A Graphical Notation for Modeling Complex Events in Business Processes

Gero Decker, Alexander Grosskopf
Hasso-Plattner-Institute
Potsdam, Germany
(gero.decker,alexander.grosskopf)@hpi.uni-potsdam.de

Alistair Barros
SAP Research Centre
Brisbane, Australia
alistair.barros@sap.com

Abstract

Using complex event rules for capturing dependencies between business processes is an emerging trend in enterprise information systems. In previous work we have identified a set of requirements for event extensions for business process modeling languages. This paper introduces a graphical language for modeling composite events in business processes, namely BEMN, that fulfills all these requirements. These include event conjunction, disjunction and inhibition as well as cardinality of events whose graphical expression can be factored into flow-oriented process modeling and event rule modeling. Formal semantics for the language are provided.

1 Introduction

Business process modeling languages have a long tradition of representing coordination requirements of enterprises through control and data flow dependencies. Graphical constructs for sequences, conditional branching, parallelism, synchronization and iteration have a natural suitability for capturing the dependencies of activities especially inside organizational boundaries. With requirements for coordination extending across collaborating business partners, process languages have evolved to support asynchronous dependencies between inter-operating processes, primarily through message exchanges. It is not uncommon for process languages nowadays to support more generally asynchronous occurrences, or events, considering widely adopted languages such as Business Process Modeling Notation (BPMN) [2], UML Activity Diagrams [1] and Business Process Execution Language (BPEL) [3].

Events in a process model express triggering dependencies with other process models, not only as part of its initiation and termination, but at other stages of its execution. The trigger for the initiation of a process is an event consumed by the process, while the termination of a process yields an event produced by the process, allowing it to be consumed outside. Execution of intermediate stages of a process can similarly involve event consumption and production.

With trends towards coupling business process with real-world aware applications using sensor networks and physical device tracking (radio frequency tags) notably, limitations of the current event handling in process languages are apparent. Take an example of a manufacturing process where faults need to be escalated for timely resolution. These are normally collected and separately processed through coded logic, however the different levels of event monitoring and action escalation could be integrated into processes coordinating relevant business activity. Thus, consumption of events at different time periods triggering sets of activities could be made more visible. So too could critical faults that override non-critical faults, escalating one level of monitoring and action over another. Such scenarios featuring composite events and in conjunction, disjunction and inhibition relations cannot directly supported through the simple event handling in current process languages.

In parallel to the evolution of process definition languages in the field of Business Process Management, complex event processing has developed through Active Databases, Message Brokering and Message-Oriented Middleware. For general description of complex events in these fields, event pattern languages such as Rapide [11] and Snoop [6] were introduced. However, only textual representations are available for event languages leaving open how they can be suitably expressed in business process languages. Without this in place, business processes with limited event handling are difficult to align well with complex event specifications, with the result of specification stove-pipes.

In order to allow a seamless integration of event pat-

terns into graphical process modeling languages, we propose a graphical notation for expressing event patterns, namely BEMN, the Business Event Modeling Notation. Support of complex events has been inspired from our previous work in developing event patterns [5]. These expressed common eventing scenarios in business processes and an assessment of BPMN and BPEL showed that only few of these scenarios are supported.

The remainder of this paper is structured as follows. Section 3 discusses related work, before section 4 provides an overview over BEMN. Section 5 introduces formal semantics and section 6 an assessment of BEMN regarding its suitability for expressing real-world eventing scenarios. Finally, section 7 concludes and gives an outlook to future work.

2 Event Consumption in BPMN and BPEL

Important eventing concepts like subscription to events, matching of events and finally the consumption of events is present in flow-oriented modeling languages. E.g. the Business Process Execution Language (BPEL [3]) However, these languages can only capture simple eventing scenarios as shown in [5] where typically only individual events are considered.

The Business Process Execution Language (BPEL [3]) and BPMN include the notion of events. E.g. in BPEL `invoke`, `receive` and `onMessage` activities specify production and consumption of message events. An initial `receive` and `onMessage` activity with the attribute `createInstance` set to `yes` defines a WSDL port type / operation combination that is relevant for the instantiation of BPEL processes: As soon as a message of that particular combination arrives, a process instance is created.

We can find three typical steps for the consumption of events in BPEL: i) A *subscription* to events (e.g. incoming messages) is initiated. Subscriptions for those messages that lead to process instantiation are initiated at deployment-time of a process definition. Subscriptions for those messages that are consumed by a running process instance are normally initiated as soon as the respective `receive` or `onMessage` activity is reached. BPEL provides correlation mechanisms, where different process instances can subscribe to message of the same type (port type / operation). ii) An *event occurs*, i.e. a message arrives or a timeout occur. iii) The *event is matched* by a subscription. This determines that the event will be consumed by a particular process instance. Either it is consumed by an already existing process instance or a new instance is created as result of that event. In the former case, the BPEL engine checks the correlation info included in the message and routes it to the corresponding instance.

As a forth step we could also consider unsubscription. If an event is not awaited any longer the subscription is taken back. A typical scenario could be that at a given moment in the process a message of one out of a set of different types could be consumed. In this case, there is a subscription for every type and as soon as one message arrives no message of the other types is waited for.

BPEL only considers individual events: It is checked on a per-message basis if a message matches a registered subscription (based on the port type, operation and correlation sets) and only one message is consumed in a `receive` or `onMessage` activity. This is different to what is demanded e.g. by the event patterns Conjunction or Inhibition [5]. In the case of Conjunction, different events have to be present for matching taking place. This includes atomicity: All or none of the events are consumed. Inhibition checks the absence of other events, which is also not supported in BPEL.

BPMN allows more event types than BPEL: In addition to messages, timeouts and exceptions that are also present in BPEL, BPMN also comes with rule events and it even allows to extend the language with custom defined event types. Since BPMN is basically a graphical notation without defined execution semantics, it is unclear how and when subscription for events is handled in BPMN. Anyway the BPMN assumes that all messages are persistent. Thus they are kept until a process instance is ready to consume them. As it is the case for BPEL, BPMN distinguishes start events and intermediate events as two kinds of event consumption: start events lead to process instantiation and intermediate events are consumed by a running process instance. BPMN also allows to specify message consumption and production through send and receive activities. A more in-depth assessment of BPMN and BPEL can be found in [5].

3 Related Work

The field of Complex Event Processing (CEP) comes with a set of languages and architectures for describing and efficiently executing complex event rules. A good reference for CEP is the book by Luckham [11], where he also introduces Rapide, an event pattern language. A framework for detecting complex event patterns can be found e.g. in [12]. Considerable work on event pattern languages can also be found in the field of active databases. In [6] the event algebra Snoop is introduced and compared with other event languages. However, these languages only come with textual representations,

therefore being unsuited for process modelers who are used to graphical languages.

Nevertheless, there have been several proposals for using event rules in the business process domain [8, 7, 9], especially for exception handling [4]. Flow-oriented modeling languages and event rule languages are compared in [10] regarding their control flow capabilities. Strengths and weaknesses are identified along the five dimensions expressibility, flexibility, adaptability, dynamism and complexity.

Flow-oriented business process modeling languages have a long tradition in the field of Business Process Management (BPM). The Business Process Modeling Notation (BPMN [2]) is considered the de-facto standard. It comes with notational elements for concepts such as declarations for event consumption and event production, repetitions and cancellation. These concepts are also present in event rules and therefore notational elements can be reused for allowing a seamless integration of the flow-oriented and the event-rule-based modeling paradigm for business processes.

4 Language Overview

4.1 Language Constructs

The Business Event Modeling Notation (BEMN) serves to define event rules. Each rule consists of a event pattern description, specifying combinations of events that can be matched, as well as a set of output event declarations indicating what kind of events are to be produced as a result of the firing of a rule. Figure 1 shows all language constructs of BEMN.

Input event declarations specify events to be observed. Precedence relationships specify a time constraint between two objects: the source object must have occurred before the target object. Inhibition relationships specify that the source object must not have occurred before the target object. OR operators and AND operators logically relate incoming relationships. As an example imagine two input event declarations A and B preceding an OR operator which in turn precedes a third input event declaration C. A or B need a corresponding event preceding a corresponding event for C in order for the pattern to be matched. Also if there is a corresponding event for both A and B as well as for C, the pattern would match.

Groupings are sets of objects with additional constraints. Filters attached to groupings provide constraints for the events corresponding to the declarations contained in the grouping. There are different types of filters: time-related, process-data-related, environment-data-related and other filters. Each filter comes with an expression describing the constraint. Filters can also be directly attached to individual event declarations, which is an abbreviation for a grouping only containing one object. Furthermore, groups can be typed as repetitions. Finally, input event declarations can be placed at group boundaries ("exception event declarations"). In this case, a corresponding event serves as an inhibitor for the grouping.

Output event declarations specify the events to be produced upon firing of a rule.

4.2 Examples

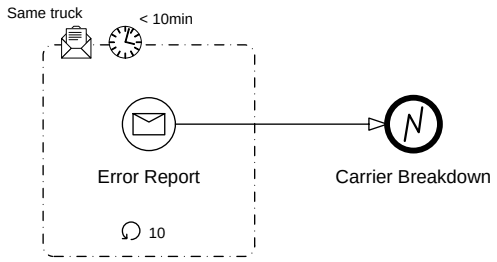

Figure 2. Example 1: Carrier breakdown

Figure 2 shows a BEMN example. Error reports of trucks are to be collected. However, immediate action is only needed as soon as a threshold is reached. In this case 10 error reports of the same truck have to appear within 10 minutes, resulting in the production of a carrier breakdown event that in turn can be used as process instantiation or cancellation trigger in a process model.

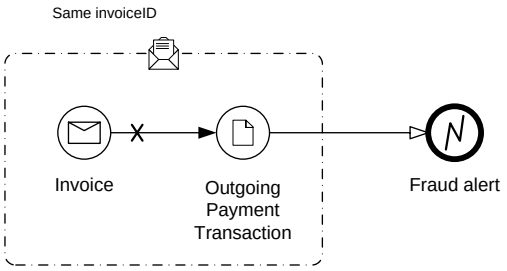

Figure 3. Example 2: Fraud alert

Figure 3 shows a simple fraud alert scenario. Every time an outgoing payment transaction happens without having received a corresponding invoice before, the alert is raised.

BEMN can both be used in stand-alone diagrams but is also supposed to be integrated into BPMN process models. Figure 4 illustrates how flow-oriented modeling and event rule modeling go hand in hand. In this example, an order request is processed involving a logistic

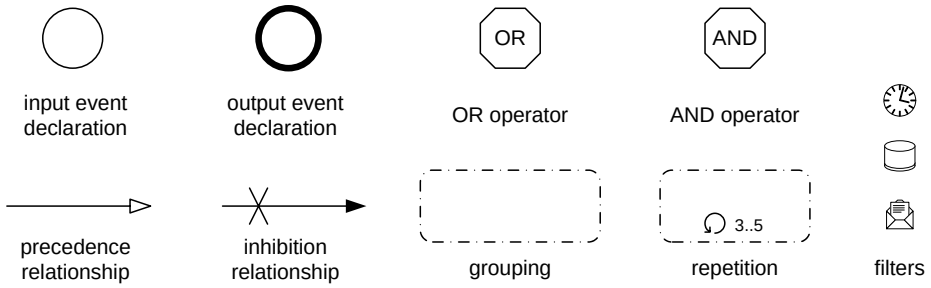

Figure 1. BEMN constructs

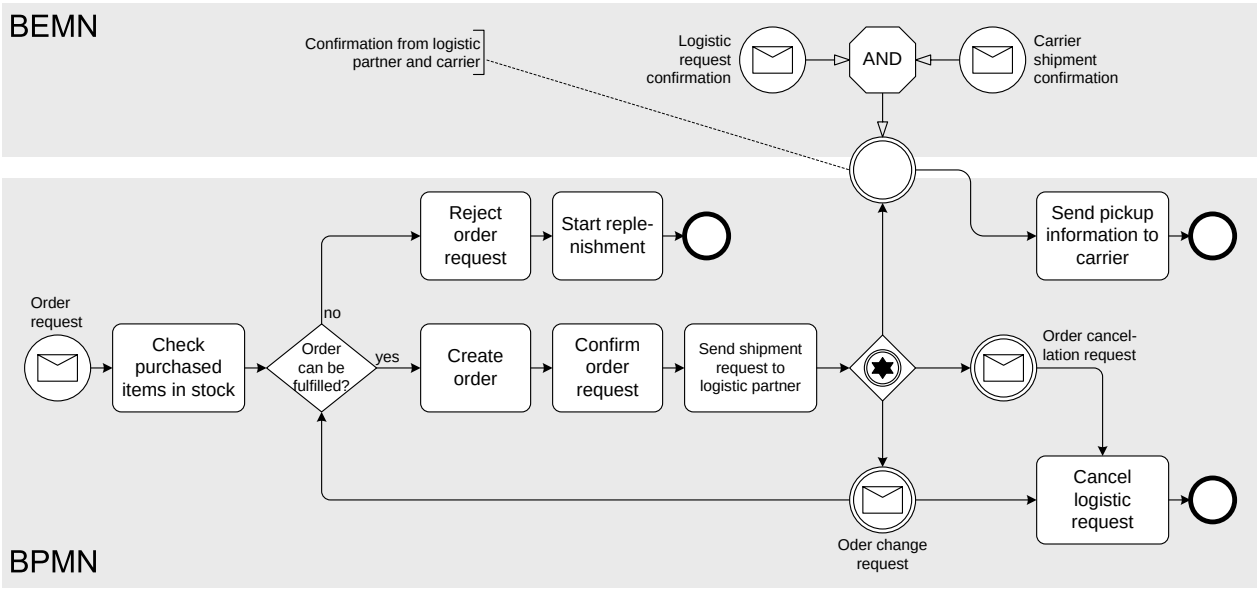

Figure 4. BPMN diagram enhanced with event conjunction expressed in BEMN

partner and a carrier. Based on the current purchased items in stock the order can be fulfilled or not. If not the order request is simply rejected and replenishment is triggered. If the order can be fulfilled the request is confirmed and a shipment request is sent to a logistic partner. Now three events can happen: 1) An order change request comes in, leading to the cancellation of the logistic request and a re-evaluation whether the order can be fulfilled. 2) An order cancellation request arrives, also leading to the cancellation of the logistic request. 3) Both confirmations from the logistic partner and the actual carrier which was selected by the logistic partner are available. Only in this third case the pickup information is sent to the carrier.

In this example we see the occurrence of event conjunction (cf. [5]). Two elementary events form a composite event which in turn is consumed in the business process. Having the possibility to specify such event patterns as integral parts of process models dramatically reduces the complexity of such models.

When modeling the same scenario using classical BPMN, events could only be consumed one at a time. In this particular example we see that the composite event declaration direct follows an event-based XOR-gateway. This gateway indicates a racing condition between the three event declarations.This makes the motivation for atomicity of composite events obvious. When expanding the same semantics into a pure BPMN diagram, the number of elements in the diagram would be significantly higher.

5 Formal Semantics

This section is going to define an abstract syntax for BEMN models and give these models a formal execution semantics. As part of that, *core event composition models* will be introduced as a special class of BEMN models. The idea behind is that the semantics of such

a model can be directly given and that core models represent the unit of execution: Matching and firing of a core model will result in a single transactional step, while firing of non-core models might involve several steps. Non-core models therefore represent a set of core models. An algorithm will be provided for this transformation.

5.1 Abstract Syntax

An event composition model EC is a tuple $EC = (O, ED, Op, ED^I, ED^O, Op^A, Op^O, Gr, gt, gc_{min}, gc_{max}, E_{Gr}, cons, Pr, Inh, F, F_{Gr})$, where

- O is a set of objects which can be partitioned into disjoint sets of event declarations ED and operators Op,
- ED can be partitioned into disjoint sets of input event declarations ED^I and output event declarations ED^O,
- Op can be partitioned into disjoint sets of AND operators Op^A and OR operators Op^O,
- $Gr \subseteq \wp(O)$ is a set of sets of objects, so called groupings,
- $gt : Gr \to \{r, n\}$ is a function specifying the type of a grouping: repeated or normal,
- $gc_{min}, gc_{max} : Gr \to \mathbb{N}$ are two functions specifying the cardinality of a repeated grouping (minimum number and maximum number),
- $ED_{Gr} : ED^I \to Gr$ is a partial function linking an input event declaration to a grouping, hence making it an exceptional event declaration for that grouping,
- $cons : ED^I \to \{once, always\}$ is a function determining whether an event should be consumed once or potentially multiple times,
- $Pr \subseteq O \times O$ is the precedence relation on objects,
- $Inh \subseteq O \times O$ is the inhibition relation on objects,
- F is a set of filters and
- $F_{Gr} : F \to Gr$ is a function linking a filter to a grouping.

We introduce the auxiliary functions $in_{Pr}, in_{Inh}, in, out_{Pr}, out_{Inh}, out : O \to \wp(O)$, where $in_{Pr}(o) := \{x \in O \mid x \ Pr \ o\}$ and $out_{Pr}(o) := \{x \in O \mid o \ Pr \ x\}$, in_{Inh} and out_{Inh} analogously and $in := in_{Pr} \cup in_{Inh}$, $out := out_{Pr} \cup out_{Inh}$. The constraints below are assumed to be satisfied by any BEMN model:

- every input event declaration that is not attached to a grouping has at least 1 outgoing relationship and has a path of Pr and Inh relationships to an output event declaration, i.e. $\forall e \in ED^I \setminus dom(ED_{Gr}) \ [|out(e)| \geq 1 \wedge \exists o \in ED^O \ (e(Pr \cup Inh)^+ o)]$,
- every output event declaration has at least 1 incoming and exactly 0 outgoing relationships, i.e. $\forall e \in ED^O \ [|in(e)| \geq 1 \wedge |out(e)| = 0]$,
- every operator has at least 1 incoming and 1 outgoing relationship, i.e. $\forall o \in Op \ [|in(o)| \geq 1 \wedge |out(o)| \geq 1]$,
- in each grouping there is at most one object that has an incoming relationship with a source outside the grouping and at most one object that has an outgoing relationship with a target outside the grouping, i.e. $\forall g \in Gr \ [|\{o \in g \mid in(o) \setminus g \neq \emptyset\}| \leq 1 \wedge |\{o \in g \mid out(o) \setminus g \neq \emptyset\}| \leq 1]$,
- if two groupings contain the same objects then one grouping must be fully contained in the other, i.e. $\forall g_1, g_2 \in Gr \ [g_1 \cap g_2 \neq \emptyset \Rightarrow g_1 \subset g_2 \vee g_2 \subset g_1]$,
- the minimum cardinality of a grouping must be at least 1 and the maximum cardinality must be greater or equal the minimum cardinality, i.e. $\forall g \in Gr \ [1 \leq gc_{min} \leq gc_{max}]$,
- an input event linked to a grouping must not have incoming relationships, i.e. $\forall e \in dom(ED_{Gr}) \ [|in(e)| = 0]$,
- Pr and Inh are disjoint, i.e. $Pr \cap Inh = \emptyset$,
- Pr is acyclic,
- source and target of an Inh are not the same object,
- for every output event declaration there must be at least one input event declaration with a path of Pr relationships to that declaration, i.e. $\forall e \in ED^O \ [\exists e_2 \in ED^I \ (e_2 \ Pr^+ \ e)]$,
- for every input declaration there exists a path of Pr and Inh relationships containing at most one Inh relationship to any following output declaration, i.e. $\forall e \in ED^I, o \in ED^O \ [e(Pr \cup Inh)^+ o \Rightarrow e \ Pr^+ \ o \vee \exists (e_1, e_2) \in Inh \ (e \ Pr^* \ e_1 \wedge e_2 \ Pr^* \ o)]$ and
- for every input declaration contained in a grouping with an event declaration attached to it and for every event declaration attached to a grouping there exists a path of Pr relationships to any following output declaration, i.e. $\forall e \in ED^I, o \in ED^O \ [e(Pr \cup Inh)^+ o \wedge (\exists g \in G \ (e \in g \wedge g \in range(ED_{Gr})) \vee e \in dom(ED_{Gr})) \Rightarrow e \ Pr^+ \ o]$.

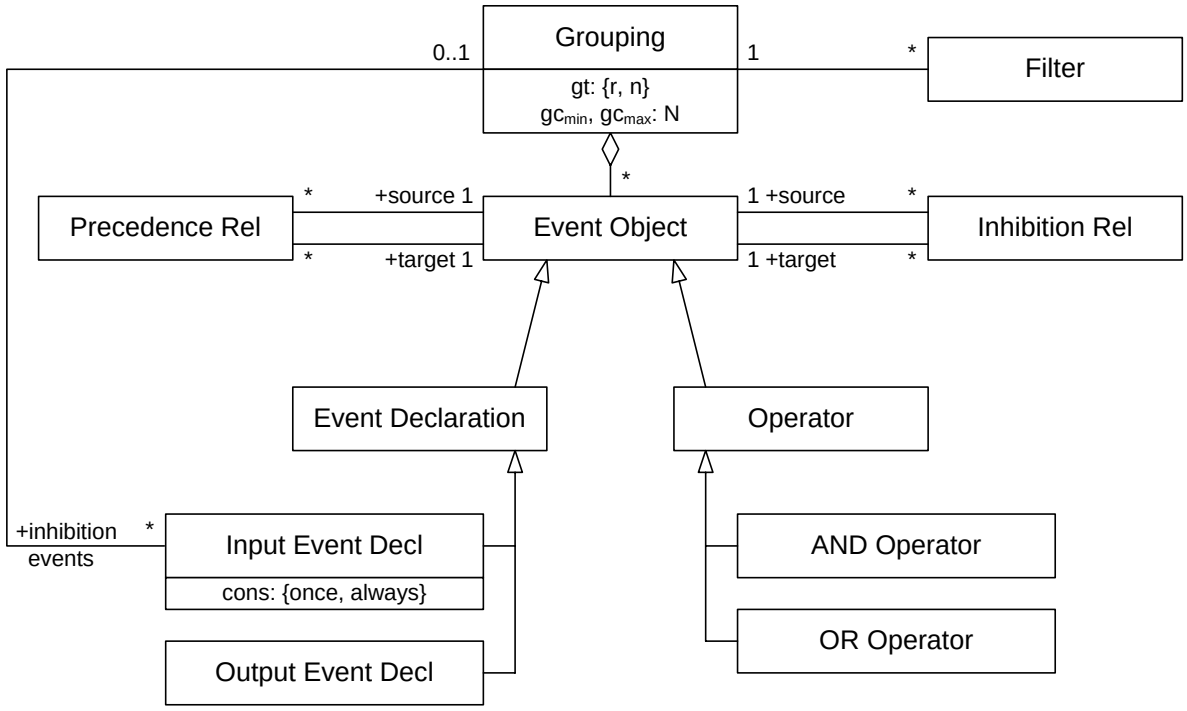

Figure 5. BEMN meta-model

5.2 Core event composition models

A core model is a BEMN model where the following constraints hold:

- every input event declaration has at most 1 incoming and exactly 1 outgoing relationship, i.e. $\forall e \in ED^I\ [|in(e)| \leq 1 \wedge |out(e)| = 1]$,
- every output event declaration has exactly 1 incoming relationship, i.e. $\forall e \in ED^O\ [|in(e)| = 1]$,
- there is only 1 end event declaration, i.e. $|ED^O| = 1$,
- there is no OR operator, i.e. $Op^O = \emptyset$,
- no intermediate event is linked to a grouping, i.e. $ED_{Gr} = \emptyset$ and
- no grouping is of type repetition, i.e. $\forall g \in Gr\ [gt(g) = n]$.

We introduce the auxiliary functions *med* and *ied* assigning a set of input event declarations to an event composition model EC, where we call $med(ec) := \{e \in ED^I \mid \exists o \in ED^O\ (e\ Pr^+\ o)\}$ the match declarations and $ied(EC) := ED^I \setminus med(EC)$ the inhibitor declarations for EC.

5.3 Semantics of Core Event Composition Models

During execution we deal with a stream of events where event rules can be matched and lead to firing, hence the production of new events. Firing happens atomically. We introduce the notion of subscriptions for enabling event rules, i.e. only if a subscription to a certain rule exists, this rule might fire. Subscriptions are subject to so called subscription scopes. In the process-oriented world, a scope would typically be a process instance or an activity instance, but alternatively it could also be a group of process instances or even the whole system. Scopes are important for the cardinality of consumption of individual events. Events that correspond to match declarations might be used in one firing per scope or an arbitrary number of times. E.g. an incoming order request should only be processed once, while a product out of stock event might be consumed multiple times as context information.

We formally represent our execution environment as follows:

- E is a set of events,
- EM is a set of event composition models,
- Sub is a set of subscriptions,

- Sc is a set of subscription scopes, where $global \in Sc$ is the global scope,
- $sub : Sub \rightarrow Sc \times EM$ is a function linking a subscription to a scope and an event model,
- $< \subset (E \cup Sub \cup \{now\} \cup Time)^2$ is a relation fully temporally ordering events, subscriptions, the current time and any other point in time and
- $Consumed \subseteq Sc \times E$ is a relation indicating that an event was already consumed for a subscription scope.

A subscription scope could be a process instance, an activity instance or a group of process instances. We introduce a boolean function $applies : F \times \wp(E) \rightarrow \{true, false\}$ indicating whether the criteria of a given filter applies to the given set of events.

A set of events M can lead to firing of a subscription $s \in Sub$ with its corresponding BEMN model and scope $(sc, em) = sub(s)$ iff there is a bijective function $map_M : med(em) \rightarrow M$, the match condition $C_M(sc, em, map_M)$ is fulfilled and there does not exist an injective function $map_I : ied(em) \rightarrow (E \setminus M)$ such that the inhibition condition $C_I(sc, em, map_I \cup map_M)$ is fulfilled. As a result of the firing an event of the type given in the output event declaration of em is produced and all those events are marked as consumed for sc the corresponding input event declaration of which is defined as *once*.

The match and inhibition conditions are defined as follows:

$$
\begin{aligned}
&C_M(sc, em, map_M) := \forall ed \in range(map_M) \\
&\quad [cons(ed) = once \Rightarrow \\
&\quad\ (sc, map_M(ed)) \notin Consumed] \wedge \\
&\quad \forall (ed_1, ed_2) \in Pr^+ \ [\{ed_1, ed_2\} \subseteq med(sc) \Rightarrow \\
&\quad\ map_M(ed_1) < map_M(ed_2)] \wedge \\
&\quad \forall f \in F \ [C_F(em, f, map_M)]
\end{aligned}
$$

$$
\begin{aligned}
&C_I(sc, em, map) := ied(em) \neq \emptyset \wedge \\
&\quad \exists (o_1, o_2) \in Inh \ (\forall ed_1 \in prec(o_1), ed_2 \in succ(o_2) \\
&\quad\ [map(ed_1) < map(ed_2)] \wedge \\
&\quad\ \forall (ed_1, ed_2) \in Pr^+ \ [ed_2 \in ipr(o_1, em) \Rightarrow \\
&\quad\ map(ed_1) < map(ed_2)] \wedge \\
&\quad\ \forall f \in F \ [C_F(em, f, map_M \cup \\
&\quad\ (map_I \cap (ipr(o_1, em) \times E)))])
\end{aligned}
$$

$$
\begin{aligned}
&C_F(em, f, map) := \\
&\quad applies(f, range(map \cap (F_{Gr}(f) \times E))) \\
&prec(o) := \{ed \in ED^I \mid \\
&\quad ed \ Pr^* o \ \wedge \neg \exists ed' \in ED^I \ (ed \ Pr^+ \ ed' \cap ed' \ Pr^+ \ o)\} \\
&succ(o) := \{ed \in ED^I \mid \\
&\quad o \ Pr^* ed \ \wedge \neg \exists ed' \in ED^I \ (o \ Pr^+ \ ed' \cap ed' \ Pr^+ \ ed)\} \\
&ipr(o, em) := \{ed \in ied(em) \mid \\
&\quad \exists ed_2 \in prec(o) \ (ed \ Pr^* \ ed_2)\}
\end{aligned}
$$

5.4 Example

Figure 6 shows a sample core model em. Its formal representation is

- $O = \{a, b, c, i, op, x\}$, $ED = \{a, b, c, i, x\}$, $ED^I = \{a, b, c, i\}$, $ED^O = \{x\}$, $Op^A = \{op\}$, $Op^O = \emptyset$,
- $Gr = \{\{a\}, \{b\}, \{c\}, \{i\}\}$, $gt = Gr \times \{n\}$, $E_{Gr} = \emptyset$,
- $cons = ED^I \times \{once\}$,
- $Pr = \{(a, op), (b, op), (op, c), (c, x)\}$,
- $Inh = \{(i, b)\}$,
- $F = \{A, B, C, I\}$ and $F_{Gr} = \{(A, \{a\}), (B, \{b\}), (C, \{c\}), (I, \{i\})\}$.

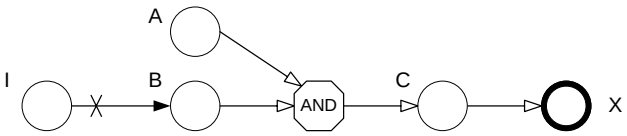

Figure 6. Sample core model

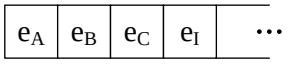

Figure 7. Sample event stream

We see that the event types A, B, C, I are actually represented by filters for the individual event declarations.

Now imagine a stream of events $[e_A, e_B, e_C, e_I]$ (see Figure 7), a subscription s and a subscription scope sc, where $sub(s) = (sc, em)$. The events e_A, e_B, e_C, e_I are of types A, B, C, I, i.e. $applies(A, \{e_A\}) = true$ etc. None of the events has been consumed yet for the subscription scope sc, i.e. $(sc, e_A) \notin Consumed$ etc. Following the order in the event stream, we know that $e_A < e_B$ and $e_B < e_C$ etc. Furthermore, we see that the match declarations are $med(em) = \{a, b, c\}$ and the inhibitor declarations are $ied(em) = \{i\}$. The function assigning events to match declarations would be $map_M = \{(a, e_A), (b, e_B), (c, e_C)\}$.

The match condition C_M is fulfilled as e_A, e_B and e_C have not been consumed yet for sc, the Precedence constraints are not violated, i.e. $e_A < e_C$ and $e_B < e_C$, and all filter criteria are satisfied.

The only possible map_I is $\{(i, e_I)\}$. When looking at the tuple $(o_1 = i, o_2 = b) \in Inh$ we see that the inhibition condition C_I is not fulfilled as $e_I < e_B$ is not the case.

5.5 Translating Non-core Models into Core Models

We already defined the execution semantics for core models. The semantics of non-core models is given through a translation to a set of core models. Figure 8 illustrates these translation rules:

1. repetitions are resolved through the duplication of the structure contained in the grouping,
2. in the case of more than one outgoing or incoming relationship AND operators are introduced,
3. OR operators with more than one outgoing relationship are replaced by a combination of OR and AND operators,
4. OR operators are resolved by creating several event composition models,
5. several output event declarations are resolved through the creation of several event composition models,
6. exceptional event declarations attached to groupings are resolved by introducing AND operators in combination with Inh relationships.

Especially the proposed translation of cardinality ranges for repeated groupings leads to a huge number of core models. However, optimization in implementations is easily possible through the introduction of pattern matching counters for each subscription.

6 Validation

In previous work [5] we have presented a catalog of typical eventing scenarios in business processes. We are going to investigate which of these scenarios are supported in BEMN.

In the case of *Event Conjunction* two or more events have to have occurred in order to be matched. This is directly supported in BEMN through the AND operator. If there are alternatives of events that have to have occurred in order to be matched we deal with *Event Disjunction*. This is supported through the OR operator. For *Event Cardinality* a specified number of events have to have occurred in order to be matched. This could either be a fixed number of events or a range of numbers. The fixed number case is supported through repeated groupings where the minimum and maximum cardinality are equal. The other case where a range of numbers is given where the maximum number is higher as the minimum number. In the case of *Inhibiting Event* an event can only be matched in the absence of another specified event. This is supported in BEMN through inhibition relationships.

For *Event – Event Time Relation* two events can only be matched if their occurrence happens within or outside a given timeframe. This is supported through groupings with a time filter attached to it. The special case of this pattern, where one event always has to have occurred before the other is supported through precedence relationships. *Event – Subscription Time Relation* is also supported through groupings with a time filter attached to it. However, as opposed to the previous pattern, the filtering function will not compare the timestamps of pairs of events but rather compare timestamps of events with the timestamp of the subscription. In the case of *Event – Consumption Time Relation* the events' timestamps are compared to *now*, which is also included in the $<$ relation. *Event – Absolute Time Relation* demands that an event can only be matched if it occurs before or after an absolute point in time. Absolute points in time are included in the $<$ relation, too.

For *Event – Event Data Dependency* two events can only be matched if their data is in a specified relation. This is supported through groupings with data dependencies attached to it. In the case of *Event – Process Instance Data Dependency* an event can only be matched if its data is in a specified relation to the control data of the subscribing process instance. In analogy *Event – Environment Data Dependency* demands that the event's data has to be in a specific relation to environment data. These two patterns are covered in corresponding data filters.

The two Consumption Patterns *Consume Once* and *Consume Multiple* are supported through the attributes *once* and *always* as well as through the subscription scopes. If an event e is to be consumed exactly once, then the corresponding event declaration ed must have $cons(ed) = once$ and the subscription scope must be set to *global*. For *Consume Multiple* either *always* is set for the event declaration or for every consumption a different subscription scope is chosen.

We have implemented a graphical editor based on the GMF framework[1] for an important subset of BEMN. This tool also implements the transformation of non-core to core models and of core models to logical ex-

[1] See `http://www.eclipse.org/gmf/`

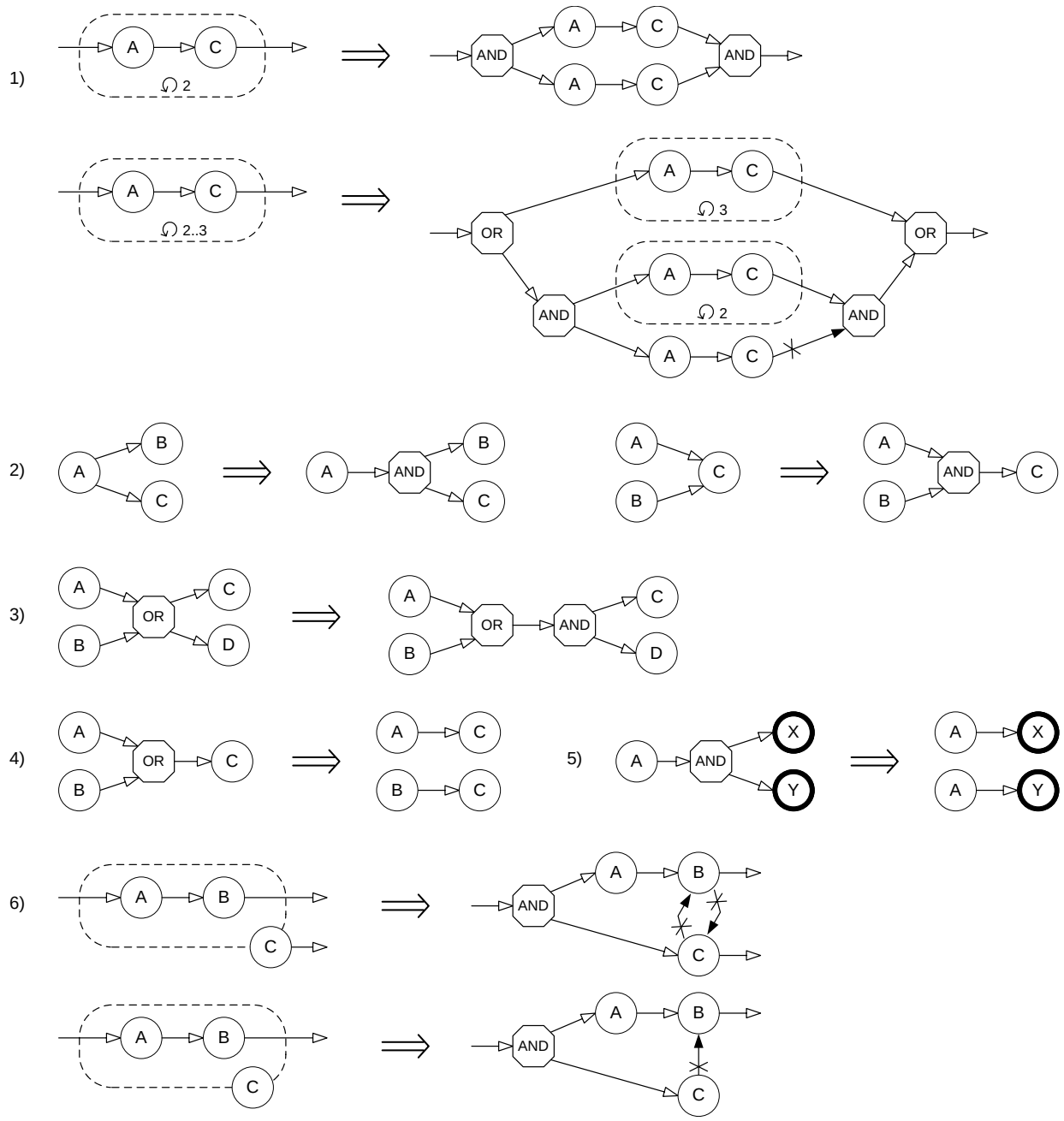

Figure 8. Rules for translating non-core models into core models

pressions were implemented.

We have validated the notational elements for the language through a user survey involving practitioners from the Business Process Management domain. In this survey alternative notational elements were presented to the users and they selected those that were most intuitive in their opinion for representing different eventing scenarios.

7 Conclusion

In this paper we have presented a new language for describing complex event patterns, namely the Business Event Modeling Notation (BEMN). A graphical notation was introduced and the language was given a formal semantics. The semantics was defined in two steps: (i) A mapping from core event models to logical expressions was provided. (ii) Non-core event models

have to be transformed into a set of core event models. Mapping rules for such a transformation were provided.

We have used a previously defined requirements framework for assessing the suitability of the language for expressing common eventing scenarios in business processes. BEMN supports most of the requirements and therefore has a high suitability for real-world use cases. Further validation was provided through prototypical implementation and a user survey concerning the notational elements.

As flow-oriented process modeling and event pattern modeling might be alternatively used to capture the same scenario, it has to be further studied in what particular situation what paradigm is more suited. Furthermore, the joint usage of both paradigms requires combined model verification and management techniques.

References

[1] UML 2.0 Superstructure Specification. Technical report, Object Management Group (OMG), August 2005.

[2] Business Process Modeling Notation (BPMN) Specification, Final Adopted Specification. Technical report, Object Management Group (OMG), February 2006. `http://www.bpmn.org/`.

[3] T. Andrews, F. Curbera, H. Dholakia, Y. Goland, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, S. Thatte, I. Trickovic, and S. Weerawarana. Business Process Execution Language for Web Services, version 1.1. Technical report, OASIS, May 2003. `http://www-106.ibm.com/developerworks/webservices/library/ws-bpel`.

[4] J. Bae, H. Bae, S.-H. Kang, and Y. Kim. Automatic control of workflow processes using eca rules. *IEEE Transactions on Knowledge and Data Engineering*, 16(8):1010–1023, 2004.

[5] A. Barros, G. Decker, and A. Grosskopf. Complex Events in Business Processes. In *Proceedings 10th International Conference on Business Information Systems (BIS)*, number 4439 in LNCS, pages 29–40, Poznan, Poland, April 2007. Springer.

[6] S. Chakravarthy and D. Mishra. Snoop: An expressive event specification language for active databases. *Data Knowledge Engineering*, 14(1):1–26, 1994.

[7] G. Kappel, S. Rausch-Schott, and W. Retschitzegger. Coordination in workflow management systems - a rule-based approach. In *Coordination Technology for Collaborative Applications - Organizations, Processes, and Agents [ASIAN 1996 Workshop]*, pages 99–120, London, UK, 1998. Springer-Verlag.

[8] G. Kappel, S. Rausch-Schott, and W. Retschitzegger. A framework for workflow management systems based on objects, rules and roles. *ACM Comput. Surv.*, 32(1es):27, 2000.

[9] G. Knolmayer, R. Endl, and M. Pfahrer. Modeling processes and workflows by business rules. In *Business Process Management, Models, Techniques, and Empirical Studies*, pages 16–29, London, UK, 2000. Springer-Verlag.

[10] R. Lu and S. Sadiq. A Survey of Comparative Business Process Modeling Approaches. In *Proceedings 10th International Conference on Business Information Systems (BIS)*, number 4439 in LNCS, pages 82–94, Poznan, Poland, April 2007. Springer.

[11] D. Luckham. *The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems*. Addison-Wesley, 2001.

[12] P. R. Pietzuch, B. Shand, and J. Bacon. A Framework for Event Composition in Distributed Systems. In *Proceedings of the 4th International Conference on Middleware (MW'03)*, Rio de Janeiro, Brazil, 2003.