GRUVe: A Methodology for Complex Event Pattern Life Cycle Management

Sinan Sen and Nenad Stojanovic

FZI Research Center for Information Technology
Haid-und-Neu-Straße 10-14, 76131 Karlsruhe, Germany
{sinan.sen,nstojano}@fzi.de
http://www.fzi.de/ipe

Abstract. With the rising importance of recognizing a complex situation of interest near real-time, many industrial applications have adopted Complex Event Processing as their backbone. In order to remain useful it is important that a Complex Event Processing system evolves according to the changes in its business environment. However, today's management tasks in a Complex Event Processing system related to the management of complex event patterns are performed purely manually without any systematic methodology. This can be time consuming and error-prone.

In this paper we present a methodology and an implementation for the complex event pattern life cycle management. The overall goal is the efficient generation, maintenance and evolution of complex event patterns. Our approach is based on a semantic representation of events and complex event patterns combined with complex event pattern execution statistics. This representation enables an improved definition of relationships between patterns using semantic descriptions of patterns and events.

1 Introduction

Complex Event Processing (CEP) is the analysis of events from different event sources in near real-time in order to generate immediate insight and enable immediate response to changing business conditions [1]. For example *Transportation and Logistics*, *Financial Front Office*, *Telecommunication* or *Customer Risk Management* are successful application areas of CEP. Events are classified into simple (atomic) events and complex events. According to Chandy et. al. [2] an event indicates a significant change in the state of the universe. Sensor data, network signals, credit card transactions or application processing information are examples for simple events. 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. For example, a credit card fraud event is detected from incoherent use of a single credit card in different places by combining these events into a complex event.

To detect a pattern over events properly is the most important capability of CEP systems. This capability enables a just-in-time reaction to occurred situations. In order to recognize these situations so called complex event patterns

B. Pernici (Ed.): CAiSE 2010, LNCS 6051, pp. 209–223, 2010.

(CEPATs) have to be defined. These patterns[1] are used in order to process the events and aggregate them to more high level complex events. A pattern is an expression formed by using a set of events (either simple or complex) and a set of event operators [3]. They resemble knowledge about the reactivity of the system. For example the pattern[2]

(A AND B) happen WITHIN 2 Minutes

contains two events *A* and *B*, a logical Operator *AND* and a window operator *WITHIN*.

In order to cope with the evolving nature of business environments we need effective and efficient support for advanced CEPAT management. However, today's management tasks in a CEP system related to the generation, maintenance and evolution of CEPATs are performed manually without any systematic methodology and tool support. So far there exists no academic approach dealing with the management and evolution of CEPATs. Also the vendors providing CEP systems neglect the issue of management and evolution. They are focused rather on runtime than on design time issues. CEPATs can be compared to business rules in the sense that they both are indispensable for today's business. In the business rules domain the management and the evolution of rules have been proven to be crucial in order to optimize the process of rule generation, modification and evolution (see [4],[5],[6],[7]). However, a straightforward application of existing business rule management systems to CEPAT management is not a viable solution. This is due to the nature of CEPATs. Moreover, the existing business rule evolution approaches do not tackle capabilities such as recommendation of new rules, the reusability of existing rules, especially for large rule sets, and finally the usage-driven evolution of rules which is part of our methodology.

In this paper we present a methodology for the management of CEPATs which supports the whole life cycle of a CEPAT: starting from its generation, throughout its usage including its evolution. The methodology treats CEPATs as knowledge artefacts that must be acquired from various sources, represented, shared and evaluated properly in order to enable their evolution. We argue that such a methodology is a necessary condition for making a CEP system efficient and keeping it alive. In a nutshell our approach is a semantic-based representation which enables us to a) make the relationships between CEPATs explicit in order to reuse existing pattern artefacts b) automatically suggest improvements in a CEPAT, based on necessity derived from usage statistics.

The paper is organized as follows: In section 2 we present related work. In section 3 we define requirements for a CEPAT management methodology followed by section 4 where we describe our methodology consisting of different phases including the semantic CEPAT representation and where we outline its importance. In section 5 we present the implementation state of our work and in section 6 we conclude the paper and give an outlook for the continuation of this line of research.

[1] Pattern and complex event pattern are used synonymously in this article.

[2] For the sake of convenience the pattern is represented in a pseudocode form.

2 Related Work

In the followed section we discuss related work from different fields of research relevant for this paper, namely current approaches in event representation, business rule evolution and rule management. We also take a look at the existing CEP systems.

Event representation: There exist different approaches for event representation. Distributed publish/subscribe architectures like presented in [8], [9] and [10] represent an event as a set of typed attributes. Each individual attribute has a type, name and value. The event as a whole has purely syntactical and structural value derived from its attributes. Attribute names and values are simple character strings. The attribute types belong to a predefined set of primitive types commonly found in programming languages. In our opinion this kind of event model is not sufficient for CEPAT management since it does not provide any explicit semantics. The only justification for choosing this typing scheme is the scalability aspect. The event stream engine AMIT (see [11] and [12]) is targeting the high-performance situation detection mechanism. Via the offered user interface one can model business situations based on event, situations, lifespans and keys. Within AMIT an event is specified with a set of attributes and it is the base entity. All events belong to a certain class (a class contains events with similar characteristics) and have relationships to other events. Attributes can be references to other objects as well. This approach is close to ours in the sense that it describes the importance of generalization, specialization and relationships of events.

Event representation in existing CEP systems: There is a large number of CEP systems, both academic and commercial. The Aurora/Borealis [13] system has the goal to support various kinds of real-time monitoring applications. Data items are composed of attribute values and are called tuples. All tuples on a stream have the same set of attributes. The system provides a tool set including a graphical query editor that simplifies the composition of streaming queries using graphical boxes and arrows. The Cayuga [14] system can be used to detect event patterns in the event stream as well. An event here is a tuple of attribute value pairs according to a schema. The goal of the Stream [15] project is to be able to consider both structured data streams and stored data together. The Esper [16] CEP engine supports events in XML, Java-objects and simple attribute value pairs. There are also many commercial CEP vendors like Tibco, Coral8/Aleri, StreamBase or Progress Apama providing tool suites including development environments and a CEP engine. Most of the described systems use different SQL-like or XML based complex event pattern definitions, which makes it difficult to understand and read them. They do not provide a semantic model for events and event patterns either. The management of event patterns is restricted to create, modify and delete. Furthermore they do not use any underlying management methodology and do not tackle the issues of reusability and evolution of patterns.

(Business) Rule evolution and its importance: The Agile Business Rule Development methodology [17] is aimed at harvesting rules and producing an executable (albeit incomplete) rule set as early as possible. It is based on five cycles: Harvesting, Prototyping, Building, Integration and Enhancing. Within the cycles, the activities are Discovery, Analysis, Authoring, Validation and Deployment. The methodology considers mostly the harvesting of rules and does not consider their evolution. Also the refinement and the reusability of rules are not part of the methodology. There are also many other approaches in the area of (business) rules evolution or management like [4], [5], [6], [7], [15], [17]. They consider either some aspects of (business) rules or describe the necessity of evolution and maintenance. We of course incorporate relevant aspects from these works in our methodology where applicable.

3 Requirements: What Do We Need and Why Is It Important?

Compared to other IT-systems, CEP systems still lack in support of tools allowing users to reconfigure a system easily or to refactor services and components [18]. This is also valid for CEPAT generation, maintenance and management. David Luckham[3] considers the validation and the management of event processing rules especially when dealing with large sets of event processing rules as one of the challenges in the area of CEP [19]. Luckham defines CEPAT management as writing correct patterns, their efficient execution, correct changes, ensuring logical consistency and absence of redundancies.

From our point of view the management of CEPATs requires two basic functionalities: efficient generation of patterns, including the incremental development of new patterns and the continual improvement of existing patterns. These functionalities in turn require methods and tools for: checking the validity of new patterns, including the entire repository, reuse of existing patterns and discovery of the need for changes in patterns and realizing them. In the following we briefly describe each of these requirements:

- **Ensure an anomaly-free repository:** Anomalies in general are symptoms of probable errors. Efficient generation of patterns means that the newly created patterns semantically suit to the existing patterns. This can be ensured through resolving two types of anomalies: contradiction and redundancy. In order to ensure an anomaly-free repository a formal model is needed that is expressive enough to detect anomalies.
- **Deal with weak information needs and reuse of existing knowledge:** The creation of new patterns can be augmented by an incremental approach which will enable smooth refinements of existing patterns in order to sustain patterns being as relevant as possible for the problem at hand.

[3] David Luckham is one of the most prominent experts in the area of CEP - http://www-ee.stanford.edu/∼luckham/

– **Enable continual improvement:** As already explained patterns will change in time and there is a need for methods of continually updating patterns in order to ensure their relevance for new situations. CEP systems are designed for near real-time environments. Iterative refinement of the generated CEPATs enables a smooth modification of an initial CEPAT in order to make it as relevant as possible to the given task of the CEPAT. A continual improvement of the CEPATs enables an adaptation of the patterns to the internal or external changes in order to keep the CEP system useful. The methodology should also consider the effective detection of problems in the usage of patterns, to enable suggesting how to evolve them.

These requirements cause some additional requirements for the implementation of the methodology that are described below:

– **High level graphical representation of events and patterns:** In order to express rules succinctly in a way that makes them understandable, we need proper graphical development environments. A business expert (a non-technician) should be able to express her/his need for a complex event pattern by using this environment. The complexity of underlying CEP engines needs to be hidden from the user in order to increase the usability of the tool. Furthermore the tool must support the refinement and the reusability of patterns in a graphical way as well.
– **Independent modeling of events and patterns:** The event and pattern model has a major impact on the flexibility and usability of a CEP system. Nowadays event patterns are mostly realized in XML, Java-Objects or SQL-like CEP languages. However, these representations do not support the management of event patterns. In order to assure that, a model is needed which allows the explicit definition of event and event pattern semantics and which supports reasoning about events and event patterns. This model is used at design time for event and event pattern representation and is transformed into the CEP engine specific languages during the pattern deployment phase.

We believe a methodology and a tool-set based on these requirements will shorten the time of the CEPAT development, improve the quality of CEPATs and enable also non-domain experts to develop, maintain and evolve CEPATs. In the next section we present a methodology implementing these requirements.

4 GRUVe: The Methodology

We define the CEPAT life cycle management as a process covering the generation, the refinement, the execution and the evolution of CEPATs. The focus is on supporting the user by creating new patterns, increasing the reusability of existing pattern artefacts and to give hints for pattern evolution. As illustrated in figure 1, the GRUVe[4] methodology consists of four main phases. These phases

[4] Acronym for Generation Refinement Usage eVolution of complex (e)vent patterns.

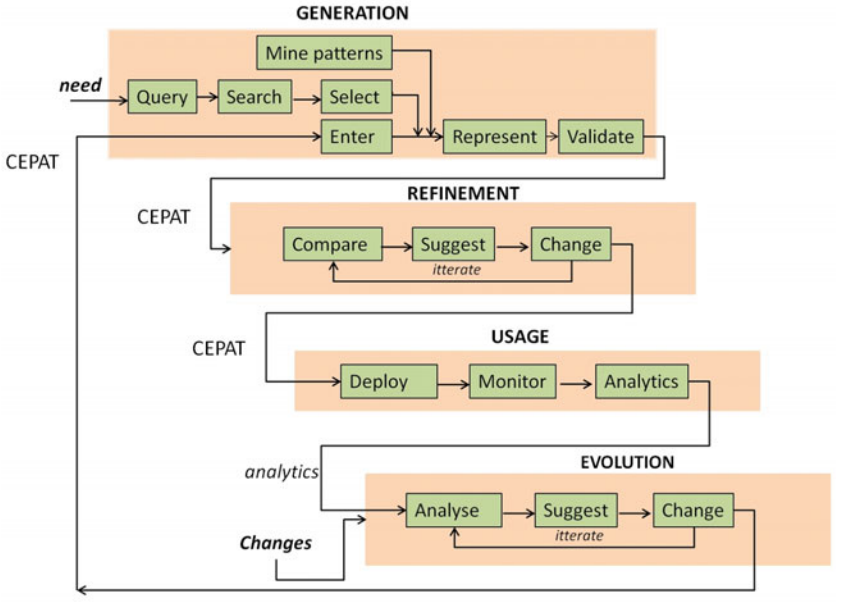

Fig. 1. Phases of the CEPAT life cycle

form a feedback loop enabling continual flow of information collected in the entire life cycle of a CEPAT. The main idea is to enable non-technicians to *search* incrementally for the most suitable form of requested CEPATs and to continually improve their quality taking into account changes that might happen in the internal or external world. Obviously the basis for the development of the methodology were the requirements given in the previous section. The methodology has been influenced by our past work in the area of knowledge management and ontology evolution [20], [21]. Below we describe these phases in more detail.

4.1 Generation Phase

The life cycle of CEPATs starts with their development which is classified into three categories (see figure 1 block *Generation*). The simplest way to develop a CEPAT is a manual development which can be done only by business experts. However, we cannot expect that an arbitrary user spends time finding, grouping and ordering the events in order to create their desired CEPAT. In order to do that the user must be aware of the way of combining events, he/she must find the right type of events, foresee and solve the intermediate conflicts that might appear and order events in a right way. A more user-oriented approach can be obtained by reusing existing CEPATs, the so-called search-based approach. Here, the users should be able to specify their complex goals without considering how they can be realized. By selecting *similar* existing CEPATs at the start of the CEPAT development process, the users could realize their request much faster. For instances if the user intends to generate a new pattern for a fraud situation

he/she can search for events which belong to the type *fraud* from a certain event source. In that way the generation of a CEPAT could be realized as a straightforward search method, where the specificities of the CEPAT domain are used to define a search space and to restrict the search in this space. Besides CEPATs developed by experts or users explicitly, there are also implicit CEPATs in the domain, reflected in the behavior of the system. These can be discovered through the analysis of log files, etc. Various data mining methods can be used for the extraction of such knowledge. Independent of how CEPATs are identified, in order to deal with them they have to be represented in a suitable format which supports their management. Therefore, the generated complex event patterns must be represented in an internal, CEP platform independent way.

RDFS-based complex event pattern representation. A well defined event pattern model has a major impact on the flexibility and usability of CEP tools [18]. Contemporary event and pattern models lack the capability to express the event semantics and relationships to other entities. Although the importance of such a semantic event representation is demonstrated in practice [12], there does not exist any systematic work on this topic so far. Still most of the existing event models consider an event as an atomic unstructured or semistructured entity without explicitly defined semantics. Also the existing CEP languages are product specific languages designed solely for the matching process rather than for their own management and evolution. Our semantic model for event and pattern representation is based on RDFS[5]. RDFS in general has the advantage that it offers formal and explicit definitions of relations between concepts. We use our RDFS based pattern representation at different stages within the methodology: increasing the reusability of existing pattern artefacts, validation of defined patterns and identification of relations between two patterns in order to evolve them. We believe the explicit representation of events through RDFS will enable CEP tools to provide a qualitatively new set of services such as verification, justification and gap analysis. These systems will be able to weave together a large network of human knowledge and will complement this capability with machine processability. Various automated services will then aid users in achieving their goals by accessing and providing information in a machine understandable form. It is important to note that semantic-based representations not only define information but also add expressiveness and reasoning capabilities. In general, reasoning tasks can be used to verify a model (i.e. a set of CEPATs). Consistency checking, detection of redundancies and refinement of properties are just some of these reasoning capabilities.

The upper-level ontology contains a set of concepts *Event*, *EventOperator*, *EventSource*, *EventType* and a set of event properties. Each property may have a domain (denoted by $\sqsubseteq \exists$) concept as well as a range (denoted by $^{-} \sqsubseteq$) concept (see figure 2). These concepts can be specialized in order to define new types, sources and operators.

[5] RDF Schema (RDFS) is an extensible knowledge representation language providing basic elements for the description of ontologies.

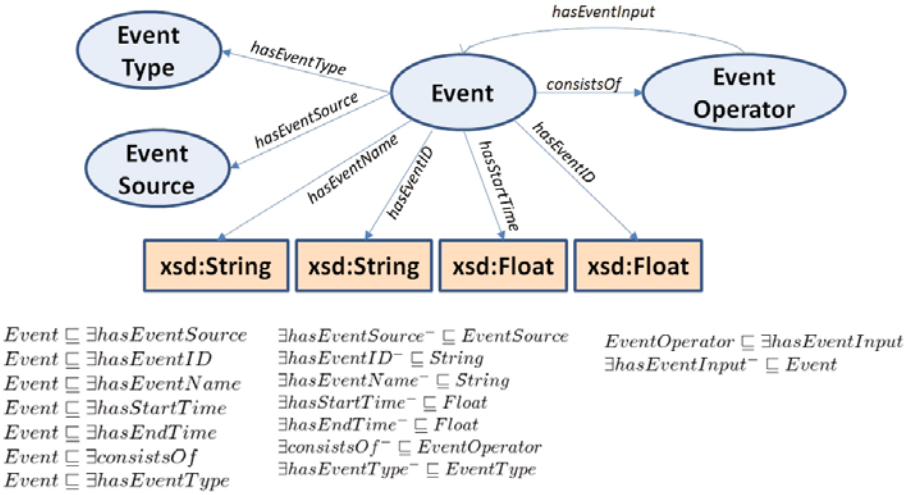

Fig. 2. Upper-level event and CEPAT ontology

Definition 1: An event is defined as a 7-tuple $E := (I, N, ST, ET, ES, EO, T)$ where:

- I is a numeric event id
- N is a alphanumeric event name
- ST is numeric event start time of an event
- ET is numeric event end time of an event
- EO is a reference to the event operator concept
- ES is a reference to the event source concept
- T is a reference to the the event type concept

A concrete event occurrence, simple or complex, is an instantiation of E. It has an unique id and an event name. The start time indicates when the event occurred. The end time of an event is relevant only for complex events and indicates the triggering of a complex event which might be different from the start time for an event with a duration. In comparison to a complex event that contains a complex event pattern via a reference to the concept EO a simple event does not contain any reference to the concept EO. In this sense the concept EO indicates whether an event is simple or complex with a proper complex event pattern.

In order to classify events of similar characteristics we use event types and event sources. The event type characterizes a class of event objects.

Definition 2: The event type is defined as a tuple $T := (H_t, A_t)$ where:

- H_t is an acyclic event type hierarchy
- A_t is a set of event type specific attribute $< at_1, ..., at_n >$

An event source is an entity that sends events e.g., a software module, a sensor, a web service etc. The definition of the event source is similar to the definition of the event type.

Definition 3: 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 type specific attribute $< as_1, ...as_n >$

A complex event is composed of an event operator via the *consistsOf* property. An event operator is further classified into operator classes according to the nature of the operator. Currently the model contains operators defined in [3]. An event operator again has events as input. This allows us to build more complex event patterns recursively.

Definition 4: The event operator is defined as a tuple $EO := (H_{eo}, A_{eo})$ where:

- H_{eo} is an acyclic event operator hierarchy
- A_{eo} is a set of event operator specific attribute $< ao_1, ...ao_n >$

One of the key idea of the approach is having a well defined representation of the complex event patterns in order to recognize different relations between them. The information about the relation can be used for getting more precise information on top of the pattern statistics knowledge.

A pattern is formed as a tuple (E, EO) where E is a set of events and EO is a set of event operator.

Let:

- Γ be the set of all CEPAT triples within the Knowledge Base (KB) representing all defined CEPATs.
- ϵ_i represent all event triples and σ_i all operator triples created for a CEPAT instance.
- τ_i be the set triples of a CEPAT instance containing all event and operator triples, i.e. $\tau_i = \epsilon_i \cup \sigma_i$.

Then a CEPAT instance can be seen as a finite subset of Γ, i.e $\tau_i \subseteq \Gamma$. On top of that definition we define the relations between two CEPATs:

- subsumption: A complex event pattern $CEPAT_1$ subsumes another complex event pattern $CEPAT_2$, i.e. $\tau_1 \subset \tau_2$
- overlapping: A complex event pattern $CEPAT_1$ have a common overlap with another complex event pattern $CEPAT_1$, i.e $\exists x, y \in \epsilon : (x \subset \tau_1 \wedge x \subset \tau_2) \wedge (y \subset \tau_1 \wedge y \not\subset \tau_2)$
- disjointness: Two complex event patterns do not have an overlap, i.e $\tau_1 \cap \tau_2 = \emptyset$
- equalness: Two complex event patterns represent the same situation, i.e. $\tau_1 \equiv \tau_2$

Regarding the *Generation* phase, the main strength of our approach lies in the presented semantic-based nature of our CEPAT representation which enables formal representation of the relations between existing CEPATs and consequently better understanding of that information space. It increases the reusability of existing patterns enabling a faster development of new patterns.

4.2 Refinement Phase

As we already mentioned the process of creating CEPATs is a kind of weakly-defined search operation in an information space rather than a search with very precisely defined queries. Lessons learned from the Information retrieval community show that the right mechanism for the former case is the incremental refinement of queries [20], based on various feedbacks that can be obtained in that process. It would mean that the main task in the CEPAT creation process is not the very precise formulation of a pattern in the first place but an iterative improvement of the CEPAT. This corresponds to the second requirement from section 3. As presented in figure 1 this phase has the CEPAT as input created in the previous phase. It allows the user to fine tune the CEPAT if he/she is not sure about the quality of the created CEPAT. Let us assume our goal is to build new patterns N1 and N2. We further assume that there exist two CEPATs, P1 and P2, in the repository[6].

```
P1- Stock.VW.value=117 AND Stock.Daimler.value=34.78
    AND Stock.Porsche.value=47.90
P2- Blog.Twitter.Tweet.Person="Barack Obama" AND
    Blog.Twitter.Tweet.Person="Nicolas Sarkozy"
```

We can use the previously defined relations in order to demonstrate the refinement for the following new patterns and show how relevant hints can be given about related patterns.

```
N1- Stock.VW.value=117 AND Stock.Porsche.value=47.90
N2- Blog.Twitter.Person="Angela Merkel"
    AND Blog.Twitter.Person="Barack Obama"
```

The following relations exist between the four patterns:

- *subsumption(P1, N2)=true*
- *disjoint(P1,N2)=true*
- *disjoint(P2,N1)=true*
- *overlapping (P2, N2)=true*

The result of this relation detection is that in the first case P1 is presented to the user in order to be reused. In the second case P2 can be shown in order to inform the user that there exists another pattern that can be relevant for him as well. By using the RDF relation *RDF:instanceOf* we also can identify patterns that are different at the instance level but equivalent at the class level. For instance if we

[6] The pattens in this example are presented in a pseudocode form.

change the pattern artefact *Stock.VW.value=117* into *Stock.VW.value=118* we are still able to detect the subsumption relation based on the class information although they differ at the instance level.

4.3 Usage Phase

Once created, CEPATs must be deployed in a CEP engine, after being transformed into the corresponding syntax. However in order to ensure the quality of created patterns, it is necessary to monitor what happens to a CEPAT or a set of CEPATs in a CEP engine at runtime. Nowadays there is no monitoring of the defined CEPATs e.g., if they had been executed in a CEP engine. It is not obvious to see how often certain patterns have been triggered and which parts of the pattern have been executed how often. It is not available either which patterns are going to be executed next. However we believe information of how often a pattern was triggered or how high the current degree of fulfillment is might be essential for the pattern evolution. The goal of this phase is to track as much as possible of this information and process it in the context of the defined CEPATs. For this reason we think a component may be developed that might capture all the pattern related information. These statistics can be used within the *Evolution* phase in order to evolve and optimize a pattern.

4.4 Evolution Phase

A pattern that has not become rapidly obsolete must change and adapt to the changes in its environments, user's needs, etc. Therefore, if a pattern aims at remaining useful it is essential that it is able to accommodate the changes that will inevitably occur. Developing patterns and their applications is expensive but evolving them is even more expensive. The experiences show that the traditional software system maintenance costs exceed the development costs by a factor of between two and four. There is no reason to assume this should be any different for CEP systems when they are used during a longer period of time. Facilitating those changes is complicated if large quantities of events and CEPATs are involved. The Evolution phase is responsible for coping with changes that may impact the quality of a CEPAT. The causes of the changes usually lay in the highly changeable internal and external factors. In a more open and dynamic business environment the domain knowledge evolves continually. The basic sources that can cause changes in a business system are:

- **The environment:** The environment in which systems operate can change.
- **The user:** Users' requirements often change after the system has been built, warranting system adaption. For example hiring new employees might lead to new competencies and greater diversity in the enterprise which the system must reflect.
- **The process:** The business applications are coupled around the business processes that should be reengineered continually in order to achieve better performances.

The goal of this phase is to use the analytics provided by the Usage phase and suggest to the user some improvements to evaluated CEPATs. While a good design may prevent many CEPAT errors, some problems will not pop out before CEPATs are in use. Therefore, the relationship with the usage of a CEP-based system is paramount when trying to develop useful CEPATs and cannot be neglected. One of the key ideas of the approach is having a well defined representation of the complex event patterns in order to recognize different relations between them. The information about the relations can be used for getting more precise information on top of the pattern statistics knowledge.

The most important analysis is the comparison with the usage data of related patterns. We give here an example: Let us assume that P1 and N1 (from the previously section) are in the event repository. Let us further assume that P1 has been triggered 10 times in a period and N1 1000 times. Obviously there might be a problem in the usage of the pattern P1. What can be concluded by comparing usage data from these two patterns is that part *Stock.VW.value=117 AND Stock.Porsche.value=47.90* in the pattern N1 had been fulfilled many times but the combination with the simple subpattern *Stock.Daimler.value=34.78* is rather critical. The system can discover such a discrepancy and suggest the user to replace this event. Moreover, the system can suggest which similar event is the most appropriate one. Obviously, judging whether a pattern is still correct based on its usage requires a kind of metrics. Our next step will be to develop a comprehensive notion of the quality of a CEPAT, which will alleviate the Evolution phase.

5 Implementation of the Methodology

The current version of our CEPAT life cycle management environment supports the Generation and Refinement phase implemented as a web application. The implementation supports the transformation of patterns and events into our RDFS representation, as described in section 4.1.

The CEPAT management environment consists of the CEPAT Input Section, the CEPAT Design Section, the CEPAT Refinement Section and the CEPAT Statistics Section (see figure 3). The pattern input section provides the event nodes and the operator nodes in order to build a new CEPAT.

Within the pattern design section new patterns can be created. Patterns can be generated by selecting the required event or operator nodes from the pattern Input Section and connect them. Event nodes should be connected to an operator node where each event node is connected to at most one operator node. An operator node can be connected to several event nodes but can only be connected to either an action node or another operator node. The pattern generated in fig. 3 shows the pattern N1 from section 4.2. The results of the refinement are shown in the CEPAT Refinement Section. It shows the pattern P1 from section 4.2. As soon as the user starts building a new pattern our system calculates the next similar patterns related to his current situation based on the relations. The results are presented in a graphical way. The user can see the details of the presented patterns by klicking on them and reuse parts or all of the selected pattern.

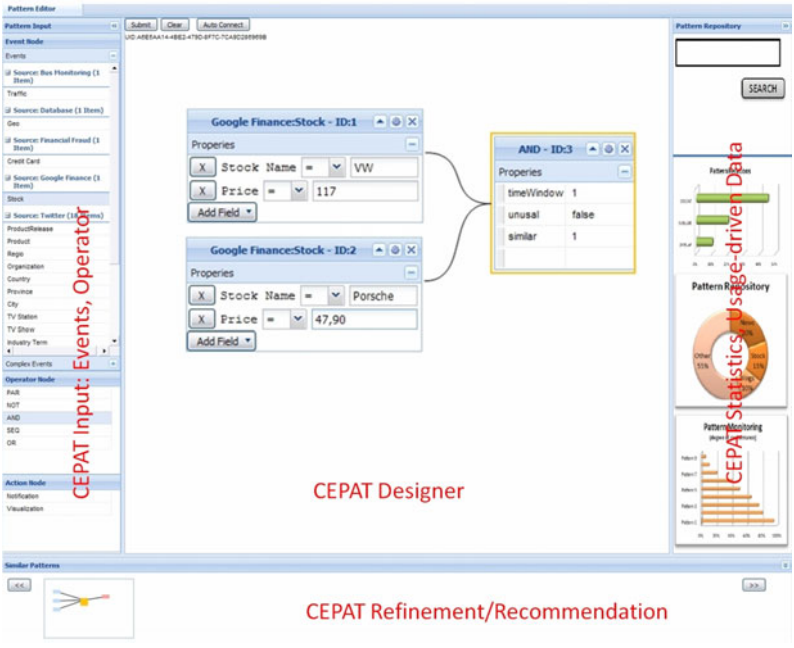

Fig. 3. Complex event pattern management environment including the pattern N1 from section 4.2

6 Conclusion and Future Work

The most important capability of CEP (Complex Even Processing) systems is to detect an occurred situation properly. Today's management tasks in CEP systems related to the evolution of complex event patterns (CEPATs) are performed manually without any systematic methodology and tool support. So far there is not any academic approach dealing with the management and evolution of CEPATs. In this paper we presented a methodology for CEPAT life cycle management supporting the generation, maintenance and evolution of CEPATs. In a nutshell the approach is a semantic-based representation of CEPATs which enables us to make the relationships between CEPATs explicit in order to reuse existing pattern artefacts and automatically suggest improvements for a CEPAT, in the case that it is necessary, based on usage statistics. We present the current implementation status of our methodology that supports the generation and the refinement phase.

Our long-term goal is to underline the importance of a methodology for CEPAT management through a proper reference implementation. Therefore we will extend the phases of our methodology presented in this paper in the following way:

- **Generation phase:** To cover also the implicit knowledge within the domain we will use existing data mining technologies for generating CEPATs automatically. They can be adapted by the expert later on. Furthermore we will develop advanced CEPAT search strategies in order to find existing pattern knowledge in the pattern repository.
- **Refinement phase:** Beside the reusability of patterns it is also useful to give experts hints about missing patterns for closing the pattern knowledge gap and to be up-to-date.
- **Usage phase:** In order to support the evolution we will develop a statistical manager component that monitors the execution of patterns and build some proper analytics for the evolution phase.
- **Evolution phase:** For the evolution of patterns we will introduce quality metrics for patterns in order to identify CEPAT candidates for update.

Acknowledgments. The research presented in this paper was partially funded by the European Commission in the project VIDI (http://www.vidi-project.eu/), EP-08-01-014.We would like to thank to Ljiljana Stojanovic for her comments and contribution to this work. We also would like to thank Dominik Riemer, Ruofeng Lin and Weiping Qu for their contribution to the implementation.

References

1. Luckham, D.C.: The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems. Addison-Wesley Longman Publishing Co., Inc., Boston (2001)
2. Chandy, K.M., Charpentier, M., Capponi, A.: Towards a theory of events. In: DEBS 2007: Proceedings of the 2007 inaugural international conference on Distributed event-based systems, pp. 180–187. ACM, New York (2007)
3. Chakravarthy, S., Mishra, D.: Snoop: An expressive event specification language for active databases. Data Knowl. Eng. 14(1), 1–26 (1994)
4. Sobieski, J., Krovvidy, S., McClintock, C., Thorpe, M.: Karma: Managing business rules from specification to implementation. AAAI/IAAI 2, 1536–1547 (1996)
5. Wan-Kadir, W.M.N., Loucopoulos, P.: Relating evolving business rules to software design. J. Syst. Archit. 50(7), 367–382 (2004)
6. Lin, L., Embury, S., Warboys, B.: Business rule evolution and measures of business rule evolution. In: IWPSE 2003: Proceedings of the 6th International Workshop on Principles of Software Evolution, Washington, DC, USA, p. 121. IEEE Computer Society, Los Alamitos (2003)
7. Lin, L., Embury, S.M., Warboys, B.C.: Facilitating the implementation and evolution of business rules. In: ICSM 2005: Proceedings of the 21st IEEE International Conference on Software Maintenance, Washington, DC, USA, pp. 609–612. IEEE Computer Society, Los Alamitos (2005)
8. Carzaniga, A., Rosenblum, D.S., Wolf, A.L.: Design and evaluation of a wide-area event notification service. ACM Trans. Comput. Syst. 19(3), 332–383 (2001)
9. Pietzuch, P.R., Bacon, J.: Hermes: A distributed event-based middleware architecture. In: ICDCSW 2002: Proceedings of the 22nd International Conference on Distributed Computing Systems, Washington, DC, USA, pp. 611–618. IEEE Computer Society, Los Alamitos (2002)

10. Aguilera, M.K., Strom, R.E., Sturman, D.C., Astley, M., Chandra, T.D.: Matching events in a content-based subscription system. In: PODC 1999: Proceedings of the Eighteenth Annual ACM Symposium on Principles of Distributed Computing, pp. 53–61. ACM, New York (1999)
11. Adi, A., Etzion, O.: Amit - the situation manager. The VLDB Journal 13(2), 177–203 (2004)
12. Adi, A., Botzer, D., Etzion, O.: Semantic event model and its implication on situation detection. In: ECIS (2000)
13. Abadi, D., Carney, D., Çetintemel, U., Cherniack, M., Convey, C., Erwin, C., Galvez, E., Hatoun, M., Maskey, A., Rasin, A., Singer, A., Stonebraker, M., Tatbul, N., Xing, Y., Yan, R., Zdonik, S.: Aurora: a data stream management system. In: SIGMOD 2003: Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data, p. 666. ACM, New York (2003)
14. Brenna, L., Demers, A., Gehrke, J., Hong, M., Ossher, J., Panda, B., Riedewald, M., Thatte, M., White, W.: Cayuga: a high-performance event processing engine. In: SIGMOD 2007: Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data, pp. 1100–1102. ACM, New York (2007)
15. Arasu, A., Babcock, B., Babu, S., Cieslewicz, J., Datar, M., Ito, K., Motwani, R., Srivastava, U., Widom, J.: Stream: The stanford data stream management system. Technical Report 2004-20, Stanford InfoLab (2004)
16. Esper: Esper version 3.2.0, espertech inc. (2009), `http://esper.codehaus.org/` (last visited: January 2010)
17. IBM-ILOG: Agile business rule development methodology (2010), `https://www.ibm.com/developerworks/mydeveloperworks/blogs/isis/?lang=en_us` (last visited: January 2010)
18. Rozsnyai, S., Schiefer, J., Schatten, A.: Concepts and models for typing events for event-based systems. In: DEBS 2007: Proceedings of the 2007 Inaugural International Conference on Distributed Event-Based Systems, pp. 62–70. ACM, New York (2007)
19. Luckham, D.C.: What's the difference between esp and cep? (August 2006), `http://complexevents.com/?p=103` (last visited: January 2010)
20. Stojanovic, N.: Ontology-based information retrieval. Ph.D. Thesis, University of Karlsruhe, Germany (2005)
21. Stojanovic, L.: Methods and tools for ontology evolution. Ph.D. Thesis, University of Karlsruhe, Germany (2004)