



Universiteit
Leiden
The Netherlands

Constraint-based analysis of business process models

Changizi, B.

Citation

Changizi, B. (2020, February 21). *Constraint-based analysis of business process models*. IPA Dissertation Series. Retrieved from <https://hdl.handle.net/1887/85677>

Version: Publisher's Version

License: [Licence agreement concerning inclusion of doctoral thesis in the Institutional Repository of the University of Leiden](#)

Downloaded from: <https://hdl.handle.net/1887/85677>

Note: To cite this publication please use the final published version (if applicable).

Cover Page



Universiteit Leiden



The handle <http://hdl.handle.net/1887/85677> holds various files of this Leiden University dissertation.

Author: Changizi, B.

Title: Constraint-based analysis of business process models

Issue Date: 2020-02-25

2

Business Process Model and Notation

2.1 Introduction

Business Process Model and Notation (BPMN) [Gro11], also known as Business Process Modeling Notation, is a standard graphical representation of business process models. BPMN bridges the gap between visualization of the business processes and their actual implementation by providing an understandable notation for both business stakeholders and technical experts.

BPMN is based on flowcharting techniques. It allows modeling complex business processes using its diverse set of control structures, which covers concepts such as sequencing, repetition, choice, concurrency, messaging, failure, transactions, etc. BPMN has an expressive notion to define events and to associate triggers to the defined events. Furthermore, it provides means to form reusable units out of a set of elements.

The first version of BPMN is developed by the Business Process Management Initiative (BPMI) in 2004. In 2005, BPMI and the Object Management Group (OMG) merged. BPMN is maintained by OMG since then. In 2006, the BPMN specification was adopted as an OMG standard. In 2011, the final edition of BPMN

2 specification was released.

BPMN 1.2 presents a notation for modeling business processes and informally expresses the semantics of the modeling primitives. This leads to ambiguity and confusions in interpretation of a process. For instance, the authors in [vdADK02] present a deadlock situation called *vicious circle* that is caused by using convergent *inclusive gateways*. This is a class of situations where two *inclusive gateways* are connected in a cyclical way. Moreover, BPMN 1.2 specification provides no details on model serialization format.

BPMN 2, the biggest revision of BPMN so far, presents a formal definition in terms of a meta-model, that is a formal definition of the constructs and their relations in a valid model. The meta-model specifies a serialization format that enables model exchange among different BPMN 2 tools. In the context of modeling elements, BPMN 2 offers the following enhancements over previous versions:

- It expands the set of BPMN gateways with *exclusive* and *inclusive* event-based gateways.
- It enriches the set of activities by adding *business rule task*, *sequential multi-instance* activity, *event sub-process* that handles events occurring in bounding *sub-process*, and *call activity* that invokes a global *sub-process*.
- It enhances *events* by introducing *escalation*, and *complex events*, and the concept of *interrupting* and *non-interrupting* events.

Although BPMN 2 provides an explicit execution semantic, the semantics are expressed in informal fashion. This leaves rooms for interpretation for a number of issues such as deadlocks and race conditions.

In this chapter, we provide an overview to BPMN. We also present examples of process models containing semantical errors.

2.2 BPMN 2 elements

A BPMN diagram consists of a number of elements that fall into the categories of *flow objects*, *connecting objects*, *swimlanes*, and *artifacts*. A flow object can be an *event*, a *gateway*, or an *activity*.




2.2.1 Connecting objects

Connecting objects are used to connect the other BPMN elements:

- *Sequence flows* represent the occurring order of processes in a business model.
- *Message flows* are used to exchange messages between process participants.
- *Association flows* associate modeling elements to each other. For instance, a *compensation task* is associated to its task via an association flow.

2.2.2 Events

Events represent triggers occurring during execution of business processes. Events usually have a *cause* or a *result*. The representation of an *event* is a circle wherein internal markers are placed to denote triggers or results. Based on the time that events affect the flow, they fall into three categories:

-  *Start events*, which start a process;
-  *Intermediate events*, which occur between start and end of a process;
-  *End events*, which terminate a process.

Each time a process receives a new start event trigger, a new instance of the process begins to execute. Therefore, a process may have many process instances. Start events and intermediate events are *catching*, meaning that they catch a trigger in order to occur. End events and some of intermediate events are *throwing* as they throw a result. Compared to the passive nature of catching events, throwing events are *active* as they trigger themselves rather than waiting for a trigger to take place.

The following intermediate events can attach to the boundary of an activity: *message*, *timer*, *error*, *compensation*, and *signal*. In this case, they can only occur while the surrounding activity is active. *Boundary events* can either be interrupting or non-interrupting.

Interrupting events stop the execution of the activity and direct the flow out of the boundary event, while non-interrupting events do not interfere with the execution of the activity. Instead, they start the flow out of the boundary event in parallel. Another difference is that non-interrupting events can occur several times while the surrounding activity is running.

Following is the list of event types in BPMN 2:

- A *none* event has no defined trigger. It can indicate a start point, a state change or a final state. Each process can only have one *none start event*.
- ✉ A *message* event is used to model exchange of messages. A message has a specific receiver.
- △ A *signal* is broadcasted between processes. It differs from message in that a message has a specific target, but a signal is broad-casted. A thrown signal can be caught multiple times.
- 🕒 A *timer* event indicates a waiting time within the process. A timer trigger can be a specific date/time value or a duration.
- 📋 A *conditional* event occurs when a business condition becomes true.
- ➡ A *link* is a mechanism for connecting two sections of a process. A throwing link event is used at the exit point, while a catching link event as the entrance point. Using link helps keeping the model clean and prevents spaghetti models.
- ⊗ A *cancel* event is always used with a transaction sub-process. It indicates that the transaction should be canceled. A cancel event triggers a cancel intermediate event attached to the sub process boundary.
- A *terminate* event indicates that all activities in the process should be immediately ended. In this case, the process is ended without compensation or event handling.
- ⏪ A *throwing compensation* event indicates that a compensation is needed. A catching compensation event states that a compensation will occur when the event is triggered. All other boundary events occur only while the activity that they are attached to is active. In contrary, an attached compensation takes place only if the process triggers a compensation and if the activity to which compensation is attached has been completed successfully.
- ⬢ A *multiple* event summarizes several events with a single event. A catching multiple event occurs if at least one of its specified events occurs. However, a throwing multiple triggers all the defined events.

- ⊕ A *parallel multiple* event, which is added in BPMN 2, is a supplement to multiple event. A parallel multiple event is only catching. It indicates that all of the defined events are required in order to trigger this event.
- ⌖ *Escalation* is new in the BPMN 2 specification. An escalation event is used to trigger a path in middle of a process flow that requires involvement of a higher responsibility.

Based on the types, event triggers are forwarded in five different strategies:

- *Publication*: A published trigger can be caught by any catching event that matches the trigger within any scope where it is published. *Message* and *signal* events triggers are forwarded this way.

Messages are created out of the pool wherein they are published. In case that a message should be received by a specific process instance, the particular instance is referred by the message.

Signals are created inside the pool wherein they are published. In general, signals are used to broadcast within and across processes, pools, and process diagrams.

- *Direct Resolution*: The timer and conditional triggers are thrown implicitly. These triggers wait for a defined time or a specific condition to trigger the related catch event, respectively.
- *Propagation*: A propagated trigger is forwarded from its origin to the innermost enclosing level that has an attached catching event that matches the trigger. Instances of events that propagate are *error* and *escalation*.

Unlike error triggers that are critical and suspend execution, escalations are non-critical and allow execution to proceed normally. If there is no catching event found for an error or an escalation trigger, the trigger is unresolved.

- *Cancellation*: When a *cancellation* occurs, all running activities terminate and all activities in the sub-process wherein cancellation applies are compensated, if they are completed successfully. In case that the sub-process is a transaction, it needs to be rolled back.
- *Compensation*: A successfully completed activity is *compensated* by its compensation handler, which is either user-defined or implicit. In latter case, the

compensation handlers of the enclosed activities are invoked in the reverse order of their execution. If an activity has not completed successfully, nothing happens and no error is raised.

2.2.3 Activities

An activity describes the type of work that needs to be done. An activity is either a task, a sub-process, or a transaction. BPMN represents the activity in a high-level of abstraction. It is not the BPMN responsibility to describe the activity details.



Tasks, which are atomic activities have several types:



A *manual task* is a task that is performed manually.



A *user task* is performed by a person with assistance of automation.



Service tasks are services such as web services or automated applications.



A *script task* is executed by a business process engine.



Business rule tasks are introduced in BPMN 2. They are performed by business rule engines.



A *send task* is a simple task with an outgoing message flow, which is used for sending messages. The task is completed after the message is sent.



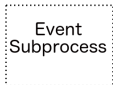
A *receive task* is a simple task with an incoming message flow, which waits for a message to arrive. Once it receives the message, the task is completed.

A *sub-process* captures a set of activities, gateways, and flows within a single activity. It hides or reveals details of business process based on being expanded or

collapsed, which is denoted using a plus sign at the bottom of the sub-process. A sub-process may only begin with a *none start* event and end with a *none end* event.



A *transaction* is a sub-process that all of its enclosed activities constitute a logical unit of operation, meaning that all the activities must be completed successfully, and if one fails, all of them need to be compensated.



Event sub-process are introduced in BPMN 2. An event sub-process behaves like a boundary event, but it resides inside a process or a sub-process rather than on their boundaries.

An event sub-process can be considered as an optional sub-process that occurs when its start event is triggered.

Similar to boundary events, an event sub-process may interrupt the containing process or run in parallel in a non-interrupting fashion, depending on the type of its start event.

In addition, it is allowed to have only one start event that is non-empty. The event types that can be used as a start event for an event sub-process are: *message*, *conditional*, *signal*, *timer*, *escalation*, *error*, *multiple*, and *parallel-multiple*. As mentioned, the only way to run an event sub-process is by triggering its start event. As a result, no incoming or outgoing sequence flow can connect to an event sub-process.

In BPMN 1.2, there are two types of sub-processes: *embedded* and *reusable*. BPMN 2 sub-processes are inherently embedded. They can only be reused if they are defined globally and are referenced by call activities.

An embedded sub-process can only contain a *none start* event. It cannot have other types of start events such as timers or messages.

Furthermore, an embedded sub-process can only be found inside a process to which it belongs. A global sub-process, on the other hand, can reside within different processes.

In BPMN 2, reusable task and sub-processes are invoked using a *call activity*. According to the BPMN 2 specification [Gro11], a call activity in BPMN 2 corresponds to the BPMN 1.2 reusable sub-process, while a sub-process in BPMN 2 corresponds to the BPMN 1.2 embedded sub-process.

A transaction has three possible outcomes:

- All the activities finish *successfully*. In this case, the process proceeds with the normal flow.
- In case of a *failure*, the compensation tasks associated to the successfully

completed activities execute. The process continues through the cancel intermediate event.

- In case that an *unexpected error* takes place, the sub-process activities are interrupted without any compensation. The process then proceeds with the intermediate error event.

An activity can be annotated using different *markers* that indicate the nature of the activity. The markers are as follows:

- The *loop* marker indicates that the attached activity executes multiple times until the loop condition holds. The condition can be evaluated either in the beginning or in the end of the activity depending on a specific attribute of the activity.
- A *compensation* marker is used to undo a completed activity.
- A *sequential multi-instance* marker defines an activity that has multiple instances created sequentially. The number of instances to be instantiated is either defined as an attribute of the activity or as the cardinality of input data items.
- A *parallel multi-instance marker* represent activities that can be executed in parallel as multiple instances. Each instance can have a different set of input parameters.
- An *ad-hoc marker* is used to represent an activity, whose inner tasks have no required order. Each task can start at any time. There is no dependency among the activities.

2.2.4 Gateways

Gateways manage the control flows within a process or sub-process by specifying the interaction among sequence flows as they converge and diverge. The list of BPMN 2 gateways follows:

2.2.5 Swimlanes and artifacts

A *swimlane* is used for organizing and categorizing activities inside a business process. A *swimlane* can be either a *pool* or a *lane*. A *pool* represents a participant in a business. *Lanes* are partitions inside a *pool*.



Data-based exclusive gateways are used to create alternative paths based on the conditions that are set on the incoming data flow. A diverging exclusive gateway, also called *decision*, routes the incoming flows to one of the mutually exclusive alternative outgoing flows. A converging exclusive gateway directs one of its incoming flows to its only outgoing flow.



Data-based inclusive gateways create alternative but also parallel paths within a process flow. A diverging inclusive gateway directs its incoming flow to one or more outgoing flows based on conditions. A converging inclusive gateway, on the other hand, awaits incoming flows to complete.



Parallel gateways are used to create and also to combine parallel flows. A diverging parallel gateway creates parallel flows, while a converging one merges the incoming flows into one outgoing flow.



Event-based gateway routes based on occurrence of events rather than on data. In addition to events, it also works with receive message task. An event-based gateway is always followed by catching events or receive tasks.



A *parallel event-based Gateway* is similar to a parallel data-based gateway with the difference that it depends on occurrence of events rather than on data.



A *complex gateway* models complex synchronization behavior. An expression is used to describe the behavior of the gateway.

Artifacts are used for adding information into the model. The followings are three types of *artifacts*:

- *Data objects*, which describe the required or the produced data in an activity.
- *Groups* are used to categorize different activities.
- *Annotations* are providing information about the model.

Example 2.2.1 *Figure 2.2.1 depicts a BPMN model consisting of two processes. The receiver process starts, waits till receiving a message from the sender process before it ends. While the sender process starts, evaluates a condition based on which it chooses to end or to send a message to the receiver process, and returns back to the condition evaluation step.*

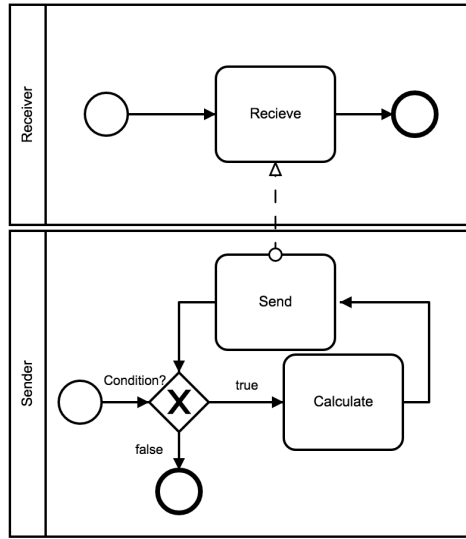


Figure 2.2.1: An example of messaging in BPMN

The desired behavior of this model is that the processes start, the message exchange occurs, and they end. However, it is possible that the sender process finishes without sending any message. In this case, the receiver process keeps waiting for a message that will never arrive. This is an example of deadlock.

In addition, the model contains a livelock, which occurs if after the receiver process receives a message from the sender process and finishes, the sender keeps going back to the sending step and does not end.