Execution Semantics

Moving Process Models from the Conceptual to the Executable

White Paper

Author: Tom Debevoise

Black Pearl Development, Inc.

BLACK PEARL
DEVELOPMENT

Execution Semantics
By: Tom Debevoise,
Executive Director
Black Pearl Development, Inc.

Contents

Execution Semantics
By: Tom Debevoise,
Executive Director
Black Pearl Development, Inc.

Introduction

We use BPMN and DMN to depict process scenarios and use cases in a manner that business analysts can understand. Yet, to become a digitized, executable process, the business details of the process model must become a technical deliverable – For example, in a purchasing use-case:

- The period of performance for a contract becomes two data elements populated from the query of a REST service.
- A demographics web form is created at the point of the user task.
- A six month projection of inventory demand becomes a regression against an array of numbers.

Yet, even after you have described the placement of these with proper BPMN activities, events, gateways, transitions, and messages, and with the proper applications of problem solving and workflow patterns, no BPMN tool can guarantee the execution-ready state of your process. Obviously, even when the process is syntactically correct, more modeling work remains. In some cases, we need to apply additional detailed analysis of the model. In other cases, elements must be adjusted to ones that are recognized by the execution engine. Additionally, the process model must accommodate the true nature of the technical characteristics of the interactions with the infrastructure.

First, to be considered execution-worthy the full picture of the process model should be completed. The 'happy path[1]' must be detailed with exceptions, escalations, and compensations. Frequently, these more complex details can obscure the intent of the process from the business community. Here the modeling team might consider maintaining separate models. Also, there are points at which processes become so unwieldy as to obscure the fundamental business nature of the process.

Next, the nature of the execution semantics of the model, as described in chapter 14 of OMG's BPMN 2.0 Specification, should be evaluated against the intended nature of the process. Execution semantics accurately prescribes how a proper BPMN engine will interpret the configuration of model elements. This description is not just the functioning of gateways and conduct of activities—there are important characteristics that connect elements of the BPMN data model with the engine. The conforming engine launches process instances and controls their internal states precisely as they are described by these 'semantics'. There are BPMN elements that can be optionally ignored by the engine. These include manual and abstract tasks, ad-hoc processes, and others. The modeling team should familiarize themselves with specifics of the target engine in this area.

The specification's execution semantics uses the token concept. Tokens traverse the sequence flows and pass through the BPMN. For some elements, especially gateways and implicit splits and merges, the execution semantics describes how the token defines the behavior of an executing process.

[1] A default scenario featuring no exceptional or error conditions

In most cases, the semantics of a category of elements is hierarchical. A looping sub-process inherits the semantics of the sub-process; a script or service activity inherits the semantics of the activity. Further, there are important aspects of the BPMN's XML data model that are critical for understanding what elements do. That said, most of the elements should be obvious to practitioners —for example, exclusive gateways have conditions, sub-processes have input and output data sets, multi-instance sub-processes have counts.

Completing the Process Model for Execution

To complete a BPMN model, anticipated exceptions must be accommodated by either:

- Catching events at the boundary of the sub-process and directing them to activities that can correct, waiting for an alleviation of the constraint, or trapping the condition;
- Adding an error condition to gateways that directs anticipated errors; or
- Referencing a standard sub-process at the escalation of any error.

These exceptions are determined by the business problem that we are solving, the nature of the internal and external services that we are connecting to, and the characteristics of the infrastructure that hosts the executable services. In addition, next to optimizing a process efficiently, managing exceptions is one of the clear benefits of the business process approach.

For example, in the BPMN fragments below, an error checking message is passed through a device cloud[2] interface to a device on a cellular network that might be 'offline' for various reasons.

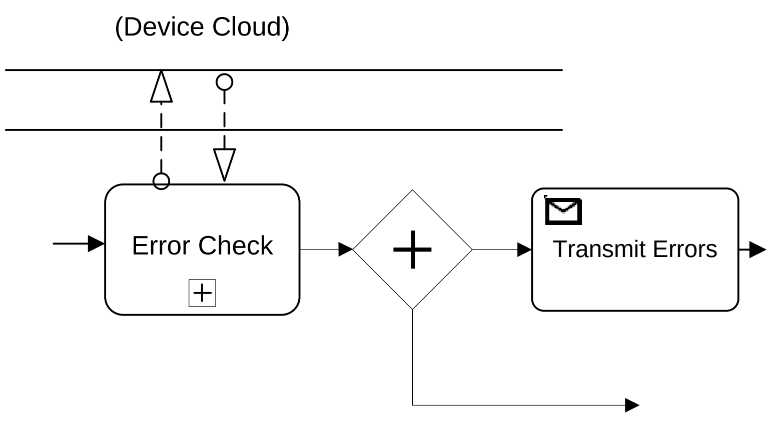


Figure 1: An analytical representation of communications to devices in the device cloud through an error checking sub process

[2] A Device Cloud is a PaaS that is usually hosted by a third party that commissions, provisions, and communicates with the IOT devices. IOT solutions often involve millions of devices with many types of device profiles.

Here, in the earlier phases of modeling, a 'happy path' level of effort was completed and now that the model is to be executable, exceptions must be accommodated. Business and technical exceptions can occur here. For instance:

- In business exceptions, a device might not have been properly commissioned and a special exception process might need to be started.
- In technical exceptions, security equipment might detect a man-in-the middle attack.

These specific known errors should be handled and might entail the modeling of additional workflows.

As mentioned earlier, the execution semantics of BPMN normally preclude the use of abstract tasks. So, abstract tasks (not shown in Figure 1:) should be replaced with scripts, services, and human tasks. Message tasks might be replaced with service calls or scripts where possible:

- A service task can communicate with other processes and services. In execution, we use the service task when a process must invoke an external service or process. Typically, this denotes a call to WSDL or RESTful services.
- The script task manipulates the values of data objects within your process. Typically, a gateway might direct the flow of the process. The script task is used to denote this change in the business process. For instance, you might set the default or lookup values of data objects at the start of a process.

For example, Figure 1: 'message' to and from the device cloud will require analysis in the execution phase. These services generally occur across four layers of infrastructure: The corporate SaaS, where user applications are hosted, a device cloud, a computing gateway, and a network or grid of connected sensors and devices. Communications at the first layer from the SaaS platform are generally performed over HTTP to REST services — hiding the details of the actual message. Com from the device cloud to the devices is frequently done with a message queue such as MQTT[3]. While our device cloud hides the complexity of the queue, the exceptions that they handle should be reflected in the exceptions of the process that calls the REST services.

The fragment from Figure 1: might entail the addition of exceptions for error and timeout handling. The nature of the error or timeout dictates the nature of the response. As shown in Figure 2, errors can often be corrected or ignored/noted and an activity restarted. In other cases, the activity can be skipped entirely. All these possibilities are easily modeled in BPMN.

In developing the proper tasks for our execution model, modelers first examine the operating characteristics of each task and select the appropriate type of activity, event or subprocess. As

[3] MQTT is a machine-to-machine (M2M)/"Internet of Things" connectivity protocol. See MQTT.org

mentioned earlier, abstract tasks do not execute in BPMN. The executing tasks are selected for their implementation characteristics. For instance,

- The transmit error values uses an enterprise email server, the original analysts-level message only depicted a one-way message;
- The evaluate com error uses a rule shape because it decides what must be done with the stack of errors from the device;
- The handle time error task uses a script because it transforms the values of the time for the next execution;
- The Device Error data object is needed to hold the values created by the process instance.

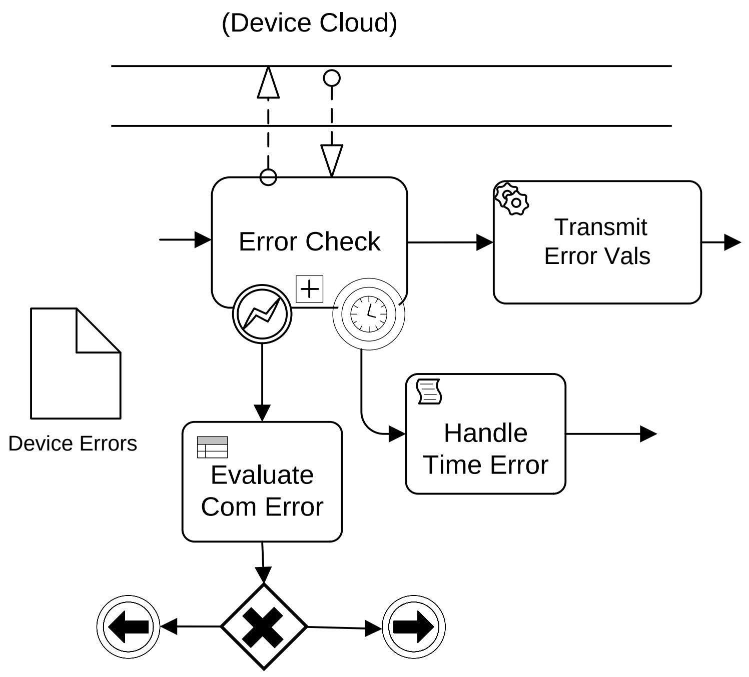
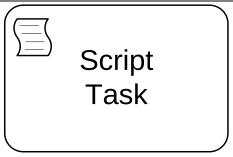
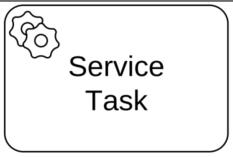
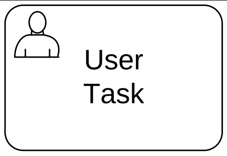


Figure 2: As error handles are added to analytical model, there are more considerations that must accommodate the realities of the technical infrastructure and the intent of the process

As our discussion illustrates, deciding what error or exceptions to trap is a matter of mastery of the details of the infrastructure where the executing process resides. For instance 'legacy' database environments might experience a number of correctable exceptions that are not experienced in the same manner as in modern cloud environments. It is a best practice to provide for a generalized exception, reusable subprocess that traps a broad set of unplanned conditions. These might be managed in an ITIL manner and modeled as event sub-processes. Still, interfaces to external entities, partners, and companies that do not provide service level agreements (SLA's) should be managed accordingly.

In summary, Table 1: presents some of the conditions that might indicate the replacement of an abstract or another task with a more specific task that follows the execution semantics of BPMN. The table also discusses how the specification prescribes task execution and completion for the different task types. It is important to match the systems operational characteristics with the proper task type.

Each task type is a type of activity. All Activities share common attributes and behavior such as states and state transitions. BPMN defines how the activities must move from instantiation through the potentials for interruption, compensation, or termination (if applicable). The outcome is that the process engine might mark the activity as completed, failed, compensated, terminated or withdrawn.

Shape	Example Usage	Design Scenario	Example Execution Semantic
	Change values of data objects at the point of a process. Or change the values of data objects outside of another flow object. Typically done in a programming language such as Xpath, Jscript or Visual Basic(.net).	Populate a complex business object from data input into a process. Transform one business object into another.	Upon instantiation, the associated script is invoked. Upon completed, the task is marked completed.
	Analogous to message send and receive tasks or the event messages throw and catch; however synchronously invokes processes and services. With this task, the process waits at the service task until a response is returned.	A message queue when the response is immediately expected. An email sent to an enterprise mail server. A message on a JMS pipe.	The service is called upon instantiation, if the service fails the activity is marked as failure.
	Process participants interact with forms of various types. The user interface include the widgets, field prompts that process participants see, are populate with the tasks.	Web screens, mobile applications that are placed on a 'task list'.	Upon instantiation, the User Task is distributed to the assigned person or group of People.

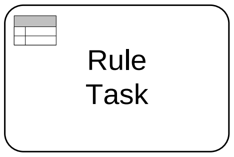
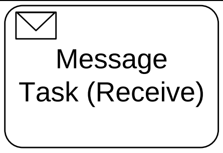
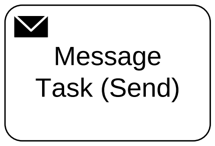
Shape	Example Usage	Design Scenario	Example Execution Semantic
	The Rule Task denotes the call of a rules engine.	Apply business rules that decide, what is the next task to perform, who should participate, what procedure should be followed.	Upon instantiation, the associated business rule is called.
 	Where asynchronous ‘throwing’ and ‘catching’ is needed to coordinate activities	Process the outcome of a participant’s activity	Upon instantiation, the associated Message is sent and the Send Task completes. If the task is a type, the ‘receive’ task waits for the message to arrive. When the Message arrives, the Receive Task completes.

Table 1: Replacements, Corrections and Variations in BPMN Models

The intent of the BPMN 2.0 designers with respect to the different task shapes is clearer when they are considered as notation that describes their execution.

Synchronous and Asynchronous Service calls

In BPMN models there are patterns for synchronous and asynchronous communications. Consider the service call for transmitting error values in Figure 2. The answering message arrives synchronously. If a response arrives later, the communication is asynchronous. Message events commonly represent asynchronous communications.

Yet, not only should the nature of the interfaces to systems be understood as in the case of the message queuing for the device cloud, the synchronicity of the process must be coordinated with activities. Processes request important services and must wait for their completion:

- Inspection of materials in supply chains often need certified lab results.
- Customers provide critical information for loan applications.
- Supply chain partners provide finished goods and services.

The process that orchestrates these must interact with the requested service in the proper way. When developing deeper understanding for execution, modelers make adjustments to the sequence of activities so that services are either synchronously or asynchronously gathered. There are two aspects to these scenarios; although the communications appears asynchronous at the business level, the API’s to these services might be technically synchronous services.

In an asynchronous service, the answer to a request can be delayed. This can present a time window for other process work to be performed. In a synchronous service, the sender waits for a response. The response should be rapid on a process time-scale; however, there are circumstances where the wait is unpredictable. This applies to human and system participants and alike.

Synchronous communication is simple— a response is returned, directly in an application interface. Asynchronous styles are more complex, delayed responses must be assigned to the waiting process instance. This is known as defining a correlation condition. The BPMN specification describes how messages need to reach a specific process instance with a correlation to identify the particular instance.

Correlations

A correlation is the connection of an event or message with a process instances. Designers need to properly define the correlation, so the process engine can assign incoming messages to the correct process instances. The execution semantics of "instance routing" proscribe the creation of a correlation key as an expression.

There are a number correlation approaches to building the correlation key(s). First there is pure, key-based: a data key value is generated for conversations, and the key is used in call and response messages. The process engine correlates the messages with the key. Key-based approaches can also use business attributes, example, order number. The semantics allow the expression to build the key, on the fly, with the technical expressions (correlation property retrieval expression) that define how to find these properties in the process variables or incoming messages. The second approach is a subscription or context-based correlation. It is built atop the key-based approach. At runtime, the correlation key instance holds a composite key that is dynamically calculated and updated whenever the underlying data objects or properties change.

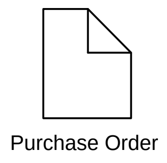
Data Modeling

In addition to completing the process model, the data elements for the process instance must be defined and associated with the process model. Not only is data critical for creating transaction information; it is a critical source of information for decisions and process payloads. In the process diagram, the device error is represented as a data object. The attributes of the data object are developed with traditional data modeling; however BPMN does not depict a data model. Data is accommodated as XML schemas in various technologies. The data definition from the modeling effort can be merged directly into XML code.

Data sets are critical for activities. In the semantics of the activities above, an input dataset must be available for activities to enter an active state. An input set is available if each of its required data inputs is available. Activities wait until the data becomes available. Similarly, output datasets, upon completion, are checked for availability and completion. If no data is output, an error is thrown.

Data Object

The objective of the data object in BPMN is to document the inputs and outputs of process activities. The data object is a rectangle with the upper right corner folded over, as shown here:



The text label for a data object is underneath the shape. Often, the current state of the data object is shown as an attribute shown in brackets under the text label. As the diagram progresses, the state of the data object can easily be read, as displayed in Figure 3.

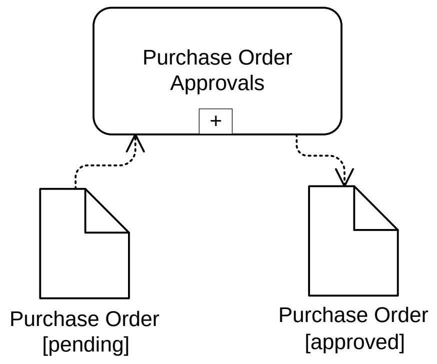


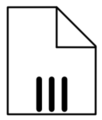
Figure 3: Use of data artifact shapes.

As with the text annotation, the association line connects the data artifact to another shape. Data Artifact shapes are often associated with tasks, gateways, events, sequence lines, or message lines. In message flow, data objects portray the “payload” or content of messages.

A data object can be associated with an activity, which signifies where the data is produced. Associating data artifacts with a gateway can show the data on which a decision is based.

Data modeling is as critical as process, decision and event modeling. A data object is a visual depiction of the modeled subject or “business entity.” A data model may depict an electronic form or a physical document. Data objects provide information about what activities need to be performed and/or what they produce. For instance, an inventory manager might requisition special items. The “requisition” would be a data item. Input and output data is a formal part of the BPMN 2.0 specification and affect execution.

In some cases, the data object denotes a collection of a data type. It uses the same base shape, but adds the multiplicity symbol: three vertical bars. For example, a set of contract documents could be illustrated with the collection symbol.



The data objects can show direction of document flow. For example, are the contracts an input to the "Legal Approval" activity or are they an output? This can be accomplished in several ways. First, the annotation association lines can have an arrow pointing to the direction of flow, as in figure 4.

The data object shape allows for additional annotation (an arrow) showing whether or the data elements are being (output) sent or received (input). Similar to the event shapes, the white (empty) arrow means receive and the black (filled) arrow means send. Figure 4 shows how these shapes are used, and how they can add detail to a diagram.

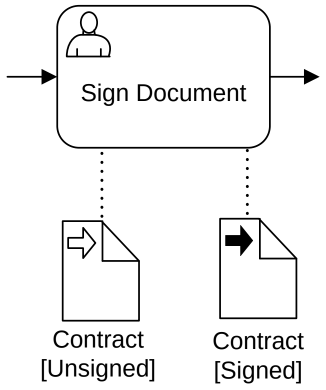


Figure 4: Data Artifacts as inputs and outputs to activity

The input and output annotations on the data objects can also be used in conjunction with the collection symbol. Therefore, there are six types of data object shapes:

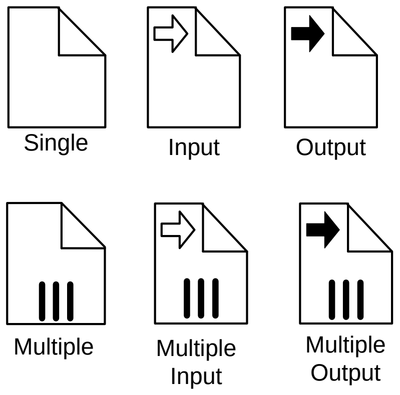


Figure 5: table of data object types

Input data object types provide detail to the activities of the process. An input or output data object references a schema element and creates the 'input and output specification'. When developing an executable process, a elements within a schema in XML and can be referenced by the data object. The specification of the input and output models the data behavior of the process.

Boolean conditions in gateways can also reference the data elements.

Data Stores

The data store artifact shown in figure 6.5 is similar to a database symbol used in modeling notations other than BPMN. The data store is a concept that represents a permanent place for data storage. For the analyst, the important detail is the data associated with permanent storage.

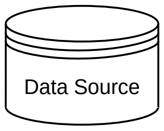


Figure 6: Data Store Artifact

The data store inherits all the data that is associated with the flow. This clearly indicates that the data is available to other people and systems outside of the process.

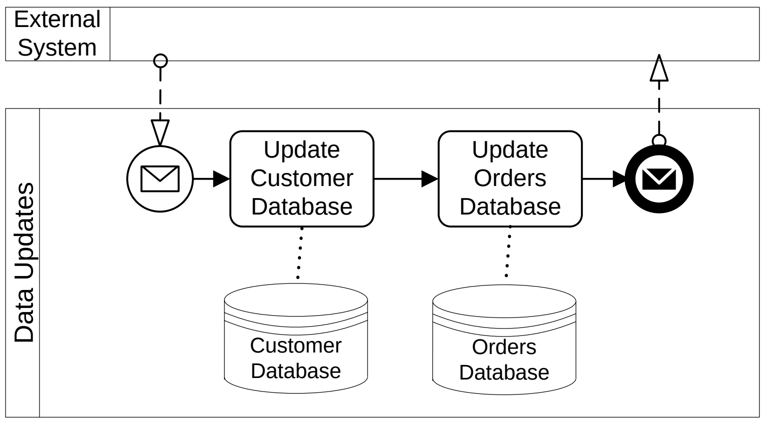
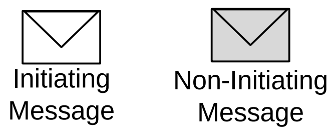


Figure 7: Data Source Artifact used to denote persistent storage.

The Data Source artifacts in figure 7 illustrate the proper use of this shape. This process interfaces with an IT business process.

Message

BPMN provides two message shapes for use as data detail. The message shape is the same envelope-like shape that is used inside a message event; and there are two forms of the message: a white shape for a message that initiates a process and a lightly filled message for 'non-initiating' messages. The two messages are shown below.



Message artifacts can be associated with any activity, event, or messaging flow. They cannot be associated with gateways or sequence flow. The direction of message flow can be shown by associating the shape with a message flow line.

As a process is modeled for the purpose of execution, the message becomes more important. Messages flows need to be sufficiently detailed to permit alignment of participants and callable processes. Messages reference a data type, according to the 'Item Definition' in the associated schema.

Figure 6.7 shows the usage of the message shape. The association lines (dotted) are used to create the relationship between the message, and where the message is used. When the manager in Figure 8 sends a *work request message*, it is received as a process start event. This is an initiating message so the envelope is white. The response task is in the work queue of the worker. When the worker begins the task, it becomes active. After the task is completed, the send notification event occurs, which sends the *completion notification* message. The completion message is shown as a non-initiating message with light shading.

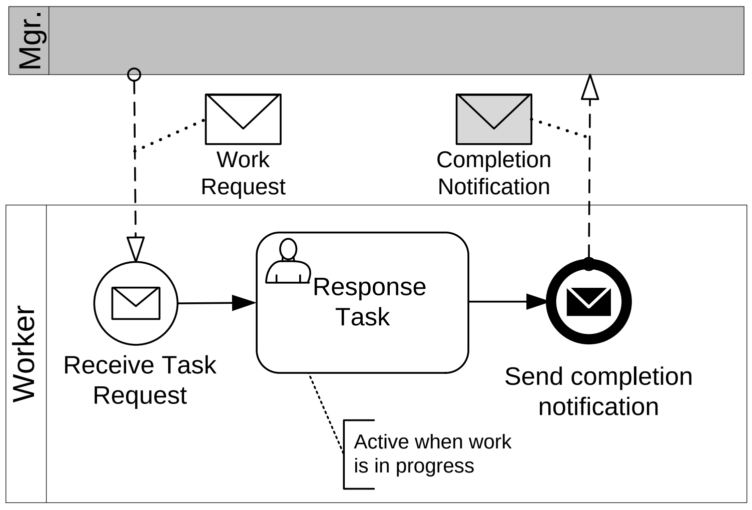


Figure 8: message artifact shape usage

Execution Semantics

Execution semantics precisely define how the BPMN 2.0 Specification expects an engine to translate the structure of a valid BPMN diagram into a process instances that are started, stopped, and cancelled. They also define how activities and events are interpreted. There are two primary concepts that explain how the engine executes the process model:

- Process instance,
- The Token Passing Model

A process engine generally assigns a computer server's resources to a process instance. These resources include one or more threads, CPU memory, disk space and perhaps utilities for parallelism and hardening and fail-over. On top of all this, there is usually a unique identifier associated with the process instance. The BPMS system will use this identifier so that if can internally correlate messaging and track the state of data variables in their various scopes. Generally there is a process instance for each iteration of a process: one per customer order or per insurance claim. However, there are some cases where a single process instance might be used to monitor events or as an overseer.

Within the running instance, the token is a tool for modeling and understanding the behavior of the internal sequence flow of processes, particularly at sequences and transitions, both implicit merges and joins and at gateways. Process semantics describe how BPMN elements interact with tokens, as they flow across the sequences of elements of the process instance. So, once you understand token passing, many of the semantics of execution are self-apparent —the definition of the shape describes how the engine is expected to execute the orchestration of activities. For instance, parallel and inclusive gateways create and consume multiple tokens and exclusive elements create and consume a single token.

Implicit Splits and Merges

By convention, a multi split can be equated like an explicit parallel split. If gateways leaving and preceding activities with multiple (>3) sequences are not used then the specification defines a convention for the evaluation. First, multiple outgoing, unconditional sequence flows behave like a parallel gateway split. As shown in figure 9 below, multiple outgoing sequences with conditions behaves like an inclusive split. A mix of multiple outgoing sequences with and without conditions is considered as a combination of a parallel and an inclusive split.

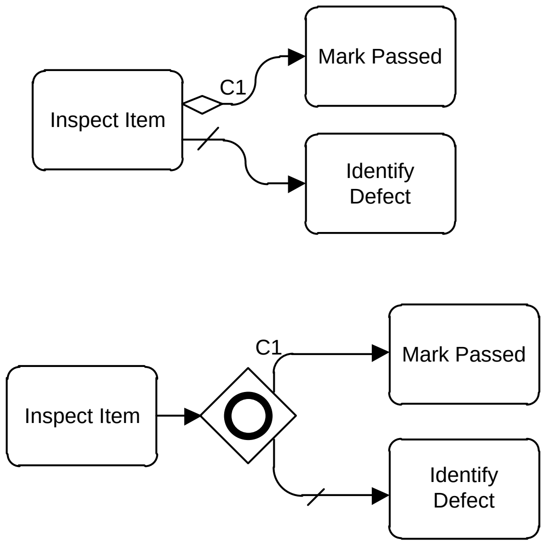


Figure 9, these two BPMN fragments are equivalent because 2 outgoing sequences with conditions behaves like an inclusive split.

As shown in the figure below there is a convention for multiple incoming flows. Similarly for implicit joins, called uncontrolled flow in the specification, the presence of multiple incoming sequences behaves as an exclusive gateway. To eliminate the ambiguity of these conditions it is suggested that gateways (other than Exclusive) should be explicitly included in the process flow. We suggest that even the exclusive gateway be used so that your diagrams are easier to understand.

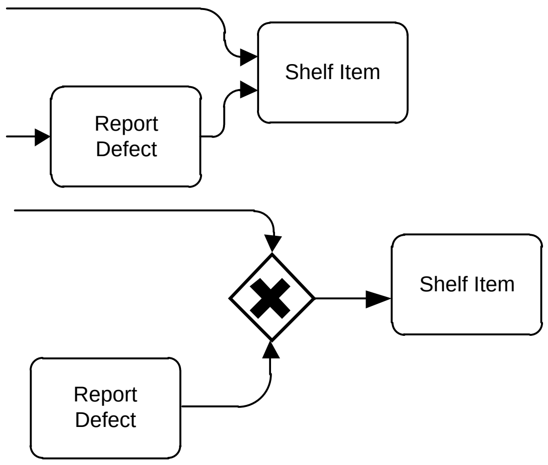
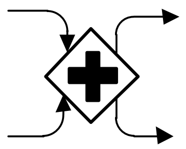
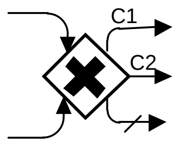
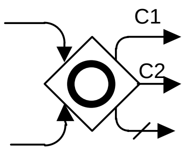


Figure 10, again, these two BPMN fragments are equivalent because 2 incoming sequences behaves like a database exclusive merge

In the process of moving an analytical model to execution, implicit splits and merges should be examined for their conformance to the expected business model. Because of the assumption that multiple merges behave as a data based inclusive merge, it is common practice to use this in models. Yet, assumptions can change over time and additional details can be placed on the process model that alters the original.

Understanding the Execution Semantics of the Gateways

The combination of the token passing model and the semantics on parallel, inclusive and exclusive splits and merges defines the universe of how BPMN expects activities and events to be orchestrated within a process. Table 2 below summarizes the semantics of the gateways and the nature of the merging and splitting of flows after the elements:

Gateway type	**Inbound-merging behavior**	**Shape Graphic**	**Outbound-Sequence splitting behavior**	**Operational Semantics**
Parallel Gateway (Fork and Join)	Synchronize multiple concurrent branches, i.e. wait till all arrive		Spawn new concurrent threads on parallel branches, each get a token	Activated when at least one token on each incoming sequence, produces exactly one token & concurrent thread at each outflow
Exclusive Gateway (Exclusive Decision (data-based) and Exclusive Merge)	Pass-through for a set of incoming branches		Activation of exactly one out outgoing branch	The first, true condition determines the sequence for the token and no other conditions evaluated. If none true the default branch is taken. Exception is thrown when none is evaluated.
Inclusive Gateway (Inclusive Decision and Inclusive Merge)	If the condition of the branch is 'true' it is a 'true' branch. Synchronizes 'true' branches of concurrent incoming branches, wait for the 'true' tokens to arrive		Spawn new concurrent threads (with a token) on parallel branches that have branch conditions that are true- a 'true' branch	The Inclusive Gateway is activated if: all incoming sequences with tokens have arrived. All conditions that evaluate to true create a thread and token, unless none are true then the default is taken. Exception is thrown when none is evaluated.

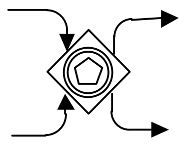
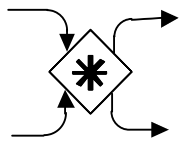
Gateway type	Inbound-merging behavior	Shape Graphic	Outbound-Sequence splitting behavior	Operational Semantics
Event-based Gateway (Exclusive Decision (event-based))	Pass-through semantics for incoming branches		Exactly one of the outgoing branches (with the triggered event) is activated afterwards	When used at the Process start as a Parallel Event Gateway, only message-based triggers are allowed
Complex Gateway (related to Complex Condition and Complex Merge)	Complex synchronization behavior, in particular race situations, similar to the		Each has a Boolean that determines what sequence receives a token	See discussion below

Table 2: Execution Semantics for BPMN gateways

In execution semantics, parallel execution means that the engine executes the process token on a parallel thread. This is true for parallel, inclusive and complex gateways. This can be an advantage to clustered environments and massively parallel server environments.

Origins of the Complex Gateway

When there are a series of paths to be executed in parallel, a complex problem arises when the need to cancel execution of one or more arises out of the execution of another. This is known as 'dead-path elimination'. The complex gateway was, in part, an attempt to model the dead path elimination (DPE) capabilities of BPEL. To accomplish this, complex gateways enable a process to synchronize N paths from M incoming transitions (N<=M). With the complex gateway, parallel (or inclusive) branches converge and outbound activity is activated once the incoming branches are complete. The result of the rest of the M minus N branches is ignored. Execution is blocked till the M incoming branches have been triggered and unblocked when their execution is completed.

Consider a use case: to declare an executive review of a product line, a business process needs to check concurrently whether the product has high liabilities, is subject to frequent quality post-manufacture returns, and has unsatisfactory internal quality checks. The outcomes of these checks is to be evaluated in such a way that as soon as two of these checks pass and the product doesn't have any high liabilities, a clean product state is declared, and the third check is aborted. This pattern is a special case of the N/M join.

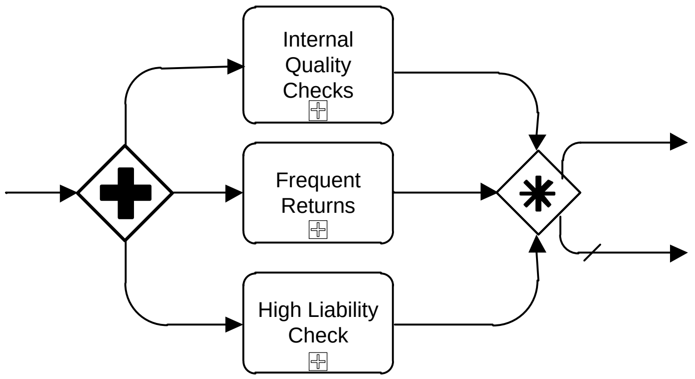


Figure 5, an example of the use of a complex gateway.

The three steps are run in parallel as threaded by the parallel split. The outcome of responses coming from each of the sub processes is evaluated by the complex gateway. This is an example of a 2 of 3 join with one mandatory condition to be met, according to the execution semantics, if the frequent returns subprocess and the High Liability Check passes, then the Internal Quality Checks would be cancelled.

You can use complex gateways to explicitly skip activities. This is simple to denote as a business rule because, in essence a decision is being made concerning the next steps to be taken. In addition, the behavior of the complex gateway is complicated to understand from a visual examination. Moreover, using OR gates that negate the sequences are also hard to understand. The complex gateway introduces a 'reset output' to abort all pending flows and creating a token on the optional reset output flow. This is a very powerful, yet subtle semantic.

Arguably, complex gateways are difficult to understand and generally not needed and can be replaced with a decision flow.

Understanding the life of the instance

A Process is instantiated when one of its Start Events occurs. BPMN defines 7 specific types of start events: None, Message, Timer, Conditional, Single, Multiple, Multiple Parallel. In conventional orchestration, each occurrence of one of these creates a new process instance and one or more tokens. In a process engine the instance is generally identified by a key and the instance data

In process thinking, an instance of a process accomplishes an objective and the instance is uniquely identified by the process engine. In BPMN, instances are started by implicit or explicit start events.

Implementing the Starting Event

A start event starts or 'instantiates' a process. Explicit and implicit start events and each will create a new process instance. The start event is the origin of the initial tokens in a process flow. The instance is a critical concept in BPM. It has its own identity and scope.

There are many considerations when managing processes in an infrastructure. Simple (empty) starts are generally used for governing and monitoring processes and these can be critical to the integrity on the process ecosystem. Large systems must coexist with system monitors such as Tivoli and HP OpenView; BPM must coordinate and centralize operations, these communicate with messaging buses. Empty starts are not generally used; however, the execution semantics imply that the process should start when the BPM server is started.

As mentioned before, the event-based gateway can also start processes. In fact, multiple groups of event-based gateways can start a process, provided they share the same correlation information.

Intermediate Events

The semantics of intermediate events call for waiting for the event to occur. Waiting starts when the point of the intermediate event is reached; for instance this could be an incoming message or signal. After the event occurs, it is consumed and sequence flow continues.

End Events

End events can consume the final token of the process, if there are other tokens then the end even cannot occur. Unless the event is a terminate event, the action completes the activity or sub-process normally and executes the action of the end event type. The completed process state drives many subsequent semantics.

Event Subprocess

Event sub-processes were a major change to the BPMN specification and created a slightly different, albeit intuitive, appearance. These elements have all the characteristics of an event followed by a sequence of tasks and events. An event sub-process is a specialized sub-process that used within a process (or sub-process). A sub-process is defined as an event sub-process when the first event in the sequence is triggered. As shown in Figure 5 below, that event can be interrupting as in a solid line circle, or non-interrupting as with the dotted line circle. Event sub-processes are located within processes or subprocess and are called out by dotted-line frames. The specification also allows the dotted line to be collapsed with the [+] notation.

As mentioned earlier, the process of defining a standard set of exceptions processes can be implemented as an event process. These can be important for creating standard responses to infrastructure errors/ITIL, for instance in the example in Figure 5 below, a generalized error sub-process handles errors. This can be added throughout the execution system to standardize the response to unhandled exceptions in the process.

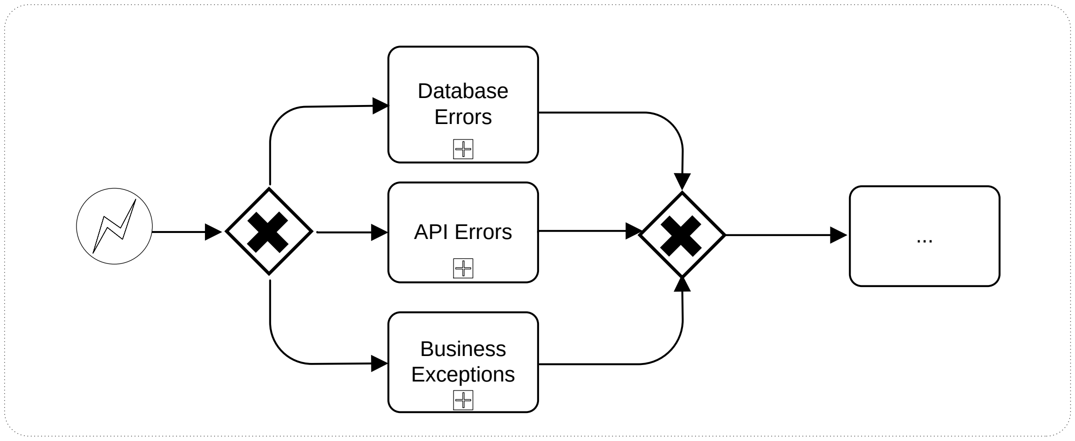


Figure 11 and event sub-process for managing error exceptions

The execution semantics of the event subprocess follow from the characteristics of the starting event. The non-interrupting event triggers the event subprocess while the enclosing process or subprocess is active. The event subprocess will cancel the enclosing subprocess, or it will execute simultaneously for non-interrupting processes. Non-interrupting event sub-processes can be triggered as frequently as needed while the process or subprocess remains active.

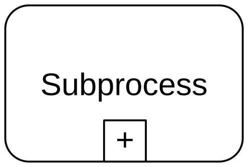
Semantically, the event sub-process:

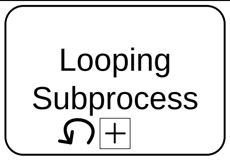
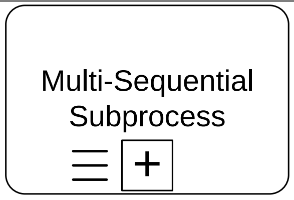
- Can have not incoming or outgoing sequence flow
- The event is started by a trigger, there are 7 triggers in BPMN semantics: Message, Error, Escalation, Compensation, Conditional, Signal, and Multiple.

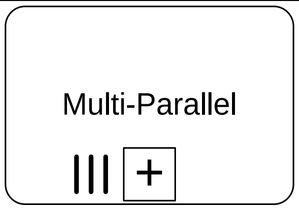
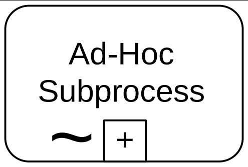
Execution Semantics for Sub-process

As processes are modeled, designers often use the sub-process to create high-level process models. There are 5 different subprocess types that are used here: the sub-process/call activity, the looping sub-process, the ad-hoc sub process, the multi-instance sub-process and the compensation sub-process.

A Sub-Process encapsulates a process and an Activity, modeled by inner elements of Activities, Gateways, Events and Sequence Flow. Once a Sub-Process is instantiated, it acts and behaves as like a normal Process. Most process engines identify the subprocess separately and can display statistics and data. The instantiation and completion of these vary by the type. Table 3 presents the execution semantics for all the sub-process types in BPMN.

Shape	Example Usage	Design Scenario	Execution Semantic
	Abstract an objective in high level diagram, objectives that obviously have a number of courses of actions	Identifiable objectives within a process. For example, a contract award subprocess	Started by a Flow token, the sole empty Start Event, gets a token upon instantiation; activities

Shape	Example Usage	Design Scenario	Execution Semantic
		would require a number of steps in a mature organization	and gateways without incoming Sequence flow get a token, A thread is instantiated for each Start Events that flows from outside, complete when all tokens consumed.
	Execute a process against a list or a set. Acts as a wrapper for an inner Activity that can be executed multiple times in sequence.	For instance, loop through a list of business objects, invoice details or customers. All standard notions of looping are supported, i.e. for, while, until, etc.	The same as the sub-process semantics, executes the inner activity as long as the loop condition evaluates to true. The 'test Before' attribute is decides when the loop condition is evaluated. The loop maximum attribute bound the iteration what can be unbounded when unset. As with any subprocess, when the activities are complete and the condition is met, a token is generated for the outgoing sequence.
	Launch a set of sequential tasks that acts as a wrapper for an Activity which has multiple instances spawned sequentially	For processing in sequence, where resources might be constrained by the resource sizes	The number of instances to be generated is evaluated once. There is a completion condition that is evaluated once an instance completes, when true the remaining instance are completed and a token is generated. There is a behavior attribute that defines when instances can throw events

Shape	Example Usage	Design Scenario	Execution Semantic
	Launch a set of sequential tasks, that acts as a wrapper for an Activity which has multiple instances spawned in parallel or sequentially	Activities that might need to use multiple resource to manage loads, queuing channels	The same as the sequential sub-process except the inner activities are execute in sequence
	Human selected sequence of related activities	Unordered list of tasks required to complete an activity	Activities within the ad-hoc sub-process are not all connected by sequences; intermediate events must have outflowing sequence. Upon execution all activities that have no incoming sequence are 'enabled'. If parallel, another enabled Activity can be selected for Execution. This is not Sequential

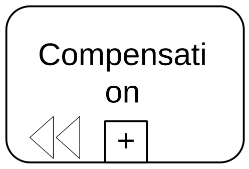
	A compensation handler is a set of activities not connected to other portions of the BPMN model.	Transaction and database record are written that must be removed because a portion on a sub process was not completed	Parent activity must be completed, when triggered the subprocess is run and scope of original activity is restored for reversing the actions of parent. Compensations are performed in reverse order of the process, according to the type of sub-process that is compensated

Table 3: Execution Semantics for the types of BPMN Sub-processes

Threads or process instances can consume many resources. If during the business analysis design period a sub process was identified that only needs a limited numbers of activities or gateways to complete, you might consider eliminating the sub-process and moving the activities outside the sub-process boundary. Another consideration that can retain the sub-process is the need for a separate enclosing scope.

Compensations

Compensations reverse steps (activities, database records) that were already completed, because their results need to be reversed. Active activities cannot be compensated, but rather need to be canceled. Yet an outcome of cancellation can be compensation of already completed activities. Compensation is performed by a compensation handler. This can either be an Event Sub-Process (for a Sub-Process or Process), or an associated compensation Activity (for any Activity). In case of a Sub-Process, its Compensation Event Sub-Process has access to the parent sub-Process data at the time of its completion (“snapshot data”).

Compensation is triggered by a throw Compensation Event, usually an error handler, a part of a cancellation, or another compensation handler. This event also specifies the activity for compensation. There can be a hierarchy, or nested hierarchy of compensations. The order of the compensation is reversed, to the extent that the sub-process type permits.

Summary

Moving a process into execution can be an extensive effort and require a precise understanding of not only BPMN but the nature of the environment, systems and workflows. From this discussion one can discern the steps needed to complete this activity. These include:

- Complete the happy path by adding error, exception, escalation and compensation handlers.
- Create process data model, integrate other data modeling efforts.

- Gather and develop an understanding of the nature of the systems and interfaces at the edges of the model, adjust synchronous and asynchronous strategies.
- Develop a standard approach to handling exceptions and errors, consider creating a standard event-driven sub-process.
- Trace the tokens across the final model. Correct the sequences of the path according to diversions from the needed behavior.

Bio-Tom Debevoise

Tom Debevoise is the Executive Director of Cloud Solutions at Black Pearl Development, Inc. He is a practitioner in advanced architectures including Internet of Things (IOT) for Smart Energy Systems Integrations, mobile solutions for Business Process Management, Business Rules Management, ERP, SOA, Business Intelligence, and Information Technology. In addition to his focus on the IOT, Mr. Debevoise has deep experience in development of enterprise information systems using Business Process Management, Decision Management, ERP, and data warehousing. Debevoise is the author of 3 books: The Microguide to BPMN, Business Process Management with a Business Rules Approach and The Data Warehouse Method published by Prentice-Hall. Mr. Debevoise provides visionary roles in complex projects for diverse clients, and has expertise in the lighting, fleet management, petroleum, telecommunications, accounting, financial management, and engineering and scientific fields.

He can be reached at tom@blackpearldevelopment.com