Design Patterns for Event-driven Enterprise Architectures

Jürgen Dunkel and Ralf Bruns
Department of Computer Science, Hannover University of Applied Sciences and Arts, Ricklinger Stadtweg 120, 30459 Hannover, Germany

Keywords: Event-driven Architecture, Complex Event Processing, Design Pattern, Enterprise Architecture.

Abstract: In recent years, event processing has gained considerable attention as an individual discipline in computer science. For event processing, generally accepted software architectures based on proven design patterns and principles are missing. In this paper, we introduce a catalogue of design patterns that supports the development of event-driven enterprise architectures and complex event processing systems. The design principles originate from experiences reported in publications as well as from our own experiences in building event processing systems with industrial and academic partners. We present several patterns on different layers of abstractions that define the overall structure as well as the building blocks for event processing systems. Finally, we discuss a general reference architecture for event-driven enterprise systems.

1 INTRODUCTION

Recently, *event processing* (EP) has established itself as an individual discipline in computer science. *Event-driven architecture* (EDA) introduces event processing as the central architectural concept (Chandy and Schulte, 2010). One key concept of EDA is the employment of *complex event processing* (CEP) as the general processing model (Luckham, 2002). CEP analyses *continuous streams* of incoming events in order to identify the presence of complex sequences of events, so called event patterns. Fundamental concepts of CEP are an *event processing language* (EPL), to express *event processing rules* consisting of *event patterns* and *actions*, as well as an *event processing engine* that continuously analyzes the event stream and executes the matching rules. If a pattern matches either a new complex event could be created or an appropriate action could be triggered.

But event-driven applications have not yet reached the maturity of well-established software architectures (Paschke, 2009), (Dunkel et al., 2009). Although several concrete software architectures have emerged there is a lack of established design patterns, best practice examples, reusable components, process models, and reference architectures offering guidelines on how to build enterprise event processing systems. In particular, design patterns and reference architectures exist only in early draft versions (Schulte and Bradely, 2009), (Paschke and Vincent, 2009), or are related to certain technologies (Coral8, 2006) and application domains (Dunkel et al., 2010).

In this paper, we are considering event processing from a software engineering perspective. We present a catalogue of patterns providing the basic structure and the building blocks of a software architecture for event processing systems. The proposed patterns can serve as general guidelines for the development of event-driven enterprise architectures. Using the pattern catalogue we derive a general reference software architecture for event processing enterprise systems.

Patterns encapsulate knowledge: they are based on the repertoire of general principles and idiomatic solutions of experienced developers. The design patterns presented here originate from experiences reported in published work as well as from our own experiences in building event processing systems with industrial and academic partners. This paper presents a pattern catalogue that provides a way of organizing solutions for general problems in EP and makes it easier to reuse them. In other words, our approach introduces a vocabulary for the architecture of EP as well as a conceptual framework that leads to a more structured approach for the design of EP systems.

After discussing related work in the next section we present in section 3 the catalogue of patterns. Finally, section 4 introduces a general reference architecture based on the proposed patterns.

Dunkel J. and Bruns R..
Design Patterns for Event-driven Enterprise Architectures.
DOI: 10.5220/0003971301870192
In *Proceedings of the 14th International Conference on Enterprise Information Systems* (ICEIS-2012), pages 187-192
ISBN: 978-989-8565-12-9

2 RELATED WORK

General Patterns: Most of the pattern literature is focused on conventional process-oriented software architectures. But some of the general design patterns like *layers* (Gamma et al., 1998) can be applied on EDA as well. Other patterns that can be directly transferred to event processing are discussed in enterprise application integration (EAI). (Hohpe and Woolf, 2003) presents numerous patterns describing the message-based interaction between applications. Obviously, the general ideas of Message Routing and Message Transformation can be reused for event processing as well. EAI patterns, as Pipes and Filters, Message Filter, Aggregator, and Content Enricher are applied on event processing in (Coral8, 2006).

Event Processing Patterns: In recent years, a lot of work has been published reporting on manifold experiences with event-driven systems. Luckham introduced the concept of *event processing networks* (EPN) in order to cope with the complexity of real-world event-driven applications (Luckham, 2002).

A collection of CEP patterns is presented in (Coral8, 2006) for illustrating the capabilities of CCL, their proprietary event processing language. Because the patterns are that closely tied to a certain technology, they are much more recipes than patterns.

Paschke (Paschke, 2009) does not describe a catalogue of patterns but presents a multidimensional scheme for the *categorization of CEP patterns*.

Reference Architectures: A *reference architecture* provides a generic and abstract software architecture that provides a template for the design of an EP-based system. Actually, there is only few work that addresses directly a general reference architectures for event processing. (Chandy and Schulte, 2010) present a coarse-grain reference architecture for EPNs containing event producers, channels, consumers, and intermediaries. In (Paschke and Vincent, 2009) the processing entity is structured into three logical steps: event selection, aggregation, and handling.

An initiative towards professionalization of the area is the EPTS Reference Architecture Working Group founded by the Event Processing Technical Society (EPTS) in 2009 (EPTS, 2012).

3 PATTERNS

In the following we present the patterns in in an adequate structured form: first, we describe the problem in its context, then the solution, and finally the benefits and dependencies of the pattern are evaluated.

3.1 Architectural Patterns

Architectural patterns describe how the entire system can be structured and divided into components.

3.1.1 Layers

Problem and Context: First, the top-level structure of the EDA has to be determined: The different tasks and responsibilities of the resulting components have to be identified and clearly separated.

Solution: The familiar architectural pattern of *software layers* (Buschmann et al., 1998) can also be applied on event processing systems. The overall structure of an event-driven architecture consists of three different *logical* layers.

1. *Monitoring Layer:* The Monitoring Layer is responsible for capturing the events from multiple event sources as well as for the preprocessing of the raw events. First, it implements the technical access to the different event sources. In a second step, the proprietary formats of the event data are translated into standardized internal event objects.
2. *Event Processing Layer:* The Event Processing Layer is responsible for the core processing of continuous event streams using CEP, which is responsible for event *pattern matching*.
3. *Event Handling Layer:* The Event Handling Layer is responsible for appropriate reactions on matched event patterns. Mostly, a service in the enterprise systems that performs simple business functions or entire business processes is triggered.

Evaluation: The *Layers* pattern describes the global structure of an EDA and decouples the key responsibilities: detecting events, processing events and reacting on events: the Monitoring Layer does not understand the further processing, the Event Processing Layer has no knowledge about the technical details of the event sources, and the Event Handling Layer does not know fine-grained events and has no idea how complex business events have been processed by the above layers. The pattern can also be find in other EDA work, e.g. (Schulte and Bradely, 2009), (Paschke and Vincent, 2009), (Michelson, 2006).

3.1.2 Agents

Problem and Context: The processing of events is realized by a set of event processing rules. To reduce

complexity and to make event processing scalable an approach is required for distributing event processing rules to different execution environments.

Solution: David Luckham has introduced the concept of event processing agents (EPA) in (Luckham, 2002). An EPA is a software component specialized on event stream processing with its own rule engine and rule base. An event processing network (EPN) connects several EPAs to constitute an architecture for event processing. Event processing agents communicate with each other by exchanging events.

Evaluation: Event processing agents are a central and well-known CEP concept (Etzion and Niblett, 2010), (Luckham, 2002). Agents can be considered as a distribution pattern defining the relation between rules and runtime environments. Firstly, agents provide an approach for modularizing and structuring rules in the Event Processing Layer. Light-weighted agents with few rules improve comprehensibility and maintainability. Usually, the rules contained in a certain agent fulfill a coherent domain-specific task. Secondly, agents make the system faster and more scalable by exploiting distributed system architectures.

3.1.3 Pipeline

Problem and Context: To cope with complexity of event processing, a general approach for structuring the Event Processing Layer is needed.

Solution: A pipeline can be used as the general event processing model. A pipeline is of a chain of processing elements, arranged so that the output of an element serves as the input of the next. This approach applies the well-known pipes-and-filters pattern (Buschmann et al., 1998) on event processing. The different stages of the pipeline are characterized by the following properties.

Each stage knows only its direct successor. The first stage processes the raw events of the event sources. The final stage sends events to the backend systems for triggering event handling.

A stage transforms incoming simple events into more complex and meaningful events. Thus, the processing of elements of the pipeline is directly related to the hierarchy of event types. Each processing stage is implemented by an event processing agent.

Evaluation: The *Pipeline* pattern provides a general processing model for events. In particular, the pipeline stages introduce sequential processing order in CEP. The rules of a certain stage cannot be applied until the processing of the former stage has been finished. This sequential view on event processing makes the Event Processing Layer easier to understand and to maintain. Furthermore, the stages of the pipeline are good candidates for processing agents. They provide loosely coupled processing elements implementing a fire-and-forget strategy: a stage sends its results as event to its successor without knowing anything about the subsequent processing stages.

3.2 Design Patterns

Design patterns describe the general mechanisms and building blocks of event processing. They specify archetypal processing stages of the *Pipeline* pattern.

3.2.1 Event Consistency

Problem and Context: Event data can contain incorrect, meaningless or faulty data. Especially data captured from the physical world tends to be noisy and unreliable (Jeffrey et al., 2006).

Solution: A special cleaning step has to be processed by a dedicated cleaning agent to detect any relevant inconsistency. OCL constraints (OMG, 2003) can be used to define valid ranges of event values. They can be easily transformed into event processing rules (Dunkel et al., 2009). Inconsistent events can be deleted so that only completely consistent events are subsequently processed. Incorrect data is transformed into meaningful data, e.g. by using default values. So, also events with partly incorrect data are considered. Eventually, incorrect events could simply be ignored, if the subsequent event processing rules can deal with wrong event data.

Evaluation: An explicit cleaning step guarantees that only consistent events are considered in subsequent event processing steps. Thus, it simplifies the CEP rules, because they only have to deal with reasonable data. Generally, the *Consistency* pattern decouples event cleaning from event processing.

3.2.2 Event Reduction

Problem and Context: Event processing systems have to deal efficiently with extremely high volumes of event data. Therefore, one of the first stages in the processing pipeline tries to reduce the number of events as early and as drastically as possible.

Solution: There are several design patterns to reduce the set of the considered events.

- *Filtering:* All events that are not required for further processing are logically removed from the

event stream, i.e. only relevant events are passed to the next stage. Usually, filtering criteria are rather simple, just considering events of a certain type or containing data of specific ranges.

- *Content-based Routing:* propagates events only to those agents that are interested in this type of events. It provides some kind of event dispatching and splits an event stream into different substreams. In fact, the total number of events is not decreased, but each EPA has to deal with fewer events. Note that content-based routing yields stronger coupling between the agents, because they need some knowledge about subsequent processing (Bruns and Dunkel, 2010).
- *Sliding Windows:* To cope with the infinite number of data, *sliding windows* are used, i.e. a historical snapshot considering the most recent set of events. Sliding windows are well-established in event processing. Nearly all event processing languages contain language constructs for defining sliding windows. Sliding windows extract a limited set of events from an infinite stream of events.

Evaluation: Different types of filtering mechanisms reduce the number of events. All the mechanisms are well-known and established (Etzion and Niblett, 2010), (Hohpe and Woolf, 2003). Filtering should be processed in a dedicated EPA as early as possible. Content-based routing provides the controlling of event streams and can be implemented by a single dispatching agent or spread over multiple agents. In contrast, reduction windows are used in a single rule as restrictive condition.

3.2.3 Event Transformation

Problem and Context: Usually, event data is not optimized for further processing: Event sources produce events in a proprietary data format, which cannot be used directly in the CEP engines. Often, events don't carry all the data necessary in a processing step. Therefore, expensive accesses to the backend system are necessary to retrieve the missing information.

Solution: There exist two types of transformation steps to cope with the above problems.

- *Translation:* In an explicit translation step, the data format is changed to another format appropriate to further processing (e.g. XML or JSON). Note that a translation step does not add any new information to an event instance.
- *Content-enrichment:* Simple events contain only low-level data and are often incomplete for further processing. For instance, measured vehicle velocities should berelated to the allowed speed limit. Thus, in an explicit transformation step the event objects are enriched with additional structural information by accessing the backend systems. The content enrichment step improves the performance of the entire system by reducing the number of requests to the backend systems in the subsequent processing steps.

Evaluation: Event translation and content-enrichment preserve the total number of events, but change the data format or insert additional information to event objects. Content-enrichment is also established in event processing (Coral8, 2006) or message-oriented systems (Fowler, 2002). Note that added information does not put any new meaning to the events, but makes them self-contained for the next processing steps. The content-enrichment step should be located after event reduction and before the more intelligent event processing starts, which needs the enriched events.

3.2.4 Event Synthesis

Problem and Context: The main task of complex event processing is to extract a domain-specific meaning out of the observed streams of simple fine-grained and uncorrelated events. Considering a solitary event is usually of no significance, because it represents just a single incident in the physical world. Instead, according to the key idea of CEP, a set of fine-grained simple events must be correlated to a single complex event with a significant meaning (Luckham, 2002).

Solution: A set of simple events is synthesized to one complex event that has more meaning. Event correlation is strongly domain-specific, but some general aspects of correlation can be identified. First, we discuss how to determine the set of correlated events. Then two stages of correlations are distinguished: granularity shifts and semantic shifts.

- *Correlation Sets:* A certain correlation set contains all events that share a particular context. Generally, we can distinguish different dimensions for constructing correlations sets: *Domain-specific correlation* correlates events according to domain-specific properties (usually determined by constraints on the event data attributes). *Temporal correlation* takes temporal relations between events into account: Nearly all event processing languages include temporal operators to specify the chronological order of events or to consider temporal sliding windows. *Spatial correlation* considers those events that are observed at a certain location for identifying spatial behav-

ior. To consider the occurrence space of events, spatial operators must be used. Such operators should allow space coordinate windows similar to temporal sliding windows.

- *Granularity Shift:* Usually, the events emitted by the event sources are too fine-grained and uncorrelated to be meaningful for the business processes. Therefore, an important aspect of CEP is achieving a shift of the *event granularity*. (Too) many fine-grained events are aggregated to a new and more abstract composite event. Aggregation uses rather simple numerical operations such as calculating averages or sums of event data for correlating events of the same type. It reveals no new sophisticated insights in analyzing the situation caused by the arrived events.
- *Semantic Shift:* The key issue of CEP is shifting the event semantics: The single events within a certain correlation set are mapped to a new complex event that assigns the occurred events a new non-obvious and semantically richer meaning. The underlying correlation sets are based on rather complex event patterns with respect to the sophisticated knowledge of domain experts. Semantic patterns describe complex dependencies between events taking temporal constraints into account. The significance of a complex event is its occurrence, not the data it is carrying. A complex event can be used in subsequent processing or for triggering immediate reactions in operational enterprise systems.

Evaluation: Event synthesis aggregates simple events and transforms them to a new complex event on a higher level of abstraction. Event correlation is discussed generally in the context of event patterns in many works, e.g. (Luckham, 2002), (Coral8, 2006). Here, we present some general correlation mechanisms that are application independent. First, correlation sets determine the simple events to be correlated. The concept of correlations sets is well-known in other areas such as in business process modeling. Secondly, we argue that event correlation is processed in two subsequent steps of the event processing pipeline. A granularity shift just aggregates simple fine-grained events to a coarse-grained composite event still without providing new gain of insights. Then, complex event processing rules define patterns in the stream of composite events that represents a significant meaning about. Thus, a semantic shift takes place.

4 REFERENCE ARCHITECTURE

In this section, we apply the catalogue of patterns to derive a *reference architecture* for event processing. The reference architecture provides a template for the design of event processing systems. To obtain a concrete architecture for a particular domain, the reference architecture has to be customized and adapted to the specific domain requirements. Figure 1 depicts the logical view of the architecture with its essential layers and components.

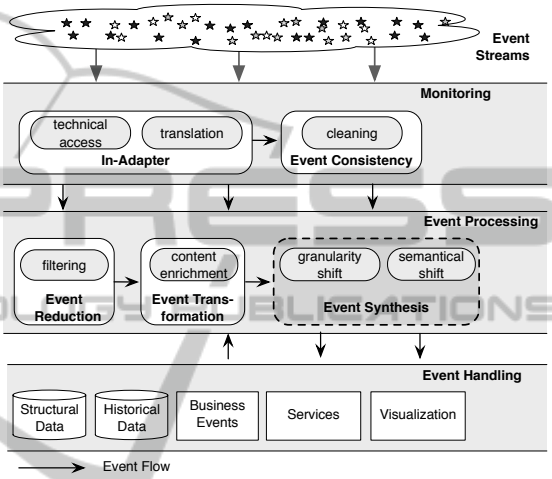

Figure 1: Reference architecture for event processing.

The global structure is based on the *Layers* pattern distinguishing the three software layers monitoring, event processing and event handling with the particular responsibilities described in section 3.1.1.

The event processing layer is organized as a processing *Pipeline* according to section 3.1.3. The pipeline provides a general processing model of events defining a number sequentially traversed stages. Each stage of the pipeline is implemented by a dedicated *Agent* (as described in section 3.1.2). Light-weighted agents allow the distribution of event processing rules between different runtime environments. Design patterns of our pattern catalogue describe the different components of each layer:

Event Monitoring: The monitoring layer is responsible for gathering events from multiple event sources as well as for the preprocessing of the raw events. The *In-Adapter* component implements the technical access to the different types of event sources. Afterwards, according to the *Event Consistency* pattern, a separate cleaning step is processed (see section 3.2.1): defect, faulty or double event objects are sorted out or corrected.

Event Processing: The event processing layer implements a pipeline, where each stage transforms sim-

ple incoming events into more complex and meaningful events. The pipeline is organized on the base of the following archetypal agents. As a first step of the pipeline an *Event Reduction* agent reduces the number of events by performing a *filtering* step (as described in section 3.2.2). Only events relevant for further processing are propagated to the next pipeline stage.

Then the relevant events pass an *Event Transformation* phase, where a *content-enrichment* step is processed (according to section 3.2.3). Structural data retrieved from the backend systems is added to the event objects to make them self-contained for the next processing steps.

Event Synthesis: The last stage in the processing chain is the most significant one: The filtered and enriched simple events are aggregated, correlated, and synthesized to new and more complex (business) events. This stage is usually complex and can consist of a processing pipeline on its own. The concrete steps are domain-specific and can only be described on a general level as discussed in section 3.2.4. First *event aggregation* provides a *granularity shift*. Then a semantic shift infers higher insight out of the set of correlated events by applying rules based on expert knowledge.

Event Handling: The event handling layer is realized by the enterprise backend systems, which implement appropriate reactions to received events (see bottom layer in Figure 1).

5 CONCLUSIONS

In this paper, we introduced a catalogue of patterns that supports the development of event-driven enterprise architectures. All patterns are described in a standardized form providing a vocabulary and a conceptual framework that leads to a more structural approach for the design of event processing systems.

We proofed the usefulness the presented pattern catalogue by deriving a general reference architecture. Our reference architecture is pattern-based and supports the developers of event processing applications in their design decisions. It facilitates the construction of systems, leads to reduced development time of event processing applications, and fosters the reuse of successful EP solutions.

REFERENCES

Bruns, R. and Dunkel, J. (2010). *Event-Driven Architecture: Softwarearchitektur für ereignisgesteuerte Geschäftsprozesse*. Springer-Verlag, Berlin Heidelberg.

Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., and Stal, M. (1998). *Pattern-Oriented Software-Architecture Volume 1: A System of Pattern*. Addison Wesley.

Chandy, K. and Schulte, W. (2010). *Event Processing: Designing IT Systems for Agile Companies*. McGraw-Hill.

Coral8 (2006). Complex event processing: Ten design patterns. http://complexevents.com/wp-content/uploads/2007/04/Coral8DesignPatterns.pdf.

Dunkel, J., Fernandez, A., Ortiz, R., and Ossowski, S. (2009). Injecting semantics into event-driven architectures. In *Proceedings of the 11th International Conference on Enterprise Information Systems (ICEIS)*, pages 70–75.

Dunkel, J., Fernandez, A., Ortiz, R., and Ossowski, S. (2010). Event-driven architecture for decision support in traffic management systems. *Expert Systems and Applications*.

EPTS (2012). Event Processing Technical Society. http://www.ep-ts.com.

Etzion, O. and Niblett, P. (2010). *Event Processing in Action*. Manning.

Fowler, M. (2002). *Patterns of Enterprise Application Architecture*. Addison-Wesley.

Gamma, E., Helm, R., Johnson, R., and Vlissides, J. (1998). *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley.

Hohpe, G. and Woolf, B. (2003). *Enterprise Integrations Patterns: Designing, Building, and Deploying Message Solutions*. Addison-Wesley.

Jeffrey, S., Alonso, G., Franklin, M., Hong, W., and Widom, J. (2006). A pipelined framework for online cleaning of sensor data streams. In *ICDE*, pages 140–142.

Luckham, D. (2002). *The Power of Events*. Addison-Wesley.

Michelson, B. (2006). Event-driven architecture overview. http://www.psgroup.com/detail.aspx?id=681.

OMG (2003). UML 2.0. Object Constraing Language (OCL) Specification. http://www.omg.org/docs/ptc/03-10-14.pdf.

Paschke, A. (2009). A semantic design pattern language for complex event processing. In *Proceedings of the AAAI Spring Symposium on Intelligent Event Processing*, pages 54–60.

Paschke, A. and Vincent, P. (2009). A reference architecture for event processing. In *Proceedings of the Third International Conference on Distributed Event-Based Systems (DEBS)*. ACM.

Schulte, R. and Bradely, A. (2009). A Gartner reference architecture for event processing networks. ID G00162454.