tau$  is application domain dependent and

needs to be defined by the pattern engineer. In our experiments we set the value of  $\tau$  to 0.3 or 30%. Another pragmatic approach in order to determine a subgraph as a best practice pattern is to identify every subgraph that is shared at least by two event patterns. The presentation of the best practice results to the pattern engineer is based on a ranking which represents the number of event patterns that includes the same subgraph. This approach would enable the pattern engineer to find also relevant best practice patterns that are rather rare and wouldn't pass the threshold value  $\tau$ .

A best practice is defined<sup>3</sup> as a technique or methodology that, through experience and research, has proven to reliably lead to a desired result. In software development, a best practice is a well-defined method that contributes to a successful step in product development. The set of *bpp* is provided to the pattern engineer at any time during the pattern modelling process. Since the detection of *bpp* is based on the event pattern relationships presented in the previous section, the detection of *bpp* is also based on the same relationship types.



**Figure 6.8:** Best practice pattern on the instance level.

Assumed we set  $\tau = 0.3 (= 30\%)$ , and calculate the *bpp* for the patterns shown in Figure 6.8. In this example the pattern repository has 8 event patterns in total. The result is that the event pattern  $SE:ENTITY(ET:PERSON, ES:TWITTER)$  is detected as the most frequent subgraph on the instance level. This is because  $\frac{|sg|}{|eps|} \geq \tau$ ,  $\frac{|3|}{|8|} \geq 0.3$ . The *bpp* is a subgraph of  $ep'$ ,  $ep''$  and  $ep'''$ . The detection of a *bpp* on the type and property level is done in a similar way.

The role of a *bpp* is to help the pattern engineer to select the most used pattern fragments from a pattern, configure and extend the pattern according to the business situation needs instead of modelling the pattern from scratch. That might help increase the speed of the pattern definition process since the best practice patterns are forming a library of most relevant pattern for a given domain. The usage

<sup>3</sup><http://searchsoftwarequality.techtarget.com/definition/best-practice>

of *bpp* can also support the building of pattern templates for different application domains like described in [Kulk11].

## 6.5 Event pattern similarity

The goal of the similarity is to find event patterns that are conceptually close but not identical to the modelling context. In computer science similarity is used as an integral part of information processing such as information retrieval or information integration.

The objective of the similarity calculation is to derive a function  $sim(ep', ep'')$ ,  $ep', ep'' \in eps$  that measures the *distance* between the event patterns  $ep'$  and  $ep''$  (see also [HZAW<sup>+</sup>06]).

**Definition: Event Pattern Similarity** Given a set of event patterns EP, the similarity of two event patterns is defined as a numerical value between 0 and 1.  $sim(ep', ep'') : EP \times EP \rightarrow [0, 1]; ep', ep'' \in EP$ .

The similarity function maps event patterns into the unit interval  $[0, 1]$ . The value 1 occurs only if  $ep' = ep''$ , which is an exact match. In case that two event patterns have nothing in common the value 0 is derived. The similarity between two event patterns is symmetric, meaning  $sim(ep', ep'') = sim(ep'', ep')$ .

To derive a more general similarity framework for event patterns we combine multiple similarity metrics in a single aggregation similarity function. This aggregation function is derived from [MaZa02] and [EHHS04].

$$sim_A(ep', ep'') := \frac{\sum_{i=1}^n \omega_i * sim_i(ep', ep'')}{n}$$

The importance of each similarity metric is defined through  $\omega$  (interval  $[0, 1]$ ). The default value of  $\omega$  is 1, which indicates that the similarity metrics are equally important for the overall similarity value.  $\omega$  provides the flexibility of adjusting the importance of each similarity calculation depending on the application domain where some similarity metrics are considered to be more important.  $n$  indicates the number of similarity measures that have been used. The similarity calculation of event patterns is derived from the similarity research in the semantic web where similarity is classified into taxonomy-based similarity, feature-based similarity and similarity based on information-content (see [ZJHN<sup>+</sup>07]). In this thesis we adapt the taxonomy-based and feature-based similarity described in [AnBK03] and [MSSS<sup>+</sup>01] to the event patterns to detect similarity between event patterns.

### 6.5.1 Taxonomy-based event pattern similarity

The taxonomy-based approaches calculate the similarity of terms by evaluating their position within a given taxonomy. They take into account that closely related terms are grouped together while distantly related terms are spaced more widely apart.

The meta model described in Chapter 5 has the benefit that the event pattern concepts are modelled as a tree structure. Since the concepts *ET*, *ES*, *EO* and *PD* are organized as a hierarchy, more general concepts should be located closer to the root of the hierarchy, while more specific ones are located closer to the leaves. A simple metric for terms arranged as nodes in a directed acyclic graph such as a hierarchy would be the minimal distance between the two term nodes so that similarity between two terms could be defined as the length of the shortest path between the two nodes (see also [PPPC07]). Alternatively, the similarity measure presented by [WuPa94] finds the most specific common concept that subsumes both considered concepts. On the other hand, [MSSS<sup>+</sup>01] introduced the upwards cotopy (UC) to measure the similarity considering their super concepts and relative places in a common hierarchy.

The taxonomy-based similarity,  $sim_{tx}$ , calculates the similarity for events and event operators used in a modelling context *mc*. The nearer the concepts are in the hierarchy the more similar they are. In order to derive the overall similarity between the modelling context *mc* and event pattern *ep* we calculate the upwards cotopy for event type, event source and event operator like described below. The upwards cotopy is the underlying measure to compute the semantic distance in a concept hierarchy.

**Event type similarity:**  $sim_{tx_{et}}(mc, ep) = \frac{H_{et}(mc) \cap H_{et}(ep)^4}{H_{et}(mc) \cup H_{et}(ep)}$  where  $H_{et}(mc) \cap H_{et}(ep)$  is the number of event type concepts (from the event type hierarchy) the event types of the modelling context *mc* and the event pattern *ep* have in common and  $H_{et}(mc) \cup H_{et}(ep)$  is the number of distinct concepts (from the event type hierarchy) *mc* and *ep* have in total.

**Event source similarity:**  $sim_{tx_{es}}(mc, ep) = \frac{H_{es}(mc) \cap H_{es}(ep)}{H_{es}(mc) \cup H_{es}(ep)}$  where  $H_{es}(mc) \cap H_{es}(ep)$  is the number of event source concepts (from the event source hierarchy) the event sources of the modelling context *mc* and the event pattern *ep* have in common and  $H_{es}(mc) \cup H_{es}(ep)$  is the number of distinct event source concepts (from the event source hierarchy) *mc* and *ep* have in total.

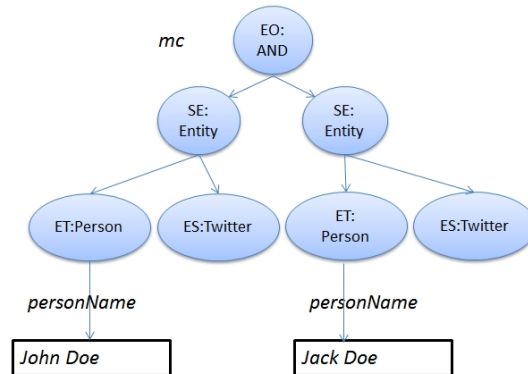
**Event operator similarity:**  $sim_{tx_{eo}}(mc, ep) = \frac{H_{eo}(mc) \cap H_{eo}(ep)}{H_{eo}(mc) \cup H_{eo}(ep)}$  where  $H_{eo}(mc) \cap H_{eo}(ep)$  is the number of event operator concepts (from the event operator hierarchy) the event sources of the modelling context *mc* and the event pattern *ep* have in common and  $H_{eo}(mc) \cup H_{eo}(ep)$  is the number of distinct event operator concepts (from the event operator hierarchy) *mc* and *ep* have in total.

<sup>4</sup> $\cap$  denotes the intersection and  $\cup$  denotes the set union.

Taking the similarity of  $ET$ ,  $ES$  and  $EO$  into account we define the taxonomy based similarity  $sim_{tx}$  as  $sim_{tx}(mc, ep) = \frac{1}{3} \cdot (sim_{tx_{et}} + sim_{tx_{es}} + sim_{tx_{eo}})$ .

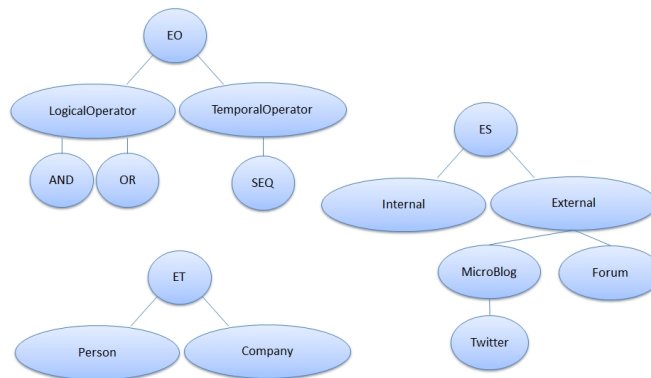
### Example

The modelling context  $mc$  in Figure 6.9 contains an event pattern where the cooccurrence (AND) of two event entities (SE:Entity) of type person (ET:Person) occurs in Twitter (ES:Twitter).



**Figure 6.9:** Example: Modelling context as input for the taxonomy-based similarity.

Since the taxonomy-based similarity is based on the concept taxonomy of the concepts used in an event pattern, Figure 6.10 displays a more specific hierarchy for  $H_{et}$ ,  $H_{es}$  and  $H_{eo}$ .

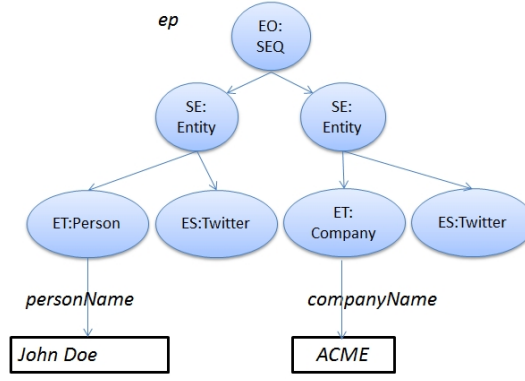


**Figure 6.10:** Example: Taxonomy of event type, event source and event operator ( $H_{et}$ ,  $H_{es}$ ,  $H_{eo}$ ).

Figure 6.11 displays the event pattern  $ep$  that will be used for the similarity calculation. The pattern describes a business situation where an event entity with event type  $ET : Person$  occurs before (SEQ) another event entity with event type  $ET : Company$ . Both events has the same event source  $ES : Twitter$ .

The set of concepts for  $mc$  and  $ep$  including the root concepts  $ET$ ,  $ES$  and  $EO$  are:

- $H_{et}(mc) = \{Person, ET\}$



**Figure 6.11:** Event pattern,  $ep \in EPS$ , which will be used for the taxonomy-based similarity calculation.

- $H_{es}(mc) = \{Twitter, MicroBlog, External, ES\}$
- $H_{eo}(mc) = \{AND, LogicalOperator, EO\}$
- $H_{et}(ep) = \{Person, ET, Company\}$
- $H_{es}(ep) = \{Twitter, MicroBlog, External, ES\}$
- $H_{eo}(ep) = \{SEQ, TemporalOperator, EO\}$

The result of the taxonomy-based similarity  $sim_{tx}(mc, ep) = \frac{1}{3} \cdot (\frac{2}{3} + \frac{4}{4} + \frac{1}{6}) = \frac{1}{3} \cdot 1,83 = 0,61 = 61\%$ . Since the concepts of an event pattern includes also properties we present as next the feature-based similarity for event patterns.

### 6.5.2 Feature-based event pattern similarity

Feature-based similarity assumes that each event type, event source and event operator have arbitrary properties called features. The more common features they have the more similar they are. While common features tend to increase the similarity, non-common features decrease the similarity. Based on [HiXH96] we calculate the similarity through the quotient of shared features and all features of  $ET$ ,  $ES$  and  $EO$ .

Given a modelling context  $mc$  and an event pattern  $ep \in EPS$  the feature-based similarity is defined as:

$$sim_{ft}(mc, ep) = \frac{1}{3} \cdot \left( \frac{ft_{etmc} \cap ft_{etep}}{ft_{etmc} \cup ft_{etep}} + \frac{ft_{esmc} \cap ft_{esep}}{ft_{esmc} \cup ft_{esep}} + \frac{ft_{eomc} \cap ft_{eopep}}{ft_{eomc} \cup ft_{eopep}} \right)$$

The function  $sim_{ft}$  delivers the intersection of features in proportion to the union of all features for  $mc$  and  $ep$ .

**Example** Taking the modelling context  $mc$  and the event pattern  $ep$  from the



previous example the result of the feature-based similarity is calculated as shown below.

The set of features for *mc* and *ep* are:

- $ft_{et}(mc) = \{personName\}$
- $ft_{es}(mc) = \{\}$
- $ft_{eo}(mc) = \{\}$
- $ft_{et}(ep) = \{personName, companyName\}$
- $ft_{es}(ep) = \{\}$
- $ft_{eo}(ep) = \{\}$

The result of the feature-based similarity  $sim_{ft}(mc, ep) = \frac{1}{3} \cdot (\frac{1}{2} + 0 + 0) = 0,166 = 16,6\%$

The aggregated similarity, where  $\omega = 1$  and  $n = 2$ , is:  $sim_A(mc, ep) = \frac{0,166+0,61}{2} = 0,388 = 38,8\%$ .

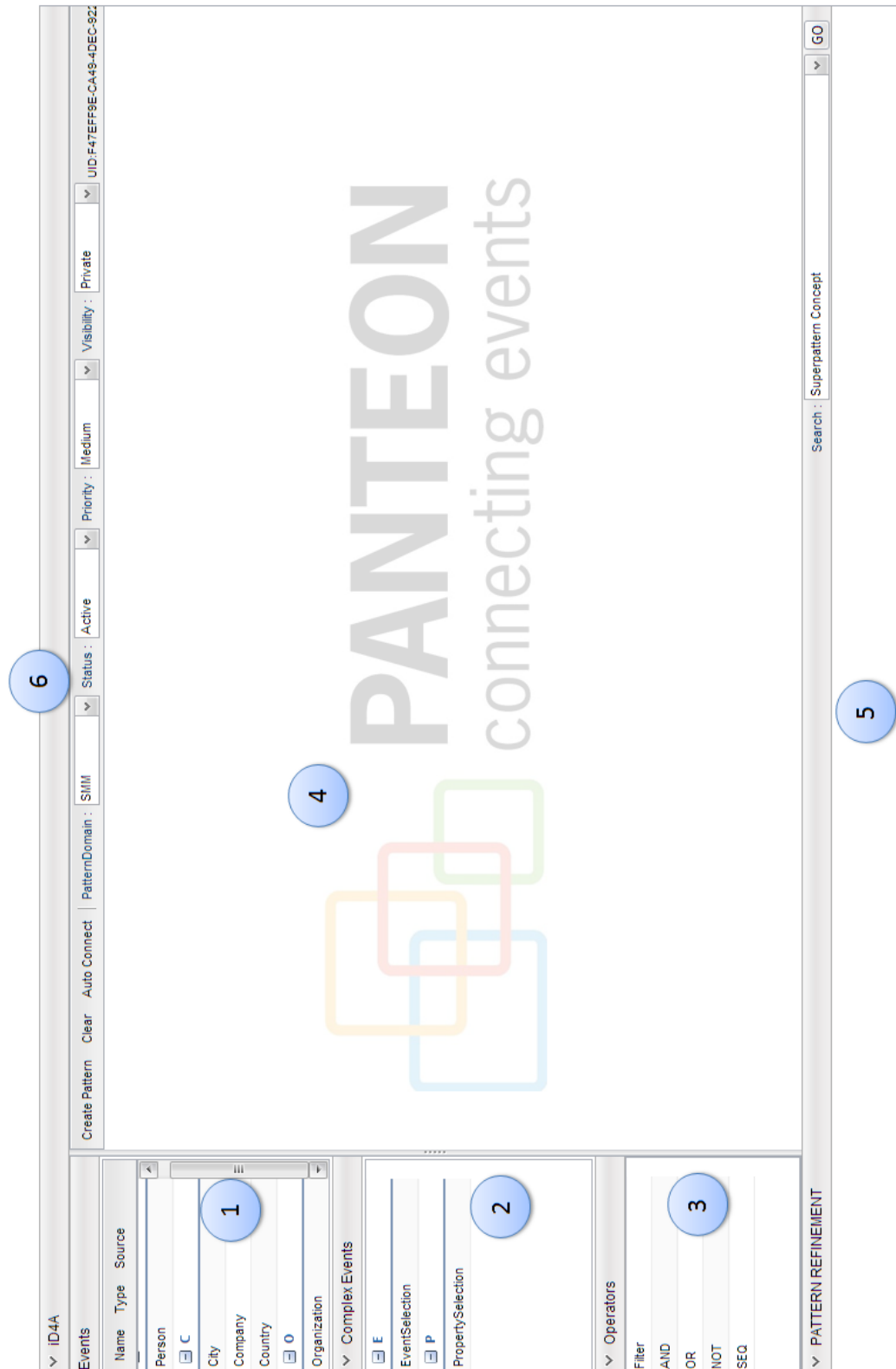
## 6.6 Implementation

In this section we describe the implementation highlights of the pattern modelling in PANTEON. We present the functionalities provided by PANTEON in order to support the pattern engineer during the pattern generation phase.

### 6.6.1 User interface

As described below the user interface for pattern modelling (see Figure 6.12) consists of 7 sections.

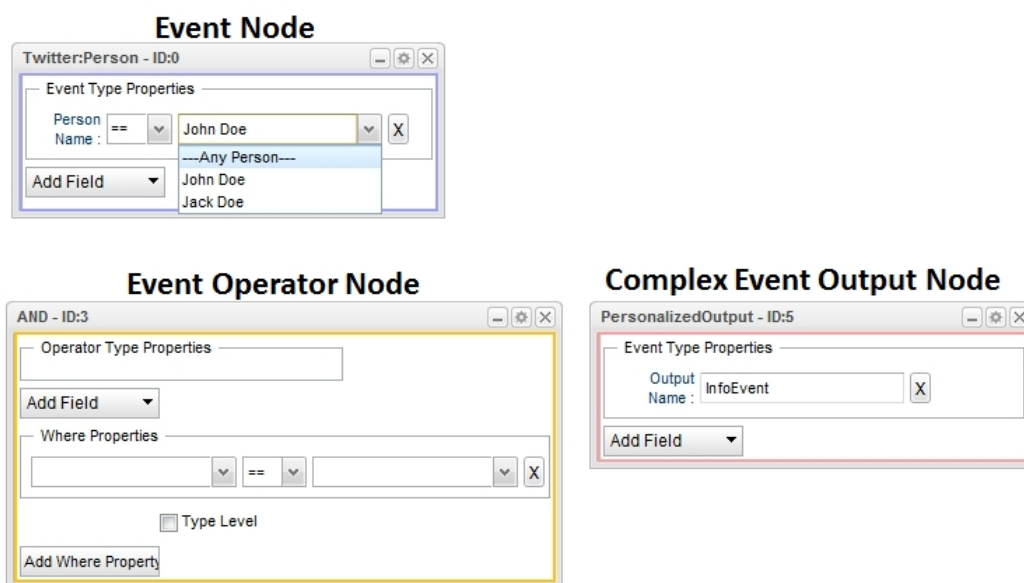
1. **Event Panel** Provides the list of all available events modelled with the PANTEON event editor (described in Section 5.4) that serve as input for new event patterns.
2. **Complex Event Panel** Provides the list of complex events for a new event pattern.
3. **Operator Panel** Provides the list of available event operators that serve as connectors within an event pattern.



**Figure 6.12:** The pattern modelling environment in PANTEON.

4. **Main Panel** Provides the design capabilities (the connection of events with event operators and their proper configuration) that are needed to design new event patterns. Events, event operators and complex events that are provided in the list are represented as a graphical node within this section like shown in Figure 6.13.
5. **Relation Panel** Provides the list of pattern relations described in Section 6.4 and Section 6.5. The results of the relation detection are displayed as selectable pattern icons.
6. **Pattern Control Panel** Provides functionalities like the deployment, auto connection of events with an event operator. Additionally the panel provides the information that is needed in order to define the pattern domain, the status (active, inactive) of the pattern, the priority (high, medium and low).

Figure 6.13 displays the graphical elements that will be used to create a new event pattern. The event node contains a list of event type and event source attributes (for example Person Name). For each attribute one can specify an attribute value. In Figure 6.13 a list of names is provided in order to define a tweet event where a person is mentioned. The event operator node provides a list of operator type



**Figure 6.13:** Graphical presentation of events, event operators and complex events.

properties (for example *count* or *timeWindow*). Constraints on the input events can be defined by using the where properties. For example one can specify that all input events should have one attribute value in common. This constraints can be defined based on the name or data type of the attribute. The complex event node provides a list of properties where custom selected properties from the input events can be added.

## 6.6.2 User interaction

The process of getting the list of relevant event pattern consists of four steps like displayed in Figure 6.14.

First the pattern engineer needs to model an initial event pattern which represents the modelling context of the user (1) (see also Figure 6.15) . In Figure 6.14 the modelling context consists of the *PersonEvent*. Once the initial event pattern is defined the next step is to select one of the relationships presented in the previous section and the proper event pattern relationship level. These relationships are:

- **Subsumption on the node type level**
- **Subsumption on the instance type level**
- **Subsumption on the property instance level**
- **Overlapping on the node type level**
- **Overlapping on the instance type level**
- **Overlapping on the property instance level**
- **Best practice pattern on the node type level**
- **Best practice pattern on the instance type level**
- **Best practice pattern on the property instance level**
- **Similar event pattern**

In Figure 6.14 the selected relationship is the subpattern relationship on the property instance level. The results of the relationship detection are presented as pattern icons in view number 2 (see also Figure 6.16). From the displayed icons one can directly see how many events and event operators are used. This is done through the different colors used for different nodes (events are displayed as blue rectangles, operator as orange rectangles, and complex events as bright-red rectangles). The details of an event pattern are presented in view number 3 (see also Figure 6.17). In view number 3 the pattern engineer is able to select the pattern as an additional input to the initial modelling context. View number 4 (see also Figure 6.18) displays the extension of the initial event pattern taking additional nodes from the pattern of interest. This process is iterative meaning that the event pattern in view number 4 can again be the modelling context in order to receive additional event patterns.

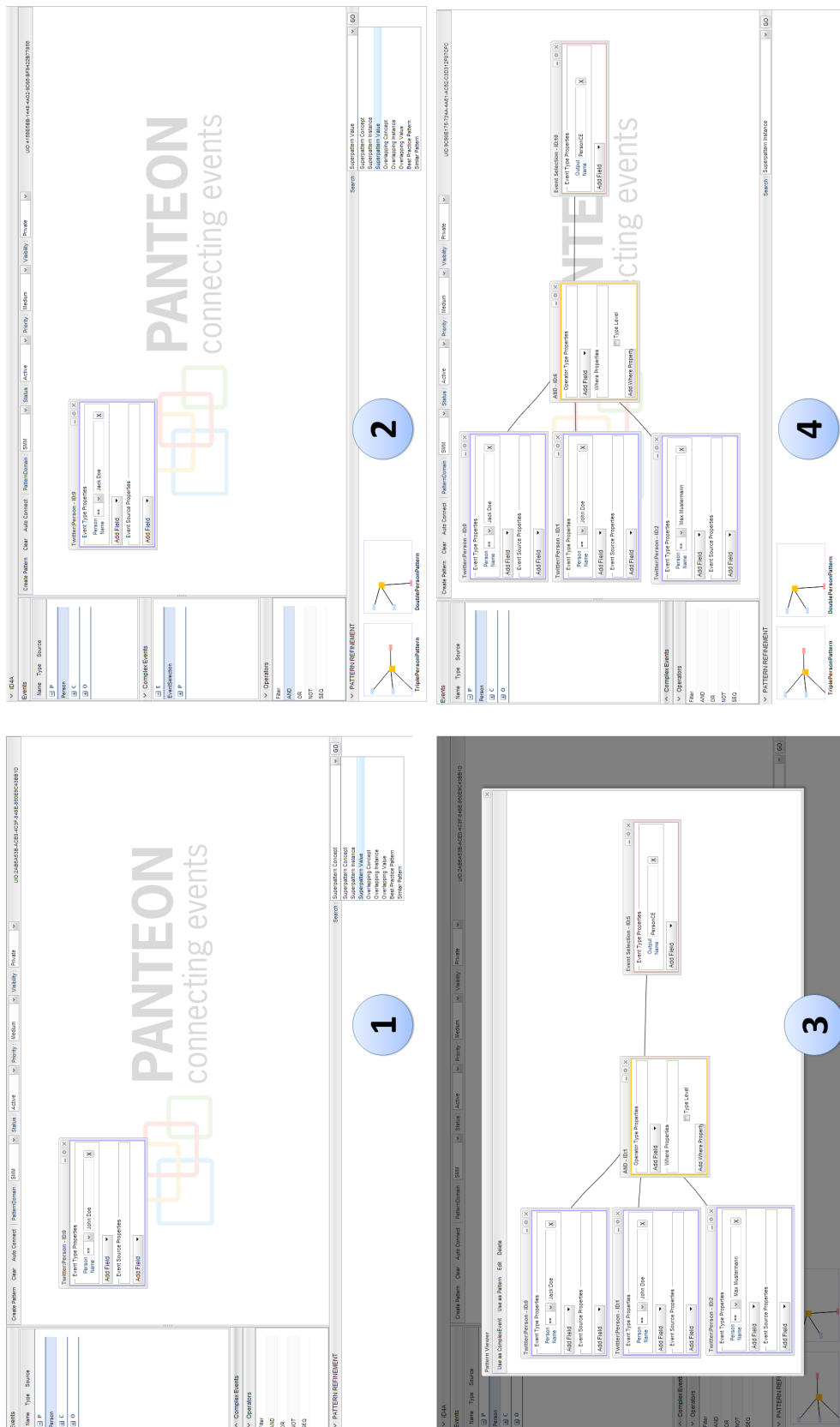


Figure 6.14: 1 - Definition of the modelling context and the relationship selection, 2 - Presentation of the results as pattern icons, 3 - Selection of the relevant event pattern from the result list and 4 - Extension of the initial modelling context with additional event nodes.

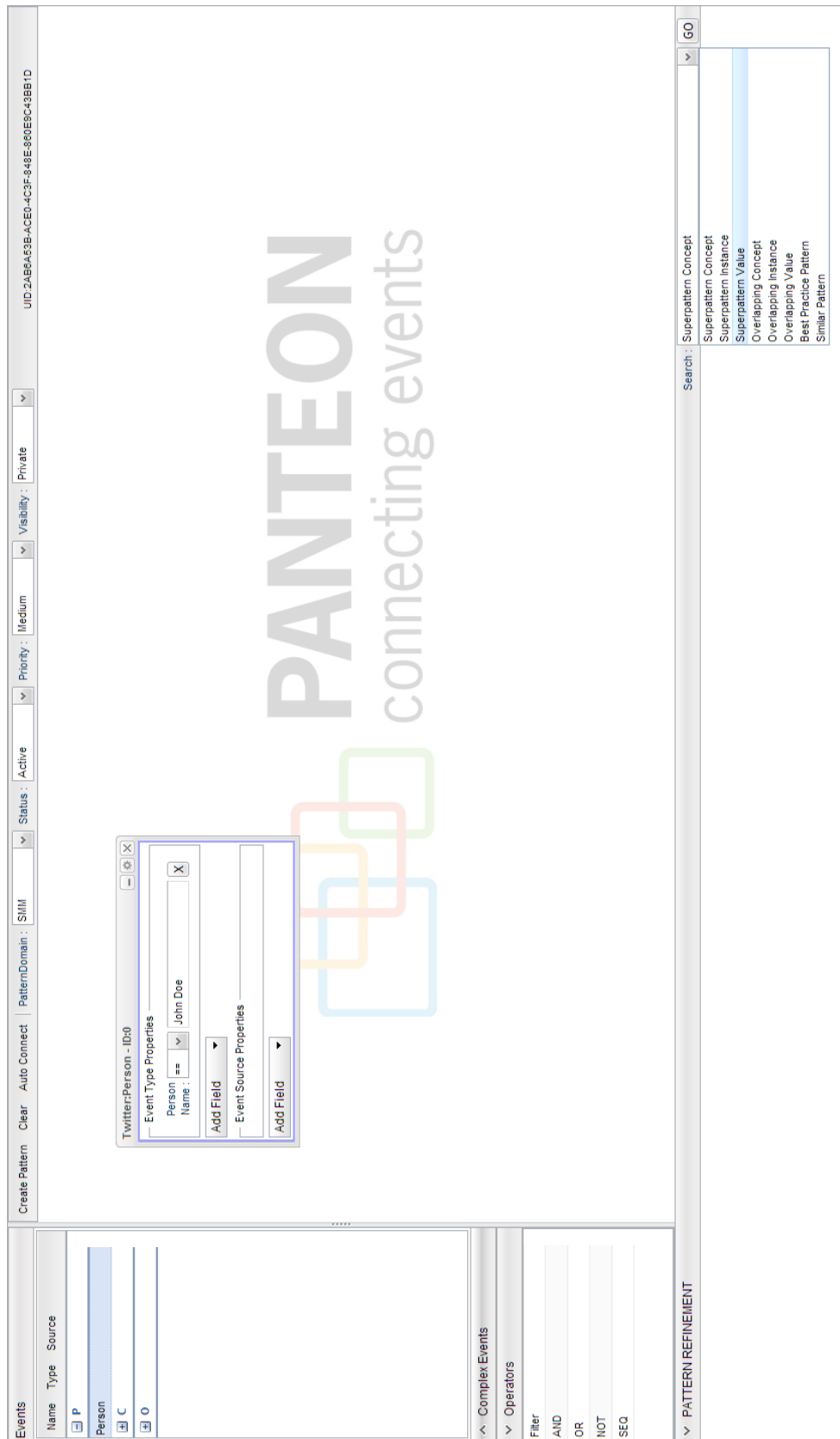


Figure 6.15: Definition of the modelling context and the relationship selection.

The screenshot displays the PANTEON software interface. At the top, there is a navigation bar with 'ID4A', 'Create Pattern', 'Auto Connect', and 'PatternDomain: SHM'. Below this, a search bar contains 'TwitterPerson - ID9'. The main area features the 'PANTEON connecting events' logo. A search results panel on the right lists several patterns, with 'Superpattern Value' highlighted. Below the search results, two pattern icons are shown: 'TriplePersonPattern' and 'DoublePersonPattern'. A dialog box titled 'TwitterPerson - ID9' is open, showing 'Event Type Properties' and 'Event Source Properties'.

Figure 6.16: Presentation of the results as pattern icons.

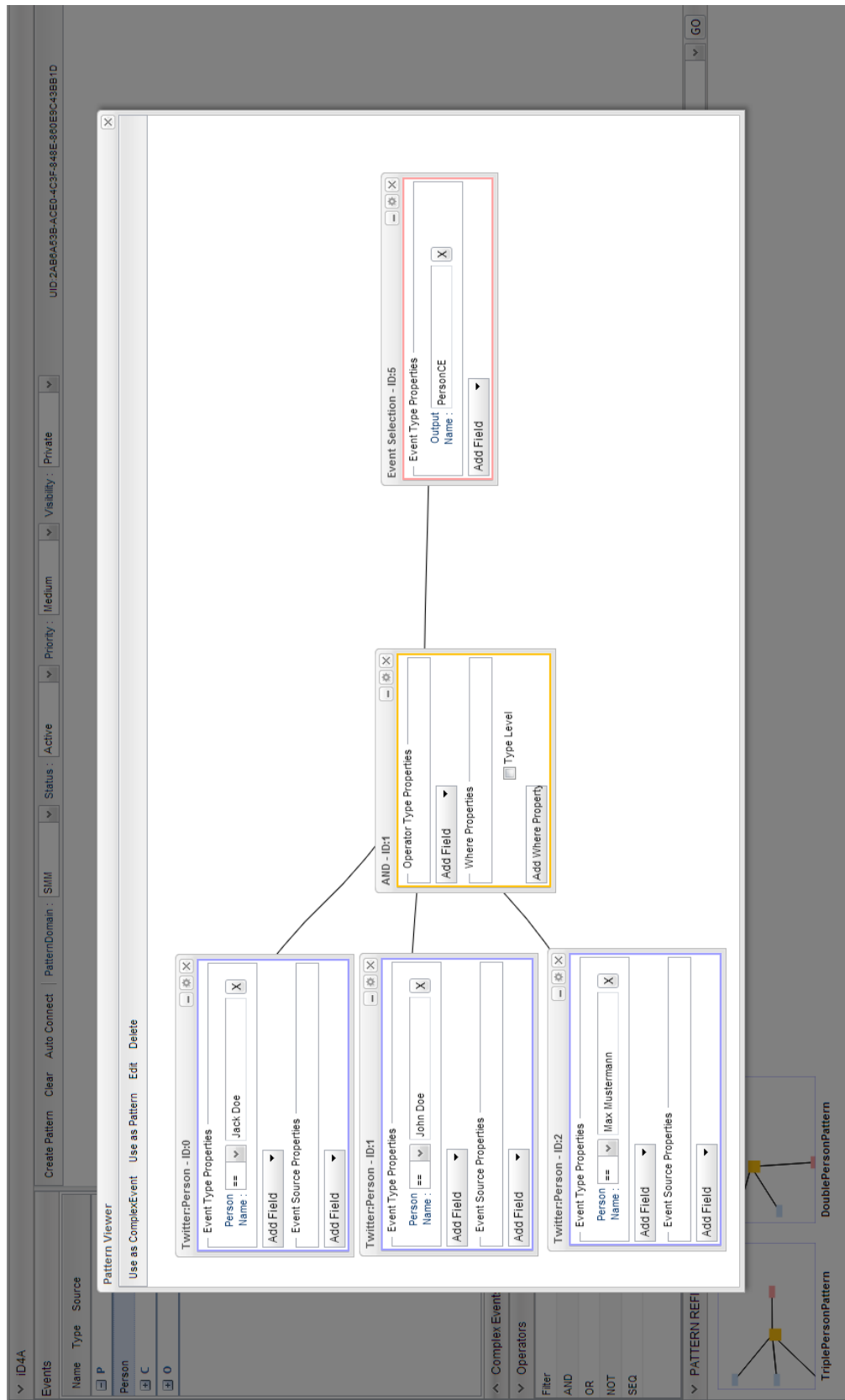


Figure 6.17: Selection of the relevant event pattern from the result list.



The screenshot displays a software interface for modeling events. The top bar includes a search field with the text "UID:9C90E175-7344-4461-ACE2-C3D312F07CFC" and a "GO" button. Below the search bar are several filters: "Create Pattern", "Clear", "Auto Connect", "PatternDomain: SIM", "Status: Active", "Priority: Medium", "Viability: Private".

The main workspace contains several event nodes, each with a title bar and a set of properties:

- Twitter:Person - ID:0**: Event Type Properties, Person Name: Jack Doe, Add Field, Event Source Properties, Add Field.
- Twitter:Person - ID:1**: Event Type Properties, Person Name: John Doe, Add Field, Event Source Properties, Add Field.
- Twitter:Person - ID:2**: Event Type Properties, Person Name: Max Mustermann, Add Field, Event Source Properties, Add Field.
- AIID - ID:6**: Operator Type Properties, Add Field, Where Properties, Add Where Property, Type Level.
- Event Selection - ID:10**: Event Type Properties, Output Name: PersonCE, Add Field.

At the bottom of the interface, there is a "PATTERN REFINEMENT" section with two diagrams:

- TriplePersonPattern**: A diagram showing a central yellow node connected to three blue nodes.
- DoublePersonPattern**: A diagram showing a central yellow node connected to two blue nodes.

The interface also features a sidebar on the left with "Events" and "ComplexEvents" sections, and a bottom status bar with "Search: Superpattern Instance" and a "GO" button.

Figure 6.18: Extension of the initial modelling context with additional event nodes.

### 6.6.3 Architecture

Figure 6.19 displays the architecture of pattern relation detection. The components are:

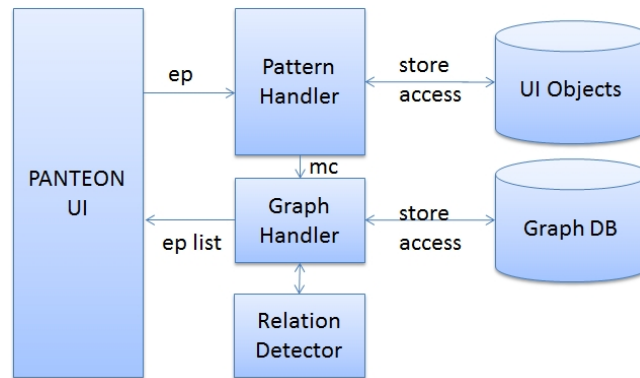
- **PANTEON UI** The PANTEON user interface was described in the previous section.
- **Pattern Handler** The pattern handler is requested by the PANTEON UI in order to execute the desired operation which are: *storage, deletion, modification or relation detection*.
- **Graph Handler** The graph handler converts the event pattern object, which is generated by the PANTEON UI into the graph object which is stored in the Graph DB. Operations like the deletion and modification of existing graph objects are also part of the graph handler. Additionally it provides an interface in order to traverse the pattern graphs for the relationship detection.
- **Relation Detector** The relation detector encapsulates the graph traversing logic for the subsumption and overlapping relations.
- **UI Objects** This is a MySQL database where the graphical event pattern objects are serialized. The purpose of this database is to store all event patterns including the id, name, time of creation, the description, the pattern domain and the serialized graphical event pattern object.
- **Graph Db** In order to detect the graph-based pattern relationships the graphically generated patterns are transformed into a graph object. The graph objects are managed in a *Neo4J NOSQL graph database*<sup>5</sup>. According to Neo Technology Neo4J offers performance improvements on the order of 1000x or more compared to relational databases.

## 6.7 Related work

Obweger et al. [OSSR10] present an approach in order to detect the similarity between single events and similarity between event sequences. In contrast to our approach the similarity calculation presented in [OSSR10] does not consider the similarity calculation based on the hierarchy of events. This is because of the underlying event model that does not provide any hierarchies. They calculate the similarity of events based on the attribute-level of event types. The attribute-level

---

<sup>5</sup>Neo4j is an open source project available in a GPLv3 Community edition, with Advanced and Enterprise editions available under both the AGPLv3 and commercial licenses, supported by Neo Technology



**Figure 6.19:** The architecture of pattern relation detection.

similarity contains lookup tables, where similarities between terms are predefined, similarity metrics for the basic data types like boolean, numeric types, strings and the time stamp of the event. In order to calculate the similarity between event sequences they extend the attribute-level similarity by taking into account the absolute and relative temporal structure of events. The temporal structure is based on the time spans between successive events. They do not consider the similarity of event types, event sources and event operators. While our goal is to support the pattern engineer during the pattern modelling process, the goal of [OSSR10] is to search for event occurrences in the past in order to detect future similar situations that has been occurred previously. This focus is again on the run-time matching process.

To the author's best knowledge there is no further approach that deals with the event pattern relationship detection.

## 6.8 Summary

In this chapter we described several event pattern relationships and described several event pattern relationship levels that enable the pattern engineer to adjust the recommendation of event patterns. We started with the graph-based event pattern relationships to cover overlapping and subpattern relations. We continued with the description of the best practice event patterns which are frequent subgraphs. The role of best practice event patterns is to create a library of patterns that are relevant for a domain of interest. The overall goal is to provide as much as possible event pattern knowledge to pattern engineer. For cases where event patterns could be relevant that are structurally not equal we presented a similarity approach based on the taxonomy of event types, sources and operators and their properties.



# 7

## Event Pattern Evolution

In this chapter we describe the event pattern evolution which is based on the monitoring of event patterns during their execution in the CEP engine. The evolution makes use of event pattern execution data to learn a usual execution behaviour of an event pattern and to identify execution outlier continuously based on ongoing pattern execution data. Once a pattern is identified as a candidate for evolution the pattern engineer is provided with additional information to adapt the identified event pattern.

### 7.1 Introduction

The quick adaptation of enterprises to their dynamic environment is an important factor for gaining competitive advantage in highly dynamic market environments. In order to ensure the competitive advantage each single enterprise information system must provide the capability of evolution and adaptation. Since CEP systems are part of the overall enterprise information system event pattern evolution should be a central pillar of the adaptable CEP system.

Increasing the efficiency and effectiveness of nowadays business are the main driver of using event processing [Luck12],[EtNi10]. Especially the declarative definition of event patterns supports the effectiveness of CEP system [ChSc09]. However, the premise is having a set of tools and methods for modelling, deploying and especially identifying the event pattern candidates that need to be updated and supporting their adaptation to the new business conditions.

In information systems it is essential that the adaptation to the changing business conditions is done in smaller, easier and more frequent steps [FaOP92]. For event patterns this could lead to the minimization of the discrepancy between the business situation that needs to be detected and the definition of the event pattern that is defined to detect the situation. Further, in information systems the bigger the discrepancy between what is needed and what is available results into undesirable situation in which a user starts disliking the system [FaOP92]. As for event patterns the discrepancy could have negative effects on the situational awareness of the enterprise.

In the next section we describe the notion of transparent event pattern matching which is the prerequisite for our event pattern evolution approach described in this chapter.

## 7.2 Transparent event pattern matching

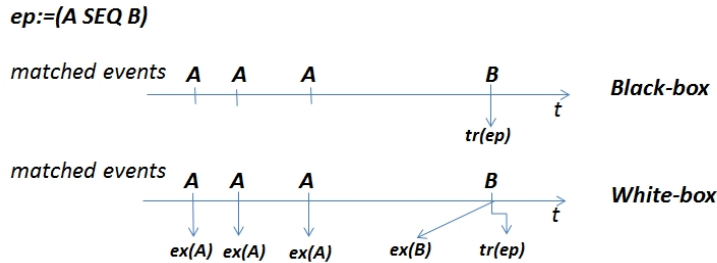
Contemporary CEP engines have the task to trigger a complex event as soon as an event pattern has been matched against received events. The detection of a complex event is either *true* or *false*. This kind of matching is like a black-box since the only interfaces that are provided are the interface for receiving events and the interface for receiving event patterns to detect a relevant situation. However, we know from the active database research that the monitoring of rules is worthwhile in order to optimize them [DiGG95]. In the domain of logic programming [EiBP91] described the concept of transparent Prolog machine in order to animate the execution and debugging of Prolog programs. Applying the concept of transparency to the pattern execution is promising not only for providing event pattern execution data for the evolution of patterns but also for a more proactive event pattern matching (see [EnEt11]). In a transparent CEP engine we need to differentiate between the full matching and the partial matching. We call the full matching of an event pattern pattern triggering and the partial matching pattern execution.

**Definition: Pattern triggering** An event pattern is triggered whenever all pattern elements are evaluated true such that the complex event can be produced as a result.

In order to trigger an event pattern the events must be occurred according to the event pattern. As described earlier, the event pattern definition contains events and the event operators. For example the pattern  $A SEQ B$  is triggered if and only if B is followed by A. Pattern triggering is that what is supported by existing CEP engines. We call these engines black-box or non-transparent CEP engines.

**Definition: Pattern execution** An event pattern is executed whenever an event pattern subgraph of the event pattern is evaluated true.

As described in Chapter 6 the minimal length of the subgraph is 1, which indicates the occurrence of an event that is part of the event pattern. We call this kind of pattern execution the white-box or transparent approach. It is what should be supported by the next generation CEP engines. Figure 7.1 displays the difference between the pattern execution (white-box) and pattern triggering (black-box).



**Figure 7.1:** Black-box vs. white-box event pattern matching.

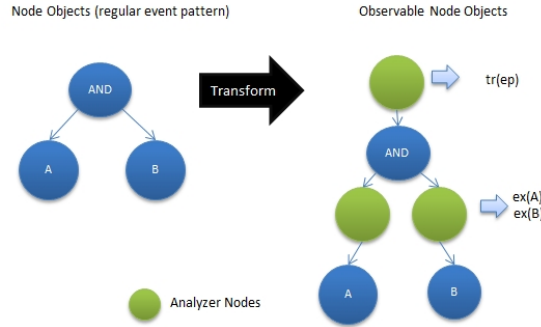
The figure displays an event pattern  $ep := (A SEQ B)$  in the black-box and white-box mode whereby in both cases the event pattern  $ep$  describes a situation where two events  $A$  and  $B$  are connected by a sequence operator  $SEQ$ . In the black-box CEP engine the pattern is triggered, indicated by  $tr(ep)$ , after the event  $B$  is received and matched. There is no additional information about the number of events that have been consumed between the first and last occurrence of event pattern relevant events. In the white-box approach the pattern is also triggered true after the event  $B$  is received and matched. However, the difference to the black-box CEP engine is that the white-box CEP engine delivers also additional execution data about the consumption of events  $A$  and  $B$  before the event pattern is triggered, indicated by  $ex(A)$  and  $ex(B)$ . As a result the white-box CEP engine registers that the event  $A$  has been occurred three times and the event  $B$  one time before the event pattern is triggered. The question is how to extend the existing black-box CEP engines to handle these additional execution data.

A naive and straightforward approach is to extend the event pattern graph in the CEP engine with additional analyzer. In order to support our evolution approach we extended the event pattern graph of the open source CEP engine *Esper*<sup>12</sup> to deliver the execution data [SeLS11]. We know that such an extension might have negative effects on the throughput of the engine. However, the performance issues of the CEP engine are not in the scope of this thesis and therefore neglected. The additional nodes that we add to the event pattern graph in Esper sends the status of the event pattern tree to an external component. The deployment of an event pattern in the engine is not influenced by the extension of the internal pattern graph. The Esper CEP engine represents an event pattern as an *Abstract Syntax Tree*. It is a connected graph without cycles.

<sup>1</sup><http://http://esper.codehaus.org/>

<sup>2</sup>We used Esper since it enables the monitoring of the pattern graphs through the implementation of new plug-ins. The ease of implementation was reason why we selected Esper.

Figure 7.2 displays the extension of the regular event pattern graph with additional *analyzer nodes*. The analyzer nodes are added between two nodes and have the task to inform the evolution component about the state change within the event pattern graph,  $ex(A)$ ,  $ex(B)$ ,  $tr(ep)$ . Our general strategy to extend the regular event pattern graph with additional analyzer was to extend each node in the regular pattern with an analyzer node. This would allow us to receive every state change in an event pattern graph.



**Figure 7.2:** Extension of the event pattern graph with additional analyzer nodes in order to receive pattern execution data.

The extension of the CEP engine might be different for different CEP engines. We do not proclaim the approach above as the gold standard. Its purpose is to deliver the data we need for the execution-driven evolution.

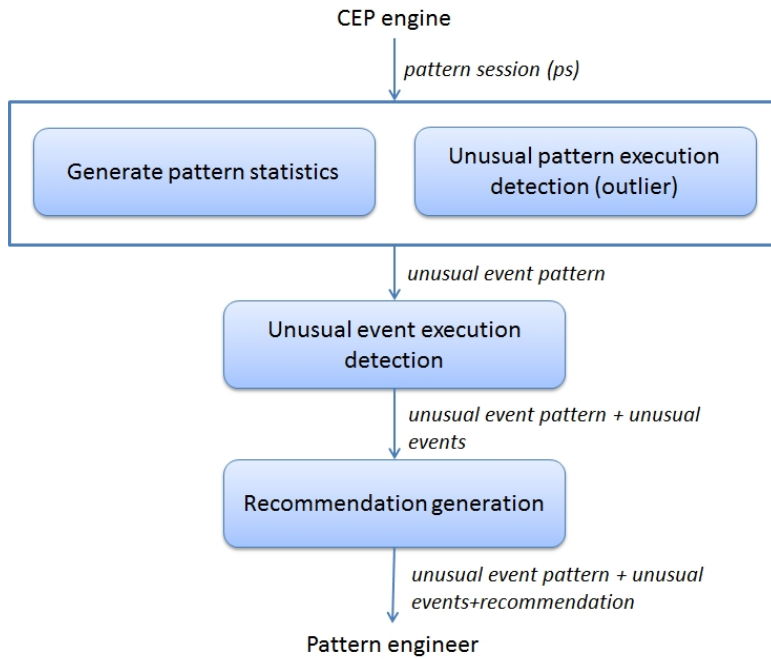
### 7.3 The process of execution-driven evolution

The evolution of event patterns consists, as described in Figure 4.2 in Chapter 4, of four tasks which are *Analyse*, *Compare*, *Suggest* and *Adapt*. While *Analyse* has the objective to build for every event pattern execution statistics from received execution data, *Compare* deals with the outlier detection in pattern execution taking into account historical execution statistics. *Suggest* aims at spotting the problem for event patterns that are candidates for evolution. *Adapt* includes the user feedback for an evolution candidate in order to adapt the event pattern if necessary. Figure 7.3 describes the process of execution-driven pattern evolution. We call the execution data that is send by the CEP engine as pattern session.

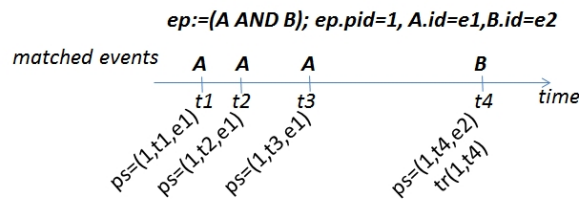
**Definition: Pattern session** A pattern session is a tuple containing the information about an event that has been consumed by the CEP engine and is part of an event pattern matching,  $PS := (pid, ts, eid)$ , where  $pid$  is the id of the pattern that has been part of the matching,  $ts$  is the timestamp of the matching and  $eid$  is the id of the event node that has been evaluated true.

A pattern session contains the detailed execution data of an event pattern. Figure 7.4 displays an example of a pattern session. The example shown in Figure 7.4





**Figure 7.3:** Overview of the evolution approach.



**Figure 7.4:** Example: Pattern session.

contains an event pattern with  $pid = 1$  and two events  $A$  and  $B$  with id  $e1$  and  $e2$ . The pattern sessions are sent at time points  $t1$ ,  $t2$ ,  $t3$  and  $t4$  whereby at  $t4$  the event pattern is triggered,  $tr(1, t4)$ .

Every pattern session that has been received by the evolution component will be used to learn the reference execution statistics and for the outlier detection. The process continues if and only if reference statistics for a given event pattern exists. In this case possible event pattern execution outlier are detected which will be described in detail in Subsection 7.3.1, Subsection 7.3.2 and Section 7.4. The outlier detection is enriched with additional information about events that caused the unusual behaviour. Finally the possible candidates for evolution are presented to the pattern engineer. In the next sections we continue with describing these steps in more detail.

### 7.3.1 Constructing the pattern execution statistics

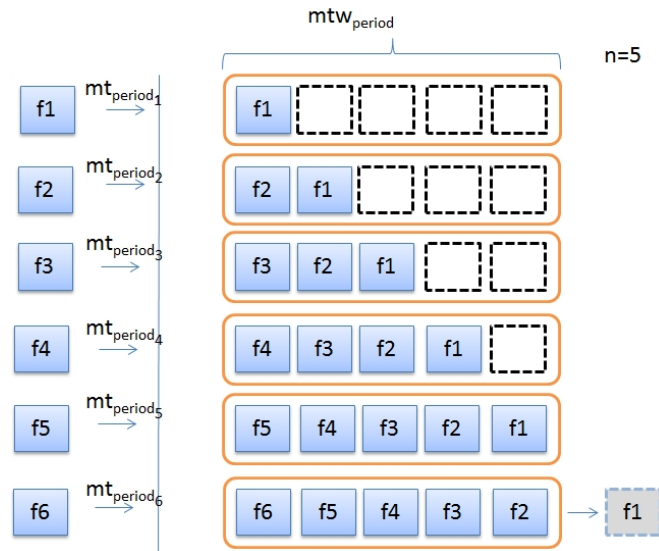
As described in the previous section the detection of evolution candidates is based on the historical execution statistics. In order to build the execution statistics we

introduce the monitoring period  $mt_{period}$  consisting of a start time  $mt_{start}$  and the end time  $mt_{end}$ . Both  $mt_{start}$  and  $mt_{end}$  are timestamps which span the duration of the monitoring period. The monitoring period is used for building the initial reference statistics which will be used for the outlier detection. The monitoring period is also the time period for which the outlier detection is done. Examples for  $mt_{period}$  can be *seconds, minutes, hours, days, weeks, months, years*.

Independently from which  $mt_{period}$  is selected, the monitoring period is repetitive and composes the *Monitoring Window*. For each  $mt_{period}$  a pattern frequency object  $f$  is created. The pattern frequency object,  $f := \{f_{tr}, f_{ex}\}$ , is composed of the total number of pattern triggering  $f_{tr}$  and the total number of pattern executions  $f_{ex}$  that includes also the total number of each executed event.  $f_{tr}$  and  $f_{ex}$  are constructed from the received pattern sessions.

**Definition: Monitoring window** The Monitoring Window, denoted as  $mtw_{period}$ , is composed of a set of pattern frequency objects  $f$  for the monitoring periods  $mt_{period_1}, mt_{period_2}, \dots, mt_{period_n}$ .

The  $mtw_{period}$  is a sliding count window with the size  $n$ . The example in Figure 7.5 displays a  $mtw_{period}$  with  $n = 5$  and 6 pattern frequency objects  $f_1 - f_6$ . Whenever the size of the monitoring window is reached and a new frequency object is received the oldest element of the  $mtw_{period}$  is deleted from the monitoring window.



**Figure 7.5:** Example: Monitoring window.

Based on the frequency objects in  $mtw_{period}$  the next step is to build the reference statistics for each event pattern. Once the reference statistic for a pattern is generated new received event pattern frequency objects is compared to generated event pattern reference statistic. As next we describe these steps in more detail.

### 7.3.2 Pattern frequency distribution and frequency clustering

The pattern frequency object  $f$  of an event pattern contains the frequency of the pattern triggering and the frequency of pattern execution within a given monitoring period.

Figure 7.6 shows an exemplified frequency distribution for the event pattern  $ep_1 := A \text{ AND } B$  and its event elements  $A$  and  $B$ .

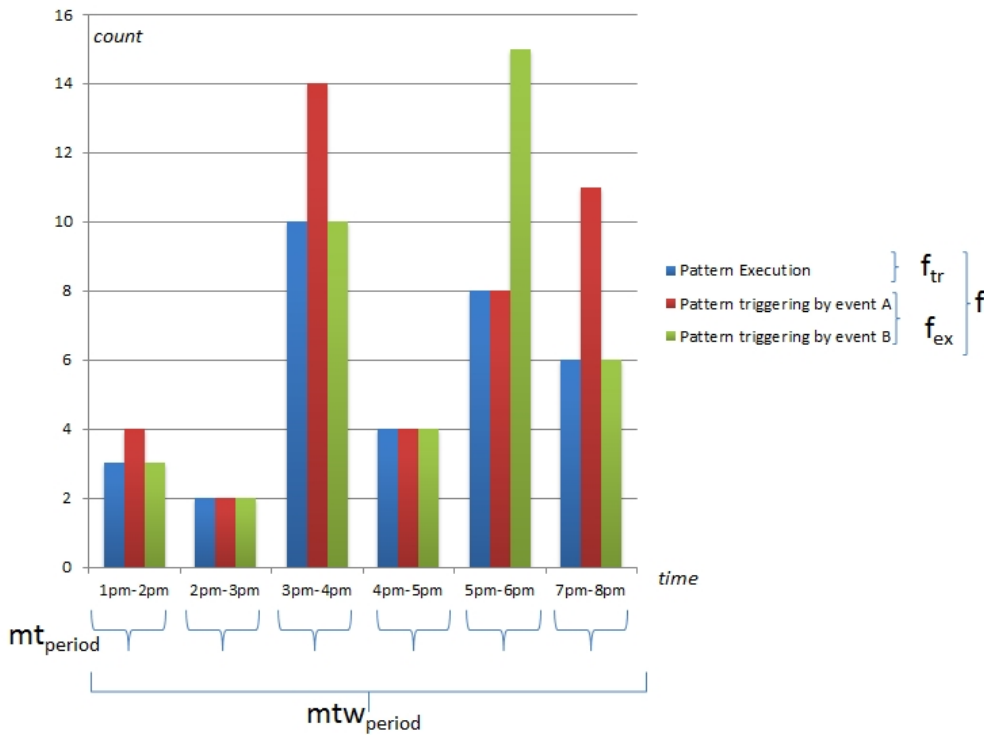


Figure 7.6: Example: Evolution time window.

In this exemplified frequency distribution the monitoring period  $mt_{period}$  is one hour and starts at  $1pm$ . The length  $n$  of the monitoring window  $mtw_{period}$  is 6. The last monitoring period ends at  $8pm$ . The figure displays the  $f$ ,  $f_{tr}$  and  $f_{ex}$ . The pattern execution  $f_{ex}$  contains the frequency of the event consumption  $A$  and  $B$ .

Based on the frequency distribution for a given monitoring windows  $mtw_{period}$  the goal is to derive a triggering cluster for usual patterns. The triggering cluster is bounded by the maximum and minimum number of pattern triggering for a given monitoring period.

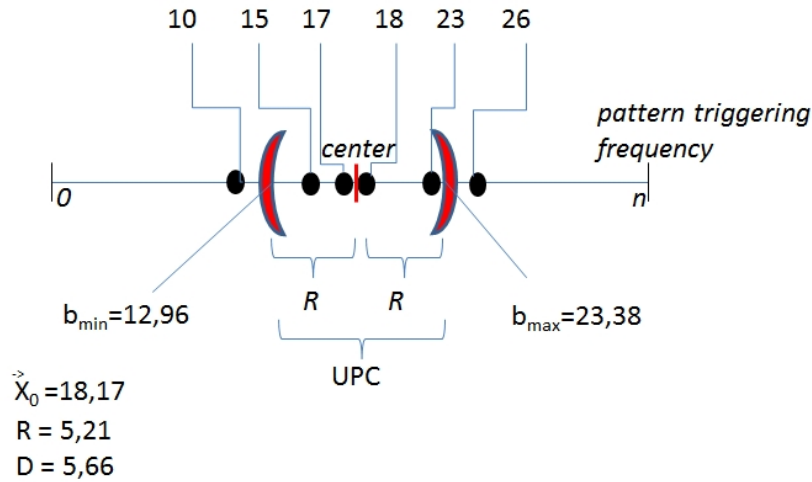
In order to detect an outlier in the pattern triggering we apply the technique of clustering known from the data mining [ZhRL97]. A cluster is a collection of data objects that are similar to one another and treated collectively as a group. To determine whether the frequency of pattern triggering is an outlier we applied the *Cluster Feature(CF)* from the BIRCH-Algorithm [ZhRL97]. BIRCH is an unsupervised data mining algorithm for performing hierarchical clustering. It has

the advantage that it incrementally and dynamically clusters an incoming data point.

CF allows to calculate measures directly, like the cluster centroid, the radius and the diameter. The cluster centroid is the euclidian center, the radius the average distance from member points to centroid and the diameter is the average pair-wise distance within a cluster. Given  $N$  d-dimensional data points in a cluster:  $\{\vec{x}_i\}$  where  $i = 1, 2, \dots, N$ , the centroid  $\vec{x}_0$ , radius  $R$  and diameter  $D$  are defined as:

$$\begin{aligned}\vec{x}_0 &= \frac{\sum_{i=1}^N \vec{x}_i}{N} \\ R &= \sqrt{\frac{\sum_{i=1}^N (x_i - x_0)^2}{N}} \\ D &= \sqrt{\frac{\sum_{i=1}^N \sum_{j=1}^N (x_i - x_j)^2}{N(N-1)}}\end{aligned}$$

The purpose of using  $\vec{x}_0$ ,  $R$  and  $D$  is to check if the pattern triggering frequency is an outlier. Figure 7.7 displays an example for a pattern frequency distribution cluster with the frequency values  $\vec{x}_i = \{10, 15, 17, 18, 23, 26\}$  and  $N = 6$ .



**Figure 7.7:** Pattern frequency cluster with centroid and radius.

The centroid  $\vec{x}_0$  of the cluster is 18.17 the radius  $R$  is 5.21 and the diameter  $D$  is 5.66. The radius  $R$  which implies that the usual pattern frequency cluster ( $UPC$ ) is in the range of  $UPC = \vec{x}_0 \pm R = [12.96, 23.38]$ . The minimal and maximal  $UPC$  values denote the minimal cluster border  $b_{min}$  and the maximal cluster border  $b_{max}$ .

The cluster is recalculated whenever a new frequency object enters the monitoring window like described in Figure 7.5. Taking into account these reference statistics we continue with the actual detection of the evolution candidates.



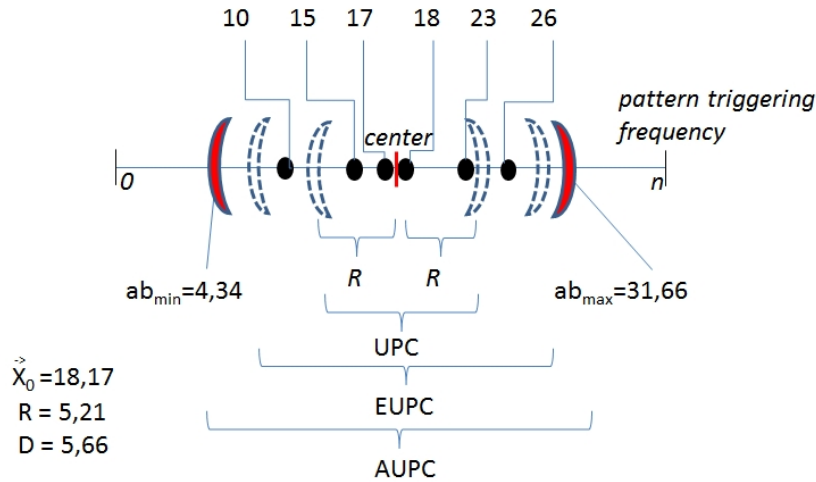
$eb_{min}$  and  $b_{min}$  are non-negative numbers with 0 as lowest value for  $eb_{min}$ . If  $eb_{min} = b_{min} - D < 0$  then  $eb_{min}$  will be set to 0. That implies that a non-occurrence of a pattern triggering within a monitoring period is possible.

Every frequency distribution that is outside these borders is marked as unusual. In Figure 7.8 the *EUPC* cluster contains all pattern triggering frequencies.

The next strategy is the most robust one and is called the absolute usual pattern cluster *AUPC*. This strategy was inspired from the Chebyshev's inequality theorem [Knut97]. Chebyshev's theorem says that in any probability distribution that at least  $\frac{1}{1/K^2}$  ( $K$  is any positive real number) of data from a sample must fall within  $K$  standard deviations from the mean. For the event pattern execution it means that there will be pattern executions that are usual but are marked as unusual which would be a false positive. In order to reduce the number of these false positives, we select the lowest pattern triggering frequency  $f_{tr_{low}}$  and highest pattern triggering frequency  $f_{tr_{high}}$  and calculate the absolute borders  $ab_{min}$  and  $ab_{max}$  by adding or subtracting  $D$  to the highest and lowest frequencies.

$$\begin{aligned} ab_{min} &= f_{tr_{low}} - D; \\ ab_{max} &= f_{tr_{high}} + D; \end{aligned}$$

The assumption is that the pair-wise average distance of each element in the cluster can be applied to the lowest and highest frequency values in order to determine the absolute range for the pattern execution.



**Figure 7.9:** Absolute usual pattern frequency cluster.

In Figure 7.9 the minimum and maximum borders are determined by adding the diameter value 5.66 to 26 and subtracting the diameter value from 10 respectively. The new absolute cluster borders are 4.34 and 31.66.

Every pattern that has a pattern triggering frequency  $f_{tr}$  outside this cluster is marked as candidate for evolution. The distance to the borders determines the evolution score which will be used for the ranking of the detected evolution candidates.

$$evo_{score}(ep) = \begin{cases} f_{tr_{ep}} - ab_{max} & \text{if } f_{tr_{ep}} > ab_{max} \\ ab_{min} - f_{tr_{ep}} & \text{if } f_{tr_{ep}} < ab_{min} \end{cases}$$

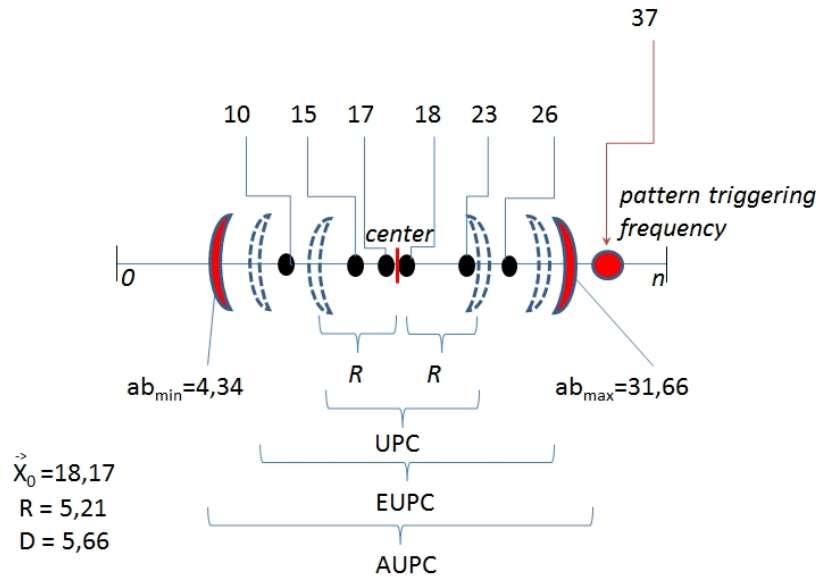


Figure 7.10: Unusual pattern execution detection.

In Figure 7.10 a new frequency distribution is received with the value 37. This value is outside of  $ab_{min}$  and  $ab_{max}$  and hence the pattern is marked unusual. The evolution value is  $evo_{score}(ep) = 37 - ab_{max} = 5.34$ . The higher the evolution score the higher the event pattern will be ranked when the list of evolution candidates is presented to the pattern engineer.

### 7.4.1 Preparation for the evolution

The outlier detection of the pattern triggering frequency itself does not provide any information why this pattern is marked as an outlier. Hence, to adapt an event pattern candidate we provide additional context information about the pattern frequency deviation. This procedure is part of the evolution preparation which consists of the identification of the events that caused the unusual behaviour followed by the generation of evolution suggestions.

The reasons for an event pattern frequency being an outlier can be:

- **Human errors** The pattern definition is not well suited to the business situation and therefore there might be an inconsistent triggering behaviour. For example the pattern can be too general meaning that it is triggered to many times under certain situations. On the other hand it can be too specific meaning it executes very rare under certain circumstances.
- **Source errors** The event source can be defective meaning that it is not reliable in sending the events that are needed for the business situation detection.

- **Changes in the environment** The system environment where the CEP is in use has been changed and the pattern is in this new context not applicable without changes.

In order to prepare the evolution phase it is essential to provide the information about the events that might lead to these unusual categories. During the preparation phase the single event execution on the pattern level is analysed in order to localize possible erroneous events.

We classify the event pattern frequency outlier detection into the following categories:

- **More Execution** An event pattern is triggered more than the maximum border of the cluster.
- **Less Execution** An event pattern is triggered less than the minimum border for the cluster.

In order to identify possible defective events we compare the current execution value of an event  $f_{ex_E}$  with the arithmetic average of the same event for the monitoring window  $mtw_{period}$ . The arithmetic average indicates a central tendency where a set of numbers cluster around some values. The arithmetic average for each event is calculated as  $A_E = \frac{1}{n} \sum_i f_{ex_{E_i}}$  where  $n$  is the size of the monitoring window and  $f_{ex_{E_i}}$  the frequency of the event  $E$  in the monitoring period  $i$ .

For each event we determine the absolute execution difference by subtracting the arithmetic average  $A_E$  from the current absolute event occurrence  $f_{ex_E}$ .

$$abs_{occurrence} = f_{ex_E} - A_E$$

The result of  $abs_{occurrence}$  indicates different types of event occurrences with different types of evolution preparation strategies.

$$Type\ of\ event\ occurrence \begin{cases} less\ occurrence & \text{if } abs_{occurrence} < 0 \\ equal\ occurrence & \text{if } abs_{occurrence} = 0 \\ more\ occurrence & \text{if } abs_{occurrence} > 0 \end{cases}$$

Based on the type of event occurrence there can be different reasons for the deviation in the event pattern triggering:

- **Event source** The event source of the event might be defective since it produces less or more events.
- **Property value** The property values are not set correctly.
- **Pattern invalidity** The whole pattern itself is not valid.



A more automatic approach would be to consider other event patterns that have the same event type and event source and check whether these patterns are marked as candidate for evolution or not. This information can be used by the pattern engineer in order to adapt the event pattern by considering related event patterns that are not marked as candidate for evolution.

Further we define the generality and speciality of an event pattern through the quotient of the number of the pattern triggering  $f_{tr}$  and the number of pattern execution  $f_{ex}$ . Assuming that  $f_{ex} \neq 0$  for a given monitoring period the quotient  $Q_{gs}$  can be defined as follows:

$$Q_{gs} = \frac{f_{tr}}{f_{ex}}$$

The value for the quotient is in the range of 0 and 1. The result can identify situations where a pattern is defined as being too specific meaning that the value of the quotient is towards 0. In an opposite way a pattern could be modelled too general meaning that the value of the quotient is towards 1. The quotient is showed to the pattern engineer in cases where the pattern is being detected as an outlier. The overall goal is to present the pattern engineer several processing related data to adapt a possible outlier.

## 7.4.2 Evolution adaptation

The evolution adaptation is the phase where the human-machine cooperation takes place. In our case the evolution detection aims at finding outliers in the event pattern triggering enriched with additional information presented to the pattern engineer. The role of the pattern engineer is, taking into account this information, to decide whether the pattern is a candidate for evolution or not.

The evolution adaptation is a process triggered by the evolution candidate detection. The process is described in Figure 7.11. When an event pattern is identified as a candidate for evolution the pattern engineer is prompted to adapt the event pattern. The pattern engineer decides whether the evolution candidates are valid for updates or not. If the evolution candidate is valid for update the next step is to adapt the event pattern taking into account the statistics of the event pattern. If it is not valid for update the pattern engineer can take into account the statistics details and derive the knowledge for a new event pattern if needed.

The event pattern definition is, as described in Chapter 4, part of the Generation phase. The more interesting question is what happens if the pattern is subject to change and what are the steps that will lead to the adaptation. Below we describe different tasks in order to adapt the event pattern.

- **Property modification** Meaning that a property value of the event will be changed to a new value since the old value is out-dated.

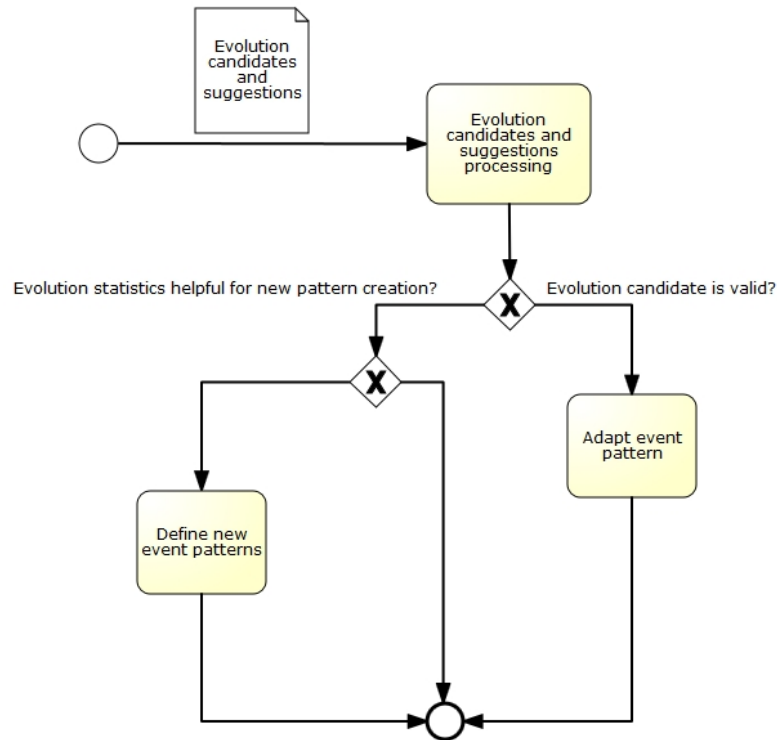


Figure 7.11: The process of evolution adaptation.

- **Event modification** Meaning that the event node will be replaced by another event or deleted.
- **Event operator modification** Meaning that the event operator configuration was not adequate and will be changed.
- **Pattern modification** Meaning that the pattern structure will be changed by removing of events and event operators or by adding of new events and event operators. The result is a new event pattern.

## 7.5 Implementation

The results of the evolution detection are presented to the pattern engineer in a graphical way. Figure 7.12 displays the main window where the pattern engineer can search for the pattern candidates that are marked as a candidate for evolution. The displayed UI consists of the following two section:

- **Search** The search part is provided in order to find an event pattern that has been created by the pattern engineer. Beside the regular search option there is also the possibility to search event patterns that are marked as candidate for evolution.

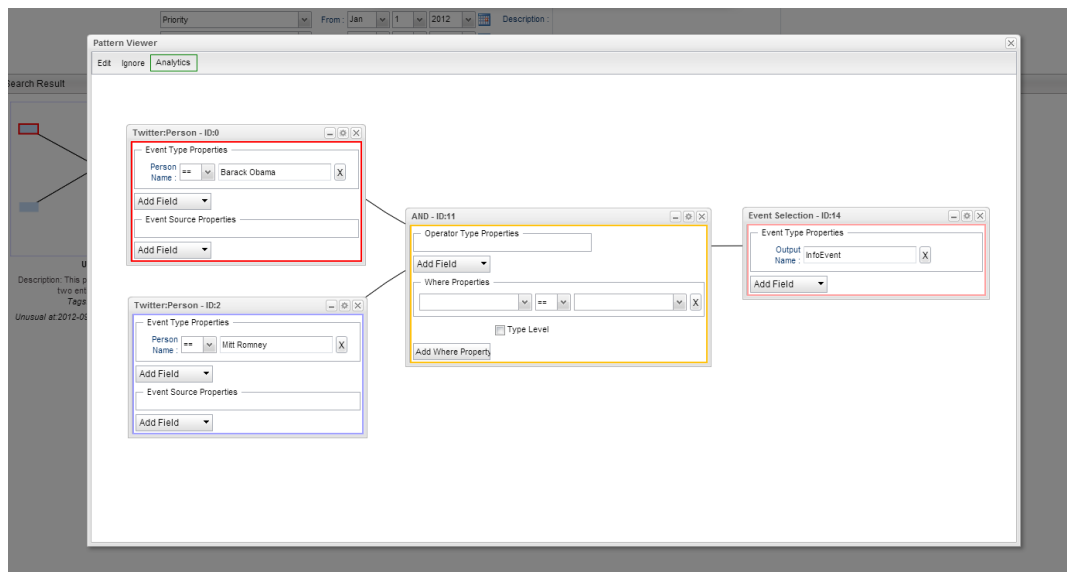
The screenshot displays the PANTEON interface, which is used for detecting evolution candidates. The interface is divided into several sections:

- Stream Designer Search:** Located at the top left, it includes a search bar and a search button.
- Search Options:** This section contains various filters and controls:
  - Evolution Search:** A checkbox that is checked, with a blue circle containing the number '2' next to it.
  - Pattern Domain:** A dropdown menu.
  - Priority:** A dropdown menu.
  - Visibility:** A dropdown menu.
  - Status:** A dropdown menu.
  - From:** A date selector set to 'Jan 1'.
  - Till:** A date selector set to 'Sep 23'.
  - Description:** A text input field.
  - Tags:** A text input field.
- Search Result:** This section displays the results of a search:
  - USElectionPattern:** A diagram showing a central yellow square connected to three other squares (red, blue, and light blue) by lines. A red circle with the number '1' is positioned above the diagram.
  - Description:** "This pattern describes a tweet containing two entities: Obama and Romney".
  - Tags:** "US, Obama, Romney".
  - Unusual at:** "2012-09-23 20:25:17.0 - Unusual Score 199".
- PANTEON Logo:** The logo "PANTEON connecting events" is displayed in the background, with a blue circle containing the number '3' next to it.

**Figure 7.12:** Overview of the detected evolution candidates with highlighting (red bordered event) the events that might lead to the detection.

- **Search results** The search results are presented as pattern icons. These icons can be clicked for further information about the event pattern. Events that might be problematic are highlighted in red. This view is presented in Figure 7.13.

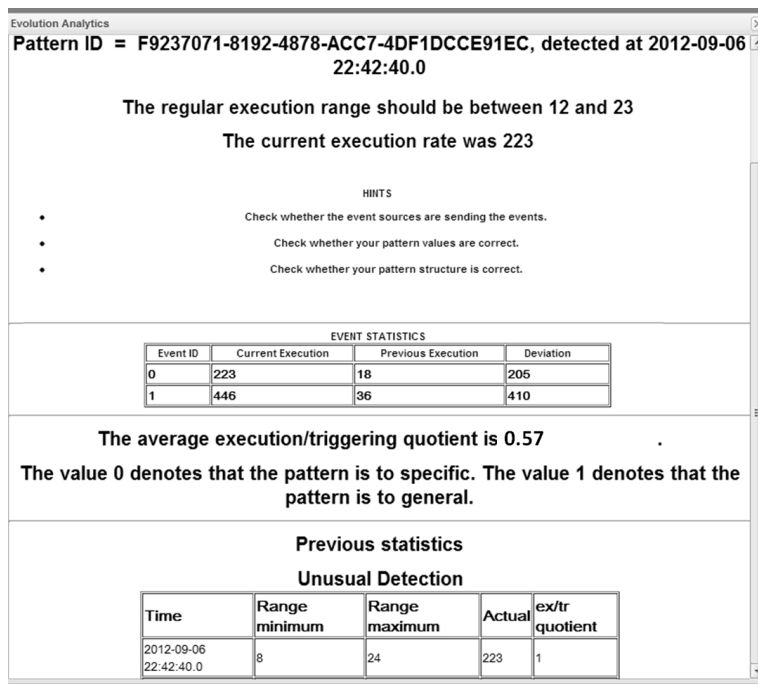
The detailed view contains the original event pattern as it was developed by the pattern engineer with highlighting (red bordered event) the event parts that might be problematic since the absolute event occurrence deviates from the historical event execution.



**Figure 7.13:** Detailed view of the detected evolution candidate.

Additional information about the detection can be delivered by clicking the *Analytics* button. Figure 7.14 displays the information that is shown for an event pattern that is detected as an outlier. In this case the number of the last event pattern execution was 223 which outside of the border 12 (minimum) and 23 (maximum). Further the execution of both events (Event ID 0 and 1) is displayed. The quotient between the current execution and triggering is 0.57 which indicates that neither there is a zero triggering nor a triggering of the event pattern whenever an event is received. Additional statistics from the previous monitoring periods are also displayed.

If the pattern is detected correctly as an outlier, the pattern engineer can click on the *Edit* button and change the original event pattern. Otherwise the pattern can be ignored (*Ignore*) since it was detected as a false positive. In this case the pattern engineer can decide whether this pattern should be excluded from the observation for a certain time period or totally.



**Figure 7.14:** Additional evolution information in order to adapt the event pattern.

## 7.6 Related Work

Turchin et al. [TuGW09] presented an approach in order to update event pattern rules automatically. The approach consists of the rule parameter prediction and the rule parameter correction. For the rule parameter prediction they used an unsupervised learning approach without any expert feedback. The parameter update is supervised which means that the expert feedback is utilized. The approach is based on a predictor-corrector type estimator that estimates the state of a dynamic system from a series of noisy events. They use the Discrete Kalman Filter [Kalm60]. In our approach the focus is to support the pattern engineer and enable a semi-automatic event pattern evolution. We consider the overall pattern structure and not only the attributes of the events in order to support the evolution. Compared to [TuGW09] we are able to detect events that caused the unusual event pattern execution. Further our approach is a semi-automatic one meaning that the details of the evolution detection are presented to the pattern engineer who determine further actions.

Vijayakumar and Plale [ViP107] address the problem of missing events in sensor and instrument streams. They propose a model based on the Kalman filter. The filter is used for modeling the input sensor streams as a time series and to predict the missing events. Wasserkrug et al. [WGET08] tackles the problem of event materialization under uncertainty. They used Bayesian network for constructing the probability space of an event history. The Monte Carlo sampling algorithm is used to approximate the materialized event probabilities. The topic of handling uncertainty in CEP was also considered by [KhBS08]. Georgakopoulos et al. [GBNC07] presents the Video Event Awareness Workbench video surveillance sys-

tem that detects automatically complex events in near real-time. Their approach is designed in order to gather information in a proactive way to deal with missing or incomplete event information. Compared to our approach these approaches are used in order to identify events that are relevant but missing. The user-centric evolution support is not the main focus of these approaches. They neither provide information about an unusual execution behaviour nor present the pattern engineer what should be done in cases where events do not occur.

## 7.7 Summary

In this chapter we described the process of event pattern evolution based on execution and triggering data. We introduced the transparent CEP engine as the prerequisite for this approach and described a way how to extend existing CEP engines towards a transparent processing style. The presented evolution approach detects pattern triggering outliers based on historical data. Event patterns that are detected as candidate for evolution are enriched with additional event statistics in order to support the adaptation of the event pattern by the pattern engineer. Finally we presented the evolution environment within the PANTEON system that detects and presents the evolution candidates for the pattern engineer and supports the pattern engineer in adapting the event patterns.

**Part III**

**Finale**





# 8

## Evaluation

In this chapter we present the evaluation of the PANTEON tool that is based on the event pattern life cycle methodology presented in Chapter 4 and includes the implementation for the approaches presented in Chapter 5, Chapter 6 and Chapter 7.

Since CEP is a rather young research area, to our best knowledge, there exists no relevant data sets for evaluation purposes. Neither for event patterns nor for event data there exist data sets which we could use for event pattern modelling and simulation purposes. Second the evaluation of management issues in CEP systems is rather unclear. As described in earlier chapters the main focus of nowadays CEP is on the matching process. Often in CEP systems performance evaluations are conducted to figure out the throughput and the number of active event patterns in the engine and so forth.

For the final evaluation we conducted a task-driven usability evaluation to find out the perceived usability, the task completion time and the failure rate. Based on these results we derived the efficiency and the effectiveness of a pattern engineer using the PANTEON tool.

### 8.1 Usability evaluation

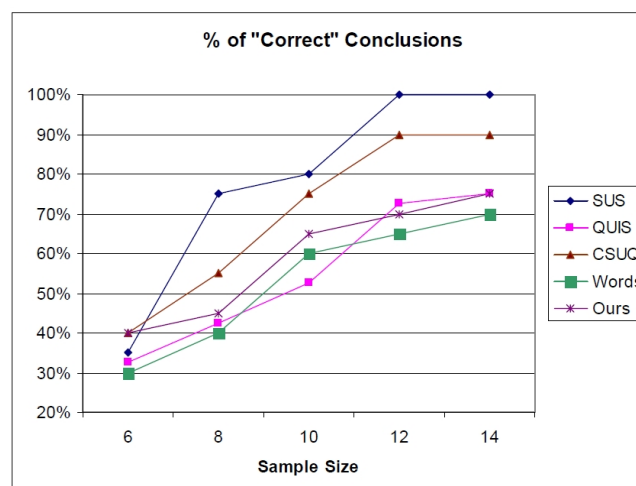
The ISO 9241 defines the usability as "the extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency and

satisfaction in a specified context of use”. Jakob Nielsen introduces a framework<sup>1</sup> where usability is a part of ”usefulness” which is composed of:

- **Learnability** How easy is it for users to accomplish basic tasks the first time they encounter the system?
- **Efficiency** Once users have learned the system, how quickly can they perform tasks?
- **Memorability** When users return to the system after a period of not using it, how easily can they reestablish proficiency?
- **Errors** How many errors do users make, how severe are these errors and how easily can they recover from the errors?
- **Satisfaction** How pleasant is it to use the system?

The usability aims at increasing the productivity of the user which is also one of the main goal of the PANTEON tool. A usability study has subjective and objective results. While subjective results are derived from the opinion and attitude scales of the participants, the objective results are based on the task completion time [Lewi95].

In order to have subjective results there exists several approaches. The user interface rating form<sup>2</sup> for interactive multimedia aims at rating the interface of a new program or one under development. The rating form contains several dimensions like *Ease of Use*, *Navigation* or *Overall Functionality*. Beside that there are



**Figure 8.1:** Data based on t-tests of random sub-samples of various sizes. Twenty sub-samples were taken at each sample size for each site and each questionnaire. What is plotted is the percentage of those 20 tests that yielded the same conclusion as the analysis of the full dataset [TuSt04].

<sup>1</sup><http://www.useit.com/alertbox/20030825.html>

<sup>2</sup><http://it.coe.uga.edu/treeves/edit8350/UIRF.htm>

also questionnaires like the Computer System Usability Questionnaire (CSUQ) [Lew95], System Usability Scale (SUS) [Broo96], Questionnaire for User Interface Satisfaction (QUIS) [ChDN88] and Microsoft's Product Reaction Card (MPRC) [BeMi02] in order to assess the perceived usability [TuSt04].

Based on the evaluation of these questionnaires in [TuSt04] we decided to use SUS for the usability evaluation. The main advantage of SUS is that it provides less questions and hence is best suited for a fast and continuous opinion capturing.

Tullis and Stetsen [TuSt04] found out that SUS has a higher accuracy with an increasing sample size than the other questionnaires. Figure 8.1 displays the percentage of the correct conclusions using 20 sub-samples of the full dataset. The questionnaire SUS reaches an asymptotes of 90-100%.

### 8.1.1 System Usability Scale (SUS)

The SUS is a low-cost scale that allows the global assessments of systems usability. SUS is a ten-item scale (see Figure 8.2) providing a subjective assessment of usability based on a 5 point Likert scale [Broo96].

	Strongly disagree				Strongly agree
1. I think that I would like to use this system frequently	1	2	3	4	5
2. I found the system unnecessarily complex	1	2	3	4	5
3. I thought the system was easy to use	1	2	3	4	5
4. I think that I would need the support of a technical person to be able to use this system	1	2	3	4	5
5. I found the various functions in this system were well integrated	1	2	3	4	5
6. I thought there was too much inconsistency in this system	1	2	3	4	5
7. I would imagine that most people would learn to use this system very quickly	1	2	3	4	5
8. I found the system very cumbersome to use	1	2	3	4	5
9. I felt very confident using the system	1	2	3	4	5
10. I needed to learn a lot of things before I could get going with this system	1	2	3	4	5

**Figure 8.2:** SUS questionnaire [Broo96].

The SUS is used after the participant evaluated the system. It is done before any debriefing and discussion take place. If a participant is not able to respond to

a particular item the centre point of the scale should be marked. The result of SUS is a single number representing the overall usability of the system. It has a range between 0 to 100. Each item has a score range from 1 to 5 (sometimes also 0 to 4). For items 1,3,5,7 and 9 the score is the scale position provided by the participants minus 1. For items 2,4,6,8 and 10 the score is 5 minus the scale position provided by the participants. The sum of the scores is multiplied by 2.5 to obtain the overall SU score (see also [Broo96]).

## 8.2 Evaluation in the context of the ALERT project

On the way to the final version of the PANTEON tool we continuously evaluated the system in order to derive insights and to continuously improve the system and to adjust the underlying methodology and methods. Beside the evaluation in the context of the ALERT project we presented in [SeSS10b] an evaluation covering the subjective and objective evaluation of the modelling and refinement aspects. Before describing the final evaluation, below we briefly describe the evaluation in the context of the ALERT project.

In the ALERT project the PANTEON tool is used in order to create interaction patterns. An interaction pattern is the description of a situation which should be detected and reported in real-time. The interaction patterns are materialized as event patterns. In order to evaluate the PANTEON tool we set up a questionnaire based on the User Interface Rating Form (UIRF)<sup>3</sup>. The UIRF covers multiple user interface dimensions and used for programs that are still under development. The main goal of this evaluation was to find out the perceived usability of the system by business users. The PANTEON tool that has been subject to evaluation provided the pattern modelling, deployment and search functionality. The evaluation didn't include the event definition, pattern relationship detection and event pattern evolution. The evaluation includes two parameters for testing:

- **Usability** - measures the quality of the user interface.
- **Functionality** - does the program do what you want it to do.

The test was done with 13 participants from the open source software development domain. These participants were rather business users than IT experts. For each of the tests described below (taken from, User Interface Rating Form, <http://it.coe.uga.edu/treeves/edit8350/UIRF.html>), we define several evaluation questions (see appendix A1). Briefly, these questions covered the following dimensions:

- **Ease of Use** The Ease of Use is the parameter which evaluates the facility with which the user can use the software and interact with it. The important

---

<sup>3</sup><http://it.coe.uga.edu/treeves/edit8350/UIRF.htm>

criteria here is the user interface which is closely related to the ease of use and usually defines the user satisfaction with the specific program to some level. This parameter evaluation ranges from the perception that the program is very difficult to use to one that is perceived as being very easy to use.

- **Navigation** Navigation is the parameter which defines whether the user can easily move through the content of the program in an intentional manner. One important aspect of navigation is orientation which actually defines the degree to which the user feels oriented within the program i.e. knows in which functionality of the program he/she is working and how to exit or go to another part of the program. Navigation can be evaluated from the perception that a program is difficult to navigate to one that is perceived as being easy to navigate.
- **Cognitive Load** The user interface is the mechanism that allows perceptual, conceptual, and physical contacts with the interactive program. In terms of cognitive load, the user interface can seem unmanageable (i.e., confusing) at one end of the continuum and easily manageable (i.e., intuitive) at the other end.
- **Mapping** Mapping is the parameter which refers to the ability of the program to track and graphically lead the user through the program. A detailed mapping system provides help for the user in understanding the degree of their interaction with the software and insight into the parts of the software which were used, and those that have not been used. Interactive programs fall in a continuum of containing no mapping function to an appropriately powerful mapping function.
- **Screen Design** Screen Design is a particularly complex dimension of interactive programs that can easily be broken down into many sub-dimensions related to text, icons, graphics, colors and other visual aspects of interactive programs. The first problem with it is that the screen design principles did not keep the pace with the rapidly changing nature of interactive technology. Secondly, creative designers may sometimes intentionally violate screen design principles for effect or to otherwise focus the user's attention. Screen design is a dimension ranging from substantial violations of principles of screen design to general adherence to principles of screen design.
- **Knowledge Space Compatibility** Knowledge space is a parameter which refers to the network of concepts and relationships that compose the mental schema a user possesses about a given phenomena, topic or process. When a novice user initiates a search for information in an interactive program, the interface should be powerful enough so that the user perceives the resulting information as compatible with his or her current knowledge space. If the information received is not perceived as relevant to the search strategies used by the user, the system will be perceived as incompatible.

- Information Presentation** The Information Presentation dimension is concerned with whether the information contained in the knowledge space of an interactive program is presented in an understandable form. This is very important because regardless of the elegance of the user interface design, the program is useless if the information it is intended for is incomprehensible to the user. Information presentation is defined as a dimension ranging from unclear to clear.
- Aesthetics** The Aesthetics parameter refers to the artistic aspects of interactive programs in the sense of possessing beauty or elegance. The aesthetics dimension of the user interface of a program is defined as ranging from displeasing to pleasing.

Figure 8.3 displays some of the results for the user interface evaluation. The whole results can be found in the appendix A2. Most of the participants evaluated the system rather positive. The usability of interaction pattern modelling, search and deployment was perceived mostly very easy.

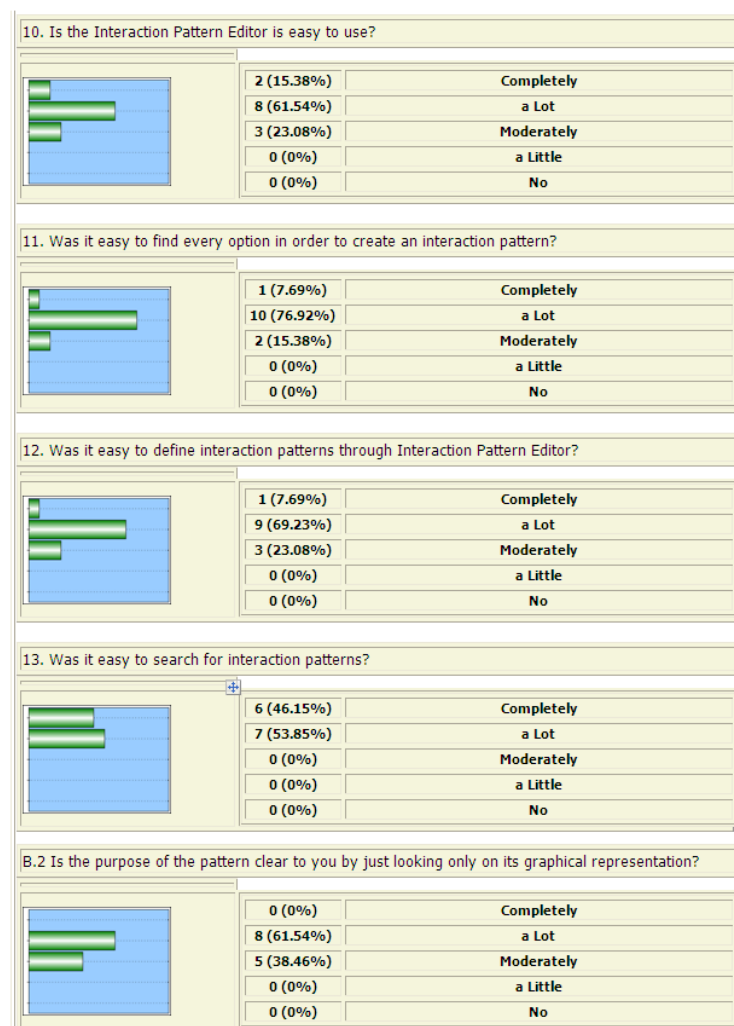


Figure 8.3: Selected results regarding the user interface.

The general conclusion of the evaluation study was that the purpose of PANTEON is mostly clear to the participants. It satisfies their expectations, particularly the need for a simpler method for new pattern definition and pattern retrieval. Furthermore, the analysis shows that the functionality of PANTEON is considered mostly as being useful. PANTEON speeds up their work and helps in resolving their problems regarding the definition of more complex patterns for real-time notifications about relevant situations. It is generally easy to use, to find every option, to define interaction patterns, to search for interaction patterns and to update patterns. Beside that the main insight for us was that offering the PANTEON tool to business users without introducing the basic CEP concepts might cause problems during the usage of the tool.

From the ALERT project we learned that a collaborative event pattern development by a bigger team could be very relevant. Taking this recommendation into account we extended the system with a user management functionality. However, we consider the collaborative event pattern development as an important feature that could be an extension of the PANTEON tool in the future.

## 8.3 Final evaluation based on SUS

In the final evaluation our goal was to get statistically significant numbers about the perceived usability and functionality of the full PANTEON tool and how well the participants perform by using different settings for the same task. According to Jakob Nielsen, 20 test users are needed in order to get statistically significant numbers<sup>4</sup>.

### 8.3.1 Evaluation methodology

For the final evaluation we created the following three evaluation settings.

- **S1 - Event pattern management with the full version of PANTEON** The full version of PANTEON provides the refinement of event pattern through the event pattern relationship detection, the detection of best practice patterns, the search functionality and the evolution support.
- **S2 - Event pattern management with the simple version of PANTEON** The simple version of PANTEON provided only the modelling and search functionality. Neither the refinement nor the evolution of event patterns were supported.
- **S3 - Event pattern management with the Esper CEP engine** The Esper CEP engine is one of the most popular CEP engines. It provides a

---

<sup>4</sup><http://www.useit.com/alertbox/20000319.html>

SQL-like language in order to define event patterns. For our evaluation we used the version written in the programming language Java. Esper does not provide any kind of user interface. An event pattern must be written as part of the programming code.

While S1 was the primary system of evaluation S2 and S3 were set up in order to put the evaluation results in a more descriptive context. However, our initial evaluation goal to compare the full PANTEON tool with a comparable system could not be realized due to the lack of similar tools with similar functionalities.

For the evaluation we acquired 20 participants including students, researchers and business users with computer science background. We selected our participants according to their knowledge about CEP and the Esper CEP engine. The participants were categorized as member of one of the following groups:

- **Experts** The participants of this group were familiar with CEP systems. All of them had experiences with the Esper CEP engine and had already defined several event patterns in Esper. All of them used the Esper engine in research projects. Hence, they were familiar very well with the concept of event patterns. Four participants had previous experience with PANTEON. The experience was only related to the manual event pattern generation using PANTEON without any system support. In this group we had 8 participants.
- **Novices** The participants of this group were not familiar with CEP systems. None of them have ever experienced with a CEP system or defined an event pattern. They have never experienced with the PANTEON tool and the underlying event pattern life cycle methodology. In this group we had 12 participants.

Having these three settings and these two groups our goal was to compare the task completion time, the perceived usability in the different settings and the failure rate per task. Since the effort to learn the Esper CEP engine is rather high we assigned to S3 only the Experts. They were familiar with the engine and its underlying event pattern language and syntax. For each setting we set up an Esper CEP engine for the event pattern matching. We further implemented a simulation component that was continuously generating the events that were needed in the different tasks in order to simulate the triggering and execution of the event pattern.

The evaluation in each setting started with an introductory part where we explained the procedure of the evaluation. The introduction was different for each setting:

- **Introduction - S1** For S1 we made an introduction to the full version of the PANTEON tool and presented the different elements of it. The participants



were asked to define several event patterns in order to become familiar with the system. Additionally we gave them the possibility to play around with the system and to experience with the pattern relation detection and the evolution functionality of the PANTEON tool.

- **Introduction - S2** For S2 we made an introduction to the simplest version of the PANTEON tool and presented the different elements of it. The participants were asked to play around with the system and to define several event patterns and to search for event patterns in order to become familiar with the system.
- **Introduction - S3** Since we assigned S3 only to participants who were familiar with CEP and the Esper engine, we printed out the Esper documentation and gave them this documentation to prepare themselves for the evaluation. We explained the event operators that will be relevant for the evaluation. The documentation, either online or the print version, could be used during the evaluation.

### 8.3.2 Scenario definition

Since there doesn't exist any data set we selected the US presidential election 2012 as our evaluation scenario. As an event source we used the Twitter data stream where Tweets are delivered continuously. Twitter has been recognized as a good source for event data. For our evaluation we were more interested in the entity patterns in the Twitter data stream. In order to detect entities each Tweet is sent through the OpenCalais<sup>5</sup> service. The OpenCalais service provides semantic information for text input. It detects beside other the following entities; *City, Company, Country, Facility, Movie, Organization, Person, PoliticalEvent, TVShow*. We parsed the RDF data which is delivered by the OpenCalais service and created for each instance an event. Each event has a property called *Name*. The value of this property is set with the value of the delivered instance.

**Task 1 - Event pattern modelling** The aim of task 1 was to model several event patterns. The events that were needed for the modelling of the event patterns have been defined beforehand. In PANTEON we used the event modeller from Chapter 5 in order to define the events. For Esper we defined the relevant events in Java objects format which is supported by Esper.

- **T1.1** Define an event pattern where a person event with the value equal to "Barack Obama" cooccurs with a country event with the value equal to "Iraq".

---

<sup>5</sup><http://www.opencalais.com>

- **T1.2** Define an event pattern where a person event with the value equal to "Barack Obama" or "Mitt Romney" cooccurs with person event with value equal to "George W. Bush".
- **T1.3** Define an event pattern where a person event with the value equal to "Mitt Romney" cooccurs with a country event with the value equals to "Afghanistan" without the occurrence of a person event where the value is equal to "Barack Obama".
- **T1.4** Define an event pattern where a person event with the value equal to "Angela Merkel" is followed by a country event with the value equal to country "Greece" and a person event with the value equal to "Hillary Clinton" within 60 seconds.

**Task 2 - Event pattern knowledge reuse** The aim of this task was to access existing event pattern knowledge and extend event patterns that have already been modelled by us. In order to do that the participants were asked to search for these patterns, and to extend these patterns with new events, event operators or property values. The pattern repository of PANTEON included 50 event patterns. The same event patterns have been modelled for S3 where Esper. For Esper we modelled the event patterns in one single file that could be used by the participants who were assigned to Esper to complete the tasks.

- **T2.1** Find the event pattern that includes at least two country events and two person events. All events are connected by the conjunction operator (AND).
- **T2.2** Find the event pattern that includes two sequence operators (SEQ) that are connected again by a disjunction operator (OR). Please replace the OR operator by an AND operator.
- **T2.3** Find the event pattern that extends the pattern which has been defined in T1.1. Extend this pattern again by connecting an arbitrary organization event to the conjunction operator (OR).
- **T2.4** Find the most used event pattern fragment containing the person "Barack Obama" and extend this fragment with an additional country event by setting the country value to "Germany".

**Task 3 - Event pattern adaptation** The aim of task 3 was to detect event patterns that are subject to evolution. For this task we set up an Esper CEP engine and fed the engine with the events from Twitter. The events were simulated based on previous entity occurrences in Twitter. For every participant we restarted the system and initialized our evolution component. The monitoring period was set to one minute and the length of the monitoring window was set to the value of 5.

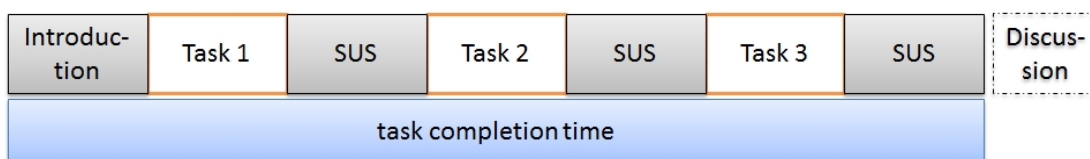
Further we created an output panel where the participants were able to see the results of the matching in simple text format.

We simulated the following situations where the participants should analyse whether there are event patterns that are not working correctly and if yes what might cause this malfunction. We simulated the following scenarios:

- **T3.1 Zero triggering** In this scenario we selected T1.1 and generated only the person events. In this case the pattern has been executed but not triggered. The participants should identify that the country event was not produced any more and therefore update the pattern definition if necessary.
- **T3.2 Doubled triggering** In this scenario we selected the T1.4 and generated two times more events than usual. As a result the pattern may lead to an information overload and therefore could be a candidate to evolution. The participants should identify this pattern and explain the reason for this pattern execution behaviour.
- **T3.3 General pattern definition** In this scenario we selected an event pattern where the person "Barack Obama" cooccurs with the country "America". During the simulation the quotient of this pattern was 1 which means that both events occur always together. This indicates that the pattern definition is too general and hence suboptimal.

### 8.3.3 Task completion procedure

Figure 8.4 displays the procedure that has been applied to all participants during the evaluation. The process has been applied to all three settings. After each task we applied the SUS questionnaire. The evaluation was concluded by a final discussion in order to receive additional feedback and suggestions.



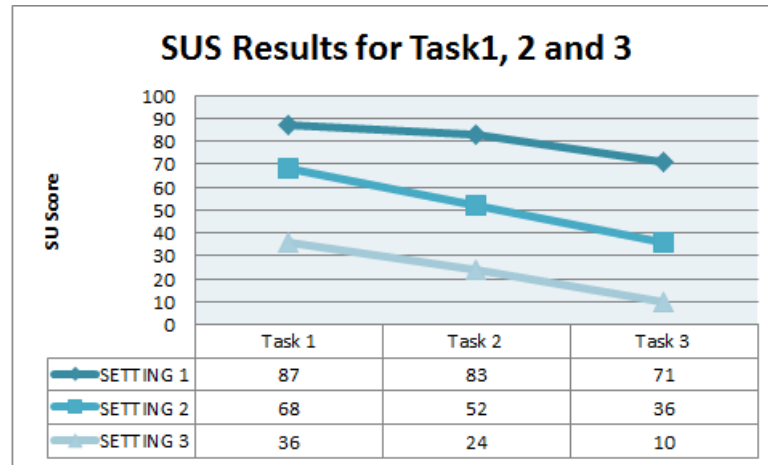
**Figure 8.4:** The process of the final evaluation.

During the evaluation we measured also the task completion time. The task completion time started whenever the participants started solving the task in PANTEON until they gave us a signal that they are finished. The task completion time was measured in seconds. The participants started with setting S2 and continued with S1. As for the setting S3 we set up a single evaluation where only the 8 participants who were familiar with the Esper CEP engine were involved and was done after S1.

### 8.3.4 Evaluation results

We had in total 120 filled in SUS questionnaires for S1 and S2. For S3 we had 24 filled in SUS questionnaires.

Figure 8.5 displays the SU score for the three settings and the three tasks. It is noticeable that the SU score decreases for all settings with increasing complexity of the tasks. The maximum possible SU score is 100. The higher the value is the better the participants evaluated the usability of the system. Remarkable



**Figure 8.5:** SU results for the three settings and tasks.

is that for S3 the SU score is far below the SU score for S1 and S2. Further it decreases to the value of 10 for task 3. As for S2 the value decreases from 68 over 52 to 36. In order to explain this discrepancy let us consider the task completion time and the failure rate for the different settings and tasks. Table 8.1 displays the minimum, maximum and the average task completion time for task 1. Although the task 1 is rather simple to solve there is a clear tendency towards

	S1	S2	S3
Min	6,5 min	7 min	7,5 min
Max	8,5 min	10,5 min	25 min
Avg	7,5 min	8,5 min	18 min

**Table 8.1:** Task completion time for task 1.

the full PANTEON version where the participants received suggestions based on the current modelling context and the pattern relationship detection presented in Chapter 6. In S1 the participants are 12% faster than the participants of S2. Compared to S3 the users of the full PANTEON version are more than two times faster. Of course being faster does not mean that they were effective. For that reason we calculated the failure rate of the participants. Task 1 was completed by all participants of each setting. Table 8.2 displays the total and the average failure rate of the three settings. The total failure rate is the number of all failures done by the participants. The average value is calculated by dividing the total number through the number of participants.

	S1	S2	S3
Failure Total	15	19	17
Failure Avg	0,75	0,95	2,13

**Table 8.2:** Failure rate for task 1 (The average value is calculated by dividing the number of total failures by the number of participants (S1 and S2 20 participants)(S3 8 participants)).

In order to have a correct event pattern both the pattern schema and value definition should be correct. While the schema definition considers whether the right number of events is connected to the event operator, the value definition considers whether the values of the events or event operators are set correctly. We didn't consider misspellings in value definition as an error. Every missing event to operator or operator to operator connection and wrong value definition increases the failure rate by 1.

Our observation was that with less tool support the failure rate is increasing too. Especially with Esper where no tool support was available the participants are not aware of existing problems with the event pattern definition. The difference in failure rate between S1 and S2 arises from the issue that in S1 the participants were able to detect pattern relationships during the modelling of new event patterns. This suggestion leads especially in the T1.4 to less failures. T1.4 was considered as the most complex event pattern and the participants in S1 made use of the pattern relationship detection and hence made less failures.

Table 8.3 displays the minimum, maximum and the average task completion time for task 2. These values show that the full PANTEON version where the user are able to detect event pattern relationships on different levels is faster in solving the defined tasks. The search and reuse of event pattern is in S1 almost 2 times faster than in S2 and about 4 times faster than in S3.

	S1	S2	S3
Min	4 min	5,5 min	14 min
Max	9 min	17 min	37 min
Avg	6,5 min	11 min	24,5 min

**Table 8.3:** Task completion time for task 2.

Further let us consider the task completion time for task 2 in the context of the failure rate. While in S1 the average failure done by each participant is 1.1 while in S2 is 1.7 and in S3 2.88. The increasing task completion time and the higher failure rate of S2 and S3 are manifested also in the SU score shown in Figure 8.5. The value decreases for S2 from 68 to 52 while for S3 the value decreases from 36 to 24. The main discrepancy between the different settings is more visible in the last task which was about the detection of pattern candidates that might be adapted. Although this scenario was a simulation based on real Twitter data it displays the role of tool support for the event pattern management evolution.

	S1	S2	S3
Failure Total	22	34	23
Failure Avg	1.1	1.7	2.88

**Table 8.4:** Failure rate for task 2 (the average value is calculated by dividing the number of total failures by the number of participants (S1 and S2 20 participants)(S3 8 participants)).

However, the increasing task completion time and the higher failure rate of S2 and S3 are manifested also in the SU score shown in Figure 8.5. The value decreases for S2 from 68 to 52 while for S3 the value decreases from 36 to 24. The main discrepancy between the different settings is more visible in the last task which was about the detection of pattern candidates that might be adapted. Although this scenario was a simulation based on real Twitter data it displays the role of tool support for the event pattern management evolution.

Table 8.5 displays the minimum, maximum and the average task completion time for task 3. Task 3 was the most complex and difficult one. The SU score for S2 in task 3 was about 36 while for the S3 the value was about 10. In contrast to that for S1 the SU score was 71.

	S1	S2	S3
Min	9 min	25	28
Max	17 min	41	44
Avg	13 min	35	37

**Table 8.5:** Task completion time for task 3.

While in S1 19 participants solved all the tasks this number was for S2 and S3 3. The participants in S2 and S3 who solved the tasks were the same persons who were very well familiar with CEP systems and had solid knowledge about CEP and event patterns.

### 8.3.5 Discussion of the results

The goal of the evaluation was to find out to what degree the PANTEON tool can increase the efficiency and effectiveness of a pattern engineer. Additionally the final evaluation delivered quantitative results <sup>67</sup> regarding the usability and functionality of the full PANTEON tool. In order to compare these results we set up two additional settings. In one setting we deactivated all supportive functionalities of the PANTEON tool while in the other we offered the plain CEP engine Esper without any additional support. Because of the complexity of the Esper language we only assigned participants to this setting who have already experienced with Esper.

<sup>6</sup><http://www.nngroup.com/articles/quantitative-studies-how-many-users/>

<sup>7</sup>According to Nielsen, 20 users offers a reasonably tight confidence interval

The main observation we made is that with increasing complexity of the tasks there is a higher demand for additional tool support. Especially in task 2 and task 3 this was observable. Participants who were using the full PANTEON tool performed in every task better, were faster in completing the tasks and made less failures.

As a result we can conclude that, independently whether the participants are familiar with CEP or not, the full PANTEON tool eases the pattern definition, enables a proper access to the underlying pattern knowledge and supports the reuse of existing pattern knowledge. The capability of the PANTEON tool to observe the pattern execution statistics and suggest event pattern improvement is of paramount importance and is very well accepted. We can say that using the PANTEON tool the pattern engineer is not only efficient but also effective in managing event patterns. However, using the full PANTEON tool does not avoid failures as shown in the previous sections. In this case additional validation and verification mechanisms are needed in order to reduce further the number of incorrectly defined patterns.

## 8.4 Performance tests

In this section we describe some performance aspects of the PANTEON tool. Although the performance issue is not the main focus of this thesis, we think that presenting some performance results could help to understand the limitations of the PANTEON tool.

The results presented in this section were carried out on a Lenovo ThinkPad T410s with Intel i5CPU M560, 2.67 GHz, 8GB of RAM running Windows 7 Professional 64 bit.

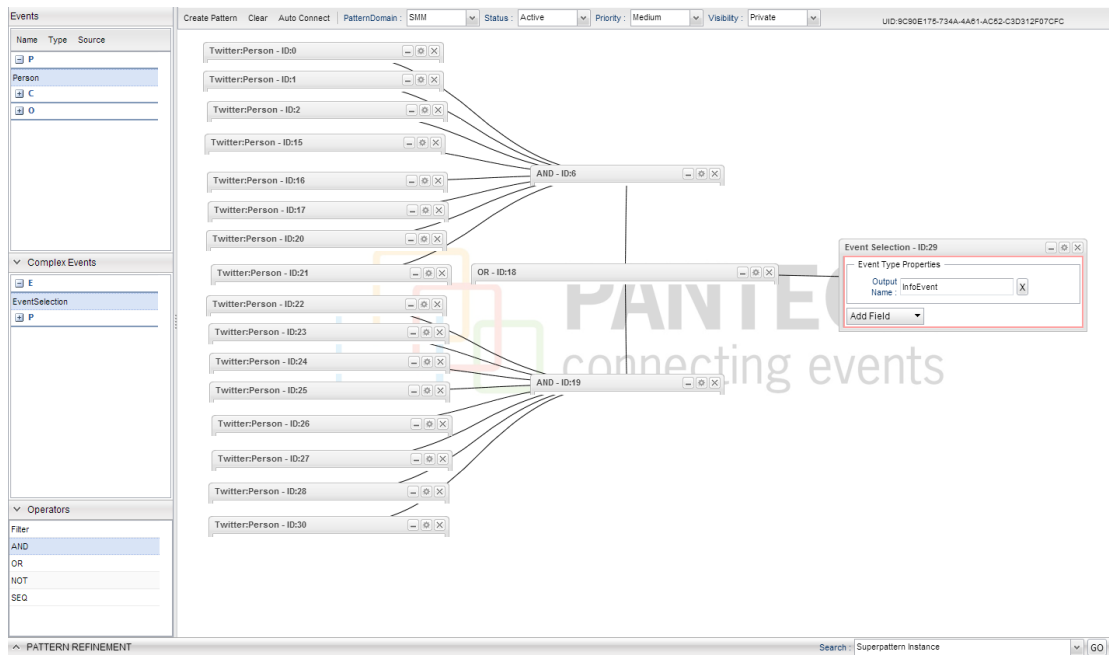
### 8.4.1 Modelling expressivity

We conducted several iterations to find out the maximum number of events and event operators that could be handled by the PANTEON tool. An event pattern with 100 events and more than 10 event operators is handled without any problems. The storage and the search (about 3500 patterns in pattern repository) needs less than 0,5 second. Using more than 120 nodes in the graphical interface leads to Google Web Toolkit(GWT) Developer Plugin<sup>8</sup> problems in the browser. However, having event patterns with such a number of events and event operators might be rather unrealistic. At least no scenario is known to us which has this amount of events and event operators. The usual way would be to split such complex scenarios into smaller pattern definition and to combine them to larger event

---

<sup>8</sup>The plugin is needed in order to run a GWT Programm in a web browser see also <https://developers.google.com/web-toolkit/gettingstarted>.

patterns. Figure 8.6 displays an event pattern with 100 events and multiple event operators. Such complex event patterns could have negative effects on the usability of the system used for modelling purposes.



**Figure 8.6:** An example of an event patten including 20 nodes.

## 8.4.2 Event pattern relation detection

For this test we implemented a pattern generator program that randomly generated event patterns. The event pattern generator was able to generate 500 distinct events with 10 properties. As for event operators the generator created patterns that include the sequence, disjunction, conjunction and negation operator with count and time window. The most expressive event pattern included 43 event operators and 800 events. The less expressive event pattern included two events and one event operator.

**Event pattern storage in Neo4J** We conducted an evaluation in order to measure the average time in order to store an event pattern as a graph in the Neo4J database. The storage of the event pattern consists of converting the output of PANTEON into a Neo4J acceptable format and the actual storage itself. Table 8.6 displays the result of the event pattern storage. For the most expressive event pattern (P2) the storage of the event pattern as a graph in Neo4J is around 600ms.

**Relationship detection** In order to detect an event pattern relationship we took as the modelling context one instance of the event pattern P2 from Table 8.6. The Neo4J database was feeded with randomly generated event patterns based on the pattern expressivity presented in Table 8.6. The size in Table 8.7 refers to the



Expressivity (P=Event pattern)	Time in milliseconds
P1: E = 2, EO = 1, SEP = 20	390 ms
P2: E = 40, EO = 5, SEP = 20	450 ms
P2: E = 800, EO = 43, SEP = 20	600 ms

**Table 8.6:** Transformation and storage of event patterns (P1,P2,P3) in Neo4J. E denotes the total number of events, EO denotes the total number of event operators and SEP denotes the number of properties for each event.

number of event patterns in the database. Table 8.7 displays the average response time. We repeated the test 100 times for each relationship detection in order to calculate the average response time.

	Size = 100	Size = 400	Size = 1600
Best practice pattern	$\leq 50$ ms	$\leq 240$ ms	$\leq 1150$ ms
Other	$\leq 20$ ms	$\leq 60$ ms	$\leq 200$ ms

**Table 8.7:** Response time for the event pattern graph relationship detection (in milliseconds).

As we can see from the Table 8.7 the response time increases linear.

### 8.4.3 Performance of the event pattern evolution component

We further conducted a test in order to have some quantitative data for the evolution component. Therefore we registered up to 5.000 event patterns and simulated their triggering and execution. For each event pattern we simulated 25000 event executions and triggered 1000 event patterns. We simulated an evolution detection every minute. For each event pattern the evolution component detected every expected evolution detection. The detection time is near real-time meaning that it detects an event pattern as candidate to evolution in less than 2 seconds after the last relevant event has arrived the component.

## 8.5 Conclusion

In this chapter we presented the evaluation of the PANTEON tool. We described previous evaluations where several aspects have been evaluated in order to collect the user satisfaction and further improvement suggestion. Further we presented a task-driven usability evaluation where several tasks have been defined and should be solved by the participants of the evaluation. We found out that with increasing task complexity the users' efficiency and effectiveness decrease enormously if no proper tool support is available. We defined several settings in order to measure the perceived usability, the task completion time and the success rate. In order to measure the usability we used the SUS questionnaire [Broo96].



# 9

## Conclusion

### 9.1 Summary

The main motivation behind this thesis came from the issue that the main research in Complex Event Processing (CEP) is mostly focused on an efficient matching of events by neglecting the management of event patterns. Event patterns are from the business point of view the most valuable asset in order to detect critical business situations. Unless this issue is not investigated the barriers for using CEP systems will be rather high. A pattern engineer who deals with the modelling and the maintenance of event patterns requires supportive tools, methods and methodologies in order to handle the complexity of event patterns in an efficient and effective way.

The starting point of this thesis was the analysis of existing research regarding the event pattern management. However, in 2008 when we started this research there were significantly less approaches targeting the event pattern management in CEP. However, we defined the necessity of an efficient and effective event pattern management by considering related areas like business rule management and active databases. Furthermore we defined requirements based on several research papers (see Chapter 3). Based on this requirement analysis we defined in Chapter 4 a methodology for event pattern life cycle management similar to the software development methodologies. Our methodology consists of three main phases namely Generation, Execution and Evolution forming a feedback loop. For each phase we defined tasks in order to support the pattern engineer in an efficient and effective

way. In Chapter 5 we presented a meta model for representing event patterns. This model is based on concept hierarchies in order to enable the relationship detection between event patterns. The relationships are used both during the modelling of new event patterns and later on during their evolution and adaptation.

In order to detect relationships we made use of graph theoretical concepts. Event patterns that are based on our meta model are represented as a graph. In Chapter 6 developed methods for detecting relationships between event patterns. We introduced the subsumption and the overlapping relationships in order to support the pattern engineer during the definition of new event patterns. These relationships enable the pattern engineer to access existing pattern knowledge and make use of it by refining a new event pattern. We further introduced the similarity relationship between event patterns in order to also access event patterns that are structurally not equal. The similarity relationship is based on the taxonomy and the features of events and event operators. In order to build libraries of most used event patterns we introduced the concept of best practice patterns where frequently used patterns are presented to the pattern engineer for further usage.

In order to evolve event patterns we introduced in Chapter 7 the concept of the transparent CEP engine. This can be considered as a paradigm shift from a black-box CEP engine where only the result of the matching is received to a white-box approach where every event consumption regarding an event pattern is monitored. Based on the concept of the transparent CEP engine we introduced the execution-driven event pattern evolution where past statistics for event pattern triggering and execution are compared to the current triggering and execution rate in order to detect deviations. Based on this deviation detection we enriched the pattern statistics with the quotient of event pattern triggering and execution, the frequency of event consumption and provide this information to the pattern engineer. Taking this information into account the pattern engineer is able to adapt the event pattern definition.

The methodology, language and the several methods were implemented in the PANTEON tool that provides a high-level graphical interface for event pattern management. It follows the graph-based programming paradigm where event patterns are defined as a graph where events, event operators and complex events are connected to each other forming a valid event pattern. The tool is able to detect inconsistencies in event pattern definition during the modelling of event patterns.

We conducted several evaluations in order to find out the perceived usability and functionality of the PANTEON tool. The pre-evaluations were helpful to get initial user feedback and to improve the methodology, language and the methods. They delivered helpful insights for the overall adjustment of our research. The final evaluation showed that the features and the user interface of the PANTEON tool are easy to learn and very helpful. Participants using the full PANTEON tool were faster in finishing the tasks, had a better success rate and a lower failure rate.

In the nutshell the main achievements of this thesis are:

- **The necessity of event pattern management;**
- **An event pattern life cycle methodology;**
- **A meta model for the management of event patterns;**
- **The relationship detection between event patterns;**
- **Evolution of event patterns based on pattern execution statistics;**
- **A high-level graphical user interface simplifying the modelling, deployment and evolution of event patterns;**

## 9.2 Future direction

This thesis is a first step towards an efficient and effective event pattern management. We provided an easy to use tool that is based on a profound methodology, language and methods. However, there is still space for future improvements:

**Collaborative pattern development** During several evaluations we found out that the participants are interested in collaborative pattern modelling. This could be especially relevant for organizations where the pattern knowledge is distributed across several departments. The final event pattern could be composed of different pattern fragments that are modelled in different departments by different pattern engineers. Although the PANTEON system enables the reuse of event patterns the collaborative development raises further open issues like real-time collaborative modelling and consistency checking of collaborative patterns which need to be researched.

**Domain dependent user interfaces** One of our assumptions was that proper user interfaces could increase the acceptance of CEP systems. The user interface of PANTEON is designed for business users who operate in the context of CEP systems. However, there is a big movement towards the offering of real-time situation detection for every day life where new kinds of high-level user interfaces are needed for hiding the underlying CEP technology from the user. The pattern definition could be implicit meaning that the user will not define the pattern explicitly like in PANTEON but rather the patterns are defined automatically by monitoring the click behaviour of the user.

**Pattern modelling driven by data mining and existing models** We have already described in Chapter 4 that data mining is one of the most important technologies that could ease the definition of event patterns. However, the challenge is how to detect an event pattern that is relevant although it is not frequently appearing.

**Event pattern evolution** The semi-automatic evolution of event patterns is from our point of view an essential part of an efficient and effective event pattern management. It helps the pattern engineer to keep an event pattern repository up-to date. Therefore the approach presented in this thesis needs to be further evaluated in different business settings in order to find the right size of monitoring window and monitoring period since these parameters are rather application and domain dependent.

**Extension of the event pattern management life cycle** Our event pattern management life cycle starts with the actual need to define an event pattern. However, like presented in [ViKD10], described in Section 4.5 and the need itself requires additional methodological approaches. The pattern engineer must be supported also during the initial need identification to find the relevant business situations that need to be detected through event pattern detection.

### 9.3 Conclusion

In this thesis we presented an approach to support the pattern engineer systematically during the event pattern modelling, deployment and evolution. The graph-based representation of event patterns based on our meta model enables the reuse of existing pattern knowledge and supports the pattern engineer to be more efficient and effective during pattern modelling. In order to keep the event pattern repository up-to-date our event pattern evolution approach detects event pattern execution outliers and supports the pattern engineer to adapt event patterns. Overall the approaches presented in this thesis increase the efficiency and effectiveness of the pattern engineer during the event pattern life cycle.

# A

## Appendix

### A.1 ALERT Questionnaire

## Introduction

*The PANTEON interaction pattern editor is a design environment to define, deploy and evolve interaction patterns. An interaction pattern is the description of a situation which should be detected and reported and is important to the user. For the definition of an interaction pattern PANTEON provides a set of events and a set of event operators in order to model an interaction pattern. The deployment of an interaction pattern will be done in the CEP engine which matches an interaction pattern against incoming ALERT events. The owner of the interaction pattern is informed about the occurrence of the situation as soon as it happens.*

*Complex Event Processing (CEP) deals with the analysis of streams of continuously arriving events with the goal of identifying instances of predefined meaningful patterns.*

*The goal of this evaluation is to find out users perception about defining and updating interaction patterns using the PANTEON tool. The actual matching and notification is not part of PANTEON. Only the design relevant issues are part of this evaluation.*

**Please take a moment to first fill in the basic information about yourself and then answer the questions.**

1. Profession:

---

2. Company:

---

3. Experience and Expectations

1- Have you ever used any kind of system with alerting/notification functionalities in general? (like RSS feeds, BTS notification, Email notification, Google Alerts etc.)

Yes,  No

If yes, which system did you use and for what purposes?

---

2- Have you ever used a Complex Event Processing system?

Yes,  No

If yes, which system did you use and for what purposes?



---

3- If you are a software developer, do you use any kind of alerting/notification system in your software development projects?

Yes,  No

If yes, which system did you use and for what purposes?

---

4- Do you think a system with alerting/notification functionalities would increase the awareness of the team members in a software development project?

Yes,  No

Why?

---

5- Which functionality should an alerting/notification system provide? (multiple selection possible)

Real-time functionality

Expressive language to define the alerts/notifications

Multiple alerting/notification channels (like SMS, Email etc.)

A graphical user interface to define and change alerting/notification rules

Recommendation of alerting/notification rules

Other:

---

6- Do you think an alerting/notification system in general would be a useful system for your daily business life?

Yes,  No

Why?

---

7- What do you expect from a tool to define and maintain interaction patterns? An interaction pattern is the description of a situation which should be detected and reported and is important to the user. (multiple selection possible)

To ease the definition of new patterns

- To ease the maintenance of existing patterns
- To define better patterns
- To search for patterns
- To save time

Other:

---

8- How would you like to define a new interaction pattern? (multiple selection possible)

- Using a graphical editor
- Coding the pattern using a programming language

If you selected both when do you prefer to use which one?

---

#### 4. Interaction Pattern Editor Quality

PART A:

A.1 What do you think who should be the primary user of Interaction Pattern Editor?

- Business user
- Programmer
- Don't know

Other

---

A.2 Is the purpose of Interaction Pattern Editor clear?

- No,  a Little,  Moderately,  a Lot,  Completely

A.3 Did the Interaction Pattern Editor satisfy your expectations in general?

- No,  a Little,  Moderately,  a Lot,  Completely

If no please tell us why?

---

A.4 Which expectations were satisfied? (multiple selections possible)

- To ease the definition of new patterns
- To ease the maintenance of existing patterns
- To define better patterns
- To search for patterns
- To save time

Other:

---

A.5 Do you find the functionalities of the Interaction Pattern useful?

- No,  Yes

If no please tell us why and which functionalities you missed?

---

A.6 Did you find any user interface element or functionality in the Interaction Pattern Editor irrelevant?

- No,  Yes

If yes, please tell us what elements or functionalities are irrelevant and what are your suggestions?

---

A.7 Is the Interaction Pattern Editor a tool for defining alerting/notification rules?

- No,  Yes

PART B:

B.1 Did you find that the created pattern match what you intended to create?

- No,  a Little,  Moderately,  a Lot,  Completely

B.2 Is the purpose of the pattern clear to you by just looking only on its graphical representation?

No,  a Little,  Moderately,  a Lot,  Completely

If no please tell us why?

---

5. Describe how skilful you should be in using a notification system, in order to successfully use the Interaction Pattern Editor:  
 Novice,  Advanced Beginner,  Competent,  Advanced,  Expert
6. Describe how skilful you should be in creating alerting/notification rules for successfully define interaction patterns:  
 Novice,  Advanced Beginner,  Competent,  Advanced,  Expert
7. Describe how skilful you should be in using other pattern editors, in order to successfully use the Interaction Pattern Editor:  
 Novice,  Advanced Beginner,  Competent,  Advanced,  Expert
8. Do you think that the Interaction Pattern Editor could speed up your work?  
 No,  a Little,  Moderately,  a Lot,  Completely
9. Do you think that the definition of an interaction pattern could reduce the time needed for resolving a problem?  
 No,  a Little,  Moderately,  a Lot,  Completely
10. Is the Interaction Pattern Editor is easy to use?  
 No,  a Little,  Moderately,  a Lot,  Completely
11. Was it easy to find every option in order to create an interaction pattern?  
 No,  a Little,  Moderately,  a Lot,  Completely
12. Was it easy to define interaction patterns through Interaction Pattern Editor?  
 No,  a Little,  Moderately,  a Lot,  Completely
13. Was it easy to search for interaction patterns?  
 No,  a Little,  Moderately,  a Lot,  Completely
14. Was it easy to find your active patterns?  
 No,  a Little,  Moderately,  a Lot,  Completely
15. Was it easy to update your pattern in Interaction Pattern Editor?  
 No,  a Little,  Moderately,  a Lot,  Completely

**16.** If the answer to any of the questions 8-15 is no, a Little or Moderately, please tell us why:

---

**17.** Was it easy to get all the patterns you are interested by using the search functionality of the Interaction Pattern Editor?

No,  a Little,  Moderately,  a Lot,  Completely

**18.** Were all of the search criteria included in the search results for the Interaction Pattern Editor?

No,  a Little,  Moderately,  a Lot,  Completely

**19.** Do you understand search results when searching patterns in Interaction Pattern Editor?

No,  a Little,  Moderately,  a Lot,  Completely

**20.** If the answer to any of the questions 17-19 is no, a Little or Moderately, please tell us why:

---

**21.** How much time did you spend to solve all the tasks?

**22.** Was the task description easy to understand:

No,  a Little,  Moderately,  a Lot,  Completely

Suggestions to improve the task description:

---

**23.** Do you have any other suggestions to improve the Interaction Pattern Editor?

---


**Thanks for filling in this questionnaire!**

## **A.2 ALERT Questionnaire Results**


## Questionnaire Interaction Pattern Editor

### Questionnaire Interaction Pattern Editor


3.1. Have you ever used any kind of system with alerting/notification functionalities in general? (like RSS feeds, BTS notification, Email notification, Google Alerts etc.)

	1 (16.67%)	No
	5 (83.33%)	Yes

3.2 Have you ever used a Complex Event Processing system?


	6 (100%)	No
	0 (0%)	Yes

3.3 If you are a software developer, do you use any kind of alerting/notification system in your software development projects?


	2 (33.33%)	No

	4 (66.67%)	Yes

3.4 Do you think a system with alerting/notification functionalities would increase the awareness of the team members in a software development project?


	0 (0%)	No
	6 (100%)	Yes

3.5 Which functionality should an alerting/notification system provide? (multiple selection possible)


	5 (27.78%)	Recommendation of alerting/notification rules
	4 (22.22%)	A graphical user interface to define and change alerting/notification rules
	3 (16.67%)	Multiple alerting/notification channels (like SMS, Email etc.)
	6 (33.33%)	Real-time functionality

3.6 Do you think an alerting/notification system in general would be a useful system for your daily business life?




	<b>1</b> <b>(16.67%)</b>	<b>No</b>
	<b>5</b> <b>(83.33%)</b>	<b>Yes</b>

3.7 What do you expect from a tool to define and maintain interaction patterns? An interaction pattern is the description of a situation which should be detected and reported and is important to the user. (multiple selection possible)

	<b>3</b> <b>(16.67%)</b>	<b>To save time</b>
	<b>4</b> <b>(22.22%)</b>	<b>To search for patterns</b>
	<b>2</b> <b>(11.11%)</b>	<b>To define better patterns</b>
	<b>3</b> <b>(16.67%)</b>	<b>To ease the maintenance of existing patterns</b>
	<b>6</b> <b>(33.33%)</b>	<b>To ease the definition of new patterns</b>

3.8 How would you like to define a new interaction pattern? (multiple selection possible)

	<b>2</b> <b>(28.57%)</b>	<b>Coding the pattern using a programming language</b>
	<b>5</b> <b>(71.43%)</b>	<b>Using a graphical editor</b>

A.1 What do you think who should be the primary user of Interaction Pattern Editor?

?	0 (0%)	Don't know
	5 (83.33%)	Programmer
	1 (16.67%)	Business user

**A.2 Is the purpose of Interaction Pattern Editor clear?**

?	0 (0%)	Completely
	2 (33.33%)	a Lot
	3 (50%)	Moderately
	1 (16.67%)	a Little
	0 (0%)	No

**A.3 Did the Interaction Pattern Editor satisfy your expectations in general?**

?	0 (0%)	Completely
	2 (33.33%)	a Lot
	4 (66.67%)	Moderately
	0 (0%)	a Little
	0 (0%)	No

**A.4 Which expectations were satisfied? (multiple selections possible)**

?	2 (14.29%)	To save time
	4 (28.57%)	To search for patterns
	1	To define better

?		
	(7.14%)	patterns
	1 (7.14%)	To ease the maintenance of existing patterns
	6 (42.86%)	To ease the definition of new patterns

A.5 Do you find the functionalities of the Interaction Pattern useful?

?		
	0 (0%)	No
	6 (100%)	Yes

A.6 Did you find any user interface element or functionality in the Interaction Pattern Editor irrelevant?

?		
	6 (100%)	No
	0 (0%)	Yes

A.7 Is the Interaction Pattern Editor a tool for defining alerting/notification rules?

--	--

?	0 (0%)	No
	6 (100%)	Yes

B.1 Did you find that the created pattern match what you intended to create?

?	0 (0%)	Completely
	3 (50%)	a Lot
	3 (50%)	Moderately
	0 (0%)	a Little
	0 (0%)	No

B.2 Is the purpose of the pattern clear to you by just looking only on its graphical representation?

?	0 (0%)	Completely
	3 (50%)	a Lot
	3 (50%)	Moderately
	0 (0%)	a Little
	0 (0%)	No

5. Describe how skillful you should be in using a notification system, in order to successfully use the Interaction Pattern Editor:

	0 (0%)	Expert

?	0 (0%)	Advanced
	4 (66.67%)	Competent
	2 (33.33%)	Advanced Beginner
	0 (0%)	Novice

6. Describe how skilful you should be in creating alerting/notification rules for successfully define interaction patterns:

?	0 (0%)	Expert
	0 (0%)	Advanced
	5 (83.33%)	Competent
	1 (16.67%)	Advanced Beginner
	0 (0%)	Novice

7. Describe how skilful you should be in using other pattern editors, in order to successfully use the Interaction Pattern Editor:

?	0 (0%)	Expert
	0 (0%)	Advanced
	2 (33.33%)	Competent
	3 (50%)	Advanced Beginner
	1 (16.67%)	Novice

8. Do you think that the Interaction Pattern Editor could speed up your work?

0	
---	--

?	(0%)	<b>Completely</b>
	3 (50%)	<b>a Lot</b>
	3 (50%)	<b>Moderately</b>
	0 (0%)	<b>a Little</b>
	0 (0%)	<b>No</b>

9. Do you think that the definition of an interaction pattern could reduce the time needed for resolving a problem?

?	1 (16.67%)	<b>Completely</b>
	2 (33.33%)	<b>a Lot</b>
	2 (33.33%)	<b>Moderately</b>
	1 (16.67%)	<b>a Little</b>
	0 (0%)	<b>No</b>

10. Is the Interaction Pattern Editor is easy to use?

?	1 (16.67%)	<b>Completely</b>
	3 (50%)	<b>a Lot</b>
	2 (33.33%)	<b>Moderately</b>
	0 (0%)	<b>a Little</b>
	0 (0%)	<b>No</b>

11. Was it easy to find every option in order to create an interaction pattern?

?	1	<b>Completely</b>
---	---	-------------------

?	(16.67%)	
	4 (66.67%)	a Lot
	1 (16.67%)	Moderately
	0 (0%)	a Little
	0 (0%)	No

12. Was it easy to define interaction patterns through Interaction Pattern Editor?

?	1 (16.67%)	Completely
	3 (50%)	a Lot
	2 (33.33%)	Moderately
	0 (0%)	a Little
	0 (0%)	No

13. Was it easy to search for interaction patterns?

?	3 (50%)	Completely
	3 (50%)	a Lot
	0 (0%)	Moderately
	0 (0%)	a Little
	0 (0%)	No

14. Was it easy to find your active patterns?

?	2 (33.33%)	Completely
	4 (66.67%)	a Lot

?		
	0 (0%)	Moderately
	0 (0%)	a Little
	0 (0%)	No

15. Was it easy to update your pattern in Interaction Pattern Editor?

?	1 (16.67%)	Completely
	3 (50%)	a Lot
	2 (33.33%)	Moderately
	0 (0%)	a Little
	0 (0%)	No

17. Was it easy to get all the patterns you are interested by using the search functionality of the Interaction Pattern Editor?

?	0 (0%)	Completely
	5 (83.33%)	a Lot
	1 (16.67%)	Moderately
	0 (0%)	a Little
	0 (0%)	No

18. Were all of the search criteria included in the search results for the Interaction Pattern Editor?

	1	Completely
--	---	------------



?	(16.67%)	
	4 (66.67%)	a Lot
	1 (16.67%)	Moderately
	0 (0%)	a Little
	0 (0%)	No

19. Do you understand search results when searching patterns in Interaction Pattern Editor?

?	0 (0%)	Completely
	4 (66.67%)	a Lot
	2 (33.33%)	Moderately
	0 (0%)	a Little
	0 (0%)	No

21. Was the task description easy to understand:

?	0 (0%)	Completely
	4 (66.67%)	a Lot
	2 (33.33%)	Moderately
	0 (0%)	a Little
	0 (0%)	No

### **A.3 EPTS event processing use case template**

### Pattern Detection

There are lots of different types of patterns that can be detected. Simple patterns could be that a property value is above/below a specific threshold, or that a specific event occurred. More complex patterns could involve temporal or spatial windows, with events occurring (or not) in a specific order, or a calculation over a window exceeding a threshold, etc. Please describe your patterns here. Examples of types of patterns are given below.

Question	Response
Discover <b>associations/relationships</b> between events	
<b>Correlate</b> events using common event properties, or perform <b>matching</b> between two or more streams	
Discover <b>specific sequences</b> of events based on event types and/or values of event properties	
Discover <b>trends over sequences</b> of events (e.g. An event property value is consistently rising)	
<b>Other</b> Describe any other types of patterns/thresholds/triggers used.	



# Bibliography

- [ABBC<sup>+</sup>04] A. Arasu, B. Babcock, S. Babu, J. Cieslewicz, M. Datar, K. Ito, R. Motwani, U. Srivastava and J. Widom. STREAM: The Stanford Data Stream Management System. Technical Report 2004-20, Stanford InfoLab, 2004.
- [ACcC<sup>+</sup>03] D. J. Abadi, D. Carney, U. Çetintemel, M. Cherniack, C. Convey, S. Lee, M. Stonebraker, N. Tatbul and S. Zdonik. Aurora: A New Model and Architecture for Data Stream Management. Band 12, Secaucus, NJ, USA, August 2003. Springer-Verlag New York, Inc., S. 120–139.
- [AdBE00] A. Adi, D. Botzer and O. Etzion. Semantic Event Model and its Implication on Situation Detection. In *Proceedings of the 8th European Conference on Information Systems, Trends in Information and Communication Systems for the 21st Century (ECIS), Vienna, Austria, July 3-5, 2000*, S. 320–325.
- [AdEt02] A. Adi and O. Etzion. The situation manager rule language. In *Proceedings of the International Workshop on Rule Markup Languages for Business Rules on the Semantic Web, 14 June 2002, Sardinia, Italy (In conjunction with the First International Semantic Web Conference ISWC2002 and hosted by SIG2 of the OntoWeb Network, 2002*.
- [AFRS11] D. Anicic, P. Fodor, S. Rudolph and N. Stojanovic. EP-SPARQL: a unified language for event processing and stream reasoning. In S. Srinivasan, K. Ramamritham, A. Kumar, M. P. Ravindra, E. Bertino and R. Kumar (Hrsg.), *WWW*. ACM, 2011, S. 635–644.
- [AgIS93] R. Agrawal, T. Imieliński and A. Swami. Mining association rules between sets of items in large databases. In *Proceedings of the 1993 ACM SIGMOD international conference on Management of data, SIGMOD '93, New York, NY, USA, 1993*. S. 207–216.
- [Amar11] R. J. Amarnath Gupta. *Managing Event Information: Modeling, Retrieval, and Applications*. Morgan and Claypool Publishers. 2011.

- [AnBK03] T. Andreasen, H. Bulskov and R. Knappe. From Ontology over Similarity to Query Evaluation pp. 39-50, in R. Bernardi, M. Moortgat (Eds.): 2nd CoLogNET-ElsNET Symposium - Questions and Answers: Theoretical and Applied Perspectives, Amsterdam, Holland, 2003.
- [ArNH86] J. D. Arthur, R. E. Nance and S. M. Henry. A Procedural Approach to Evaluating Software Development Methodologies: The Foundation. Technischer Bericht, Blacksburg, VA, USA, 1986.
- [ASSA+99] M. K. Aguilera, R. E. Strom, D. C. Sturman, M. Astley and T. D. Chandra. Matching events in a content-based subscription system. In *PODC '99: Proceedings of the eighteenth annual ACM symposium on Principles of distributed computing*, New York, NY, USA, 1999. ACM, S. 53–61.
- [BDGH+07] L. Brenna, A. Demers, J. Gehrke, M. Hong, J. Ossher, B. Panda, M. Riedewald, M. Thatte and W. White. Cayuga: a high-performance event processing engine. In *SIGMOD '07: Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data*, New York, NY, USA, 2007. ACM, S. 1100–1102.
- [BeMi02] J. Benedek and T. Miner. Measuring desirability: New methods for evaluating desirability in a usability lab setting. In *Usability Professional Association, UPA 2002, Orlando, FL, USA*, 2002.
- [BlSa05] E. Blomqvist and K. Sandkuhl. Patterns in Ontology Engineering: Classification of Ontology Patterns. In *Proceedings of the International Conference on Enterprise Information Systems 2005*, Miami Beach, Florida, May 24-28 2005.
- [Bond76] J. A. Bondy. *Graph Theory With Applications*. Elsevier Science Ltd. 1976.
- [Broo96] J. Brooke. SUS: A quick and dirty usability scale. In P. W. Jordan, B. Weerdmeester, A. Thomas and I. L. McLelland (Hrsg.), *Usability evaluation in industry*. Taylor and Francis, London, 1996.
- [ChCC07] K. M. Chandy, M. Charpentier and A. Capponi. Towards a theory of events. In *DEBS '07: Proceedings of the 2007 Inaugural International Conference on Distributed Event-Based Systems*, New York, NY, USA, 2007. ACM, S. 180–187.
- [ChDN88] J. P. Chin, V. A. Diehl and K. L. Norman. Development of an instrument measuring user satisfaction of the human-computer interface. In *Proceedings of the SIGCHI conference on Human factors in computing systems, CHI '88*, New York, NY, USA, 1988. ACM, S. 213–218.

- [ChEA11] M. K. Chandy, O. Etzion and R. von Ammon. 10201 Executive Summary and Manifesto – Event Processing. In K. M. Chandy, O. Etzion and R. von Ammon (Hrsg.), *Event Processing*, Nr. 10201 der Dagstuhl Seminar Proceedings, Dagstuhl, Germany, 2011. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany.
- [ChMi94] S. Chakravarthy and D. Mishra. Snoop: an expressive event specification language for active databases. *Data Knowl. Eng.* 14(1), November 1994, S. 1–26.
- [ChSc09] M. K. Chandy and R. W. Schulte. *Event Processing: Designing IT Systems for Agile Companies*. McGraw-Hill Professional. 2009.
- [DaJB96] T. H. Davenport, S. L. Jarvenpaa and M. C. Beers. Improving Knowledge Work Processes. *Sloan Management Review* 37(4), 1996, S. 53–65.
- [DiGG95] K. Dittrich, S. Gatzju and A. Geppert. The active database management system manifesto: A rulebase of ADBMS features. In T. Sellis (Hrsg.), *Rules in Database Systems*, Band 985 der *Lecture Notes in Computer Science*, S. 1–17. Springer Berlin Heidelberg, 1995.
- [Dixo00] N. Dixon. *Common Knowledge: How Companies Thrive by Sharing What They Know*. Harvard Business School Press. 2000.
- [DyMA05] T. Dybå, N. B. Moe and E. Arisholm. Measuring software methodology usage: challenges of conceptualization and operationalization. In *ISESE*. IEEE, 2005, S. 447–457.
- [EHHS04] M. Ehrig, P. Haase, M. Hefke and N. Stojanovic. Similarity for Ontologies - a Comprehensive Framework. In *Workshop Enterprise Modelling and Ontology: Ingredients for Interoperability, at PAKM 2004*, 2004.
- [EiBP91] M. Eisenstadt, M. Brayshaw and J. Paine. *The transparent Prolog machine - visualizing logic programs*. Intellect Books. 1991.
- [Elli04] G. Elliott. *Global Business Information Technology: An Integrated Systems Approach*. Pearson Education. 2004.
- [EmSh03] S. Embury and J. Shao. Analysing the Impact of Adding Integrity Constraints to Information Systems. In J. Eder and M. Missikoff (Hrsg.), *Advanced Information Systems Engineering*, Band 2681 der *Lecture Notes in Computer Science*, S. 175–192. Springer Berlin Heidelberg, 2003.
- [EnEt11] Y. Engel and O. Etzion. Towards proactive event-driven computing. In *Proceedings of the 5th ACM international conference on Distributed event-based system*, DEBS '11, New York, NY, USA, 2011. ACM, S. 125–136.

- [EtNi10] O. Etzion and P. Niblett. *Event Processing in Action*. Manning Publications Company, 2010.
- [FaOP92] E. Falkenberg, J. Oei and H. Proper. Evolving Information Systems: Beyond Temporal Information Systems. In A. M. Tjoa and I. Ramos (Hrsg.), *Database and Expert Systems Applications*, S. 282–287. Springer Vienna, 1992.
- [GBNC07] D. Georgakopoulos, D. Baker, M. Nodine and A. Cichoki. Event-driven Video Awareness Providing Physical Security. *World Wide Web* 10(1), März 2007, S. 85–109.
- [Gera06] D. L. A. Gerald V. Post. *Management Information Systems: Solving Business Problems With Information Technology*. Irwin/McGraw-Hill. 4. Auflage, 2006.
- [GGMO<sup>+</sup>02] A. Gangemi, N. Guarino, C. Masolo, A. Oltramari and L. Schneider. Sweetening Ontologies with DOLCE. In *Proceedings of the 13th International Conference on Knowledge Engineering and Knowledge Management. Ontologies and the Semantic Web, EKAW '02*, London, UK, UK, 2002. Springer-Verlag, S. 166–181.
- [HaKS00] D. E. Harter, M. S. Krishnan and S. A. Slaughter. Effects of Process Maturity on Quality, Cycle Time, and Effort in Software Product Development. *Manage. Sci.* 46(4), April 2000, S. 451–466.
- [HCRS<sup>+</sup>94] J. Herbsleb, A. Carleton, J. Rozum, J. Siegel and D. Zubrow. Benefits of CMM-Based Software Process Improvement: Initial Results. Technischer Bericht, Software Engineering Institute, Carnegie Mellon University, 1994.
- [HiXH96] H. Hirakawa, Z. Xu and K. Haase. Inherited Feature-based Similarity Measure based on large semantic hierarchy and large text corpus. In *Proceedings of the 16th conference on Computational linguistics - Volume 1, COLING '96*, Stroudsburg, PA, USA, 1996. Association for Computational Linguistics, S. 508–513.
- [HMSH<sup>+</sup>11] W. R. van Hage, V. Malaisé, R. Segers, L. Hollink and G. Schreiber. Design and use of the Simple Event Model (SEM). *J. Web Sem.* 9(2), 2011, S. 128–136.
- [HZAW<sup>+</sup>06] M. Hefke, V. Zacharias, A. Abecker, Q. Wang, E. Biesalski and M. Breiter. An Extendable Java Framework for Instance Similarities in Ontologies. In Y. Manolopoulos, J. Filipe, P. Constantopoulos and J. Cordeiro (Hrsg.), *Proceedings of the 8th International Conference on Enterprise Information Systems: Databases and Information Systems Integration (ICEIS 2006), May 23-27, 2006, Paphos, Cyprus*. Setúbal, Portugal, 2006, S. 263–269.



- [Kalm60] R. E. Kalman. A New Approach to Linear Filtering and Prediction Problems. *Transactions of the ASME - Journal of Basic Engineering* (82 (Series D)), 1960, S. 35–45.
- [KhBS08] N. Khoussainova, M. Balazinska and D. Suci. Probabilistic Event Extraction from RFID Data. In *IEEE 24th International Conference on Data Engineering, 2008. ICDE 2008.*, april 2008, S. 1480–1482.
- [KhSt09] M. E. Kharbili and N. Stojanovic. Semantic Event-Based Decision Management in Compliance Management for Business Processes. In *AAAI Spring Symposium: Intelligent Event Processing, 2009*, S. 35–40.
- [KlHi01] H. K. Klein and R. Hirschheim. Choosing Between Competing Design Ideals in Information Systems Development. *Information Systems Frontiers* Band 3, March 2001, S. 75–90.
- [Knut97] D. E. Knuth. *The art of computer programming, volume 1 (3rd ed.): fundamental algorithms*. Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA. 1997.
- [Kulk11] A. A. Kulkarni. ARCADE - Abstraction and Realization of Complex Event Scenarios Using Dynamic Rule Creation. In *Proceedings of the 5th ACM International Conference on Distributed Event-Based System, DEBS '11*, New York, NY, USA, 2011. ACM, S. 23–28.
- [Lew95] J. R. Lewis. IBM computer usability satisfaction questionnaires: psychometric evaluation and instructions for use. *Int. J. Hum.-Comput. Interact.* 7(1), Januar 1995, S. 57–78.
- [LiEW03] L. Lin, S. Embury and B. Warboys. Business Rule Evolution and Measures of Business Rule Evolution. In *IWPSE '03: Proceedings of the 6th International Workshop on Principles of Software Evolution*, Washington, DC, USA, 2003. IEEE Computer Society, S. 121.
- [LiEW05] L. Lin, S. M. Embury and B. C. Warboys. Facilitating the Implementation and Evolution of Business Rules. In *ICSM '05: Proceedings of the 21st IEEE International Conference on Software Maintenance*, Washington, DC, USA, 2005. IEEE Computer Society, S. 609–612.
- [Luck01] D. C. Luckham. *The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA. 2001.
- [Luck12] D. Luckham. *Event Processing For Business. Organizing the Real-Time Enterprise*. John Wiley&Sons, Inc., Hoboken, New Jersey. 2012.

- [Lund06] A. Lundberg. Leverage Complex Event Processing to Improve Operational Performance. *Business Intelligence Journal* 11(1), 2006, S. 55–65.
- [MaML89] H. Marathe, T.-K. Ma and C.-C. Liu. An algorithm for identification of relations among rules. In *Tools for Artificial Intelligence, 1989. Architectures, Languages and Algorithms, IEEE International Workshop on*, oct 1989, S. 360–367.
- [Mark01] M. L. Markus. Toward a theory of knowledge reuse: Types of knowledge reuse situations and factors in reuse success. *Journal of Management Information Systems* Band 18, 2001, S. 57–93.
- [MaZa02] A. Maedche and V. Zacharias. Clustering Ontology-Based Metadata in the Semantic Web. In *PKDD '02: Proceedings of the 6th European Conference on Principles of Data Mining and Knowledge Discovery*, London, UK, 2002. Springer-Verlag, S. 348–360.
- [MKFB<sup>+</sup>12] A. Moraru, K. Kenda, B. Fortuna, L. Bradesko and M. Skrjanc. Supporting Rule Generation and Validation on Environmental Data in EnStream. In *9th Extended Semantic Web Conference, ESWC 2012*, 2012.
- [MSSS<sup>+</sup>01] A. Maedche, S. Staab, N. Stojanovic, R. Studer and Y. Sure. SEAL - A Framework for Developing SEMantic Web PortALs. In B. Read (Hrsg.), *Advances in Databases*, Band 2097 der *Lecture Notes in Computer Science*, S. 1–22. Springer Berlin Heidelberg, 2001.
- [NPLP87] T. A. Nguyen, W. Perkin, T. J. Laffey and D. Pecora. Knowledge base verification. *AI Mag.* 8(2), Juni 1987, S. 69–75.
- [OSSK<sup>+</sup>11] H. Obweiger, J. Schiefer, M. Suntinger, P. Kepplinger and S. Rozsnyai. User-oriented rule management for event-based applications. In *Proceedings of the 5th ACM international conference on Distributed event-based system, DEBS '11*, New York, NY, USA, 2011. ACM, S. 39–48.
- [OSSR10] H. Obweiger, M. Suntinger, J. Schiefer and G. Raidl. Similarity searching in sequences of complex events. In *Research Challenges in Information Science (RCIS), 2010 Fourth International Conference on*, May 2010, S. 631–640.
- [PaDi99] N. W. Paton and O. Diaz. Active database systems. *ACM Comput. Surv.* Band 31, March 1999, S. 63–103.
- [PiBa02] P. R. Pietzuch and J. Bacon. Hermes: A Distributed Event-Based Middleware Architecture. In *ICDCSW '02: Proceedings of the 22nd International Conference on Distributed Computing Systems*, Washington, DC, USA, 2002. IEEE Computer Society, S. 611–618.

- [Pidd00] M. Pidd. *Tools for Thinking - Modelling in Management Science*. Wiley, 2000.
- [PPPC07] T. Pedersen, S. V. S. Pakhomov, S. Patwardhan and C. G. Chute. Measures of semantic similarity and relatedness in the biomedical domain. *J. of Biomedical Informatics* 40(3), 2007, S. 288–299.
- [Prob98] G. Probst. *Practical Knowledge Management*. Prism, Arthur D. Little, 1998.
- [RiHD02] C. K. Riemenschneider, B. C. Hardgrave and F. D. Davis. Explaining Software Developer Acceptance of Methodologies: A Comparison of Five Theoretical Models. *IEEE Trans. Softw. Eng.* 28(12), Dezember 2002, S. 1135–1145.
- [RoFW97] D. Rosca, M. Feblowitz and C. Wild. Decision Making Methodology in Support of the Business Rules Lifecycle. In *Proceedings of the 3rd IEEE International Symposium on Requirements Engineering, RE '97*, Washington, DC, USA, 1997. IEEE Computer Society, S. 236–246.
- [RoSS07] S. Rozsnyai, J. Schiefer and A. Schatten. Concepts and models for typing events for event-based systems. In *DEBS '07: Proceedings of the 2007 inaugural international conference on Distributed event-based systems*, New York, NY, USA, 2007. ACM, S. 62–70.
- [Rozs08] S. Rozsnyai. *Managing Event Streams for Querying Complex Events*. Dissertation. Vienna University of Technology, Vienna, Austria, 2008.
- [SAJP<sup>+</sup>02] E. Söderström, B. Andersson, P. Johannesson, E. Perjons and B. Wangler. Towards a Framework For Comparing Process Modelling Languages. In *14th International Conference on Advanced Information Systems Engineering, CAiSE 2002, volume 2348 of LNCS*. Springer, 2002, S. 600–611.
- [ScKR99] J. B. Schafer, J. Konstan and J. Riedi. Recommender systems in e-commerce. In *Proceedings of the 1st ACM Conference on Electronic Commerce, EC '99*, New York, NY, USA, 1999. S. 158–166.
- [SeLS11] S. Sen, R. Lin and B. F. Shemrani. Complex event pattern evolution based on real-time execution statistics. In *Proceedings of the 5th ACM International Conference on Distributed Event-Based System, DEBS '11*, New York, NY, USA, 2011. ACM, S. 375–376.
- [SeMa09] S. Sen and J. Ma. Contextualised Event-driven Prediction with Ontology-based Similarity. In *Intelligent Event Processing, Papers from the 2009 AAAI Spring Symposium*, 2009, S. 73–79.

- [SeSL09] S. Sen, N. Stojanovic and R. Lin. A graphical editor for complex event pattern generation. In *Proceedings of the Third ACM International Conference on Distributed Event-Based Systems, DEBS '09*, New York, NY, USA, 2009. ACM, S. 41:1–41:2.
- [SeSS10a] S. Sen, N. Stojanovic and B. F. Shemrani. EchoPAT: a system for real-time complex event pattern monitoring. In *Proceedings of the Fourth ACM International Conference on Distributed Event-Based Systems, DEBS '10*, New York, NY, USA, 2010. ACM, S. 107–108.
- [SeSS10b] S. Sen, N. Stojanovic and L. Stojanovic. An approach for iterative event pattern recommendation. In *Proceedings of the Fourth ACM International Conference on Distributed Event-Based Systems, DEBS '10*, New York, NY, USA, 2010. ACM, S. 196–205.
- [SeSt10] S. Sen and N. Stojanovic. GRUVE: A Methodology for Complex Event Pattern Life Cycle Management. In B. Pernici (Hrsg.), *Advanced Information Systems Engineering*, Band 6051 der *Lecture Notes in Computer Science*, S. 209–223. Springer Berlin Heidelberg, 2010.
- [SeTo08] S. Sen and S. Tomcic. An Enterprise Knowledge Management Platform On Top Of The Ontology-Based Similarity. In J. Baumeister and M. Atzmüller (Hrsg.), *LWA*, Band 448 der *Technical Report*. Department of Computer Science, University of Würzburg, Germany, 2008, S. 9–14.
- [SFSS09] A. Scherp, T. Franz, C. Saathoff and S. Staab. F—a model of events based on the foundational ontology dolce+DnS ultralight. In *Proceedings of the fifth International Conference on Knowledge Capture, K-CAP '09*, New York, NY, USA, 2009. ACM, S. 137–144.
- [Snow04] D. Snowden. Organic Knowledge Management: Part I The ASHEN Model: an enabler of action. *Knowledge Management* 3(7), 2004.
- [SSAM<sup>+</sup>11] N. Stojanovic, L. Stojanovic, D. Anicic, J. Ma, S. Sen and R. Stühmer. Semantic Complex Event Reasoning - Beyond Complex Event Processing. In D. Fensel (Hrsg.), *Foundations for the Web of Information and Services*, S. 253–279. Springer Berlin Heidelberg, 2011.
- [Svio90] J. J. Sviokla. An examination of the impact of expert systems on the firm: the case of XCON. *MIS Q.* 14(2), Juni 1990, S. 127–140.
- [TePa09] K. Teymourian and A. Paschke. Towards semantic event processing. In *Proceedings of the Third ACM International Conference on Distributed Event-Based Systems, DEBS '09*, New York, NY, USA, 2009. ACM.

- [TSSG12] M. Trampus, S. Sen, N. Stojanovic and M. Grobelnik. Visualisation of Online Discussion Forums. In Y. Charalabidis and S. Koussouris (Hrsg.), *Empowering Open and Collaborative Governance*, S. 157–179. Springer Berlin Heidelberg, 2012.
- [TuGW09] Y. Turchin, A. Gal and S. Wasserkrug. Tuning complex event processing rules using the prediction-correction paradigm. In *Proceedings of the Third ACM International Conference on Distributed Event-Based Systems*, DEBS '09, New York, NY, USA, 2009. S. 10:1–10:12.
- [TuSt04] T. S. Tullis and J. N. Stetson. A comparison of questionnaires for assessing website usability. *Proceedings of the Usability Professionals Association (UPA) 2004 Conference*, June 2004, S. 7–11.
- [ViKD10] K. Vidačković, I. Kellner and J. Donald. Business-oriented development methodology for complex event processing: Demonstration of an integrated approach for process monitoring. In *Proceedings of the Fourth ACM International Conference on Distributed Event-Based Systems*, DEBS '10, New York, NY, USA, 2010. ACM, S. 111–112.
- [ViP107] N. Vijayakumar and B. Plale. Prediction of missing events in sensor data streams using Kalman filters. In *1st Int'l Workshop on Knowledge Discovery from Sensor Data, in conjunction with ACM 13th Int'l Conference on Knowledge Discovery and Data Mining*, 2007.
- [Wagn00] C. Wagner. End users as expert system developers? *J. End User Comput.* 12(3), July 2000, S. 3–13.
- [Wagn07] C. Wagner. Breaking the knowledge acquisition bottleneck through conversational knowledge management. *Innovative Technologies for Information Resources Management*, 2007, S. 200.
- [Wate86] D. Waterman. *A Guide to Expert Systems*. Addison Wesley. 1986.
- [WeJa06] U. Westermann and R. Jain. E - A Generic Event Model for Event-Centric Multimedia Data Management in eChronicle Applications. In *Proceedings of the 22nd International Conference on Data Engineering Workshops*, ICDEW '06, Washington, DC, USA, 2006. IEEE Computer Society, S. 106–116.
- [WGET08] S. Wasserkrug, A. Gal, O. Etzion and Y. Turchin. Complex event processing over uncertain data. In *Proceedings of the second International Conference on Distributed Event-Based Systems*, DEBS '08, New York, NY, USA, 2008. ACM, S. 253–264.
- [Wils96] R. J. Wilson. *Introduction to graph theory*. Addison Wesley Longman, New York, NY, USA. fourth edition. Auflage, 1996.

- [WoHo04] M. F. Worboys and K. Hornsby. From Objects to Events: GEM, the Geospatial Event Model. In *GIScience'04*, 2004, S. 327–344.
- [WuPa94] Z. Wu and M. Palmer. Verb semantics and lexical selection. In *32nd. Annual Meeting of the Association for Computational Linguistics*, New Mexico State University, Las Cruces, New Mexico, 1994. S. 133–138.
- [Zach87] J. A. Zachman. A framework for information systems architecture. *IBM Syst. J.* 26(3), September 1987, S. 276–292.
- [ZhRL97] T. Zhang, R. Ramakrishnan and M. Livny. BIRCH: A New Data Clustering Algorithm and Its Applications. *Data Mining and Knowledge Discovery* 1(2), 1997, S. 141–182.
- [ZJHN<sup>+</sup>07] X. Zhang, L. Jing, X. Hu, M. Ng and X. Zhou. A comparative study of ontology based term similarity measures on PubMed document clustering. In *Proceedings of the 12th International Conference on Database Systems For Advanced Applications, DASFAA'07*, Berlin, Heidelberg, 2007. Springer-Verlag, S. 115–126.







