



Bruce Silver Associates

Independent Expertise in BPM

BPMN and the Business Process Expert, Part 3: The Art of Process Modeling

Summary: BPMN's diagram semantics are expressive and precise, but the spec doesn't tell you everything you need to know to create effective models. Here we go beyond the spec with nine tips for making your process diagram say exactly what you mean. Third of six parts.

Author: Bruce Silver

Company: Bruce Silver Associates

Created on: 19 November 2007

Author Bio



Dr Bruce Silver is an independent industry analyst and consultant focused on business process management software. He provides training on process modeling with BPMN through BPMessentials.com, the BPM Institute, and Gartner conferences, and is the author of The BPMS Report series of product evaluations available from the BPM Institute.

In the first two installments of this series, we saw why BPMN is important to the Business Process Expert and got an overview of the notation. In this part, we'll look beyond the spec to suggest some best practices for making your BPMN models most effective.

The art of effective process modeling depends on what you are trying to do. Unlike traditional notations, which presuppose a particular methodology, BPMN is methodology-neutral, and can be used for multiple purposes. The first, which I call Level 1, is simply *qualitative description*, a diagram of the as-is (or proposed to-be) business process that stakeholders can gather around, discuss, and improve. Level 1 models may ignore exceptions and show only the "happy path," and may not include every step, just those significant for discussion and qualitative analysis.

The second, which I call Level 2, describes all the activities and flows in the process, including exceptions, and pays particular attention to their sequential and concurrent relationships. The goal of Level 2 modeling is a complete business-intelligible description of the process sufficient for quantitative analysis. Level 2 models contain all the detail required for accurate simulation, but are not by themselves executable. Level 2 modeling is still a business – or BPX – function. It does not require technical knowledge of the implementation of each activity, essentially just the activity's name, performer, and possible exceptions. The rules embodied in the BPMN spec mostly apply to modeling at Level 2.

Level 3 modeling refers to using BPMN in *executable process design*, typically in a BPM Suite. It is similar to Level 2 modeling, but most tools that leverage BPMN as part of their executable design environment diverge here and there from the spec, since not everything that can be drawn in BPMN may be executable on the BPMS's process engine, and even

Bruce Silver Associates

BPMS Watch www.brsilver.com/wordpress
BPMN Training www.bpmessentials.com/

500 Bear Valley Road, Aptos CA 95003

Tel: 831.685.8803 Fax: 831.603.3424 E-mail: bruce@brsilver.com

when the engine can implement the BPMN semantics, the executable design may not describe it in accordance with BPMN. Thus BPMN at Level 3 is, today at least, vendor-specific in its details.

So in this article, when I talk about the art of effective process modeling, I really mean at Level 2.

Here are nine best practices to get you going.

1. Make the important information visible in the diagram. Label everything.

BPMN is primarily a diagramming notation, but it also provides attributes for each diagram element. Many of these attributes are not displayed in the diagram itself, but are really placeholders for detail needed for technical implementation or simulation analysis. Even if modeling at Level 2 or 3, all of the important features of the process should be described, at least qualitatively, in the diagram itself. Shared understanding is always goal number one, and typically that comes from stakeholders gathered around a table looking at a printed version of the diagram. They are not accessing the diagram through a modeling tool, which shows the attributes in property sheets, and they are not reading through 200 pages of documentation generated by the modeling tool. They are looking at a printed diagram, so if a key bit of detail is not shown there, it may as well not be in the model.

As a practical matter, this rule translates to “label everything” – not just activities but subprocesses, gateways, events, and sequence flows. All of these have a Name attribute that corresponds to the object’s label in the diagram. The BPMN spec never requires these objects to have labels, but does require, for instance, population of non-diagram attributes like gateway conditional expressions, message references, and error codes. Whether you choose to comply with those spec requirements or not, best practice is to create *labels* for gateway outputs that suggest, in business-intelligible terms, the meaning of each path. Unlike non-printing attributes, labels clarify *in the diagram* the meaning of a message event, the timeout of a timer event, or the source of an error.

2. Make your diagrams valid. BPMN has rules; learn to follow them.

A process modeling tool is not the same as a drawing tool. You can download free BPMN stencils for Visio, but they don’t bring with them the underlying semantics and rules of BPMN, so you can create invalid diagrams and never know it. A true modeling tool has a function called Validate, and if your diagram contains spec violations it will list them for you. Sure, a few of the rules in the spec may be arbitrary or even misguided – such as requiring non-printing attributes but not labels, as discussed above – but most of them merely enforce the semantic precision that is the key to linking business-intelligible diagrams to executable implementation. A model is not a doodle. It has rules, so follow them.

3. Make your diagrams hierarchical, using subprocesses.

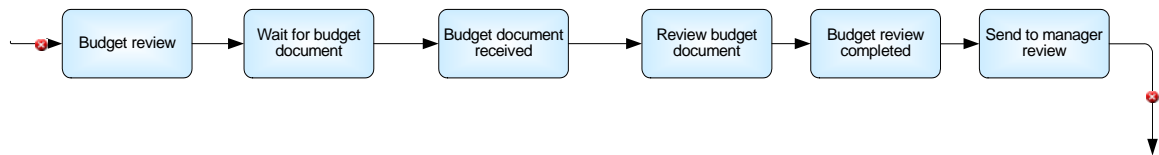
This is admittedly a personal bias, to which some others do not subscribe. When the as-is process is first captured by putting stickies on the wall, what you have is a “flat” process model, with all the detail at one process level. Of course, you can’t take it in all at once, but instead have to walk around the room looking at one fragment at a time. Some people build their BPMN models this way, too, but it seems counter to the whole notion of BPM as a management discipline, which advocates looking at processes end-to-end. In traditionally

stovepiped organizations, BPM is a reaction to the problem of not seeing the forest for the trees, and flat BPMN models that do not provide an end-to-end view are not a good answer.

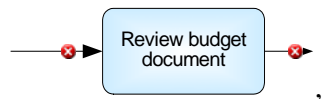
Instead, you should be able to model the end-to-end process on one page using coarse-grained-subprocesses, and drill down into each subprocess on a separate page of the diagram. Because it can render a subprocess either as a collapsed activity shape or expanded into its detailed flow – and repeat that drilldown as many levels as necessary – BPMN allows process description at multiple levels of detail, accessed by zooming in and out in the tool, without losing the integrity of a single end-to-end process model.

4. Use tasks to represent work. Label them VERB-NOUN.

In BPMN, a task represents *work* done in the process, an action. It is not a function, not a state, and not a handoff to other participants in the same process. The state of a task (started or completed) and the sequential flow of work are *implicit in the notation itself*; you don't need to add tasks for them. In fact, it is incorrect to do so. Nevertheless, from beginners in BPMN you may see task sequences like this:



Instead, the correct sequence should be this:



or if the budget document is received from outside the process instead of from a prior step, maybe this:



In BPMN, a message event (or, equivalent, a Receive task) inherently means *wait for the event and then continue*. A User task is inherently enabled to start when its incoming sequence flow reaches it, and the sequence flow out of it means that the task is completed. So diagrams should not insert tasks to signify those states, since they are implicit in the notion of sequence flow itself.

5. Reserve the verbs Send and Receive in task names (labels) for activities that either send a message or wait for (receive) a message.

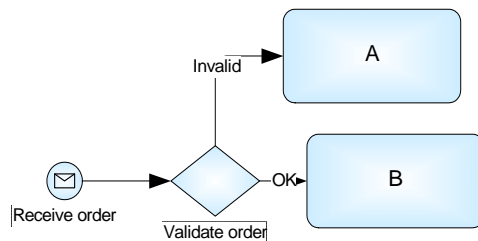
In BPMN, a “message” simply means a signal to or from an entity outside the process. The signal does not have to be a SOAP or JMS message. It could be a fax or a phone call. What is significant is that the fact that the signal means communication with an *external* entity. That entity could be the requester of the service that the process represents, such as a customer placing an order. Or it could represent an external service invoked by the process.

Information passed by a task to another task downstream is not sending a message. In fact, the sequence flow by itself delivers process information to the task, so the upstream task doesn't have to “send” anything.

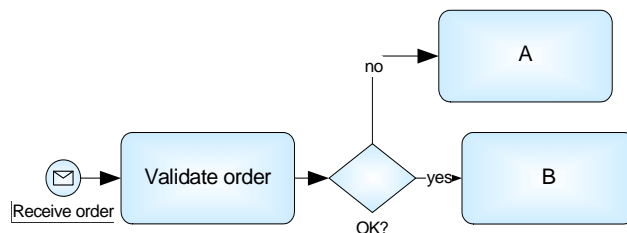
BPMN defines special task types called Send and Receive used for inter-process communications. A Send task is the same as a “throwing” message event. Typically, it is not performed by a user but is assumed to be an automated activity that sends the message as soon as the sequence flow into it arrives. A Receive task is the same as a “catching” message event. It too is not performed by a user but is assumed to represent an event listener that pauses the flow until the event arrives, and then continues.

6. Use gateways to represent pure routing logic.

Gateways do not perform work. They are not assigned to a resource such as a person or engine. They do not *make* decisions, but control the flow *following* a decision. However, it is not uncommon to see things like this:



This is incorrect. Validating the order, whether performed by a person or a business rule engine, is work, and it requires a task in BPMN. A gateway following the decision task can then route the instance this way or that depending on the result.

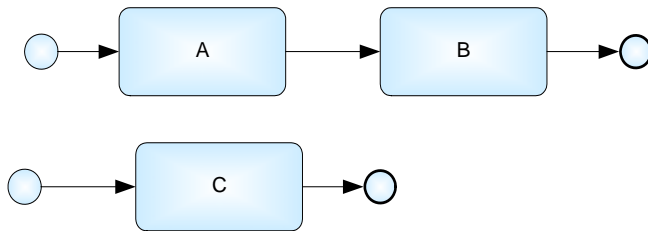


Also, sometimes you see complex decision logic, such as that used to validate an order, represented as a network of gateways in which each gateway represents a single rule in the decision. This is allowed in BPMN, but in general is not best practice, for several reasons. One, it is inherently procedural, whereas in the real decision the rules are often declarative. Two, it embeds the internal decision logic in the process, where it is best practice to externalize business rules so they can be maintained independently of process, particularly when the rules represent policies that cross process boundaries. And three, it makes the diagrams unnecessarily complex. Just use a single task to represent the decision, and follow it with a gateway to route the subsequent workflow.

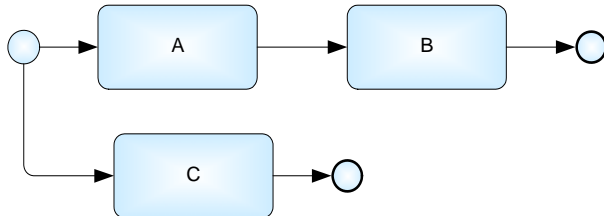
7. Avoid multiple start events.

The BPMN spec allows multiple start events in a process or subprocess, but recommends they be “used sparingly and that the modeler be aware that other readers of the Diagram may have difficulty understanding the intent of the Diagram.” I’ll go further and say, “just don’t do it,” since the semantics are, more often than not, inherently ambiguous.

In the one case where the semantics are unambiguous, a single start event can be used to signify the same thing. You could see something like this in a process or subprocess:

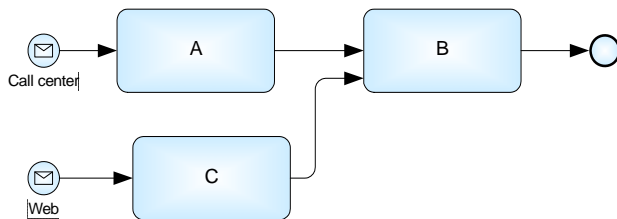


Multiple None start events (no trigger symbol inside) means trigger all of the start events when the process is instantiated. But that's the same thing as a single None start with multiple sequence flows out of it, signifying parallel paths from the start.



Thus the multiple start events are not incorrect, just superfluous.

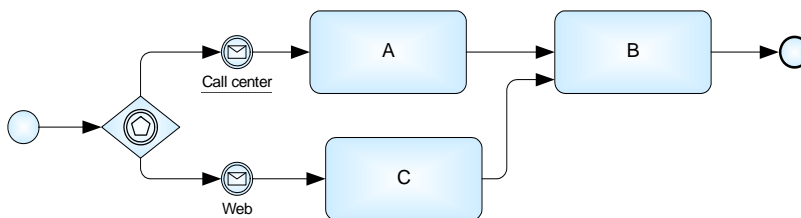
Now what does this diagram mean?



Usually the modeler's intent is that the process can be triggered one of two ways, such as through the call center or through the web. Some initial processing might be channel-dependent, but all channels merge to a common downstream process. But is this the correct way to draw it?

The BPMN spec says each start event is an independent event and generates a process instance. Thus you could argue that if the second event occurs after the first one, it generates a separate instance of the process, so the diagram above would be correct. But the spec also says that if a process requires two independent events to occur in order to start, you model this with two start events that flow to a common activity with a special merge attribute set. So in some cases, multiple start events mean the same instance and in other cases they mean different instances. That's bad spec-writing.

Fortunately, BPMN 1.1 provides a less ambiguous solution to our modeler's use case: an event gateway used to "bootstrap" the process. (In BPMN 1.0, an event gateway could only be used in the middle of a process, not to instantiate it.)



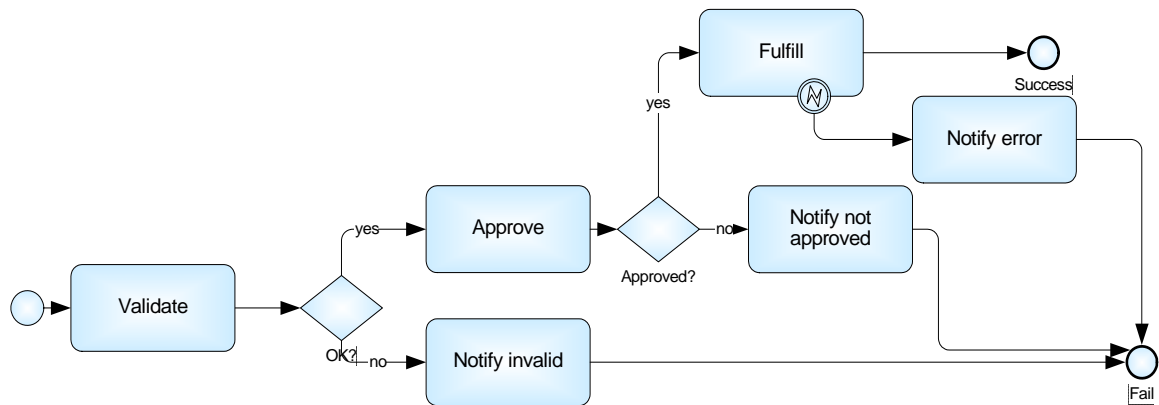
The event gateway means wait for one of the following events – typically message events – and take the sequence flow from the first one to occur. If another of the listed events occurs later, it triggers a new process instance... unambiguously.

Thus there is no good reason to ever use multiple start events.

8. Use multiple end events to classify end states.

This might seem inconsistent with the previous recommendation, but it's really not. An end event does not by itself end a process or subprocess. If the process or subprocess contains parallel paths, *all* of them must reach an end event in order for it to complete. Thus there is an implicit AND-join of all end events in a process or subprocess.

Even if there are no parallel paths, it is quite common to draw separate end events for each alternative path. On the other hand, you can draw sequence flows from each path in the diagram to a single end event, and there is nothing wrong with that, either. But it often clarifies the diagram to simply draw end events representing each possible end state of a process or subprocess, such as success or failure, and labeled accordingly. In a subprocess, particularly, where a gateway following the subprocess governs subsequent processing of the end-to-end process, this helps make the meaning of the diagram understood by all.



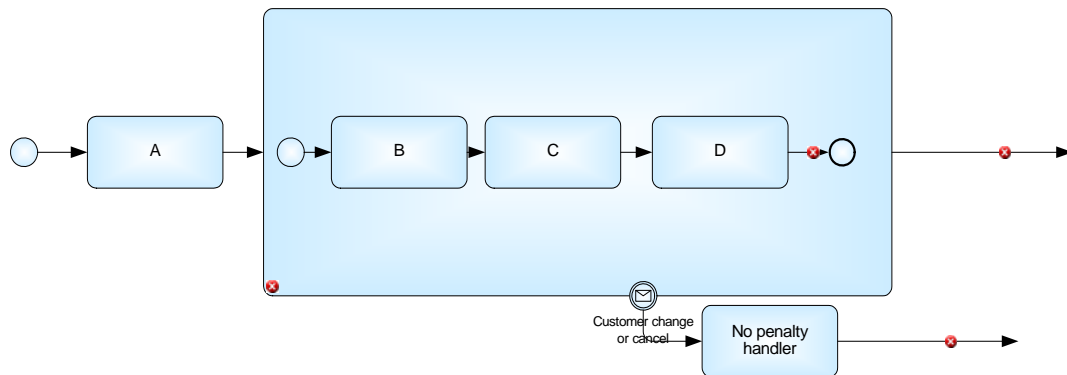
9. Use subprocesses to scope events.

This best practice will make more sense after the next article in this series, but it's so valuable it's worth repeating: An attached intermediate event – timer, message, error... whatever – can only be detected by the process when the activity it is attached to is active. If the event occurs before the activity starts, or after it ends, BPMN says ignore it. If the event occurs while the activity is running, the BPMN says abort the activity and continue down the sequence flow out of the event, called the exception flow.

The activity doesn't have to be a task. It can be a *subprocess*. In fact, it can be a subprocess created solely for the purpose of defining the fragment of the process where a particular event should result in a particular exception flow.

Here's an example. Suppose you have an order handling process with steps A to Z and you want to indicate that between steps B and D the customer can change or cancel the order with no penalty, and between steps E and G a change or cancellation has a penalty, and after that any change or cancellation requires a completely new process. In BPMN this is easy to model. You simply enclose the sequence from B to D in a subprocess and attach a message event (external signal) to it with a no-penalty exception flow. You also enclose the sequence from E to G in another subprocess and attach a message event to it with a penalty exception

flow. That's it! Don't tell me, as traditional BPA tool vendors often do, that BPMN events are too complicated for the Business Process Expert.



For attached timer event, the activity that is attached to has additional significance. An attached timer event is typically used to specify deadline-triggered processing. Usually the deadline is specified not as a specific date/time but as a time interval after the activity starts. So it's the activity that determines when the clock starts and when it gets shut off. If in a process with steps A to Z you want to trigger some exception flow if step C is not completed within 2 hours after the process (i.e., step A) starts, you can simply enclose A through C in a subprocess and attach a timer event to it.

Creating effective BPMN diagrams sometimes requires going beyond the spec. Learning how to harness the expressiveness of the notation is critical to shared understanding and business-IT alignment.

Bruce Silver