Complex Events in Business Processes

Alistair Barros[1], Gero Decker[2], Alexander Grosskopf[1]

[1] SAP Research Centre Brisbane, Australia
(alistair.barros,alexander.grosskopf)@sap.com
[2] Hasso-Plattner-Institute, Potsdam, Germany
gero.decker@hpi.uni-potsdam.de

Abstract. Flow-oriented process modeling languages have a long tradition in the area of Business Process Management and are widely used for capturing activities with their behavioral and data dependencies. Individual events were introduced for triggering process instantiation and activities. However, real-world business cases drive the need for also covering complex event patterns as they are known in the field of Complex Event Processing. Therefore, this paper puts forward a catalog of requirements for handling complex events in process models, which can be used as reference framework for assessing process definition languages and systems. An assessment of BPEL and BPMN is provided.

1 Introduction

In order to flexibly adapt changing business requirements, companies are in need of IT systems that allow rapid reconfiguration. Business Process Management (BPM) puts process models into the center of attention capturing the activities that have to be carried out as well as their behavioral and data dependencies. The functionality of the available IT systems is invoked by process execution engines, which turns process models into central configuration artifacts for the enterprise systems.

Over the past years different process definition languages have been proposed. They tackle different levels of detail ranging from high-level models for business analysts to executable process models. Prominent examples are e.g. the Business Process Modeling Notation (BPMN [2]), UML 2.0 Activity Diagrams ([1]) and the Business Process Execution Language (BPEL [3]). All of these languages incorporate the notion of events for triggering process instantiation or steps within a process instance. Events in the form of message exchanges or timeouts are very common in executable languages. In the case of higher-level modeling languages there is the possibility to also consider coarser grained business events, such as "goods have arrived".

Events are a way to loosely interconnect different process instances: Events produced in one process instance are consumed by one or several other process instances. Furthermore, composite events, i.e. the combination of different interrelated events, must be handled in process models, too. As activities can normally be decomposed into flows of sub-activities, we also need the possibility to handle

different levels of events in process models. As an example the high-level event "goods have arrived" might be decomposed into a number of "line item has been stored in warehouse" events.

Process definition languages have been benchmarked for their suitability regarding common scenarios in processes. However, a benchmark regarding the eventing capabilities of process definition languages is still missing. Therefore, this paper puts forward requirements derived from real-world business cases.

The remainder of this paper is structured as follows: Section 2 highlights related work before section 3 recapitulates on how events are consumed in business processes. In section 4 a catalog of patterns for composite events in business processes is presented and assessments for BPEL and BPMN are given. Finally, section 5 concludes and gives an outlook to future work.

2 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 [6], where he also introduces Rapide, an event pattern language. A framework for detecting complex event patterns can be found e.g. in [7]. Considerable work on event pattern languages can also be found in the field of active databases. In [5] the event algebra Snoop is introduced and compared with other event languages.

It is argued that process definition languages could be superseded by event pattern languages. However, flow-oriented languages, i.e. languages where the flow relation between activities is at the center of attention, have a long tradition in the field of Business Process Management and their suitability has been studied extensively [11, 8]. The fact that the most commonly used formalism in the field of business processes, namely Petri nets ([10]), is also flow-oriented underlines the importance of the flow-oriented paradigm.

Seeing message exchanges between process instances as the main event type in service-oriented architectures, the Correlation Patterns introduced in [4] describe the relationship between these communication events and the structure of process instances. The relationship between individual patterns in this paper to respective Correlation Patterns will be given in section 4.

3 Event Consumption in Business Processes

As already mentioned in the introduction, state-of-the-art process definition languages 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 process instances (depicted in Figure 1): i) A *subscription* to events (e.g. incoming messages) is initiated. ii) An *event occurs* (e.g. a message arrives). 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.

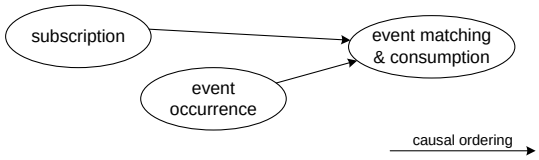

Fig. 1. Event handling in process instances: three steps

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.

We know that the consumption of an event cannot happen before the occurrence of that event nor before a subscription. These causal ordering constraints are also depicted in Figure 1. However, we leave open at this stage, whether the subscription has to precede the event occurrence. The architectural implication for allowing subscriptions after the actual event occurrence, is that the system has to store the event for later consumption. Such an architecture might not be desired since it is hard to tell how long an event should be stored.

The enlisted steps can also be found in BPEL. 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. Normally, the next two steps happen at once in the case of BPEL: as soon as a message has been matched it is routed to a process instance. However, the specification leaves it open to the process engine implementers if also messages can be consumed by a `receive` or `onMessage` activity that have arrived before that activity was reached.

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 supported in event rules in the field of Complex Event Processing. Event rules specify patterns of events that have to be matched. E.g. it is required that five corresponding messages of type customer complaint are present within a given timeframe for a given event rule to fire. Event rules enable hierarchical event architectures: Several low level events are matched in event rules producing higher-level events. In business scenarios at least two different levels of events should be present. At a low level we find

individual events, e.g. the arrival of a container detected by an RFID station, whereas on a higher level business events such as "sufficient containers available for shipment" are considered within process models. Unfortunately, such event hierarchies are not present in BPEL, only one level of events is considered. A remedy could be using event aggregation components as a separate architectural component in addition to a process execution engine. However, a seamless way of modeling processes and event aggregation is necessary in order to provide a consistent view to the process experts.

BPMN does not have the capability to express different levels of events, either. However, it 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.

The BPMN specification does not explicitly state how often an event can be consumed by process instances. But as the specification intends to map BPMN to BPEL, we assume that every event is only consumed once like in BPEL.

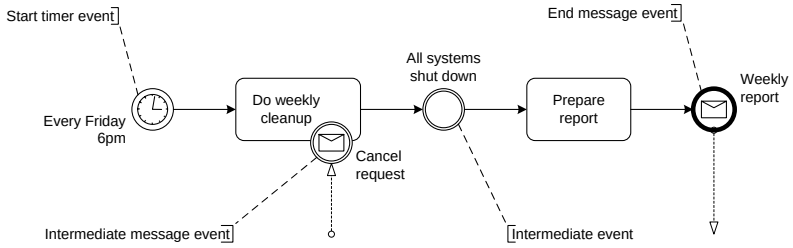

Fig. 2. Event consumption and production in BPMN

4 Patterns for Composite Events

This section introduces a set of patterns as reference framework for assessing process definition languages regarding their support for composite events in business processes. We have seen that both event consumption and production are present in process languages. The patterns enlisted in this section only focus on the consumption side. Each of the patterns comes with a short description, examples and an assessment of BPEL and BPMN.

4.1 Co-occurrence Patterns

This set of patterns describes scenarios where several events have to be considered in order to decide whether a pattern matches or not.

1. Event Conjunction. Two or more events have to have occurred in order to be matched. The order of occurrence is irrelevant. This pattern is similar to the Atomic Consumption pattern from [4].

Examples. (a) As part of the management of a shipment delivery to transient storage nodes, when the number of carriers collectively yielding the minimal outbound shipment has arrived, an event is raised to alert the relevant carriers to confirm delivery pick-up.

(b) If goods have been delivered which were previously canceled, a delivery exception event is raised.

Assessment of BPEL. Only one event is consumed at a time in BPEL processes. A workaround for some scenarios can be that e.g. several `receive` activities are placed within a `flow` constructs (i.e. in parallel). For process instantiation scenarios a `pick` representing the alternative occurrence sequences could be used. In this solution, we run into the problem that the instance is created as soon as the first message arrives. Event Conjunction demands atomicity: Only in the presence of all demanded events, an action in the process should be taken. Therefore, there is no direct support for this pattern in BPEL.

Assessment of BPMN. The situation for BPMN is similar to that of BPEL. Atomic consumption of several events is not possible in BPMN processes. Therefore, no support for this pattern, either. Similar workarounds like in BPEL are possible, though.

2. Event Cardinality. A specified number of events of the same type that are all subject to the same constraints have to have occurred in order to be matched. Event Cardinality is a special case of Event Conjunction. There are two flavors of Event Cardinality: (i) a fixed number is specified (ii) a range of numbers is specified. In the latter case a set of events can be matched as soon as the minimum number of events are present. However, if more events are available at the moment of matching, all available events are matched (as a maximum number the upper limit of the range). The fixed number and the range of numbers might be known at design-time or only at runtime.

Examples. (a) GSM stations send status report events. Some events indicate errors due to minor technical malfunctions. If more than a threshold number of errors are reported, an event is raised for trouble-shooting.

(b) Requests for purchase of small items are not processed immediately but are batched, and subsequently trigger ordering when a certain number of purchase requests is reached.

Assessment of BPEL. There is no support for this pattern in BPEL since we find the same problem like in the case of Event Conjunction: only one event is consumed at a time. Similar workarounds would be needed to implement event

cardinality (e.g. using while constructs) but the constraint of atomic matching can not be fulfilled.

Assessment of BPMN. In analogy to the case of BPEL, there is no support for this pattern in BPMN.

3. Event Disjunction. There are alternatives of events that have to have occurred in order to be matched. The Workflow Pattern "Deferred Choice" ([9]) is a special case of this pattern, where alternative individual events are waited for.

Examples. (a) The shipment planning process is started either automatically at a scheduled time before or at an earlier time determined by the shipment scheduler (e.g. in case of additional stock variances).

(b) The ordering process of an online video-on-demand marketplace uses one of several payment instruments obtained from a customer's profile. When the one of these clears credit check, the transaction phase of ordering can proceed.

Assessment of BPEL. The `pick` construct in BPEL allows to define alternative event types. The first matching event is consumed and the process resumes. This semantics implements the Deferred Choice pattern. We conclude that there is direct support for Event Disjunction in BPEL.

Assessment of BPMN. The event-based gateway has similar semantics like the `pick` construct in BPEL. Moreover, BPMN has a multiple event type. It can be triggered by alternative events and used for process instantiation scenarios. Hence, there is also direct support in BPMN.

4. Inhibiting Event. An event can only be matched in the absence of another specified event. This inhibiting event is not consumed.

Examples. (a) A fraud alert is raised if an invoice paid event was detected without a corresponding invoice approved event.

(b) A passenger's seat allocation on a flight is flagged if the passenger cannot be located through the search/alert passenger process and the departure gate of the flight is closed. Commencement of seat cancellation triggers retrieval of the passenger's baggage, although the passenger may still be allowed to board the flight if the baggage has not yet been retrieved. Baggage retrieval signifies completion of seat cancellation and the passenger is not allowed to board the flight from that point.

Assessment of BPEL. Matching of messages to subscriptions is done on a per-message basis in BPEL. I.e. no other messages are considered when deciding whether it matches or not. This leads to the situation that there is no direct support for the Inhibiting Event pattern in BPEL. However, in some cases the notion of cancellation can emulate an inhibiting event: in a certain scope of the process, incoming messages are dealt with by event handlers throwing an exception which in turns causes the scope to be canceled. This does not work for the case of process instantiation. The inhibiting event should cause an instance not to be created.

Assessment of BPMN. Like in the case of BPEL, there is no direct support for this pattern in BPMN. However, we can also think of workarounds using

intermediate events that are attached to activities. A running activity would then be canceled as soon as the specified event occurs.

4.2 Time Relation Patterns

These patterns describe common time-related constraints for event patterns. The moment an event occurs might be relevant for deciding whether a pattern matches a certain group of events.

5. Event – Event Time Relation. Two events can only be matched if their occurrence happens within or outside a given timeframe. A special case is where the event of one type always has to have occurred before the other. This pattern only appears as additional constraint for Event Conjunction.

Examples. (a) For a supermarket chain, suppliers of certain categories of stock are responsible for replenishing stock to predefined thresholds, monitored by suppliers. If an event is raised that line-item sub-category falls within a certain threshold and has order notification from the supplier for replenishment within a certain time since the threshold was reached, the replenishment process terminates without exception.

(b) Customers in an online video-on-demand marketplace are served by media content brokers. A broker who cannot fulfill a request may forward it to other brokers. Forwarded requests need to be fulfilled within a certain time of the request issued by the customer, otherwise the request is no longer current.

(c) Buy and sell events arising from the stock market of a customer portfolios are automatically correlated within a certain time of their occurrence by an investment management process, otherwise they are ignored.

Assessment of BPEL. Since this pattern always requires the presence of an Event Conjunction there is no support for this pattern in BPEL. Possible workarounds could include the usage of timeouts, i.e. `onAlarm` events: As soon as the first message is consumed, the timer is started. If the timeout occurs before the arrival and consumption of the second event, cancellation takes place. Such a workaround only works if only those messages are consumed that arrived after the corresponding `receive` activity was reached. In the other cases it is not known how much earlier the first message arrived before reaching the `receive` activity.

Assessment of BPMN. In analogy to BPEL there is no support for this pattern in BPMN. A similar workaround could be used: an intermediate timer event is attached to an activity containing the second intermediate event.

6. Event – Subscription Time Relation. An event can only be matched if it occurs within a given timeframe relative to the moment of subscription, e.g. an event must occur within 5 minutes after the moment of subscription. Alternatively it is specified that it occurs outside a given timeframe relative to the moment of subscription, e.g. an event must have occurred at least 10 days before the moment of subscription. This pattern is only relevant for cases where events are to be consumed by already running process instances.

Examples. (a) A company assessment process determines business properties. When it reaches the point of stock prize evaluation it consumes all stock prize updates of the last 2 months for that company to calculate average growth and variance.

(b) A process supporting tax return applications accepts input for the application up until the point where a final version of the tax return is prepared. Thus, any information relating to tax returns such as exemptions or investments can be asynchronously consumed during the process.

Assessment of BPEL. It is not specified when the subscription to a message is initiated in BPEL. We only know that the latest moment of subscription is when the `receive` or `onMessage` activity is reached. If we use a `pick` construct in combination with an `onMessage` and `onAlarm`, we can define a duration how long the engine should maximally wait for the message starting at the moment of reaching the `onAlarm` activity. This covers the case where the message should arrive within a given time after the subscription. All other cases are not directly supported in BPEL. We conclude that there is partial support for this pattern.

Assessment of BPMN. The event-based gateway in BPMN directly corresponds to the `pick` construct in BPEL. Therefore, we also find partial support for this pattern in BPMN.

7. Event – Consumption Time Relation. An event can only be matched if it occurs at least a certain time before the moment of consumption. Alternatively it is specified that it occurs at most a certain time before consumption. This pattern is especially important for process instantiation scenarios.

Examples. (a) Ad-hoc stock requests which occur between regular replenishment cycles are processed together for allocation to existing shipments of the next cycle.

(b) Within a certain time of stock pick-up for the next shipment time, the relevant carriers are expected to report pick-up and commencement of delivery. Outstanding carriers are contacted for determination of whether alternative transportation should be triggered.

Assessment of BPEL. There is no support for this pattern in BPEL.

Assessment of BPMN. There is no support for this pattern in BPMN.

8. Event – Absolute Time Relation. An event can only be matched if it occurs before or after an absolute point in time.

Examples. (a) Incoming calls into a service hotline during business hours are handled by the local support center. Calls outside business hours are directed to global support centers.

(b) A scheduled upgrade of a system is set for a specific time. Warning are sent out to users that log in within that specific time frame.

(c) During fixed times on weekdays, processes are triggered for major roads with interchangeable lanes (for left/right side) to have their lane allocations set.

Assessment of BPEL. The `onAlarm` construct in BPEL has both a duration semantics (the timeout occurs after a certain time period has passed) and a

deadline semantics (the timeout occurs as soon as a certain deadline is reached). The latter can be used to easily express one flavor of the pattern: A `pick` construct in combination with an `onMessage` and an `onAlarm` can implement cases where the message has to arrive before a given deadline. Since this implementation does not cover the situation where a message has to arrive after a given point in time, we opt for partial support for this pattern.

Assessment of BPMN. In analogy to BPEL we can use a combination of an event-based gateway with an intermediate event and an intermediate timer event. The other flavor of the pattern is not directly supported, either. Therefore, we also have partial support for this pattern in BPMN.

4.3 Data Dependency Patterns

For the following set of patterns we assume that events carry additional data. For instance, incoming messages carry message content and a "goods have arrived" event carries information about the supplier and the corresponding order. The patterns describe how data dependencies constrain the matching of events.

9. Event – Event Data Dependency. Two events can only be matched if their data is in a specified relation. This pattern only appears as additional constraint for Event Conjunction. It is similar to Key-based and Property-based Correlation in [4].

Examples. (a) An incoming order confirmation or order rejection event is matched with the outgoing order request event with the same OrderID.

(b) An investigation for provision of land tenure (e.g. land planned for a school site) involves complex searches for related tenure applications and future land actions planned, based on geographic locality (e.g. railway line planned or environmental regulations in the locality) and "neighborhood" parcels of land within a geographic locality (shopping center planned in the same block). Keys for data correlation accordingly vary.

Assessment of BPEL. Since this pattern always requires the presence of an Event Conjunction there is no support for this pattern in BPEL. In general, correlation sets can be defined for constraining the messages matched for a running process instance. However, since only a combination of two or more events should be matched that fulfill the data constraint, it would be invalid to accept a first message independently of the data constraint. Therefore, receiving any message of a desired port type / operation and then using this message for initializing the correlation set which in turn is used for matching the second one, cannot be a valid workaround.

Assessment of BPMN. Unlike BPEL, BPMN does not provide any support for correlation. Therefore, data dependencies between two events cannot be expressed in BPMN.

10. 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.

Examples. (a) A Customer Payment Details event is consumed by the process that reference the same Customer within its process context.

(b) A reply to an asynchronous request is routed to the process instance that holds correlation data within its process context matching the event data.

(c) An Order Cancellation affects the process instance that holds the correct Order identifier in its context.

Assessment of BPEL. Correlation sets are a means to restrict subscriptions to messages with specific content. E.g. a customer ID can be included in a message and only those messages are accepted that belong to a particular customer. We conclude that there is direct support for this pattern in BPEL.

Assessment of BPMN. BPMN simply lacks the notion of correlation and therefore does not support this pattern.

11. Event – Environment Data Dependency. An event can only be matched if its data is in a specified relation to data that can be accessed by different process instances. This pattern is especially important for process instantiation.

Examples. (a) Email requests from premium customers start premium handling processes, other start normal handling processes.

(b) A shipper processes shipment request events only if they come from known business partners.

(c) Only invoices referencing a valid order start an approval process.

Assessment of BPEL. Whether a message triggers process instantiation is only decided based on the port type / operation combination. BPEL does not allow to further constrain such consumption using correlation information. Therefore, there is no direct support for the pattern in BPEL. A workaround could be that messages first trigger process instantiation and are then checked for their content. If the content does not fulfill the constraint the process instance is terminated.

Assessment of BPMN. It is not possible to constrain the consumption of events based on data attached to them. Therefore, there is no support for this pattern, either.

4.4 Consumption Patterns

The Consumption Patterns describe how often an event can be consumed.

12. Consume Once. An event can only be consumed at most once by one out of all process instances.

Examples. (a) An order request event is to be processed exactly once.

(b) Activation for a newly provided credit card event is required only once.

(c) An event notifying the breakdown of a carrier should be consumed once by a specific breakdown-service agent.

Assessment of BPEL. Every incoming message is routed to at most one process instance and is consumed by at most one `receive` or `onMessage` activity. Hence, there is direct support for this pattern.

Assessment of BPMN. In the previous section we mentioned the assumption that every event is consumed by at most one BPMN process instance. This results in direct support for this pattern.

13. Consume Multiple Times. An event is consumed several times (possibly within the same process instance). This pattern is similar to the Multiple Consumption pattern from [4].

Examples. (a) Events signifying share buy/sell recommendations are consumed many times by an investment monitoring process for different customer portfolios.

(b) A traffic monitoring system provides traffic updates for its subscriber carriers involved in delivering goods.

(c) Changes to a shipments are broadcasted to its stakeholders including different carriers, storage nodes, final consignments and stock-to-shelf dispatchers.

Assessment of BPEL. In the case of BPEL, every message is consumed at most once. Therefore, BPEL does not support this pattern.

Assessment of BPMN. Due to our assumption that every event is consumed by at most one BPMN process instance, we conclude that there is no support for this pattern in BPMN.

Composite Event Patterns	BPEL	BPMN
1. Event Conjunction	–	–
2. Event Cardinality	–	–
3. Event Disjunction	+	+
4. Inhibiting Event	–	–
5. Event – Event Time Relation	–	–
6. Event – Subscription Time Relation	+/–	+/–
7. Event – Consumption Time Relation	–	–
8. Event – Absolute Time Relation	+/–	+/–
9. Event – Event Data Dependency	–	–
10. Event – Process Instance Data Dependency	+	–
11. Event – Environment Data Dependency	–	–
12. Consume Once	+	+
13. Consume Multiple Times	–	–

Table 1. Composite Event Pattern support in BPEL and BPMN

5 Conclusion and Outlook

This paper has introduced a set of patterns describing common eventing scenarios in business processes. In analogy to other sets of patterns in the field of Business Process Management, they can be used to evaluate process definition languages and systems. Table 1 summarizes the assessment of BPEL and BPMN

that we carried out in section 4. Direct support for a pattern is denoted as "+", partial support as "+/–" and no support as "–". It turns out that BPEL and BPMN support similar patterns, while a wide range of patterns are not supported by both languages. This underlines the initial assumption that only very basic eventing scenarios can be captured. We argued that modeling process logic and describing complex event patterns should not occur independently of each other since both aspects are essential for process experts to capture the overall process context. As a result, we see the need to closely integrate event pattern descriptions into executable process definition languages such as BPEL and higher-level modeling languages such as BPMN.

Future will especially focus on integrating more sophisticated eventing mechanisms into BPMN. As part of that, graphical representations for event patterns will be proposed.

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. A. Barros, G. Decker, M. Dumas, and F. Weber. Correlation Patterns in Service-Oriented Architectures. In *Proceedings of the 9th International Conference on Fundamental Approaches to Software Engineering (FASE)*, Braga, Portugal, March 2007.
5. S. Chakravarthy and D. Mishra. Snoop: An expressive event specification language for active databases. *Data Knowledge Engineering*, 14(1):1–26, 1994.
6. D. Luckham. *The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems*. Addison-Wesley, 2001.
7. 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.
8. N. Russell, W. M. van der Aalst, A. ter Hofstede, and P. Wohed. On the Suitability of UML 2.0 Activity Diagrams for Business Process Modelling. In *Proceedings 3rd Asia-Pacific Conference on Conceptual Modelling (APCCM 2006)*, volume 53 of *CRPIT*, pages 95–104, Hobart, Australia, 2006.
9. W. M. P. van der Aalst, A. H. M. ter Hofstede, B. Kiepuszewski, and A. P. Barros. Workflow Patterns. *Distributed and Parallel Databases*, 14(1):5–51, 2003.
10. W. v. d. van der Aalst and K. v. van Hee. *Workflow Management: Models, Methods, and Systems (Cooperative Information Systems)*. The MIT Press, January 2002.
11. P. Wohed, W. M. van der Aalst, M. Dumas, A. ter Hofstede, and N. Russell. On the Suitability of BPMN for Business Process Modelling. In *Proceedings 4th International Conference on Business Process Management (BPM 2006)*, LNCS, Vienna, Austria, 2006. Springer Verlag.