Summer 6-15-2016

CONCEPTUAL MODELING OF EVENT PROCESSING NETWORKS

Christian Janiesch
University of Würzburg, christian.janiesch@uni-wuerzburg.de

Jakob Diebold
Karlsruhe Institute of Technology, jakob.diebold@student.kit.edu

CONCEPTUAL MODELING OF EVENT PROCESSING NETWORKS

Research

Christian Janiesch, Julius-Maximilians-University of Würzburg, Würzburg, Germany, christian.janiesch@uni-wuerzburg.de

Jakob Diebold, Karlsruhe Institute of Technology, Karlsruhe, Germany, jakob.diebold@student.kit.edu

Abstract

Complex Event Processing (CEP) enables analyses with high velocity on high volume and varied data. It is an established technology to enable flexible event-driven systems. These systems can provide real-time analytics for business process management or support the realization of cyber-physical systems in the context of the Internet of Things. While the technology is maturing rapidly, the design of event-based systems is still in its infancy and complex. In particular, there is no established means to support the early stages of the IS development process through conceptual modeling. Currently, any comprehensive graphical specification of the involved event processing networks (EPN) is tool-dependent and creates vendor lock-in effects before an informed decision for a software product can be made. Current conceptual modeling options are limited to generic flow diagrams and methods from related areas. In this paper, we describe a method to conceptually model EPN which allow not only the filtering and projection but also the translation, enrichment, and aggregation as well as the splitting and composition of events. This artifact enables the design of EPN prior to deciding for a particular CEP product alleviating vendor lock-in effects. We present meta models excerpts and a notation as well as sample code in Esper to emphasize their serializability. The results are applied to a sample process.

Keywords: Complex Event Processing, Event Processing Agent, Event Processing Network, Conceptual Modeling.

1 Introduction

Current problems in big data processing and business analytics require new approaches to handle data. Complex Event Processing (CEP) enables real-time processing of varied high velocity and high volume data. CEP is an established technology to enable flexible event-driven systems providing analytics for business process management, enterprise systems or sophisticating cyber-physical systems in the context of the Internet of Things. CEP provides a processing platform running in parallel to existing IT infrastructure based on the principle of loose coupling (Janiesch et al., 2012).

CEP is not tied to particular data specifications but it can operate with a variety of data inputs as events, be it from IT systems, sensors, or human beings. The interaction of CEP and other systems is also based on events: CEP as well as other systems can act as a producer and as a consumer of events. Technologically, events are data objects with attributes and data components. Events signify activities that have taken place or are thought of having taken place. Events can also signify the absence of activities (Etzion and Niblett, 2010, Luckham, 2002).

CEP comprises a set of techniques for making sense of the behavior of a system by deriving higher-level knowledge – so-called complex events – from lower-level system events in a timely and online

fashion (Etzion and Niblett, 2010, Luckham, 2002). Semantically, CEP queries can be formulated as event processing networks (EPN) which consist of individual event processing agents (EPA). These are later be implemented as SQL-like queries to process new event data. In contrast to regular databases, CEP focuses on processing data on arrival rather than on request,

The technology is maturing rapidly, yet the design of event-based systems is still in its infancy and complex. While there is some consensus on the demeanor of the basic EPAs, there is no established means to unambiguously specify them in an EPN other than raw code (Janiesch et al., 2012, Krumeich et al., 2014).

However, in order to achieve consensus in the early stages of the design process, it is important to be able to conceptually discuss EPN inputs, outputs as well as – and in particular – the filtering and transformations that take place in an EPN. Currently, as a means of shared representation there is no conceptual modeling method for EPN available which is software independent. Hence, for the development of a CEP system, the system's behavior cannot be modelled before the decision for an execution engine can be made. This creates undesired vendor lock-in effects.

In this paper, we derive modeling requirements from the CEP domain, compare them with a review of the available body of knowledge related to specifying EPAs, and define modeling constructs which allow the tool-independent modeling of an EPN. The contribution, the artifact, of this research, thus, is a method which can be used to specify EPN conceptually. Such a method enables IS developers to design a CEP system's event processing as well as its interaction with other IT systems such as an enterprise system or a business process management system.

Since modeling is not necessarily supposed to be an end to itself, it is useful if conceptual models can be used as stubs or skeletons for further implementation or if they are executable altogether. We also take this into consideration when discussing the model as well as possible instantiations.

In the following, we derive requirements for EPN modeling and match them to research findings of related work. Based on the open issues identified, we develop constructs to model EPN. We specify meta model excerpts for unambiguous understanding and provide sample code in Esper EPL to emphasize on their serializability as well. The results are applied to a sample process before we discuss the adequacy and limitations of the results. The paper closes with an outlook on ongoing and future work.

2 Foundations and Related Work

2.1 Conceptual Modeling and Research Methodology

In order to facilitate any IS development process, development groups must agree on some shared form of representation. Based on this representation, ideas, thoughts, opinions, objectives, and beliefs about the object system can be exchanged and discussed (Hirschheim et al., 1995). Conceptual modeling methods are considered to be a suitable means of representation (Becker and Pfeiffer, 2007, Frank, 1999, Janiesch, 2007).

In general, the specification of a conceptual modeling method consists of two parts: first the modeling method's meta model specifying the models abstract and concrete syntax and second the methods' procedure model (Brinkkemper, 1996, Nuseibeh et al., 1996, Strahringer, 1996). The meta model is concerned with the representation of the model and its formalization of its constructs and their attributes and views. It is sometimes equivalently denoted as modeling grammar (Wand and Weber, 2002), language (Becker et al., 2001). The procedure model gives advice on how to efficiently create a semantically useful model in accordance with the syntax specified by the meta model (Saeki, 1995, Wand and Weber, 2002).

This research to develop such a conceptual modeling method follows the design science paradigm as introduced by Simon (1996) and applied to IS by Hevner et al (2004). Following Gregor and Hevner's (2013) knowledge contribution framework, we consider the design science research contribution of our work an improvement over existing proprietary modeling approaches as well as an exaptation of common procedures in conceptual modeling for procedural applications to EPN design which so far has been dominated by computer science research. The design search process was an iterative process of building artifacts and theories, intervention and learning, and enhancement. It was aligned with suggestions by Peffers et al. (2007). Concerning an evaluation, Gregor and Hevner (2013) argue that "a proof-of-concept may be sufficient" when judging design science research contributions. We successfully implemented our conceptual modeling method as an expository instantiation (Gabriel and Janiesch, 2016). Yet, in the future we still intend to develop testable propositions and further evaluate and improve the artifact.

2.2 General and CEP-specific Requirements

CEP systems are usually designed as multiple (interconnected) EPNs which consist of multiple EPAs. So the connection of the EPN to other systems or EPN has to be clear as they can act as a producer or as the consumer of the EPN's events. Also, it is important for any development group to quickly grasp the content of an EPN. Thus, it is important to quickly identify the corresponding EPA types. Hence, it is important *not* to design a method that only uses a single construct for all EPA similar to a simple flow chart, but to provide means to create a graphical model which can be understood and used without consulting secondary information such as attached tables or attribute values.

Consequently, apart from the inclusion of constructs for connectivity such as EPN inputs, outputs, and event channels, the most important part of the requirements engineering is the determination of the individual EPAs which have to be included in the modeling method. While Luckham (2002) only distinguishes generic patterns, rules, and constraints, which mainly provide input for serialization of queries, Etzion and Niblett (2010) distinguish several distinct logical EPA constructs: a filter EPA, six types of transformation EPAs, and several pattern detect EPAs according to their functionality. They also discuss event producers and event consumers which represent the connection of the EPN to other systems. We consider them a sensible basis for our design method.

Their Filter EPA selects individual events from the event stream. Only events, which fulfill or do not fulfill certain conditions, are passed on. Filters can be represented as separate agents, but in the definition of the EPA construct by Etzion and Niblett (2010) every agent includes a filter for incoming events. They target individual events as well as amount- or duration-based windows.

Table 1 summarizes the transformation agents by Etzion and Niblett (2010). Note that due to length restrictions this paper will only cover transformation agents.

EPA	Function
Translate	This EPA generates a single event derived from a single input event using a predefined function.
Enrich	The input event is enriched with data from external sources (e.g. from a relational database). It is an extension of a Translate EPA.
Project	The derived event only contains a subset of the attributes of the input event.
Aggregate	A collection of events is aggregated to one derived event by applying a function.
Compose	This EPA combines the information of multiple events from different events streams into a new event.
Split	A collection of output events is produced from a single event. These can be projections or clones of the incoming event.

Table 1. Event transformation agents.

2.3 Comparison with Constructs Available from Related Work

As introduced above, Sharon and Etzion (2008) and Etzion and Niblett (2010) propose an approach to model CEP in their seminal work. They focus on the event processing network (EPN) to describe the interaction of the individual event processing agents (EPA). However, the notation only distinguishes four basic elements of in an EPN: (1) event producer, (2) event consumer, (3) event processing agent, and (4) event channel. All further details are put in a textual description of each element. This requires an additional table to completely model an EPN, since most information about the event processing is described in this table. Without it, only the general flow of an event in an EPN can be reconstructed. It is hardly suitable as a means of communication since the model is not self-explanatory.

Anicic et al. (2010) propose a rule-based language to define event processing networks. Their work provides a comprehensive abstract syntax, yet they do not provide a concrete syntax. Hence, it is an approach that is not intended to (and does not) enable the design of conceptual graphical models.

Vidačković (2014) describes the Event Processing Model and Notation (EPMN) as a method for the modeling and execution of dynamic business processes based on event processing concepts. A notation for CEP, however, is not in focus but the definition of sub-processes requiring dynamic capabilities based on the Esper Event Processing Language (EPL) and XML.

Decker et al. (2007) propose the Business Event Modeling Notation (BEMN) which focusses on conjunction, disjunction, and inhibition as well as cardinality of events in conjunction with process modeling. Their notation cannot stand on its own, it uses Business Process Model and Notation (BPMN) (ISO/IEC, 2013) constructs to define event rules which can be used to enhance process management. It contains filters and logical operators though. But it does not cater for event transformation.

Appel (2014) also proposes a CEP notation in the context of process management. His Service Allocation Diagrams do not specify event processing logic in EPNs but only capture the configuration of an EPN as a whole with its roles, inputs, outputs as well as some technology dependency. The level of abstraction of his Event Stream Processing task is similar to that of a business rule task in BPMN: It acts as an anchor for detailed specification in another system. Hence, his method is geared at being used as an annotation in conjunction with BPMN and the Event-driven Process Chain and not as a stand-alone notation. However, inputs and outputs are relevant secondary constructs as already brought up by Sharon and Etzion (2008). He also includes event channels which do not exhibit processing functionality as such but allow connecting EPAs.

All of the latter three have a close link to modeling CEP behavior alongside business processes. However, neither of them includes modeling constructs which can be used to model transformation behavior. They all rather focus on event routing with logical operators (i.e., AND, OR, XOR), which are more closely related to the modeling of logical patterns covered by pattern detect EPA.

Friedenstab et al. (2012) also propose an extension of BPMN to model Business Activity Monitoring (BAM). Their modeling method enables the design of BAM functionality together with a BPMN process model. The authors present a meta model which describes the abstract syntax of the BPMN extension formally. They also provide a concrete syntax and give notation examples on how the abstract syntax can be represented. The focus of the modeling are mathematical relations of performance indicators though, not events. The work of Friedenstab et al. (2012) cannot be directly used for CEP since the behavior of metrics is different from events. Metrics only respond to mathematic transformation. Events can be split and projected based on their attributes and types. Some of their conceptualizations can be adapted to event processing though. For the expression of filters in BAM models, e.g. they propose a *Filter* class. A Filter has to contain at least one of either an *Input Limit* or a *Condition*. In order to apply the limit or condition a *Measure* can have one Filter, whereas a Filter can refer to one Measure or more. The Input Limit restricts either the time or the quantity in which the events are being processed. This enables the modeler to define windows such as ‘the last 5 events’ or ‘the last 20 seconds’.

On a query level, this subsumes the three different Filter categories of Decker et al. (2007): time, data, and environment.

Apart other constructs, Figure 1 shows the concrete syntax of a Filter including both examples of a Condition and an Input Limit. The Filter, represented by the funnel symbol, is associated to a model element. Also, they have defined a Condition construct in their BAM meta model and use it in their example to perform filter functionalities, but since the Filter provides a possibility to include conditions, we consider it more unambiguous to use only the Filter symbol.

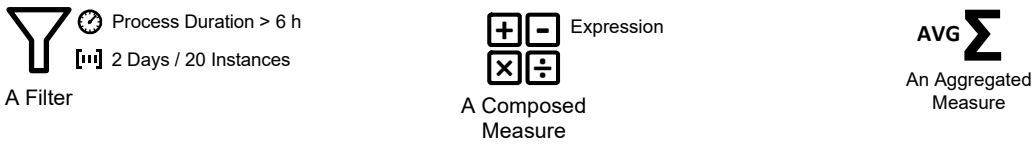


Figure 1. BAM constructs and notation by Friedenstab et al. (2012).

2.4 Summary of Requirements

As argued above, we have divided transformations into six subclasses: Translate, Enrich, Project, Aggregate, Split, and Compose. Here, Friedenstab et al. (2012) provide a good starting ground.

The *Translate* EPA performs a function on a single input event to derive an output event. To express this function, the *Composed Basic Measure* from Friedenstab et al. (2012) can be adopted since its basic function is the calculation of a new measure. Only minor adaptions are necessary to include events and more sophisticated functions. So far, the definition only allows the use of the four basic arithmetic operators (+, −, /, *). Furthermore, the name needs to be changed because the term 'Composed' can be mistaken for the Compose EPA. The two subclasses of Translate, *Enrich* and *Project*, both of them do not have a representation in the BAM modeling method.

The *Aggregate* EPA applies a function over a collection of input events and derives a single new output event. This functionality is already discussed by Friedenstab et al. (2012) as an *Aggregated Measure*. However, in BAM it is defined only to cover individual measures which are aggregated. The construct has to be adapted to handle and aggregate events.

The *Split* EPA is not included in the BAM modeling method, because its focus is on measure computation. It does not perform any computation on incoming events which may need to be split up. The *Compose* EPA joins events from two or more event streams. These input streams can contain different event types. Through a matching criterion, the two streams are composed on the event level. The so-created event goes into an outgoing event stream containing a new event type. Similarly, this has not been covered so far.

Friedenstab et al. (2012) have defined further constructs in their BAM modeling method. The construct *Frequency* is a specialization of an Aggregate EPA doing a count-based aggregation. The construct *Duration* is a Translate EPA which calculates the difference of two timestamps. It remains to be seen if these constructs provide valuable shortcuts for modeling EPN as well.

The existing definition of filters covers most requirements for CEP. The only addition required is the batch or fixed window functionality. So far, only rolling windows are supported but event processing languages also provide the possibility of handling batches. Batches also include a certain period or number of instances, but in contrast to rolling windows, an event can only be part of exactly one batch.

Transformation EPA constructs have to be developed for Enrich, Project, Compose, and Split functionality. Translate and Aggregate EPAs can be derived from prior art with minor changes. The Translate EPA requires the availability of functions besides basic arithmetic operations. Aggregate is defined for measures only, but it can be extended for the modeling of CEP.

3 Modeling Method Specification

3.1 Abstract Syntax

In the following, we discuss the meta models of our modeling method. To highlight the practical relevance of the developed constructs, we have derived code examples based on the Esper EPL (EsperTech Inc., 2014). They implement the EPAs functionality for a CEP engine.

Before we detail the individual EPA, we need to specify the constructs of event and an (abstract) EPA before going into detail on transformation constructs.

Event: Figure 2 shows the definition of the class *Event*. To process an Event, a unique identifier and a timestamp are required. The timestamp allows relating events to each other based on their occurrence. This model only defines the basic *Properties*. Depending on the system, the CEP model is designed for, it may require more attributes for every single event, such as the event source. Also, the activity, the technical system, or the estimated veracity may be of interest for the processing. Additionally, an Event has exactly one *Event Type* that defines which properties the event inherits. In an EPN, *Event Streams* connect EPA with each other as well as with event producers and consumers. We use the term event stream in the same sense as event channel. This modeling method only supports homogeneous event streams. Thus, an Event Stream only contains events of one type. Heterogeneous streams are not included in the model because different event types in one stream would require additional definitions in the EPN which would make the model less clear and understandable. Multi-type event streams can be simulated by drawing multiple single-type event streams between two EPAs.

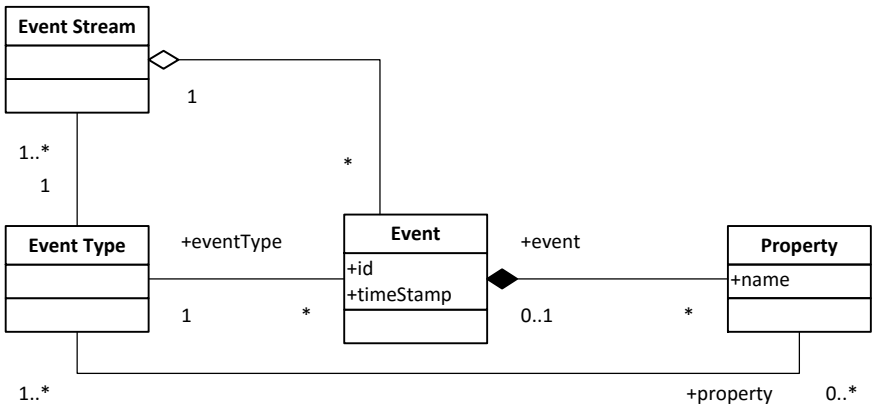
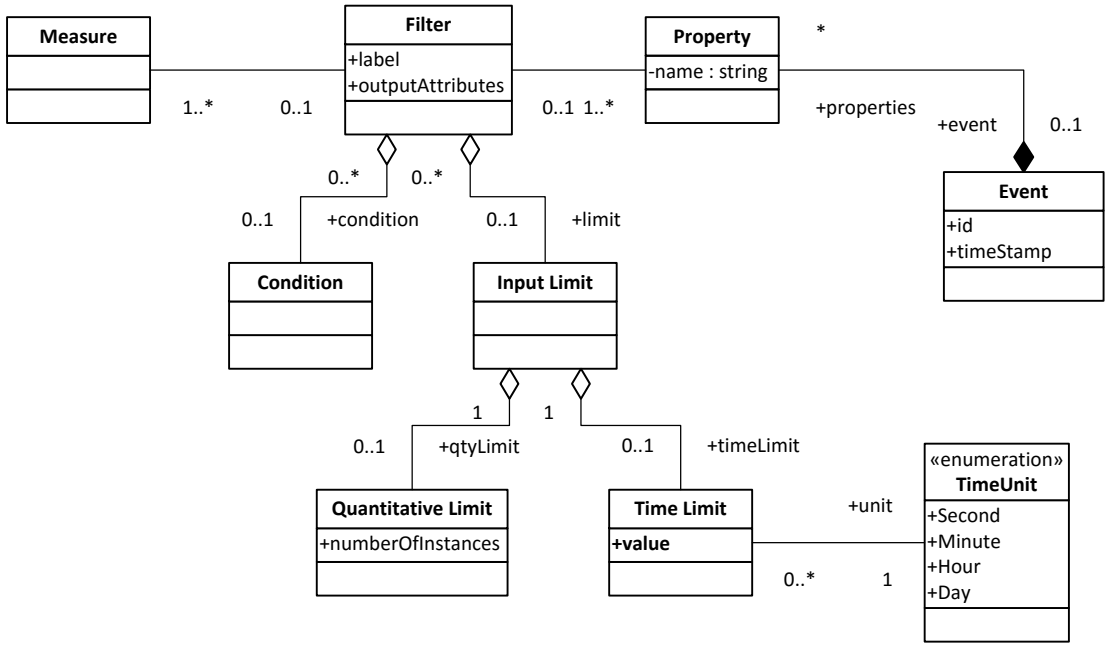


Figure 2. UML definition: Event.

Event Processing Agent: An *EPA* is the super class for all agents which are needed for an EPN. The three subclasses are *Filter*, *Transformation*, and *Pattern Detect*. A Filter is also part of every EPA, as all input streams may need to be filtered by the EPA before the actual processing step (Etzion and Niblett, 2010).

Filter and Project: The meta model for the *Filter* is based on the definition proposed by Friedenstab et al. (2012), because their semantics do not differ substantially from Filters as required by CEP. In order to enable event processing, the Filter is associated with the properties which are part of an event. For the application of the Filter, the *Property* is checked against a *Condition* or *Input Limit*. If the Condition is fulfilled or the Input Limit is not yet exceeded, the event passes through the Filter. If not, the event is filtered out. Another addition is a list of attributes in the filter, which makes it possible not only to filter events in or out as a whole, but also allows defining which attributes are passed on. Output attributes is a list of attributes which can contain all attributes of the event or only a subset. No additions can be made to the attribute list of the event. This function represents the *Project EPA*. It is

included in the Filter construct, because the only function the Project EPA has, is selecting (projecting) the attributes of an event. Therefore, an own construct for the Project EPA is not necessary and it can be included in the filter. Figure 3 shows the meta model and an example expression. The example expression (as well as all following ones) use a SQL-like query language to program EPN. Similar to SQL, properties of an event can be projected and used for selection. In terms of the meta model, `attribute1` and `attribute2` used in the examples are the names of such an Event Property.




```
select attribute1, attribute2 from
  EventStream(attribute1 > 0).win:time(5 sec)
where attribute2 < 1
```

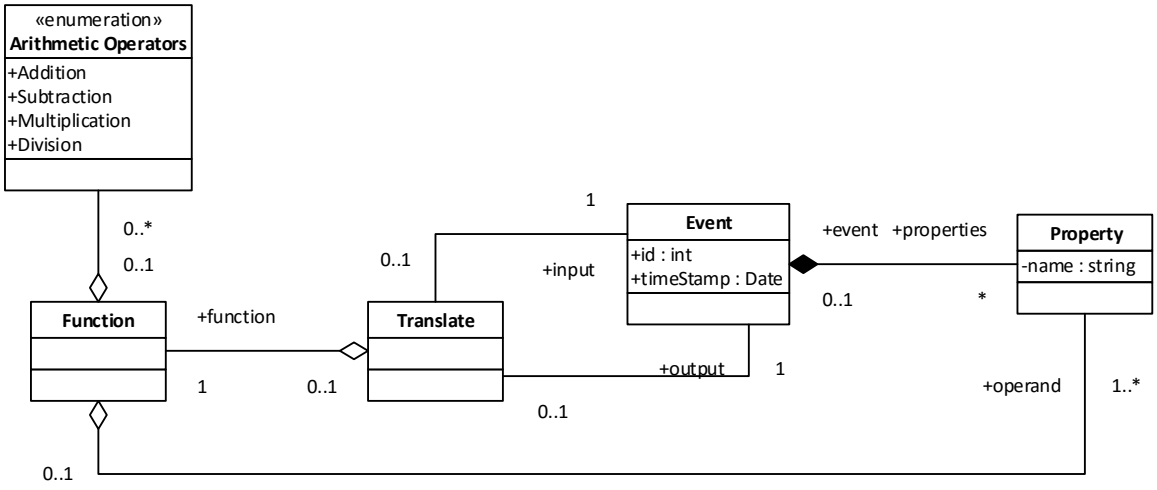
Figure 3. UML definition and Esper EPL query: Filter/ Project.

The above Esper EPL query gives a simple example: First, the `select` statement defines the output attributes of the filter. The first filter statement is in parentheses after the event stream definition. This filter statement is performed before any other processing of the query. This is especially important if the window is defined for incoming instances (e.g. `attribute1 > 0`). The input limits of the filter can be defined in the `win` statement. They can be rolling or fixed windows and for either time or a number of instances. Another filter is the `where` statement which contains all further conditions. They are processed after the two other filters.

Translate: The *Translate* EPA takes exactly one *Event* as an input and produces exactly one *Event* as an output. The UML definition is based on the structure of the Composed Measure proposed by Friedenstab et al. (2012). Except for the addition of events, the major change is the introduction of the *Function* class. Since the Composed Measure expected just a number as input, an enumeration of *Arithmetic Operators* was suitable for the calculation. Now with the handling of events, more sophisticated functions with the properties of an event have to be computed. Therefore, we added the *Function* class. It is aggregated from Arithmetic Operators and *Properties* of the input event.

To perform a function on an event in Esper EPL, expressions have to be used. An expression is defined before the actual query and can be used in the whole statement. Figure 4 shows the meta model and an example expression. It adds up the two attributes of the event.

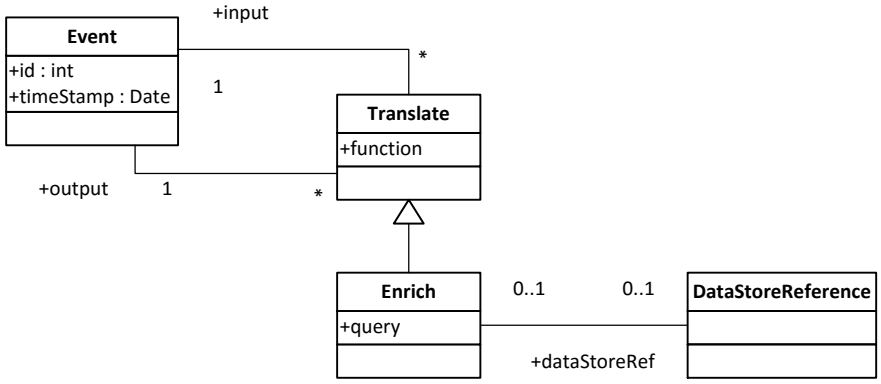
Enrich: The *Enrich* EPA is a subclass of the *Translate* EPA. Since the Enrich EPA is a translation with the addition of external data, a data source needs to be specified. Therefore, the class *DataStoreReference* is a part of the Enrich class. It references a *Data Store* as the external source of information which is added to the *Event.* Also, the Enrich EPA has the attribute query which contains the query that is executed on the external data source.




```
expression myExpression { x => x.attribute1 + x.attribute2 }
select myExpression(*) from EventStream
```

Figure 4. UML definition and Esper EPL query: Translate.

Figure 5 includes an Esper EPL query which enriches an event. In the `from` statement, the SQL database is added and in square brackets, the SQL query is defined. The attribute marked with the dollar sign is the reference to an event attribute which is matched to an attribute of the relational database. Queries are not limited to the relational model but could also include statements following an object-oriented or NoSQL model.



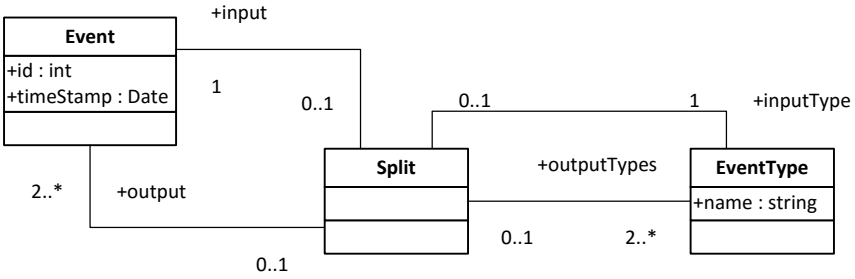

```
select sqlAttribute1, attribute2 from
  MyInputEvent, sql:ExampleDB [' select sqlAttribute1
  from Table where sqlAttribute2 = ${ attribute2 } ']
```

Figure 5. UML definition and Esper EPL query: Enrich.

Split: The *Split* EPA takes one single *Event* as input. In contrast to the Translate EPA and its subclasses, it produces more than one output event. The incoming *Event* is split into two or more outgoing Events. Those events may be copies of the incoming event or projections (containing a subset of the

original attributes). Hence, the meta model defines one input *Event* and also one input *Event Type.* But for both, output *Event* and output *Event Type,* two or more are required.

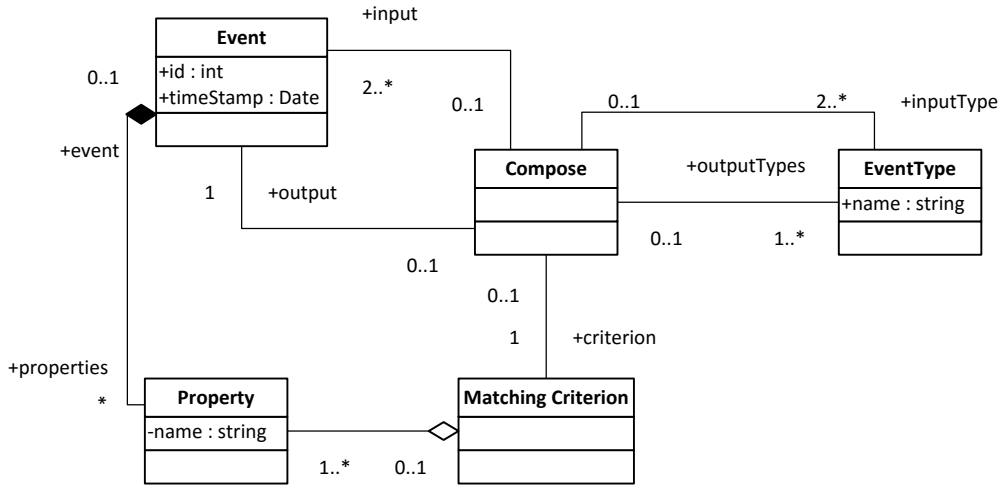
The Esper EPL query in Figure 6 shows how a Split EPA can be translated into code. The statement begins with the input stream and inserts this stream into the various output streams. Additionally, there is the option to choose the attributes which are included in the events of the output streams. This can be done in the `select` statement for each output stream. Esper provides the possibility of defining conditions to sort the input events into the different output streams. The `output all` statement is required if the event should not only be inserted into the first output stream with matching conditions but all matching output streams.




```
on InputStream
 insert into OutputStream1 select *
 insert into OutputStream2 select attribute1
 output all
```

Figure 6. UML definition and Esper EPL query: Split.

Compose: The function of the *Compose* EPA is the inverse of the Split EPA. Two or more *Events* from different *Event Types* are composed and create derived events. So the EPA has at least two input streams of at least two input *Event Types*. The class *Matching Criterion* defines how the input events are matched. The Matching Criterion has at least one *Property* which belongs to each of the input events. The events are then matched based on this property's value. Similar to the Split EPA, the operation is performed on the event level. This means that the join happens event-wise and not that the streams themselves are merged. Figure 7 shows the meta model and an example expression.



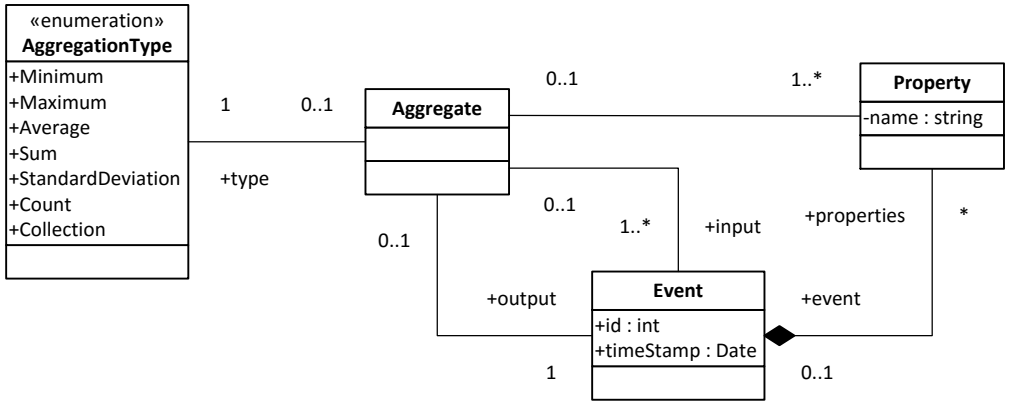

```
select * from EventStream1, EventStream2
 where EventStream1.attribute1 = EventStream2.attribute1
```

Figure 7. UML definition and Esper EPL query: Compose.

The Esper EPL query is similar to the SQL join statement. In the `from` statement, the input streams are named and in the `where` statement, the matching condition is defined. Usually, the matching condition is based on a unique attribute like an ID.

Aggregate: The *Aggregate* EPA has at least one input *Event*. All the input events are of the same *Event Type*. One or more *Properties* of this Event Type are aggregated over a defined window. In case of aggregating all input events, for some of the aggregation types the result would not be significant because the window of the aggregation is unknown. Each Aggregate EPA has an *Aggregation Type.* Apart from the classic SQL aggregation types, a set of event-specific aggregation types exists.

Figure 8 includes an Esper EPL query with an aggregation. Here, the average of attribute1 is computed grouped by attribute2. So, the average is computed separately for every distinct value which occurs in attribute2. The `group by` clause is not mandatory, but without it, the aggregation will be done over the complete set of input events, independent of other attributes contained in the event.




```
select avg(attribute1), attribute2 from EventStream
  group by attribute2
```

Figure 8. UML definition and Esper EPL query: Aggregate.

3.2 Concrete Syntax

The concrete syntax is the graphical representation of the constructs defined in the previous section. Its goal is to provide a notation to model an EPN in a clear and readable manner not only by CEP experts but also by business experts. Therefore, pictograms were chosen which express the function of the agent in a simplified way so that the user of the model can get a rough overview of the system at the first glance. For more details, labels are attached to the constructs, because not every fact can be displayed using a graphical representation. The following Figure comprises an overview of the concrete syntax of all six constructs, i.e. the notation of the EPN modeling method.

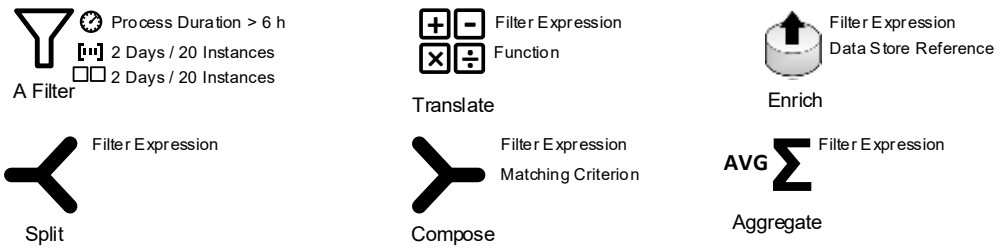


Figure 9. Notation for Filter and Transformation constructs.

The filter can be used as stand-alone EPA but it can also be included in every other type of EPA. In order to represent this filter in every notation element, all of them can have a filter expression to define

a filter only within the respective EPA. In most cases, the filter can also be defined separately for reasons of clearness or reuse.

While we have not discussed the following in the requirements analysis in detail, it is necessary to also include constructs which mark the inputs adapters and the outputs of an EPN as well as connectors to link EPA through event streams and allow further associations. They are depicted in Figure 10.

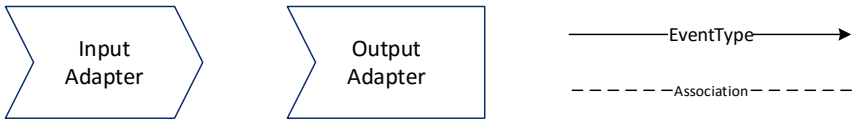


Figure 10. Further constructs: input, output, event streams, and process actuators.

The *Input Adapter* signifies the beginning of the EPN and is can be connected via an *Association* with another system acting as event producer. If only the EPN is modeled, *Input* and *Output Adapters* specify beginning and end of the EPN and can be labeled with the according event producers or consumers. The *Association* can also be used to connect an EPA with a system component such as a task in a business process or a dashboard which can be used as a shortcut for modeling instead of using an Output Adapter. The *Event Stream* connects EPAs in an EPN. Event Streams can be labeled with one EventType. If multiple event types need to be connected, multiple Event Streams have to be used.

Friedenstab et al. (2012) propose three types of process operations as direct interaction of the EPN and the business process: *Process Trigger*, *Process Abortion*, and *Process Suspension.* These can be used to indicate the start a new process instance via an API call, terminate or abort process execution or suspend the process until user intervention takes place. We have not included them in this specification since they relate to a specific application in the area of business process management.

4 Modeling Example

For a better understanding of the developed method and its use, Figure 11 provides a simple example of an EPN model with an exemplary serialization of the first three constructs. In this example we will only focus on the EPN not on the behavior of connected systems.

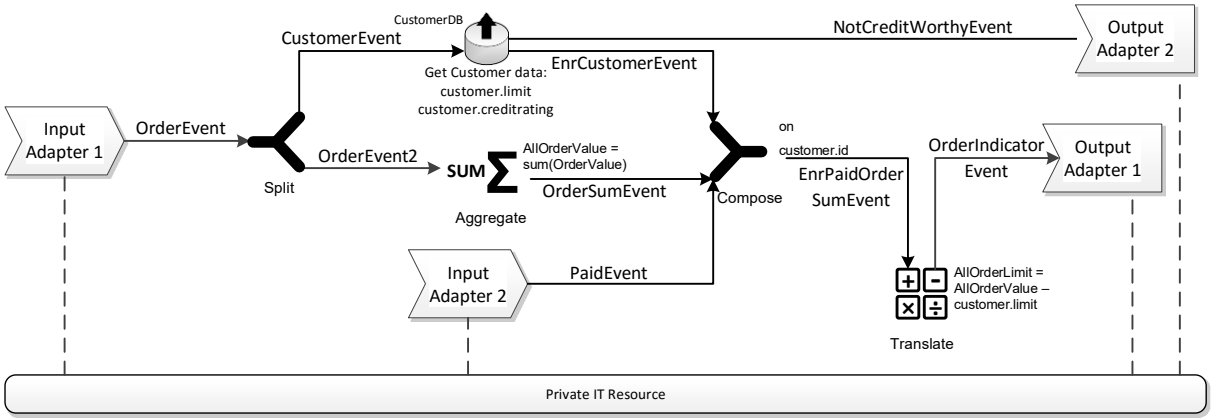


Figure 11. Example ordering process.

As it is often the case in an event-driven architecture, the concrete implementation of the consuming and producing IT systems is unknown. Hence, we consider them as a private resource. In this case, it could be a composite application which does order processing of minor goods. It provides events on order data as well as payment information. It expects event notifications on bad credit ratings to possibly abort as well as information on whether the customer's order exceeds a monetary threshold so that the minimum order value has been reached.

Accordingly, the goals for our EPN model are as follows: First, we need to immediately indicate to abort the ordering process if the customer has a bad credit rating. Second, we want to immediately identify when the customer is eligible to receive shipping once the (aggregated) value of the order(s) is high enough.

To start with, we require an order event containing the order value. The *OrderEvent* is produced by the order coming into the company's system. To parallelize processing, we split the event into an event containing only the order data (*OrderEvent2*) and an event only containing customer data (*CustomerEvent*) which is later enriched with the customer's credit limit (*EnrCustomerEvent*). If the customer turns out to be not creditworthy, the process can be directly terminated through the *NotCreditWorthyEvent*. The OrderEvent2 itself is aggregated to a sum of the current orders, which is then (re-)composed with the customer's limit and a payment received marker based on unique event IDs. This *EnrPaidOrderSumEvent* is used to calculate whether the order can be shipped out immediately or is delayed until the next day by comparing the customer's limit with the order value. The resulting event (*OrderIndicatorEvent*) provides an indicator which signifies clearance for shipping.

As motivated earlier, modeling should not necessarily be an end to itself. Hence, we have made sure that the models can be – in principle – transformed to query skeletons which can be used for a subsequent implementation. A query skeleton for the Split and Enrich EPA could look like this:

```
// Split
ON OrderEvent
INSERT INTO CustomerEvent SELECT eventID, customerID, customerName
INSERT INTO OrderEvent2 SELECT eventID, orderID, orderValue
OUTPUT ALL

// Enrich
INSERT INTO EnrCustomerEvent
SELECT (eventID, customerID, customerName,
customer.limit as customerLimit, customer.creditrating as customerCreditrating
FROM sql:customer [
  "select limit, creditrating
   from customer where CUSTOMER_ID=${customerID}"],CustomerEvent
```

The query is instantiated with the specified event types and their attributes from the EPN model. Each query block generates an output (`INSERT INTO`) which is picked up by the next block (e.g. CustomerEvent). For more details on the automation of EPN cf. Gabriel, Janiesch (2016). Further EPA of the EPN in the above model would serialize similarly according to the examples giving in Section 3.1.

This scenario could be enhanced using pattern detect constructs or tighter integration with the business process specification of the composite application. For example, the orders could be automatically cancelled if a payment is not received within a given timeframe or the detection of duplicate orders could be modeled. Also, threshold specifications and logical as well as geo-spatial operators are not included in the set of constructs presented here. Also, the integration with the process specification could be expanded so that it becomes clear which parts (i.e. activities) of the process produce and consume events. The current state of work on business process integration and pattern detect constructs for modeling EPN is available upon request but subject to future research.

5 Discussion and Conclusion

In the following, we discuss the adequacy of the developed modeling constructs. We have decided to tackle this with a bottom-up approach from the technical side. For this, we have analyzed the specification of Esper EPL Version 5.1 (EsperTech Inc., 2014) as a representative event processing language. We have extracted its most common and most prominently listed terms and clauses from its processing model and its clause references. Table 2 lists them in the left column. In the right column we show how they can be expressed with the conceptual modeling method for EPN.

Esper EPL Element	Realization in the Conceptual Modeling Method
Filters and `where` clauses	Filters are realized in the Filter EPA, `where` clauses are very general, but usually express conditions. These conditions can also be modeled by using a Filter EPA. Also every EPA includes a filter which can be applied before the actual processing step.
Time windows and time batches	Time windows and batches can be defined by a Filter and hence are also part of every EPA.
Aggregation and grouping	Aggregation can be represented by the Aggregate EPA. Grouping for aggregation operations has to be done by filtering for each group.
`Select` clause	The `select` clause is represented implicitly in the change of events types from input to output of an agent. It is represented in CEP terminology by the Project EPA which is included in the Filter in our conceptual modeling method.
`From` clause	The `from` clause in Esper defines the input stream. In the models, this is represented by the arrow symbolizing the event streams between agents.
Expressions	Expressions are used to define functions on single events. This is represented by the Translate and Enrich EPA.
Joins	Joins from different streams on event level are modeled using the Compose EPA.

Table 2. Comparison of Esper EPL with modeling method.

A clear and direct assignment of model constructs to Esper EPL constructs is not possible, but Esper EPL includes all operators necessary to translate the model into a query in a technical language and vice versa. Some of the language constructs are rather simple to represent, e.g. filters and other condition in an SQL-like statement. Others, such as grouping, have to be assembled with different model elements because it is an SQL-related function which is not necessarily required in CEP.

The output option of Esper is not yet included in the modeling method for EPNs. It enables to define when the output is produced by the CEP engine once an EPA has performed its task (e.g. “every two seconds” instead of instantly). Currently, this can only be added as a textual annotation.

Apart from thus, our notation includes all EPA conceptualizations for filtering and transformation currently present in related work. We have not encountered a filter or transformation query of Esper EPL which cannot be modeled using the abstract and concrete syntax we introduced above. In summary, we have developed a working method as an IT artifact which can be used to specify the processing behavior of EPN conceptually and tool-independent. There may be more challenges ahead when approaching pattern detect agents though. So while this set of constructs provides a promising start, the coverage of actual use cases from practice is limited to rather straight-forward and deterministic analyses which do not require pattern detection.

Since the modeling method is also supposed to be a communication vehicle, we need to test the user acceptance of both, CEP and business experts. As of now, our Visio stencils for modeling EPN have some deficiencies concerning the alignment of arrows which may hamper the understanding by less proficient users. The initial focus was on function, not form. We have also implemented the modeling method in the open source modeling software Oryx (Decker et al., 2008) which for example supports model element docking and alignment as well as model syntax checks (Gabriel and Janiesch, 2016).

Furthermore, we have also taken care to allow for the generation of code from the models. The serialization of modeling constructs into partial CEP queries was exemplified in Esper EPL when introducing each construct in Section 3.1. It is currently been tested towards the ability to automatically derive executable CEP queries (Gabriel and Janiesch, 2016). We have started to implement a model export from Oryx which creates fully executable Esper EPL queries that can be pushed to Esper’s CEP engine for execution. Internally, each EPA results in a query which creates an own event type. This event type is consumed by the next EPA. While we have not had performance issues, these occasionally tiny queries could be optimized into larger queries in the future.

References

Anicic, D., Fodor, P., Rudolph, S., Stühmer, R., Stojanovic, N. and Studer, R. (2010). "A Rule-Based Language for Complex Event Processing and Reasoning," In: *Proceedings of the 4th International Conference on Web Reasoning and Rule Systems (RR). Lecture Notes in Computer Science Vol. 6333*. Ed. by P. Hitzler and T. Lukasiewicz. Bressanone, pp. 42-57.

Appel, S. (2014). "Integration of Event Processing with Service-oriented Architectures and Business Processes," Dissertation. University of Darmstadt, Darmstadt.

Becker, J., Knackstedt, R., Holten, R., Hansmann, H. and Neumann, S. (2001). Konstruktion von Methodiken: Vorschläge für eine begriffliche Grundlegung und domänenspezifische Anwendungsbeispiele. In *Arbeitsberichte des Instituts für Wirtschaftsinformatik*. Vol. 77 Ed. by J. Becker, H. L. Grob, S. Klein, H. Kuchen, U. Müller-Funk and G. Vossen. Münster.

Becker, J. and Pfeiffer, D. (2007). Konzeptionelle Modellierung: Ein wissenschaftstheoretischer Forschungsleitfaden. In *Wissenschaftstheoretische Fundierung und wissenschaftliche Orientierung der Wirtschaftsinformatik* Ed. by F. Lehner and S. Zelewski. Berlin: Gito, pp. 1-17.

Brinkkemper, S. (1996). Method Engineering: Engineering of Information Systems Development Methods and Tools. *Information and Software Technology* 38 (4), 275-280.

Decker, G., Grosskopf, A. and Barros, A. (2007). "A Graphical Notation for Modeling Complex Events in Business Processes," In: *Proceedings of the 11th IEEE International Enterprise Distributed Object Computing Conference (EDOC)*. Annapolis, MD, p. 27.

Decker, G., Overdick, H. and Weske, M. (2008). "Oryx: An Open Modeling Platform for the BPM Community," In: *Proceedings of the 6th International Conference on Business Process Management (BPM). Lecture Notes in Computer Science Vol. 5240*. Ed. by M. Dumas, M. Reichert and M.-C. Shan. Milan, pp. 382-385.

EsperTech Inc. (2014). *Esper Reference. Version 5.1.0*. http://www.espertech.com/esper/release-5.1.0/esper-reference/pdf/esper_reference.pdf (visited on 2015-10-21).

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

Frank, U. (1999). "Conceptual Modelling as the Core of the Information Systems Discipline: Perspectives and Epistemological Challenges," In: *Proceedings of the 5th Americas Conference on Information Systems (AMCIS)*. Ed. by D. W. Haseman, D. L. Nazareth and D. L. Goodhue. Milwaukee, WI, pp. 695-697.

Friedenstab, J.-P., Janiesch, C., Matzner, M. and Müller, O. (2012). "Extending BPMN for Business Activity Monitoring," In: *Proceedings of the 45th Hawai'i International Conference on System Sciences (HICSS)*. Maui, HI, pp. 4158-4167.

Gabriel, S. and Janiesch, C. (2016). "Konzeptionelle Modellierung ausführbarer Event Processing Networks für das Event-driven Business Process Management," In: *Proceedings of the Modellierung 2016. Lecture Note in Informatics Vol. 254*. Ed. by A. Oberweis and R. Reussner. Karlsruhe, pp. 173-180.

Gregor, S. and Hevner, A. R. (2013). Positioning and Presenting Design Science Research for Maximum Impact. *MIS Quarterly*, 37 (2), 337-355.

Hevner, A. R., March, S. T., Park, J. and Ram, S. (2004). Design Science in Information Systems Research. *MIS Quarterly*, 28 (1), 75-105.

Hirschheim, R., Klein, H. K. and Lyytinen, K. (1995). *Information Systems Development and Data Modeling: Conceptual and Philosophical Foundations*. Cambridge: Cambridge University Press.

International Organization for Standardization/ International Electrotechnical Commission (ISO/IEC) (2013). *ISO/IEC 19510:2013. Information Technology -- Object Management Group Business Process Model and Notation*. http://www.iso.org/iso/catalogue_detail.htm?csnumber=62652 (visited on 2015-11-25).

Janiesch, C. (2007). "Contextual Method Design: Constructing Adaptable Modeling Methods for Information Systems Development," Dissertation, Münster.

Janiesch, C., Matzner, M. and Müller, O. (2012). Beyond Process Monitoring: A Proof-of-Concept of Event-driven Business Activity Management. *Business Process Management Journal*, 18 (4), 625-643.

Krumeich, J., Weis, B., Werth, D. and Loos, P. (2014). Event-Driven Business Process Management: Where Are We Now? A Comprehensive Synthesis and Analysis of Literature. *Business Process Management Journal*, 20 (04), 615-633.

Luckham, D. (2002). *The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems*. Boston, MA: Addison-Wesley Professional.

Nuseibeh, B. A., Finkelstein, A. and Kramer, J. (1996). Method Engineering for Multi-perspective Software Development. *Information and Software Technology*, 38 (4), 267-274.

Peffers, K., Tuunanen, T., Rothenberger, M. A. and Chatterjee, S. (2007). A Design Science Research Methodology for Information Systems Research. *Journal of Management Information Systems*, 24 (3), 45-77.

Saeki, M. (1995). "Object-Oriented Meta Modelling," In: *Proceedings of the 14th International Conference on Object-Oriented and Entity-Relationship Modellling (OOER). Lecture Notes in Computer Science Vol. 1021*. Ed. by M. Papazoglou. Gold Coast, pp. 250-259.

Sharon, G. and Etzion, O. (2008). Event-processing Network Model and Implementation. *IBM Systems Journal*, 47 (2), 321-334.

Simon, H. A. (1996). *The Sciences of the Artificial*. 3rd Edition. Cambridge, MA: MIT Press.

Strahringer, S. (1996). "Metamodellierung als Instrument des Methodenvergleichs: Eine Evaluierung am Beispiel objektorientierter Analysemethoden," Dissertation. University of Shaker, Aachen.

Vidačković, K. (2014). "Eine Methode zur Entwicklung dynamischer Geschäftsprozesse auf Basis von Ereignisverarbeitung," Dissertation. University of Stuttgart, Stuttgart.

Wand, Y. and Weber, R. (2002). Research Commentary: Information Systems and Conceptual Modeling: A Research Agenda. *Information Systems Research*, 13 (4), 363-376.