5. Connecting Objects

This section defines the graphical objects used to connect two objects together (i.e., the connecting lines of the Diagram) and how the flow progresses through a Process (i.e., through a straight sequence or through the creation of parallel or alternative paths).

5.1 Graphical Connecting Objects

There are two ways of connecting objects in BPMN: a Flow, either sequence or message, and an Association. Sequence Flows and Message Flows, to a certain extent, represent orthogonal aspects of the business processes depicted in a model, although they both affect the performance of activities within a Process. In keeping with this, Sequence Flows will generally flow in a single direction (either left to right, or top to bottom) and Message Flows will flow at a 90° from the Sequence Flows. This will help clarify the relationships for a Diagram that contains both Sequence Flows and Message Flows. However, BPMN does not restrict this relationship between the two types of Flows. A modeler can connect either type of Flow in any direction at any place in the Diagram.

The next three sections will describe how these types of connections function in BPMN.

5.1.1 Sequence Flow

A Sequence Flow is used to show the order that activities will be performed in a Process. Each Flow has only one source and only one target. The source and target must be from the set of the following flow objects: Events (Start, Intermediate, and End), Activities (Task and Sub-Process), and Gateways. During performance (or simulation) of the process, a Token will leave the source flow object, traverse down the Sequence Flow, and enter the target flow object.

- A Sequence Flow is line with a solid arrowhead that MUST be drawn with a solid single line (as seen in Figure 40).
  - The use of text, color, and size for a Sequence Flow MUST follow the rules defined in section 3.3 on page 40.

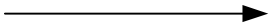

Figure 40 A Sequence Flow

BPMN does not use the term “Control Flow” when referring the lines represented by Sequence Flow or Message Flow. The start of an activity is “controlled” not only by Sequence Flow (the order of activities), but also by Message Flow (a message arriving), as well as other process factors, such as scheduled resources. Artifacts can be Associated with activities to show some of these other factors. Thus, we are using a more specific term, “Sequence Flow,” since these lines mainly illustrate the sequence that activities will be performed.

- A Sequence Flow MAY have a conditional expression attribute, depending on its source object.

This means that the condition expression must be evaluated before a Token can be generated and then leave the source object to traverse the Flow. The conditions are usually associated with Decision Gateways, but can also be used with activities.

- If the source of the Sequence Flow is an activity, rather than Gateway, then a Conditional Marker, shaped as a “mini-diamond”,” MUST be used at the beginning of the Sequence Flow (see Figure 41).

The diamond shape is used to relate the behavior to a Gateway (also a diamond) that controls the flow within a Process. More information about how conditional Sequence Flow are used can be found in in the section entitled “Splitting Flow” on page 135.

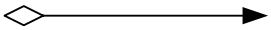

Figure 41 A Conditional Sequence Flow

A Sequence Flow that has an Exclusive Data-Based Gateway or an activity as its source can also be defined with a condition expression of Default. Such Sequence Flow will have a marker to show that is a Default flow.

- The Default Marker MUST be a backslash near the beginning of the line (see Figure 42).

Figure 42 A Default Sequence Flow

Attributes

The following are attributes of a Sequence Flow (Note that this is the complete set and it does not extend the set of common object attributes):

Attributes	Description
Id: ObjectId	This is a unique Id that identifies the object from other objects within the Diagram.
Name: String	Name is an attribute that is text description of the object.
Source: ObjectId	Source is an attribute that identifies which flow object the Sequence Flow is connected *from*; i.e., the Sequence Flow is an outgoing flow from that object. The Source MUST be from the set of the following flow objects: Start Event, Intermediate Event, End Event, Task, Sub-Process, and Decision.
Target: ObjectId	Target is an attribute that identifies which flow object the Sequence Flow is connected *to*; i.e., the Sequence Flow is an incoming flow to that object. The Target MUST be from the set of the following flow objects: Start Event, Intermediate Event, End Event, Task, Sub-Process, and Decision.

Attributes	Description
ConditionType: (None | Expression | Default): None	By default, the **ConditionType** of a Sequence Flow is None. This means that there is no evaluation at runtime to determine whether or not the Sequence Flow will be used. Once a Token is ready to traverse the Sequence Flow (i.e., the Source is an activity that has completed), then the Token will do so. The normal, uncontrolled use of Sequence Flow, in a sequence of activities, will have a None **ConditionType** (see Figure 51). A None **ConditionType** SHALL NOT be used if the Source of the Sequence Flow is an Exclusive Data-Based or Inclusive Gateway. The **ConditionType** attribute MAY be set to Expression if the Source of the Sequence Flow is a Task, a Sub-Process, or a Gateway of type Exclusive-Data-Based or Inclusive. If the **ConditionType** attribute is set to Expression, then a condition marker SHALL be added to the line if the Sequence Flow is outgoing from an activity (see Figure 41). However, a condition indicator SHALL NOT be added to the line if the Sequence Flow is outgoing from a Gateway. An Expression **ConditionType** SHALL NOT be used if the Source of the Sequence Flow is an Event-Based Exclusive Gateway, a Complex Gateway, a Parallel Gateway, a Start Event, or an Intermediate Event. In addition, an Expression **ConditionType** SHALL NOT be used if the Sequence Flow is associated with the Default Gate of a Gateway. The **ConditionType** attribute MAY be set to Default only if the Source of the Sequence Flow is an activity or an Exclusive Data-Based Gateway. If the **ConditionType** is Default, then the Default marker SHALL be displayed (see Figure 42).
(Condition is set to Expression only) ConditionExpression: Expression	If the Condition attribute is set to Expression, then the ConditionExpression attribute MUST be defined as a valid expression. The expression will be evaluated at runtime. If the result of the evaluation is TRUE, then a Token will be generated and will traverse the Sequence--Subject to any constraints imposed by a Source that is a Gateway.
Quantity: Integer: 1	The default value is 1. The value MAY NOT be less than 1. This attribute defines the number of Tokens that will be generated down the Sequence Flow.
Category*: String	This modeler MAY add one or more defined Categories that can be used for purposes such as reporting and analysis.
Documentation ?: String	The modeler MAY add text documentation about the Sequence Flow.

Table 41 Sequence Flow Attributes

Changes Since 1.0 Draft Version

These are the changes since the last publically release version:

- The Id, Source, and Target attributes, within the set of Sequence Flow attributes, were change to the type ObjectId.
- The Condition attribute, within the set of Sequence Flow attributes, was renamed to ConditionType.
- The Category attribute was added to the set of Sequence Flow attributes.

5.1.2 Message Flow

A Message Flow is used to show the flow of messages between two entities that are prepared to send and receive them. In BPMN, two separate Pools in the Diagram will represent the two entities. Thus,

- Message Flow MUST connect two Pools, either to the Pools themselves or to flow objects within the Pools. They cannot connect two objects within the same Pool.
- A Message Flow is line with a open arrowhead that MUST be drawn with a dashed single black line (as seen in Figure 43).
  - The use of text, color, size, and lines for a Message Flow MUST follow the rules defined in section 3.3 on page 40.

Figure 43 A Message Flow

The Message Flow can connect directly to the boundary of a Pool (See Figure 44), especially if the Pool does not have any process details within (e.g., is a “Black Box”).

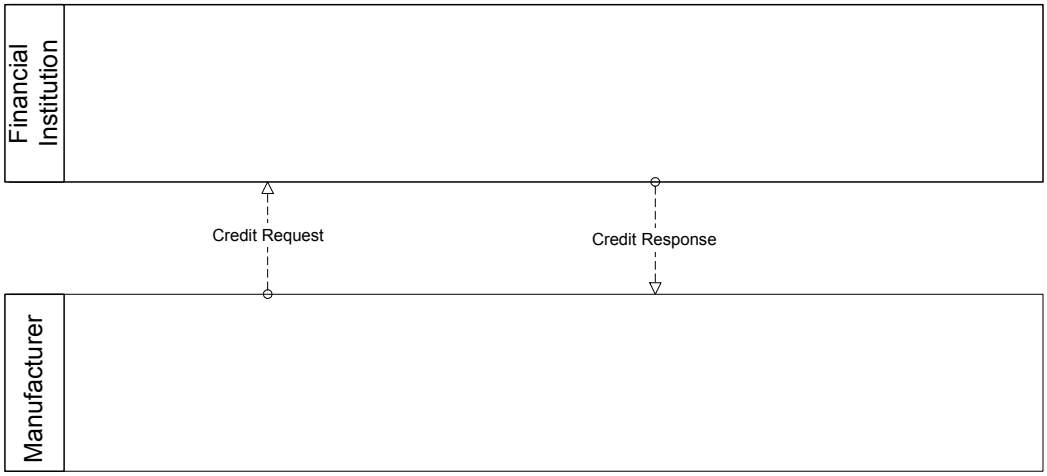

Figure 44 Message Flow connecting to the boundaries of two Pools

A Message Flow can also cross the boundary of a Pool and connect to a flow object within that Pool (see Figure 45).

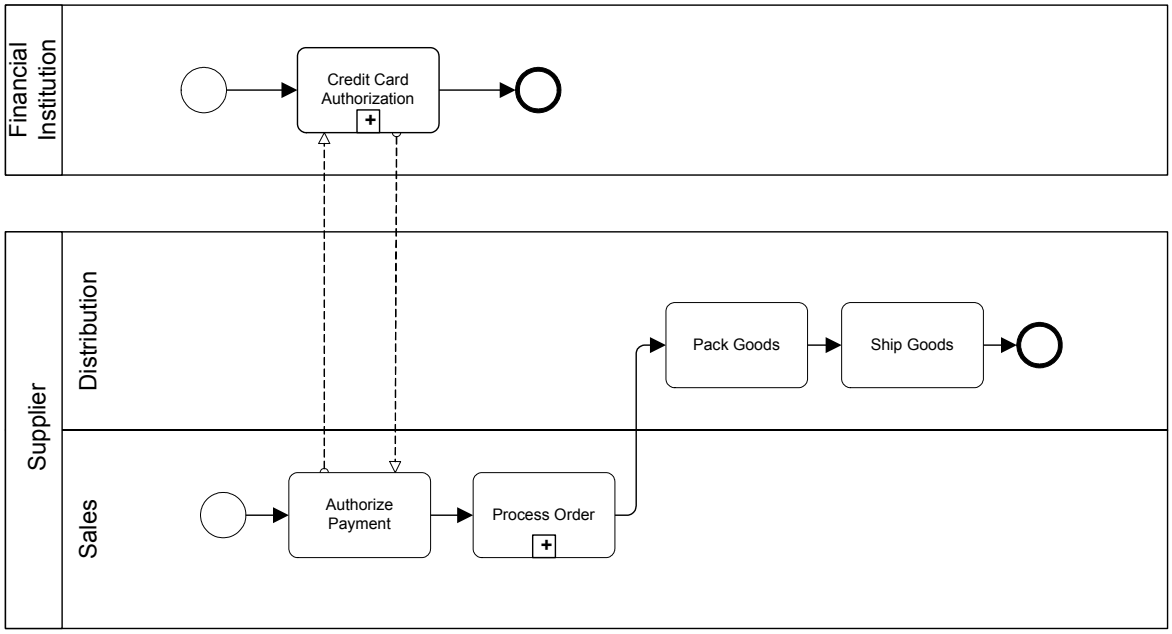

Figure 45 Message Flow connecting to flow objects within two Pools

If there is an Expanded Sub-Process in one of the Pools, then the message flow can be connected to either the boundary of the Sub-Process or to objects within the Sub-Process. If the Message Flow is connected to the boundary to the Expanded Sub-Process, then this is equivalent to connecting to the Start Event for incoming Message Flows or the End Event for outgoing Message Flows (see Figure 46).

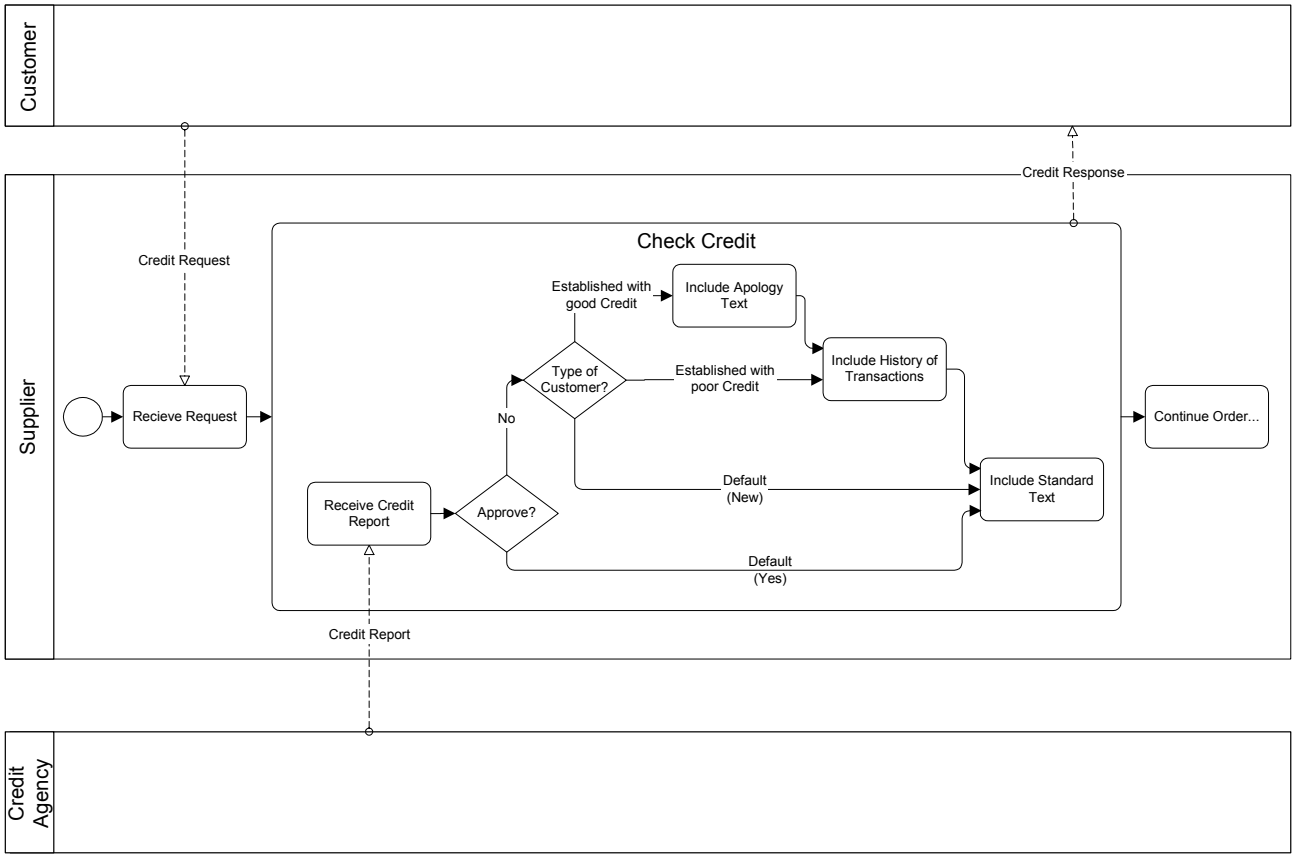

Figure 46 Message Flow connecting to boundary of Sub-Process and Internal objects

Attributes

The following table displays the identified attributes of a Message Flow (Note that this is the complete set and it does not extend the set of common object attributes):

Attributes	Description
Id: ObjectId	This is a unique Id that identifies the Message Flow from other objects within the Diagram.
Name ?: String	Name is an optional attribute that is text description of the Message Flow.
Message ?: Message	Message is an optional attribute that identifies the Message that is being sent. The attributes of a Message can be found section entitled “Message” on page 260.
Source: ObjectId	Source is an attribute that identifies the object the Message Flow is connected *from*; i.e., the Message Flow is an outgoing flow from that object. The Message Flow MAY originate from the boundary of the Pool or an object within the Pool. If the source is an object within the Pool, then the ObjectName MUST identify the Pool and the Object.
Target: ObjectId	Target is an attribute that identifies the object the Message Flow is connected *to*; i.e., the Message Flow is an incoming flow to that object. The Message Flow MAY target the boundary of the Pool or an object within the Pool. If the target is an object within the Pool, then the ObjectName MUST identify the Pool and the Object.
Category*: String	This modeler MAY add one or more defined Categories that can be used for purposes such as reporting and analysis.
Documentation?: String	The modeler MAY add text documentation about the Message Flow.

Table 42 Message Flow Attributes

Changes Since 1.0 Draft Version

These are the changes since the last publically release version:

- The Id, Source, and Target attributes, within the set of Message Flow attributes, were change to the type ObjectId.
- The Message and Category attributes were added to the set of Message Flow attributes.

5.1.3 Association

An Association is used to associate information and artifacts with flow objects. Text and graphical non-flow objects can be associated with the flow objects and flows. An Association is also used to show the activities used to compensate for an activity. More information about compensation can be found in the section entitled “Compensation Association” on page 152.

- An Association Flow is line that MUST be drawn with a dotted single black line (as seen in Figure 47).
  - The use of text, color, size, and lines for an Association MUST follow the rules defined in section 3.3 on page 40.

Figure 47 An Association

If there is a reason to put directionality on the association then:

- A line arrowhead MAY be added to the Association line. (see Figure 48).

A directional Association is often used with Data Objects to show that a Data Object is either an input to or an output from an activity.

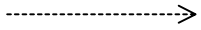

Figure 48 A directional Association

An Association is used to connect user-defined text (an Annotation) with a flow object (see Figure 49).

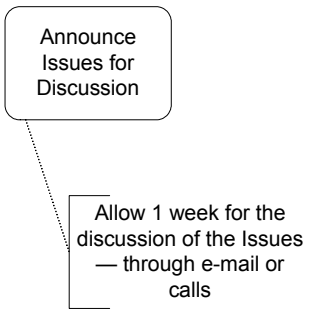

Figure 49 An Association of Text Annotation

An Association is also used to associate Data Objects with other objects (see Figure 50). A Data Object is used to show how documents are used throughout a Process. Refer to the section entitled “Data Object” on page 109 for more information on Data Objects.

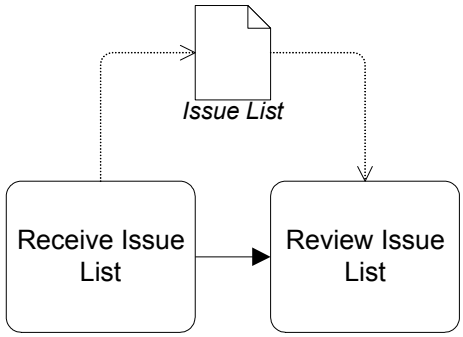

Figure 50 An Association connecting a Data Object with a Flow

Attributes

The following table displays the identified attributes of a Association (Note that this is the complete set and it does not extend the set of common object attributes):

Attributes	Description
Id: ObjectId	This is a unique Id that identifies the Association from other objects within the Diagram.
Name ?: String	Name is an optional attribute that is text description of the Association.
Source: ObjectId	Source is an attribute that identifies which object the Association is connected *from*. The set of objects that an Association MAY connect to are: Pool, Lane, all Events, Task, Sub-Process, Gateway, Sequence Flow, and Message Flow.
Target: ObjectId	Target is an attribute that identifies which object the Association is connected *to*. Associations MUST only connect to Artifacts or Compensation Activities.
Direction (None | To | From | Both): None	Direction is an attribute that defines whether or not the Association shows any directionality with an arrowhead. The default is None (no arrowhead). A value of To means that the arrowhead SHALL be at the Source object. A value of From means that the arrowhead SHALL be at the Target artifact. A value of Both means that there SHALL be an arrowhead at both ends of the Association line.
Category*: String	This modeler MAY add one or more defined Categories that can be used for purposes such as reporting and analysis.
Documentation ?: String	The modeler MAY add text documentation about the Association.

Table 43 Association Attributes

Changes Since 1.0 Draft Version

These are the changes since the last publically release version:

- The Id, Source, and Target attributes, within the set of Association attributes, were change to the type ObjectId.
- The Category attribute was added to the set of Association attributes.

5.2 Sequence Flow Mechanisms

The Sequence Flow mechanisms described in the following sections are divided into four types: Normal, Exception, Link Events, and Ad Hoc (no flow). Within these types of flow, BPMN can be related to specific "Workflow Patterns[1]." These patterns began as development work by Wil van der Aalst[2], a professor at the Eindhoven University of Technology, and Arthur ter Hofstede[3], an associate professor at the Queensland University of Technology. Twenty-one patterns have been defined as a way to document specific behavior that can be executed by a

1.http://tmitwww.tm.tue.nl/research/patterns/
2.http://tmitwww.tm.tue.nl/staff/wvdaalst/
3.http://sky.fit.qut.edu.au/~terhofst/

BPM system. These patterns range from very simple behavior to very complex business behavior. These patterns are useful in that they provide a comprehensive checklist of behavior that should be accounted for by BPM system. Therefore, some of these patterns will be illustrated with BPMN in the following sections to show how BPMN can handle the simple and complex requirements for Business Process Modeling.

5.2.1 Normal Flow

Normal Sequence Flow refers to the flow that originates from a Start Event and continues through activities via alternative and parallel paths until it ends at an End Event. The simplest type of flow within a Process is a sequence, which defines a dependencies of order for a series of activities that will be performed (sequentially). A sequence is also Workflow Pattern #1 -- Sequence[1] (see Figure 51).

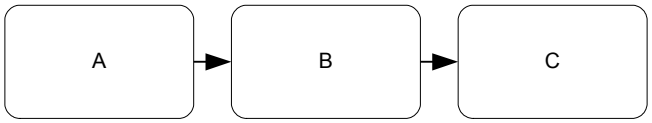

Figure 51 Workflow Pattern #1: Sequence

As stated previously, the normal Sequence Flow should be completely exposed and no flow behavior hidden. This means that a viewer of a BPMN Diagram will be able to trace through a series of flow objects and Sequence Flows, from the beginning to the end of a given level of the Process without any gaps or hidden “jumps” (see Figure 52). In this figure, Sequence Flows connect all the objects in the Diagram, from the Start Event to the End Event. The behavior of the Process shown will reflect the connections as shown and not skip any activities or “jump” to the end of the Process.

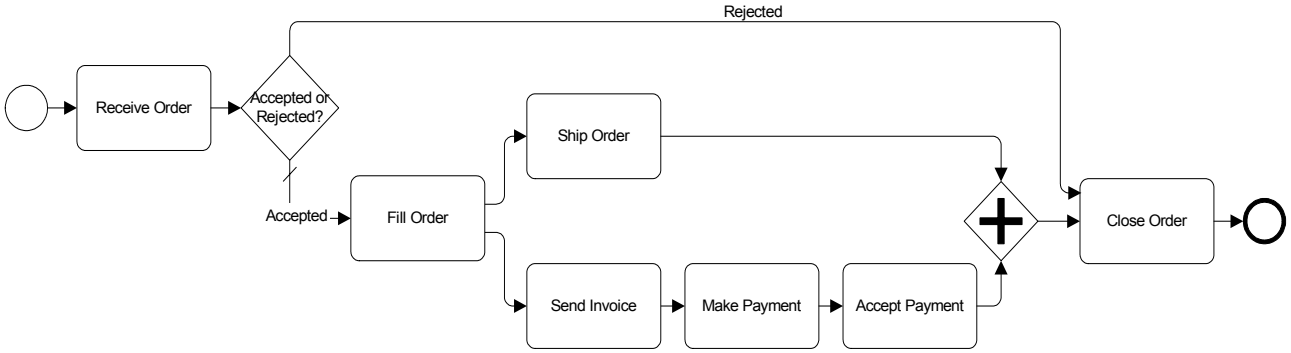

Figure 52 A Process with Normal flow

As the Process continues through the series of Sequence Flows, control mechanisms may divide or combine the Sequence Flows as a means of describing complex behavior. There are control mechanisms for dividing (forking and splitting) and for combining (joining and merging) Sequence Flows. Gateways and conditional Sequence Flow are used to accomplish the dividing and combining of flow. It is possible that there may be gaps in the Sequence Flow if Gateways and/or conditional Sequence Flow are not configured to cover all performance possibilities. In this case, a model that violates the flow traceability requirement will be considered an invalid model. Presumably, process development software or BPM test environments will be able to test a process model to ensure that the model is valid.

1. http://tmitwww.tm.tue.nl/research/patterns/sequence.htm

A casual look at the definitions of the English terms for these mechanisms (e.g., forking and splitting) would indicate that each pair of terms mean basically the same thing. However, their effect on the behavior of a Process is quite different. We will continue to use these English terms but will provide specific definitions about how they affect the performance of the process in the next few sections of this specification. In addition, we will relate these BPMN terms to the terms OR-Split (for split), Or-Join (for merge), AND-Split (for fork), and AND-Join (for join), as defined by the Workflow Management Coalition.[1]

The use of an expanded Sub-Process in a Process (see Figure 53), which is the inclusion of one level of the Process within another Level of the Process, can sometimes break the traceability of the flow through the lines of the Diagram. The Sub-Process is not required to have a Start Event and an End Event. This means that the series of Sequence Flows will be disrupted from border of the Expanded Sub-Process to the first object within the Expanded Sub-Process. The flow will "jump" to the first object within the Expanded Sub-Process. Expanded Sub-Processes will often be used, as seen in the figure, to include exception handling. A requirement that modelers always include a Start Event and End Event within Expanded Sub-Processes would mainly add clutter to the Diagram without necessarily adding to the clarity of the Diagram. Thus, BPMN does not require the use of Start Events and End Events to satisfy the traceability of a Diagram that contains multiple levels.

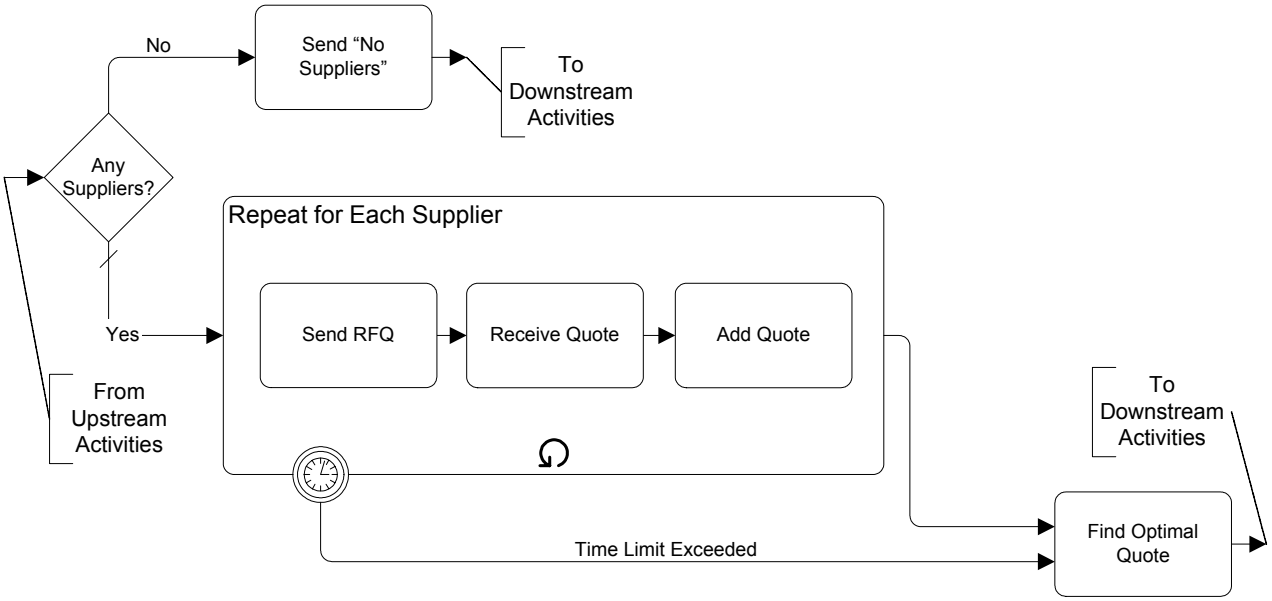

Figure 53 A Process with Expanded Sub-Process without a Start Event and End Event

Of course, the Start and End Events for an Expanded Sub-Process can be included and placed entirely within the its boundaries (see Figure 54). This type of model will also have a break from a completely traceable Sequence Flow as the flow continues from one Process level to another.

1. The *Workflow Management Coalition Terminology & Glossary.* The Workflow Management Coalition. Document Number WFMC-TC-1011. April 1999.

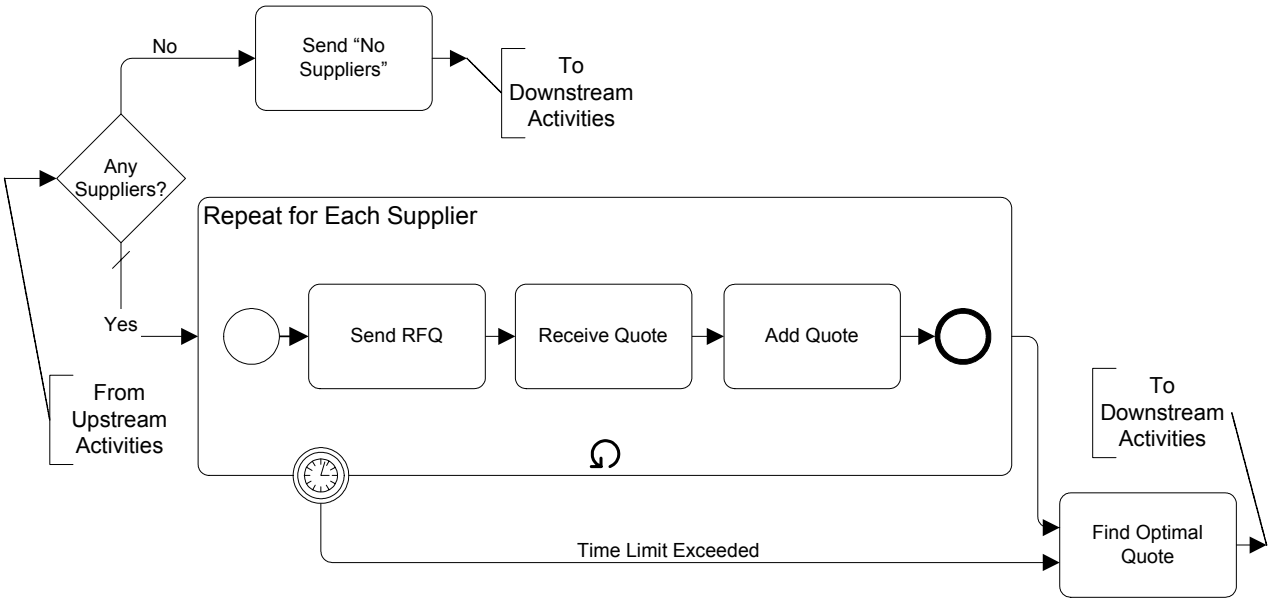

Figure 54 A Process with Expanded Sub-Process with a Start Event and End Event Internal

However, a modeler may want to ensure the traceability of a Diagram and can use a Start Event and End Event in an Expanded Sub-Process. One way to do this would be to attach these events to the boundary of the Expanded Sub-Process (see Figure 55). The incoming Sequence Flow to the Sub-Process can be attached directly to the Start Event instead of the boundary of the Sub-Process. Likewise, the outgoing Sequence Flow from the Sub-Process can connect from the End Event instead of the boundary of the Sub-Process. Doing this, the Normal flow can be traced throughout a multi-level Process.

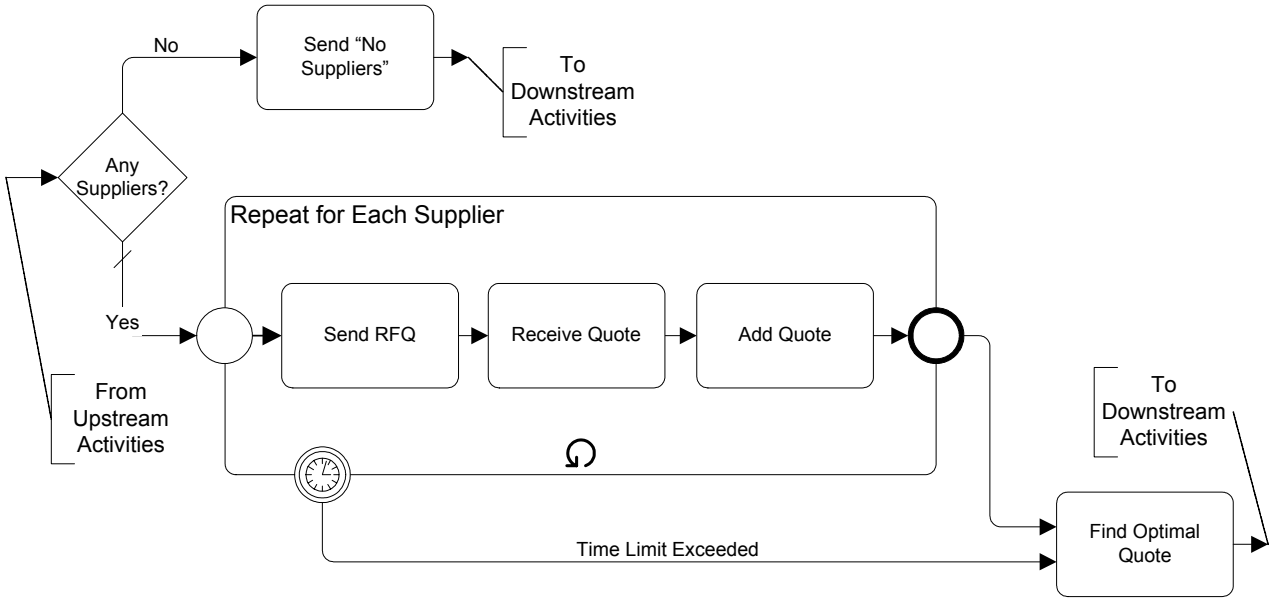

Figure 55 A Process with Expanded Sub-Process with a Start Event and End Event Attached to Boundary

When dealing with Exceptions and Compensation, the traceability requirement is also relaxed (refer to the section entitled "Exception Flow" on page 149 and "Compensation Association" on page 152).

Forking Flow

BPMN uses the term forking to refer to the dividing of a path into two or more parallel paths (also known as an AND-Split). It is a mechanism that will allow activities to be performed concurrently, rather than sequentially. This is also Workflow Pattern #2 -- Parallel Split[1]. BPMN provides three configurations that provide forking.

The first mechanism to create a fork is simple: a flow object can have two or more outgoing Sequence Flows (see Figure 56). A special flow control object is not used to fork the path in this case, since it is considered uncontrolled flow; that is, flow will proceed down each path without any dependencies or conditions--there is no Gateway that controls the flow. Forking Sequence Flow can be generated from a Task, Sub-Process, or a Start Event.

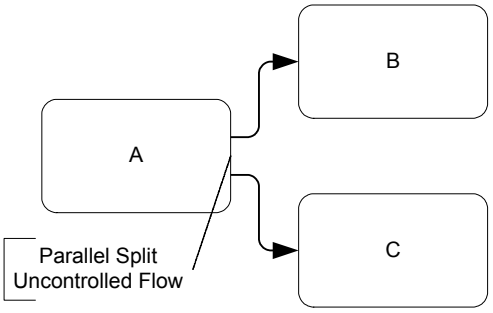

Figure 56 Workflow Pattern #2: Parallel Split -- Version 1

The second mechanism uses a Parallel Gateway (see Figure 60). For situations as shown in the Figure 57, a Gateway is not needed, since the same behavior can be created through multiple outgoing Sequence Flow, as in Figure 56. However, some modelers and modeling tools may use a forking Gateway as a “best practice.” Refer to the section entitled “Parallel Gateways (AND)” on page 101 for more information on Parallel Gateways.

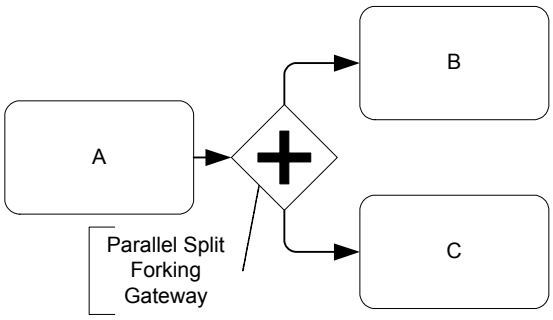

Figure 57 Workflow Pattern #2: Parallel Split -- Version 2

Even when not required as a “best practice,” there are situations were the Parallel Gateway provides a useful indicator of the behavior of the Process. Figure 58 shows how a forking Gateway is used when the output of an *Exclusive* Decision requires that multiple activities will be performed based on one condition (Gate).

1. http://tmitwww.tm.tue.nl/research/patterns/parallel_split.htm

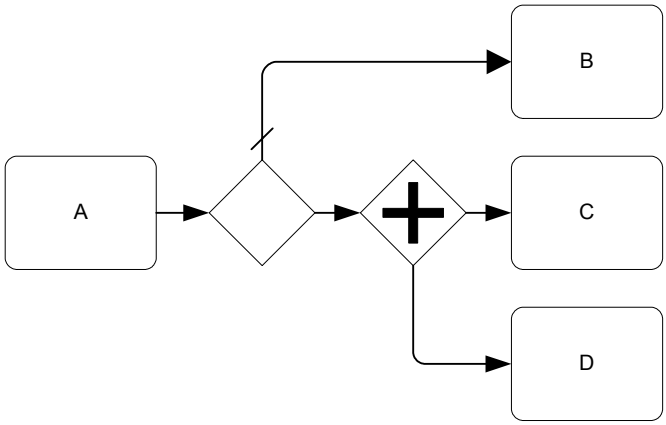

Figure 58 The Creation of Parallel Paths with a Gateway

While multiple conditional Sequence Flow, each with the exact same condition expression (see Figure 59), could be used with an *Inclusive* Gateway to create the behavior, the use of a forking Gateway makes the behavior much more obvious.

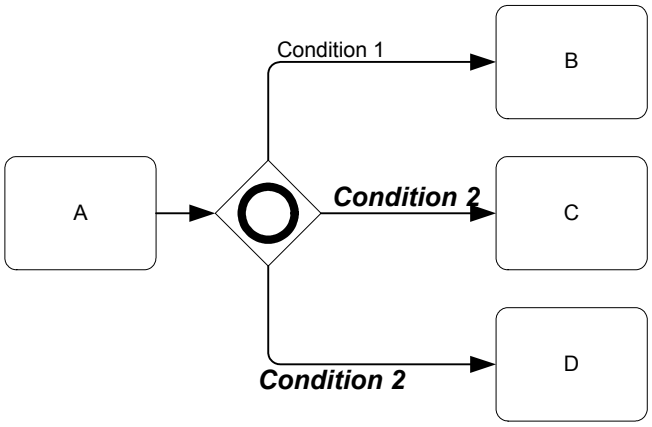

Figure 59 The Creation of Parallel Paths with Equivalent Conditions

This third version of the forking mechanism uses an Expanded Sub-Process to group a set of activities to be performed in parallel (see Figure 60). The Sub-Process does not include a Start and End Event and displays the activities “floating” within. A configuration like this can be called a “parallel box” and can be a compact and less cluttered way of showing parallelism in the Process. The capability to model in this way is the reason that Start and End Events are optional in BPMN.

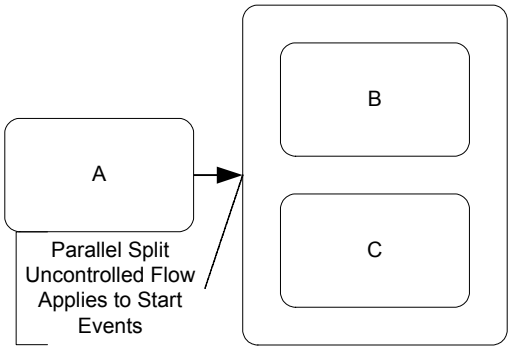

Figure 60 Workflow Pattern #2: Parallel Split -- Version 3

Most of the time, the paths that have been divided with a fork are combined back together through a join (refer to the next section) and synchronized before the flow will continue. However, BPMN provides the flexibility for advanced methods to handle complex process situations. Thus, the exact behavior will be determined by the configuration of the Sequence Flow and the Gateways that are used.

Joining Flow

BPMN uses the term joining to refer to the combining of two or more parallel paths into one path (also known as an AND-Join). A Parallel Gateway is used to synchronize two or more incoming Sequence Flows (see Figure 61). In general, this means that Tokens created at a fork will travel down parallel paths and then meet at the Parallel Gateway. From there, only one Token will continue. This is also Workflow Pattern #3 -- Synchronization[1]. Refer to the section entitled “Parallel Gateways (AND)” on page 101 for more information on Parallel Gateways.

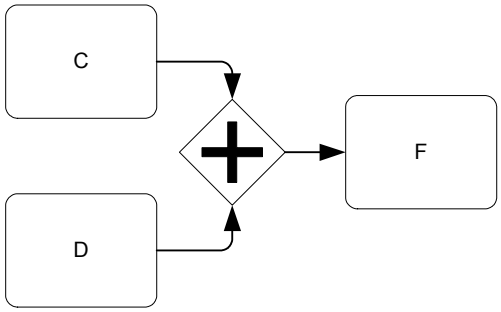

Figure 61 Workflow Pattern #3: Synchronization -- Version 1

Another mechanism for synchronization is the completion of a Sub-Process (see Figure 62). If there are parallel paths within the Sub-Process that are *not* synchronized with an Parallel Gateway, then they will eventually reach an End Event (even if the End Event is implied). The default behavior of a Sub-Process is to wait until all activity within has been completed before the flow will move back up to a higher level Process. Thus, the completion of a Sub-Process is a synchronization point.

1. http://tmitwww.tm.tue.nl/research/synchronization.htm

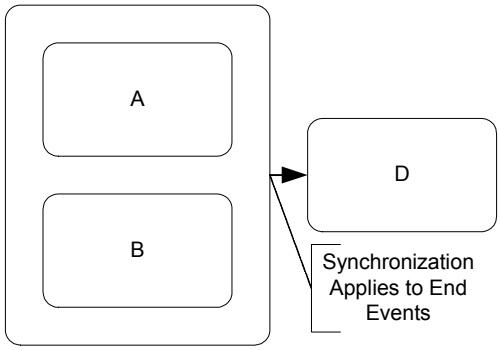

Figure 62 Workflow Pattern #3: Synchronization -- Version 2

There is no specific correlation between the joining of a set of parallel paths and the forking that created the parallel paths. For example, a an activity may have three outgoing Sequence Flows, which creates a fork of three parallel paths, but these three paths do not need to be joined at the same object. Figure 63 shows that two of three parallel paths are joined at Task "F." All of the paths eventually will be joined, but this can happen through any combination of objects, including lone End Events. In fact, each path could end with a separate End Event, and then be synchronized as mentioned above.

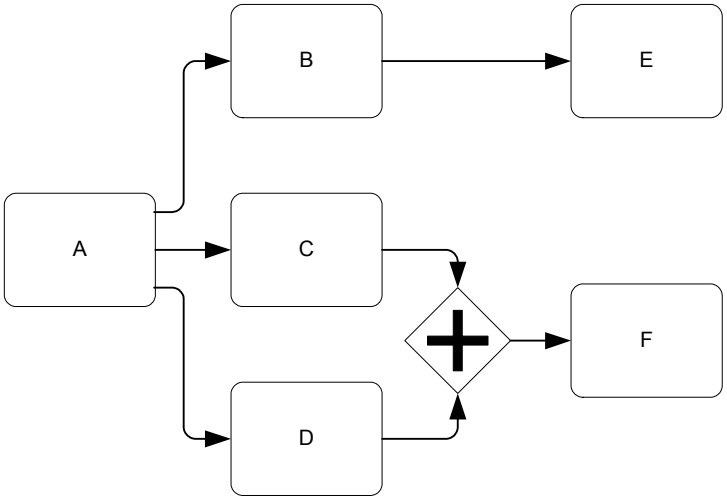

Figure 63 The Fork-Join Relationship is not Fixed

Thus, for parallel flow, BPMN contrasts with BPEL4WS, which is mainly block structured. A BPEL4WS *flow*, which map to a set of BPMN parallel activities, is a specific block structure that has a well-defined boundary. While there are no obvious boundaries to the parallel paths created by a fork, the appropriate boundaries can be derived by an evaluation of the configuration of Sequence Flows that follow the fork. The locations in the Process where Tokens of the same TokenId and all the appropriate SubTokenIds are joined with through multiple incoming Sequence Flows will determine the boundaries for a specific block of parallel activities. The boundary may in fact be the end of the Process. More detail on the evaluation of BPEL4WS element boundaries can be found in the section entitled "Mapping to BPEL4WS" on page 157.

Splitting Flow

BPMN uses the term splitting to refer to the dividing of a path into two or more alternative paths (also known as an OR-Split). It is a place in the Process where a question is asked, and the answer determines which of a set of paths is taken. It is the “fork in the road” where a traveler, in this case a Token, can take only one of the forks (not to be confused with forking—see below).

The general concept of splitting the flow is usually referring to as a Decision. In traditional flow charting methodologies, Decisions are depicted as diamonds and usually are exclusive. BPMN also uses a diamond to leverage the familiarity of the shape, but extends the use of the diamond to handle the complex behavior of business processes (which cannot be handled by traditional flow charts). The diamond shape is used in both Gateways and the beginning of a conditional Sequence Flow (when exiting an activity). Thus, when readers of BPD sees a diamond, they know that the flow will be controlled in some way and will not just pass from one activity to another. The location of the mini-diamond and the internal indicators within the Gateways will indicate how the flow will be controlled.

There are multiple configurations to split the flow within BPMN so that different types of complex behavior can be modeled. Conditional Sequence Flow and three types of Gateways (Exclusive, Inclusive, and Complex) are used to split the flow. Refer to the section entitled “Sequence Flow” on page 119 for details on conditional Sequence Flow. Refer to the section entitled “Gateways” on page 82 for details on the Gateways.

There are two basic mechanism for making the Decision during the performance of the Process: the first is an evaluation of a condition expression. There are three variations of this mechanism: Exclusive, Inclusive, and Complex. The first variation, an Exclusive Decision, is the same as Workflow Pattern #4 -- Exclusive Choice[1] (see Figure 64). Refer to the section entitled “Data-Based” on page 85 for more information on Data-Based Exclusive Gateways.

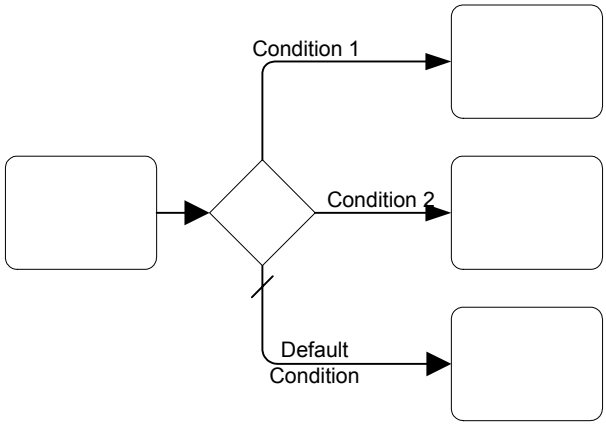

Figure 64 A Data-Based Decision Example -- Workflow Pattern #4 -- Exclusive Choice

The second type of expression evaluation is the Inclusive Decision, which is also Workflow Pattern #6 -- Multiple Choice[2]. There are two configurations of the Inclusive Decision. The first type of Inclusive Decisions uses conditional Sequence Flow from an Activity (see Figure 65).

1. http://tmitwww.tm.tue.nl/research/patterns/exclusive_choice.htm
2. http://tmitwww.tm.tue.nl/research/patterns/multiple_choice.htm

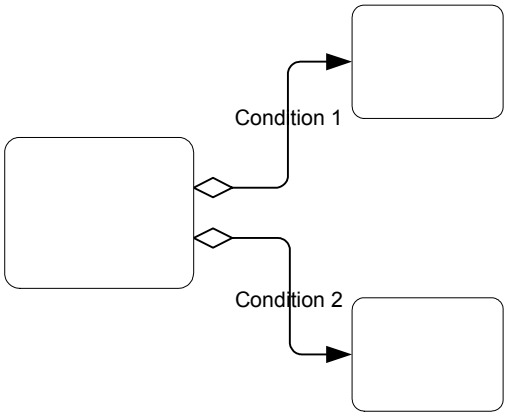

Figure 65 Workflow Pattern #6 -- Multiple Choice -- Version 1

The second type of Inclusive Decisions uses an Inclusive Gateway to control the flow (see Figure 66). Refer to the section entitled “Inclusive Gateways (OR)” on page 94 for more information on Inclusive Gateways.

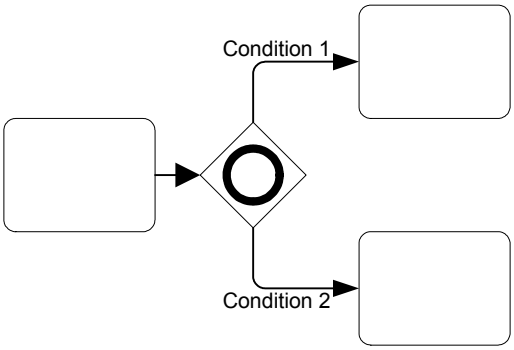

Figure 66 Workflow Pattern #6 -- Multiple Choice -- Version 2

The third type of expression evaluation is the Complex Decision (see Figure 67). Refer to the section entitled “Complex Gateways” on page 98 for more information on Inclusive Gateways.

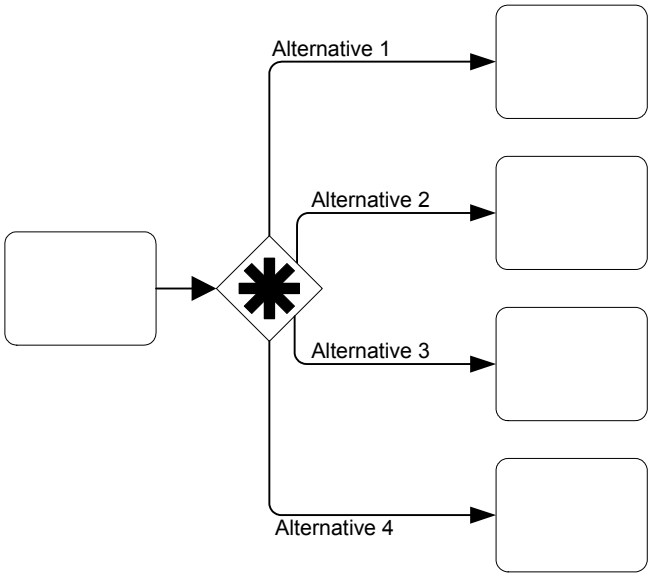

Figure 67 A Complex Decision (Gateway)

The second mechanism for making a Decision is the occurrence of a particular event, such as the receipt of a message (see Figure 68). Refer to the section entitled “Event-Based” on page 90 for more information on Event-Based Exclusive Gateways.

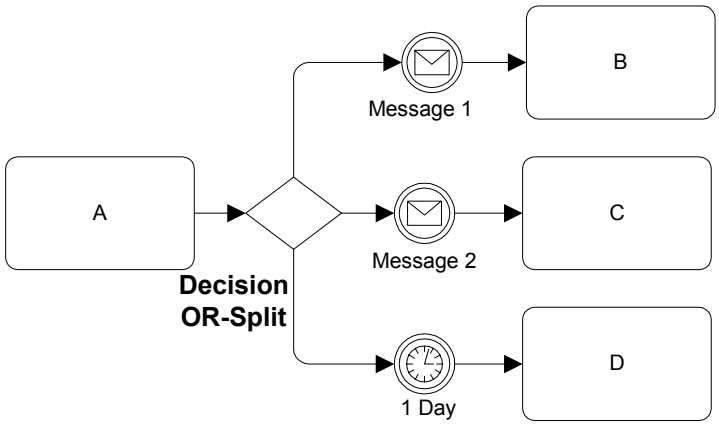

Figure 68 An Event-Based Decision Example

Merging Flow

BPMN uses the term merging to refer to the combining of two or more alternative paths into one path (also known as an a OR-Join). It is a place in the process where two or more alternative paths begin to traverse activities that are common to each of the paths. Theoretically, each alternative path can be modeled separately to a completion (an End Event). However, merging allows the paths to overlap and avoids the duplication of activities that are common to the separate paths. For a given instance of the Process, a Token would actually only see the sequence of activities that exist in one of the paths as if it were modeled separately to completion.

Since there are multiple ways that Sequence Flow can be forked and split, there are multiple ways that Sequence Flow can be merged. There are five different Workflow Patterns that can be demonstrated with merging.

The first Workflow Pattern, Simple Merge[1], The graphical mechanism to merge alternative paths is simple: there are two or more incoming Sequence Flows to a flow object (see Figure 69). In general, this means that a Token will travel down one of the alternative paths (for a given Process instance) and will continue from there. For that instance, Tokens will never arrive down the other alternative paths. BPMN provides two versions of a Simple Merge.

The first version is shown in Figure 69. The two incoming Sequence Flow for activity “D” are uncontrolled. Since the two Sequence Flow are at the end of two alternative paths, created through the upstream exclusive Gateway, only one Token will reach activity “D” for any given instance of the Process.

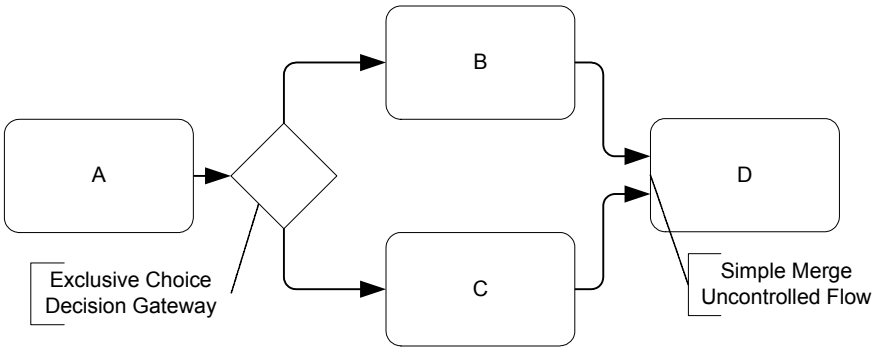

Figure 69 Workflow Pattern #5 -- Simple Merge – Version 1

If the multiple incoming Sequence Flow are actually parallel instead of alternative, then the end result is different, even though the merging configuration is the same as Figure 69. In Figure 70, the upstream behavior is parallel. Thus, there will be two Tokens arriving (at different times) at activity “D.” Since the flow into activity “D” in uncontrolled, *each Token arriving at activity “D” will cause a new instance of that activity*. This is an important concept for modelers of BPMN should understand. In addition, this type of merge is the Workflow Pattern Multiple Merge[2].

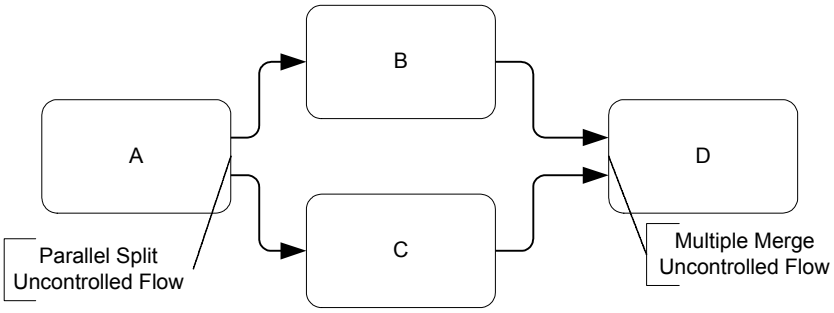

Figure 70 Workflow Pattern #7 -- Multiple Merge

1. http://tmitwww.tm.tue.nl/research/patterns/simple_merge.htm
2. http://tmitwww.tm.tue.nl/research/patterns/multiple_merge.htm

The second version of the Simple Merge is shown in Figure 71. The two incoming Sequence Flow for activity “D” are controlled through the Exclusive Gateway. Since the two Sequence Flow are at the end of two alternative paths, created through the upstream exclusive Gateway, only one Token will reach the Gateway for any given instance of the Process. The Token will then immediately proceed to activity “D.”

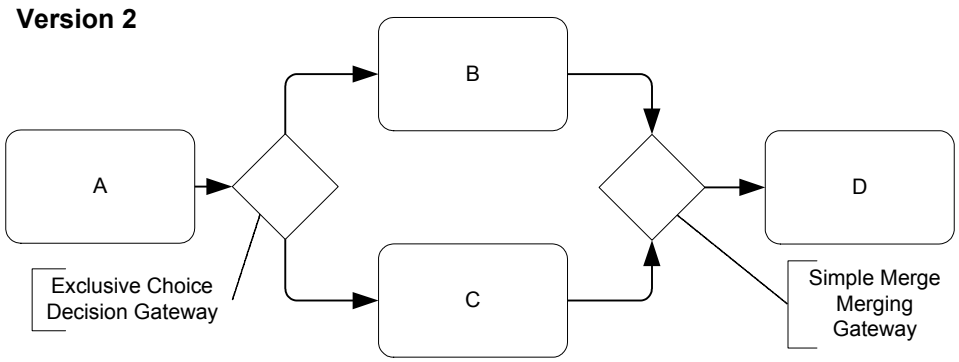

Figure 71 Workflow Pattern #5 -- Simple Merge – Version 2

Again, if the multiple incoming Sequence Flow are actually parallel instead of alternative, then the end result is different, even though the merging configuration is the same as Figure 71. In the model shown in Figure 72, there will be two Tokens arriving (at different times) at the Exclusive Gateway preceding activity “D.” In this situation, the Gateway will accept the first Token and immediately pass it on through to the activity. When the second Token arrives, it will be *excluded* from the remainder of the flow. This means that the Token will not be passed on to the activity, but will be consumed. This type of merge is the Workflow Pattern Discriminator[1].

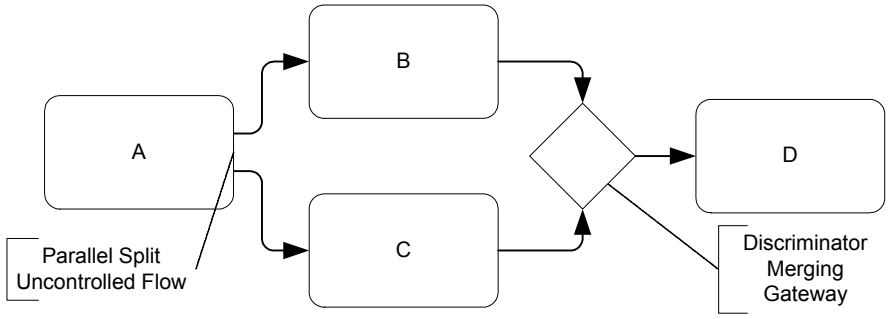

Figure 72 Workflow Pattern #8 -- Discriminator

1. http://tmitwww.tm.tue.nl/research/patterns/discriminator.htm

The fourth type of Workflow Pattern merge is called a Synchronizing Join[1]. This is a situation when the merging location does not know ahead of time how many Tokens will be arriving at the Gateway. In some Process instances, there may be only one Token. In other Process instances, there may be more than one Token arriving. This type of situation is created when an Inclusive Decision is made up stream (see Figure 73). To handle this, an Inclusive Gateway can be used to merge the appropriate number of Tokens for each Process instance. The Gateway, following the pattern Synchronizing Join, will wait for all expected Tokens before the flow will continue to the next activity. Refer to the section entitled “Inclusive Gateways (OR)” on page 94 for more information on Inclusive Gateways.

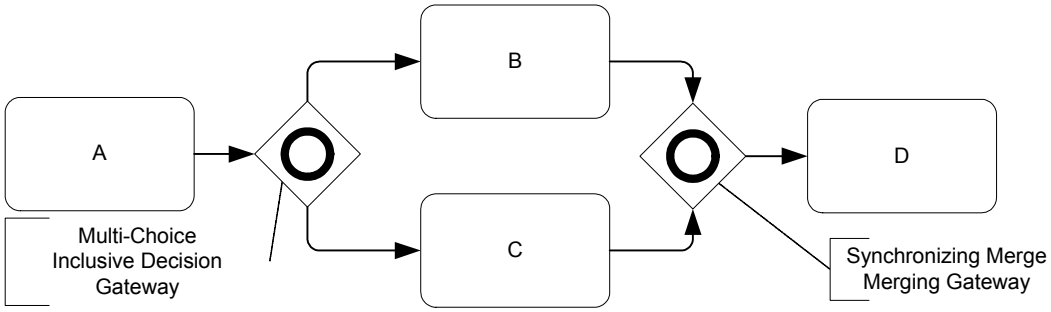

Figure 73 Workflow Pattern #9 -- Synchronizing Join

The fourth type of Workflow Pattern merge is called a N out of M Join[2]. This type of situation is more complex and can be handled through a Complex Gateway (see Figure 74). The Gateway will receive Tokens from its incoming Sequence Flow and evaluate an expression to determine whether or not the flow should proceed. Once the condition has been satisfied, if additional Tokens arrive, then will be excluded (much like the Discriminator Pattern from Figure 72). Refer to the section entitled “Complex Gateways” on page 98 for more information on Inclusive Gateways.

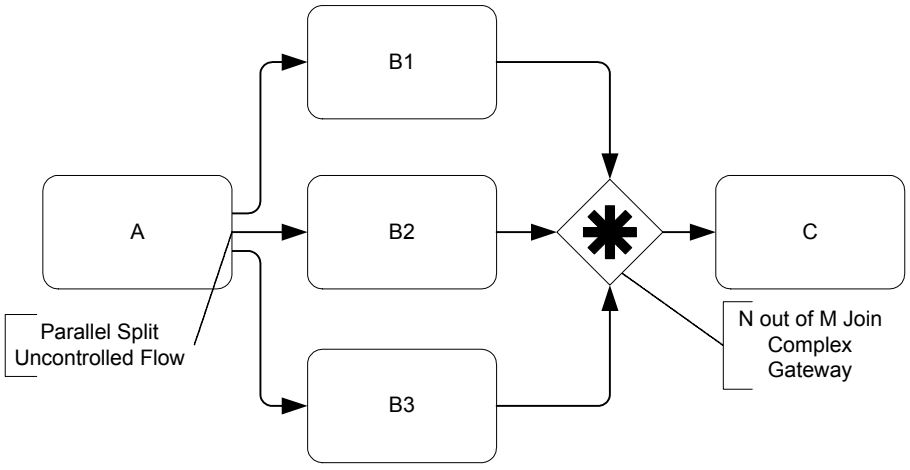

Figure 74 Workflow Pattern #8 -- N out of M Join

1. http://tmitwww.tm.tue.nl/research/patterns/synchronizing_join.htm
2. http://tmitwww.tm.tue.nl/research/patterns/n_out_of_m_join.htm

There is no specific correlation between the merging of a set of paths and the splitting that occurs through a Gateway object. For example, a Decision may split a path into three separate paths, but these three paths do not need to be merged at the same object. Figure 75 shows that two of three alternative paths are merged at Task “F.” All of the paths eventually will be merged, but this can happen through any combination of objects, including lone End Events. In fact, each path could end with a separate End Event.

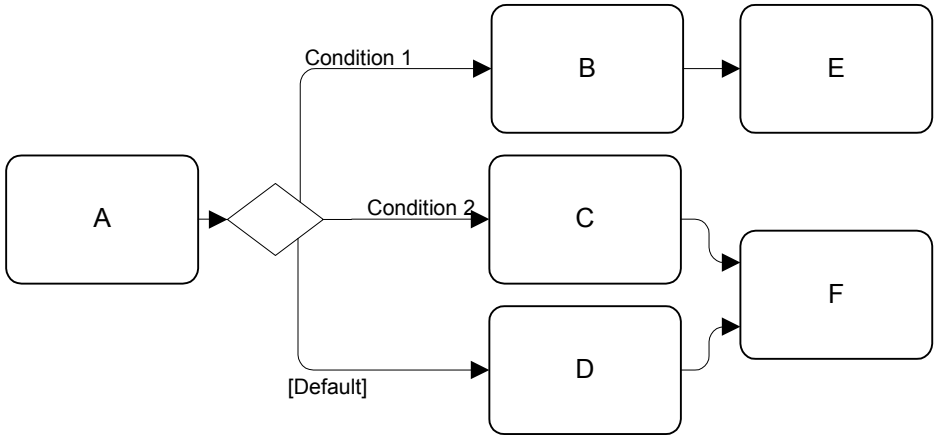

Figure 75 The Split-Merge Relationship is not Fixed

Thus, for alternative flow, BPMN contrasts with BPEL4WS, which are mainly block structured. A BPEL4WS *switch* and *pick*, which map to the BPMN Decision, are specific block structures that have well-defined boundaries. While there are no obvious boundaries to the alternative paths created by a decision in BPMN, the appropriate boundaries can be derived by an evaluation of the configuration of Sequence Flows that follow the decision. The locations in the Process where Tokens of the same identity are merged through multiple incoming Sequence Flows will determine the boundaries for a specific decision. The boundary may in fact be the end of the Process. More detail on the evaluation of BPEL4WS element boundaries can be found in the section entitled “Mapping to BPEL4WS” on page 157.

Looping

BPMN provides 2 (two) mechanisms for looping within a Process. The first involves the use of attributes of activities to define the loop. The second involves the connection of Sequence Flows to “upstream” objects.

Activity Looping

The attributes of Tasks and Sub-Processes will determine if they are repeated as a loop. There are two types of loops that can be specified: Standard and Multi-Instance.

For Standard Loops:

- If the loop condition is evaluated before the activity, this is generally referred to as a “while” loop. This means that the activities will be repeated as long as the condition is true. The activities may not be performed at all (if the condition is false the first time) or performed many times.
- If the loop condition is evaluated after the activity, this is generally referred to as an “until” loop. This means that the activities will be repeated until a condition becomes true. The activities will be performed at least once or performed many times.

For Multi-Instance Loops:

- If the MI_Ordering is serial, then this becomes much like a while loop with a set number of iterations the loop will go through. These are often used in processes where a specific type of item will have a set number of sub-items or line items. A Multi-Instance loop will be used to process each of the line items.
- If the MI_Ordering is parallel, this is generally referred to as a multiple instance of the activities. An example of this type of feature would be used in a process to write a book, there would be a Sub-Process to write a chapter. There would be as many copies or instances of the Sub-Process as there are chapters in the book. All the instances could begin at the same time.

Those activities that are repeated (looped) will have a loop marker placed in the bottom center of the activity shape (see Figure 76). Those activities that are Parallel Multi-Instance will have a parallel marker placed in the bottom center of the activity shape (see Figure 77)

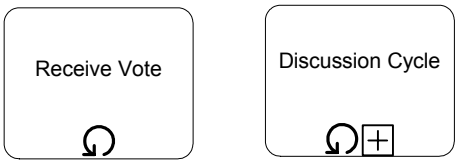

Figure 76 A Task and a Collapsed Sub-Process with a Loop Marker

Figure 77 A Task with a Parallel Marker

Expanded Sub-Processes also can have a loop marker placed at the bottom center of the Sub-Process rectangle (see Figure 78). The entire contents of the Sub-Process will be repeated as defined in the attributes.

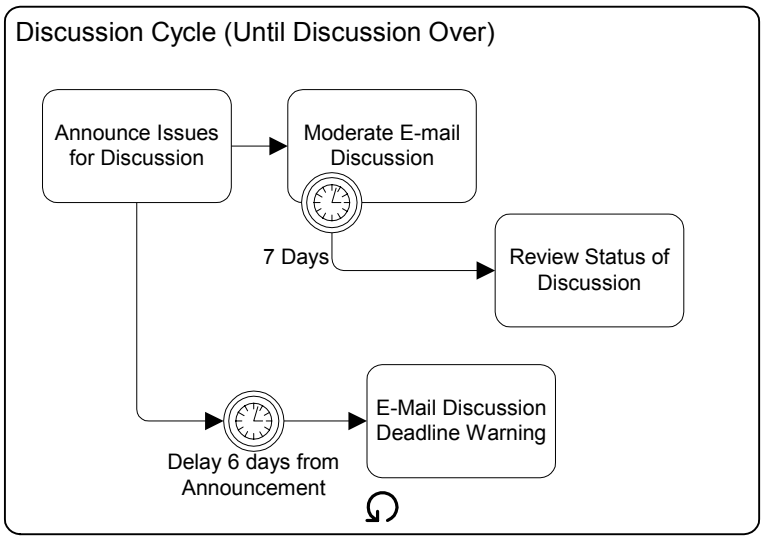

Figure 78 An Expanded Sub-Process with a Loop Marker

Sequence Flow Looping

Loops can also be created by connecting a Sequence Flow to an “upstream” object. An object is considered to be upstream if that object has an outgoing Sequence Flow that leads to a series of other Sequence Flows, the last of which turns out to be an incoming Sequence Flow to the original object. That is, that object produces a Token and that Token traverses a set of Sequence Flows until the Token reaches the same object again. Sequence Flow looping is the same as Workflow Pattern #16 -- Arbitrary Cycle[1] (see Figure 64).

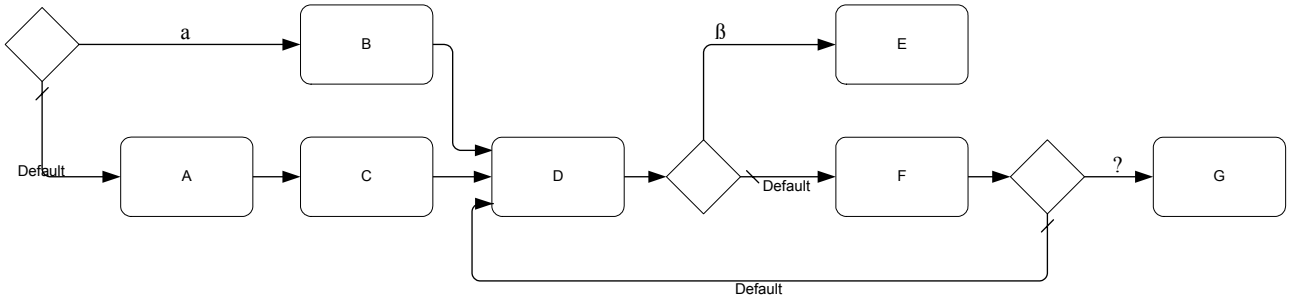

Figure 79 Workflow Pattern #16 -- Arbitrary Cycle

Usually these connections follow a Decision so that the loop is not infinite (see Figure 80). If the Sequence Flow goes directly from a Decision to an upstream object, this is an “until” loop. The set of looped activities will occur until a certain condition is true.

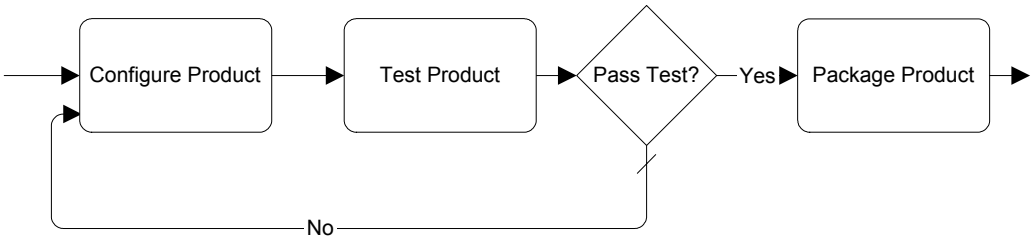

Figure 80 An Until Loop

A while loop is created by making the decision first and then performing the repeating activities or moving on in the Process (see Figure 81). The set of looped activities may not occur or may occur many times.

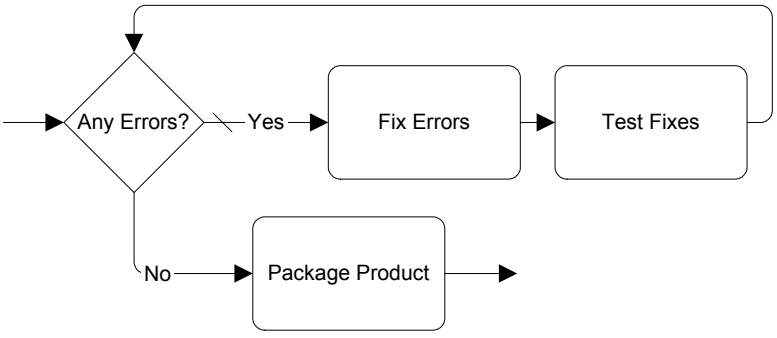

Figure 81 A While Loop

1. http://tmitwww.tm.tue.nl/research/patterns/arbitrary_cycle.htm

Sequence Flow Jumping (Off-Page Connectors and Go To Objects)

Since process models often extend beyond the lengh of one printed page, there is often a concern about showing how Sequence Flow connections extend across the page breaks. One solution that is often employed is the use of Off-Page connectors to show where one page leaves off and the other begins. BPMN provides Imtermediate Events of type Link for use as Off-Page connectors (see Figure 82--Note that the figure shows two different printed pages, not two Pools in one diagram). A pair of Link Intermediate Events are used. One of the pair is shown at the end of one page. This Event is named and has an incoming Sequence Flow and no outgoing Sequence Flow. The second Link Event is at the beginning of the next page, shares the same name, and has an outgoing Sequence Flow and no incoming Sequence Flow.

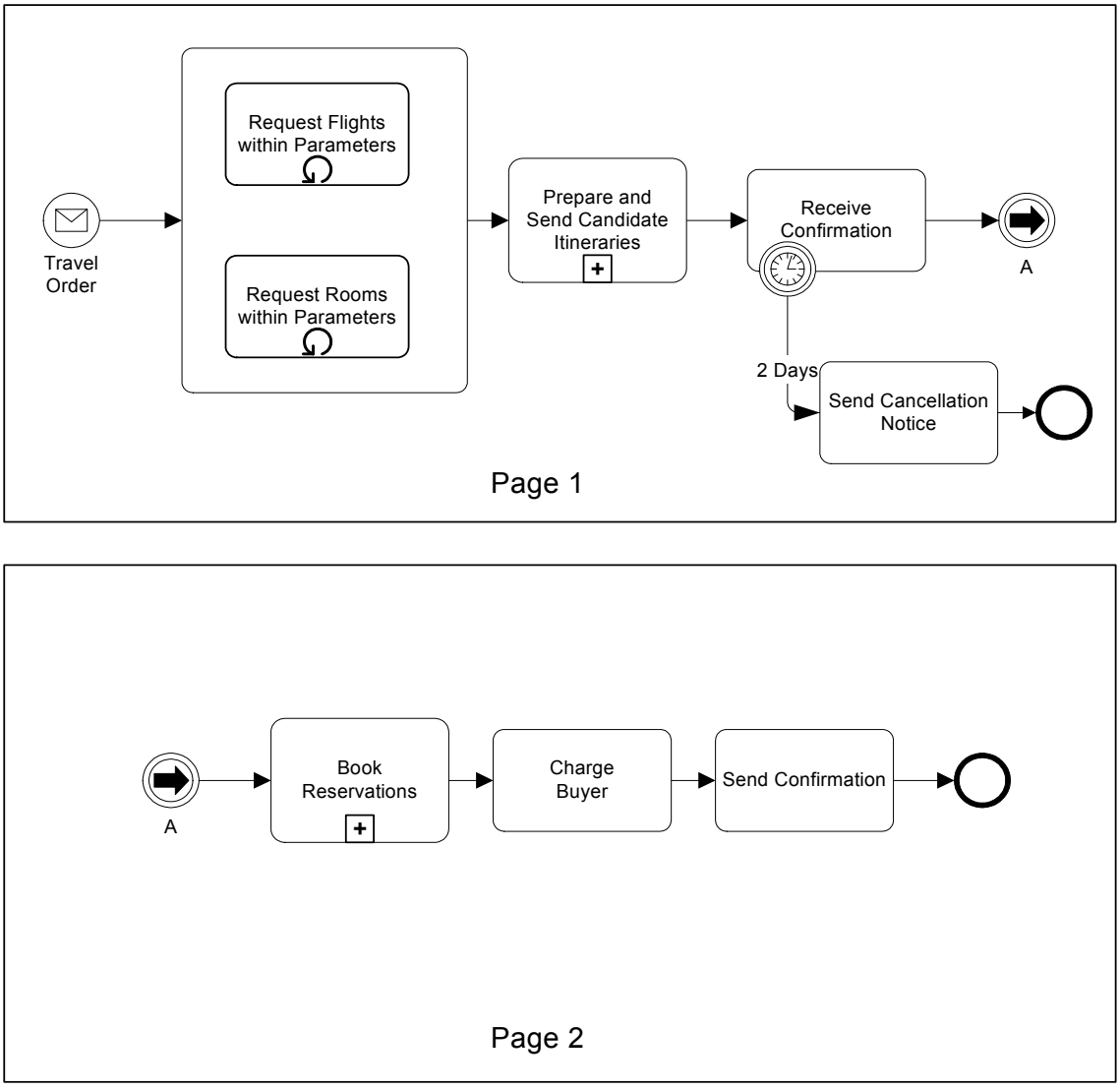

Figure 82 Link Intermediate Event Used as Off-Page Connector

Another way that Link Intermediate Events can be used is as “Go To” objects. Functionally, they would work the same as for Off-Page Connectors (described above), except that they could be used anywhere in the diagram--on the same page or across multiple pages. The general idea is that they provide a mechanism for reducing the length of Sequence Flow lines.

Some modelers may consider long lines as being hard to follow or trace. Go To Objects can be used to avoid very long Sequence Flow (see Figure 83 and Figure 84). Both diagrams will behave equivalently. For Figure 84, if the “Order Rejected” path is taken from the Decision, then the Token traversing the Sequence Flow would reach the source Link Event and then “jump” to the target Link Event and continue down the Sequence Flow. The process would continue as if the Sequence Flow had directly connected the two objects.

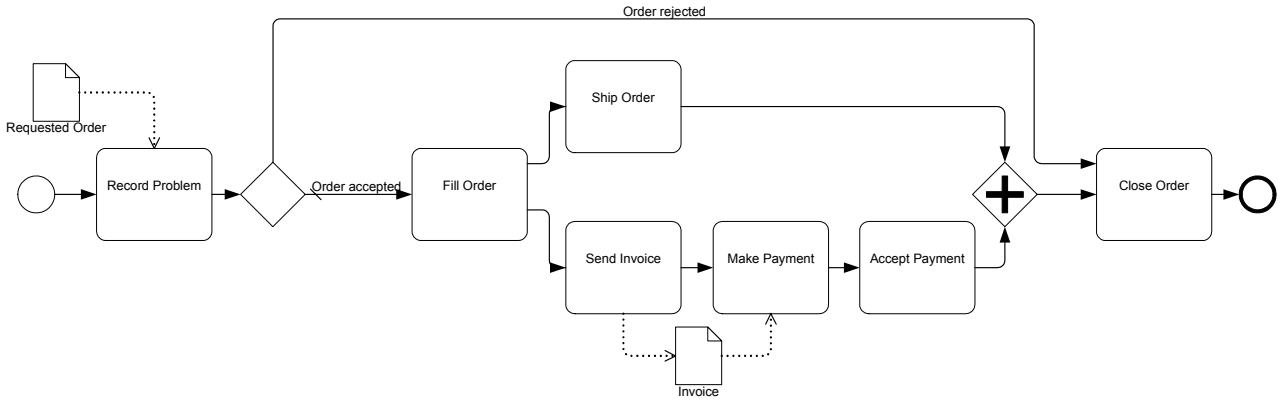

Figure 83 Process with Long Sequence Flow

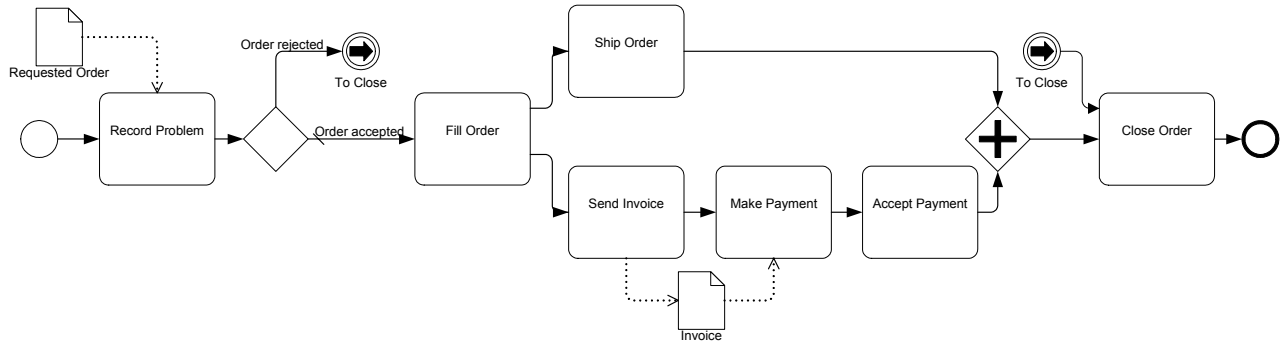

Figure 84 Process with Link Intermediate Events Used as Go To Objects

Some methodologies prefer that all Sequence Flow only move in one direction; that is, forward in time. These metholodogies do not allow Sequence Flow to connect directly to upstream objects. Some consistency in modeling can be gained by such a methdology, but situations that require looping become a challenge. Link Intermediate Events can be used to make upstream connections and create loops without violating the Sequence Flow direction restriction (see Figure 85).

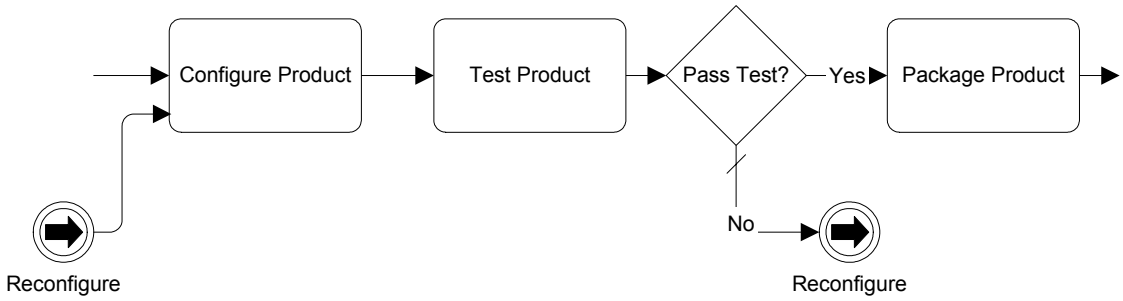

Figure 85 Link Intermediate Event Used for Looping

Passing Flow to and from Sub-Processes

This section reviews how flow will be passed between a parent Process and any of its Sub-Processes. The flow (e.g., a Token) will start at the parent Process and then move to the Sub-Process and then will move back to the parent process (see Figure 86). Most of the time the flow will reach a Sub-Process, get transferred to the Start Event of the Sub-Process, traverse the Sequence Flows of the Sub-Process, reach the End Event of the Sub-Process, and, finally, get transferred back to the parent Process to continue down the outgoing Sequence Flow of the Sub-Process object. If the Sub-Process contains parallel flows, then all the flows must complete before a Token is transferred back to the parent Process. This functionality treats the Sub-Process as a self-contained "box" of activities.

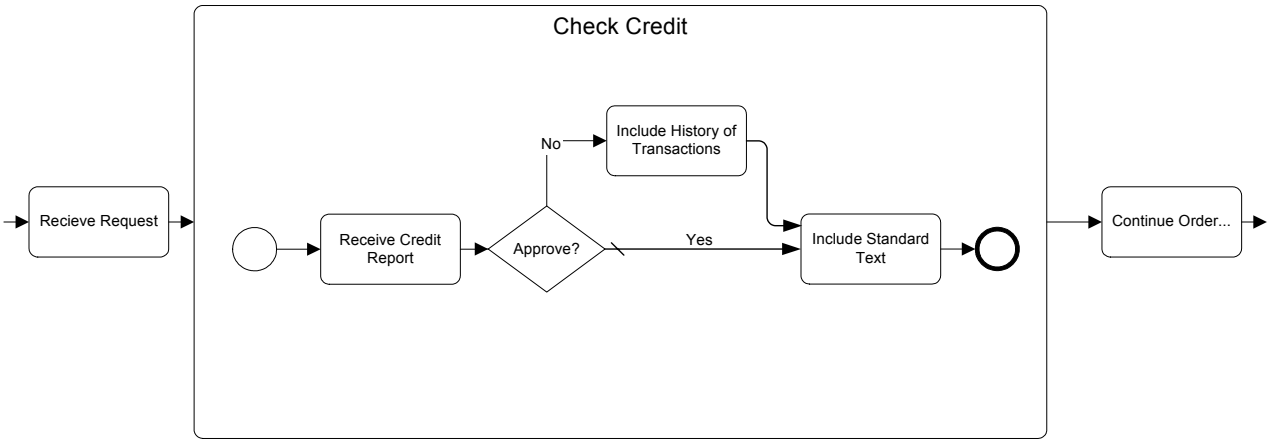

Figure 86 Example of Sub-Process with Start and End Events Inside

To make the flow between levels of a Process more obvious, a modeler has the option of placing the Start Event and the End Event on the boundary of the Sub-Process and connect the Sequence Flow from the Parent Process objects to/from these Events (see Figure 87).

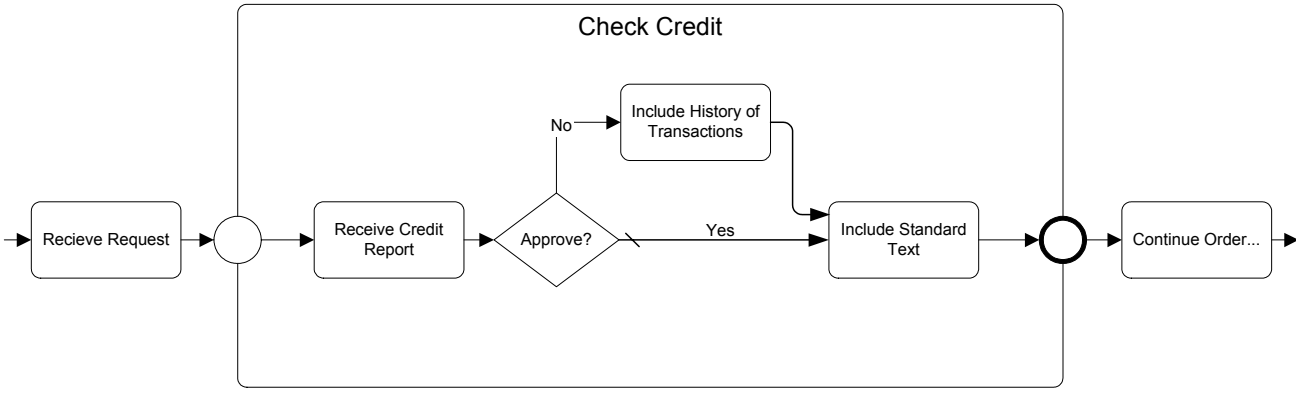

Figure 87 Example of Sub-Process with Start and End Events on Boundary

Controlling Flow Across Processes

There may be situations within a Process where the flow is affected by or dependent on the activity that occurs in another Process. That is, a Process may have to wait to start or to continue based on Link Events can also be used to pass the flow (Tokens) between processes.

(see Figure 88) The type of Workflow Pattern called a Milestone[1].

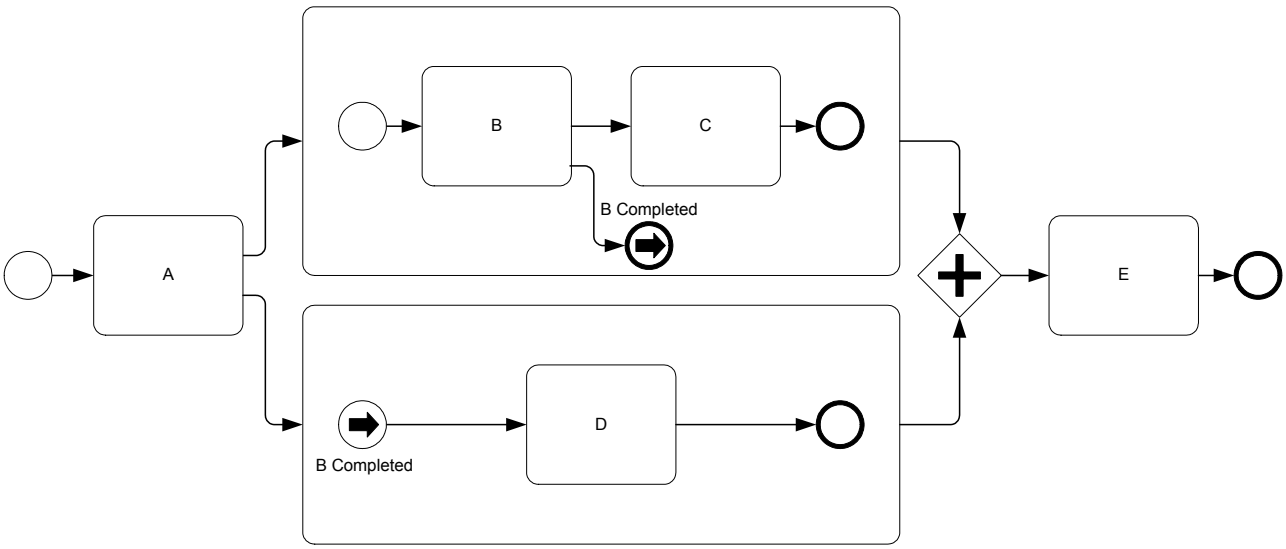

Figure 88 Link Events Used to Synchronize Behavior Across Processes

It should be noted that the flow of Tokens is somewhat unusual in Figure 88's top Sub-Process.

Avoiding Illegal Models and Unexpected Behavior

BPMN, being a graph-structured Diagram, rather than having a block-structures like BPEL4WS, provides a great flexibility for depicting complex process behavior in a fairly compact form. However, the free-form nature of BPMN can create modeling situations that cannot be executed or will behave in a manner that is not expected by the modeler. These types of modeling problems can occur because there is not a tight relationship between forks and joins or splits and merges. A block structure provides these tight relationships, but a graph-structure allows these flow control mechanisms to be mixed and matched at the discretion of the modeler. Some combinations of these control elements will create Processes that cannot be executed or will create behavior that was not intended by the modeler. The situation where alternative paths cross the implicit boundary of a group of parallel paths can cause an invalid model.

Figure 89 shows such a model. Task "D" is an activity that has two incoming Sequence Flows; one from a forked path (after a split path) and one from a split path. This can create a problem at the Parallel Gateway that precedes Task "E," which also has multiple incoming Sequence Flows. The Sequence Flow from Task "B" is crossing the implicit boundary of the fork created after Task "A." As a result, if the "Yes" Sequence Flow is taken from the Decision in the Diagram (Variation 1), then Task "E" can expect two Tokens to arrive—one from Task "C" and one from Task "D." However, if the "No" Sequence Flow is taken from the Decision (Variation 2), the Parallel Gateway will receive only one Token—one from Task "D." Since the Gateway expects two Tokens, the Process will be dead-locked at that position.

1. http://tmitwww.tm.tue.nl/research/patterns/milestone.htm

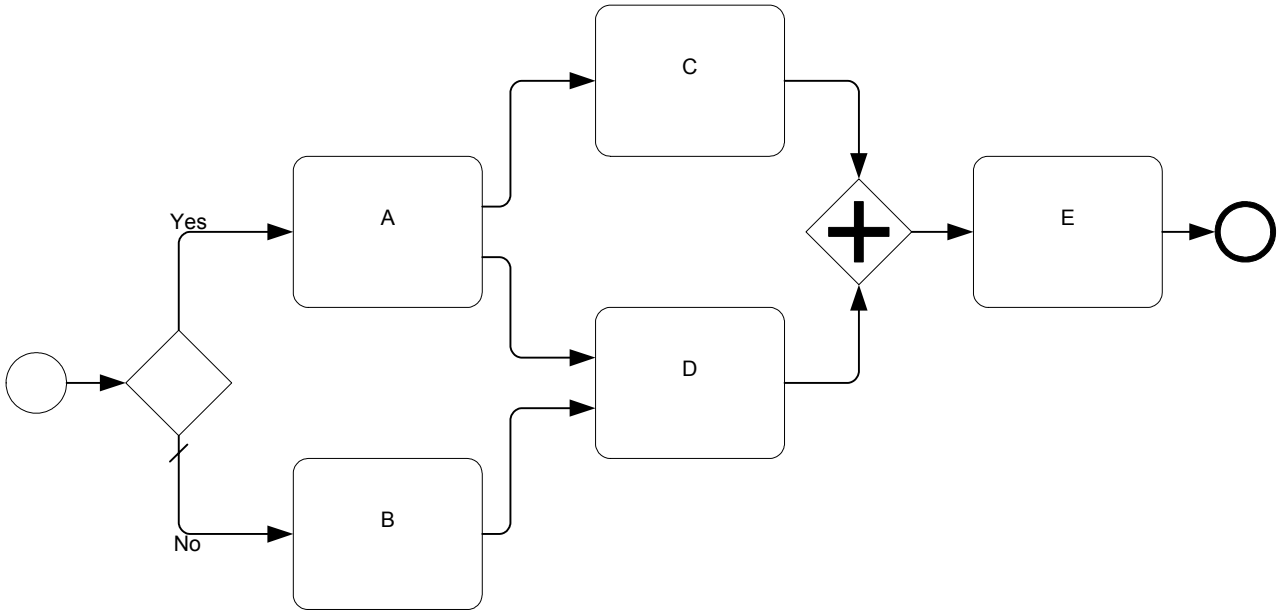

Figure 89 Potentially a dead-locked model

Another type of problem occurs with looping back to upstream activities. If the loop Decision is made within the implicit boundaries of a set of parallel paths, then the behavior of the loop becomes ambiguous (see Figure 90), since it is unclear whether Task “E” was intended to be repeated based on the loop or what would happen if Task “E” was still active when the loop reached that Task again.

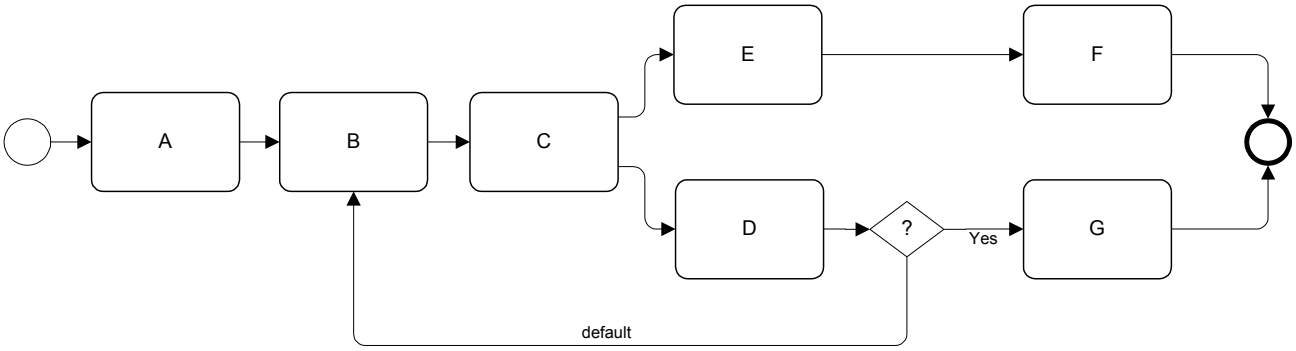

Figure 90 Improper Looping

The use of Link Events can also create unexpected behavior. In general, Link Events should be considered an advanced modeling technique and the modeler should be careful to understand how the behavior and flow of Tokens.

The figure below (see Figure 91) is a variation of Figure 88. In this figure, however, the Link End Event in the top Sub-Process is not used properly. For the top Sub-Process, there is only one Token generated and available. When the Token leaves Task “C” and arrives at the Link End Event, it is consumed by the Event, but then immediately jumps to the target Start Event that shares its name (in the bottom Sub-Process). Because the Token jumps to the other Sub-Process, there is no Token left to be transferred up to the Parent Process and continue down the outgoing Sequence Flow of the top Sub-Process. Thus, the overall Process will be stuck waiting at the Parallel Gateway for a Token that will never arrive.

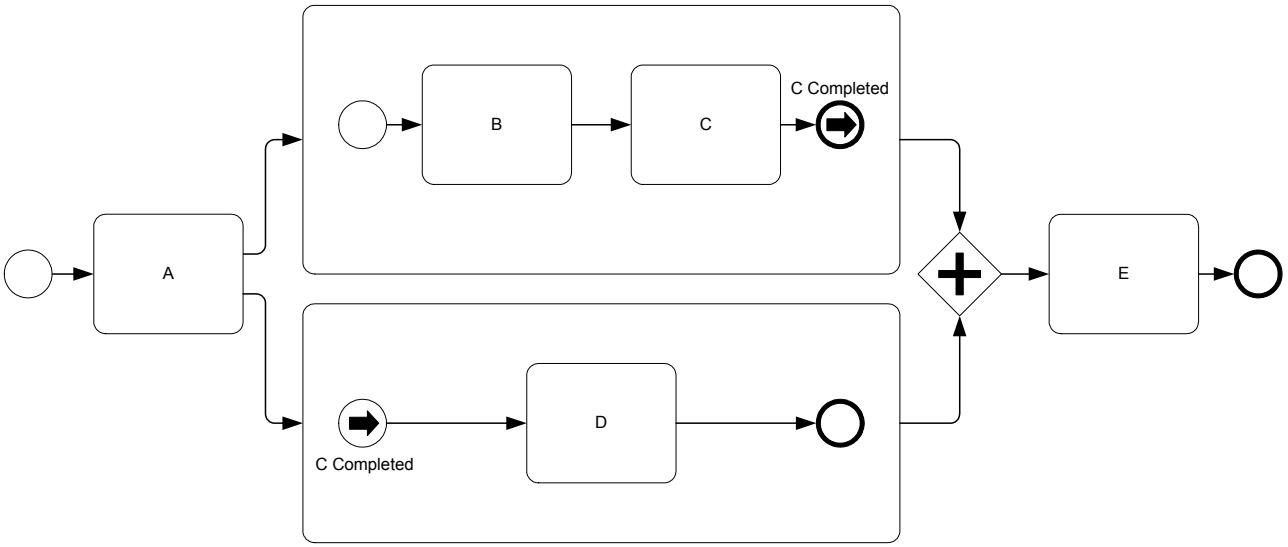

Figure 91 Improper use of a Link End Event

In general, the analysis of how Tokens will flow through the model will help find models that cannot be executed properly. This Token flow analysis will be used to create some of the mappings to BPEL4WS. Since BPEL4WS is properly executable, if the Token flow analysis cannot create a valid BPEL4WS process, then the model is not structured correctly. This is an open issue that will be resolved in a later version of the specification. The section entitled “Defining Token Generation for execution Language Mapping” on page 203 will detail the Token flow analysis. Refer to the section entitled “Open Issues” on page 231 for a complete list of the issues open for BPMN.

Changes Since 1.0 Draft Version

These are the changes since the last publically release version:

- The section entitled “Sequence Flow Jumping (Off-Page Connectors and Go To Objects)” on page 144 was added.
- The section entitled “Controlling Flow Across Processes” on page 146 was added.
- The section entitled “Avoiding Illegal Models and Unexpected Behavior” on page 147 was updated to show an example of how Link Events can cause unexpected behavior.

5.2.2 Exception Flow

Exception flow occurs outside the normal flow of the Process and is based upon an event (an Intermediate Event) that occurs during the performance of the Process. Intermediate Events can be included in the normal flow to set delays or breaks to wait for a message. However, exception flow is created by attaching the Intermediate Event to the boundary of an activity, either a Task or a Sub-Process (see Figure 92). Multiple Intermediate Events can be attached to the boundary of an activity.

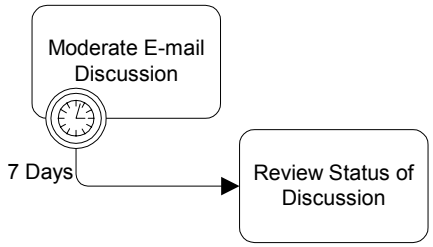

Figure 92 A Task with Exception Flow (Interrupts Event Context)

By doing this, the modeler is creating an Event Context. The Event Context will respond to specific Triggers to interrupt the activity and redirect the flow through the Intermediate Event. The Event Context will only respond if it is active (running) at the time of the Trigger. If the activity has completed, then the Trigger may occur with no response.

If there are a group of Tasks that the modeler wants to include in an Event Context, then an Expanded Sub-Process can be added to encompass the Tasks and to handle any events by having them attached to its boundary (see Figure 93).

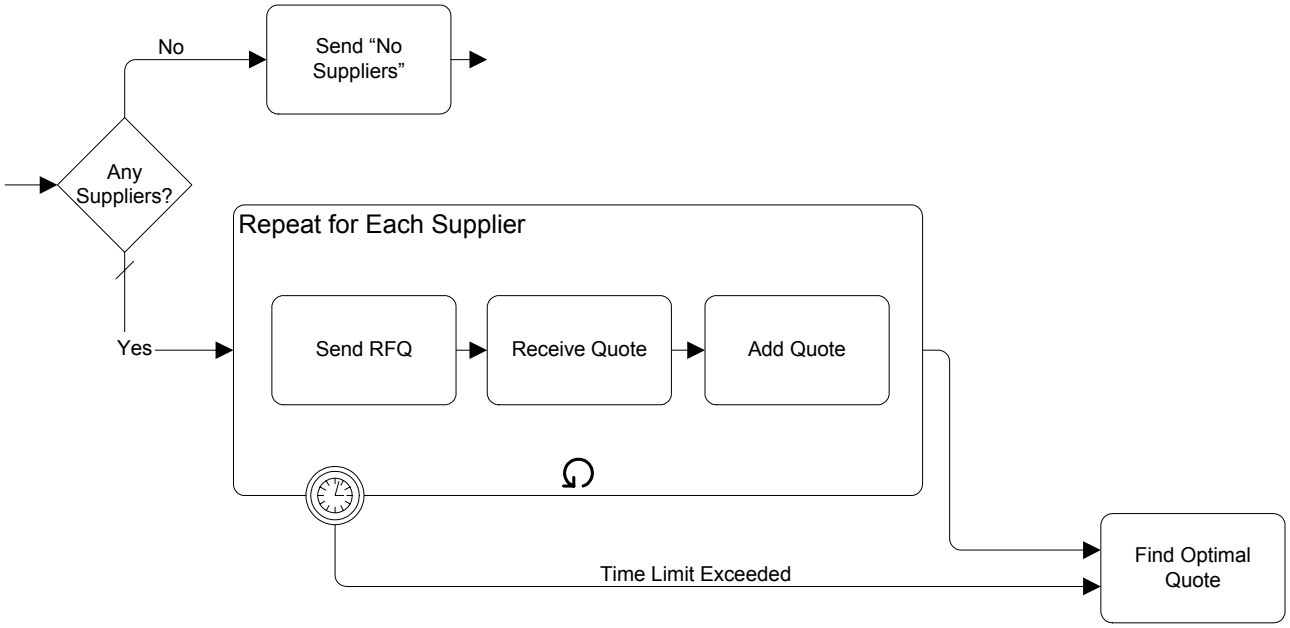

Figure 93 A Sub-Process with Exception Flow (Interrupts Event Context)

Two Triggers for Intermediate Event are used by Event Contexts at the level of the execution language (BPEL4WS): Message, and Exception (fault). A Message Event occurs when a message, with the exact identity as specified in the Intermediate Event, is received by the Process. An Exception Event occurs when the Process detects an Exception. If an Error Code is specified in the Intermediate Event, then the code of the detected Error must match for the Event Context to respond. If the Intermediate Event does not specify an Error Code, then any Exception will trigger a response from the Event Context. Other BPMN Triggers, such as a Timer, must be converted into a BPEL4WS configuration that will generate the appropriate Message or Exception.

If this event does not occur while the Event Context is ready, then the Process will continue through the normal flow as defined through the Sequence Flows.

5.2.3 Ad Hoc

An Ad Hoc Process is a group of activities that have no pre-definable sequence relationships. A set of activities can be defined for the Process, but the sequence and number of performances for the activities is completely determined by the performers of the activities and cannot be defined beforehand.

A Sub-Process is marked as being an Ad Hoc with a “tilde” symbol placed at the bottom center of the Sub-Process shape (see Figure 94 and Figure 95). Activities within the Process are disconnected from each other. During execution of the Process, any one or more of the activities may be active and they can be performed in almost any order or frequency.

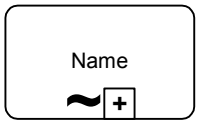

Figure 94 A Collapsed Ad Hoc Sub-Process

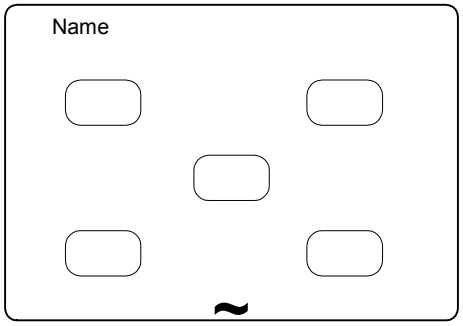

Figure 95 An Expanded Ad Hoc Sub-Process

The performers determine when activities will start, when they will end, what the next activity will be, and so on. Examples of the types of Processes that are Ad Hoc include computer code development (at a low level), sales support, and writing a book chapter. If we look at the details of writing a book chapter, we could see that the activities within this Process include: researching the topic, writing text, editing text, generating graphics, including graphics in the text, organizing references, etc. (see Figure 96). There may be some dependencies between Tasks in this Process, such as writing text before editing text, but there is not necessarily any

correlation between an instance of writing text to an instance of editing text. Editing may occur infrequently and based on the text of many instances of the writing text Task.

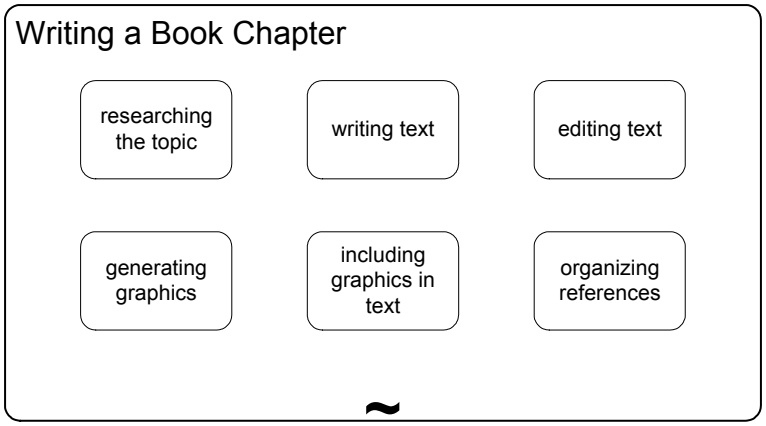

Figure 96 An Ad Hoc Process for Writing a Book Chapter

It is a challenge for a BPM engine to monitor the status of Ad Hoc Processes, usually these kind of processes are handled through groupware applications (such as e-mail), but BPMN allows modeling of Processes that are not necessarily executable and should provide the mechanisms for those BPM engines that can follow an Ad Hoc Process. Given this, at some point, the Process will have completed and this can be determined by evaluating a Completion Condition that evaluates Process attributes that will have been updated by an activity in the Process.

5.3 Compensation Association

Some activities produce complex effects or specific outputs. If the outcome is determined to be undesirable by some specified criteria (such as an order being cancelled), then it will be necessary to “undo” the activities. There are three ways this can be done:

- Restoring of a copy of the initial values for data, thereby overwriting any changes.
- Doing nothing (if nothing has be changed because the changes have been set aside until a confirmation).
- Invoking activities that undo the effects--also known as compensation.

An activity that might require compensation could be, for example, one that charges a buyer for some service and debits a credit card to do so. These types of activities usually need a separate activity to counter the effects of the initial activity. Often, a record of both activities is required, so this is another reason that the activity is not “undone.” An Intermediate Event of type Compensation is attached to the boundary of an activity to indicate that compensation may be necessary for that activity.

One of the three mechanisms for “undo” activities, Compensation, requires specific notation and is a special circumstance that occurs outside the normal flow of the Process. For this

reason, the Compensation Intermediate Event does not have an outgoing Sequence Flow, but instead has an outgoing directed Association (see Figure 97).

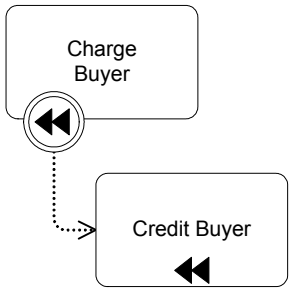

Figure 97 A Task with an Associated Compensation Activity

The target of this Association is the activity that will compensate for the work done in the source activity, and will be referred to as the Compensation Activity. The Compensation Activity is special in that it does not follow the normal Sequence Flow rules--as mentioned, it is outside the normal flow of the Process. This activity cannot have any incoming or outgoing Sequence Flow. The Compensation marker (as is in the Compensation Intermediate Event) will be displayed in the bottom center of the Activity to show this status of the activity (see the "Credit Buyer" Task in Figure 97). Note that there can be only one target activity for compensation. There cannot be a sequence of activities shown. If the compensation does require more than one activity, then these activities must be put inside a single Sub-Process that is the target of the Association. The Sub-Process can be collapsed or expanded. If the Sub-Process is expanded, then only the Sub-Process itself requires the Compensation marker--the activities inside the Sub-Process do not require this marker.

Only activities that have been completed can be compensated. The compensation of an activity can be triggered in two ways:

- The activity is inside a Transaction Sub-Process that is cancelled (see Figure 98). In this situation, the whole Sub-Process will be "rewound" or rolled back--the Process flow will go backwards and any activity that requires compensation will be compensated. This is why the Compensation marker for Events looks like a "rewind" symbol for a tape player. After the compensation has been completed, the Process will continue its rollback.
- A downstream Intermediate or End Event of type Compensation "throws" a compensation identifier that is "caught" by the Intermediate Event attached to the boundary of the activity.

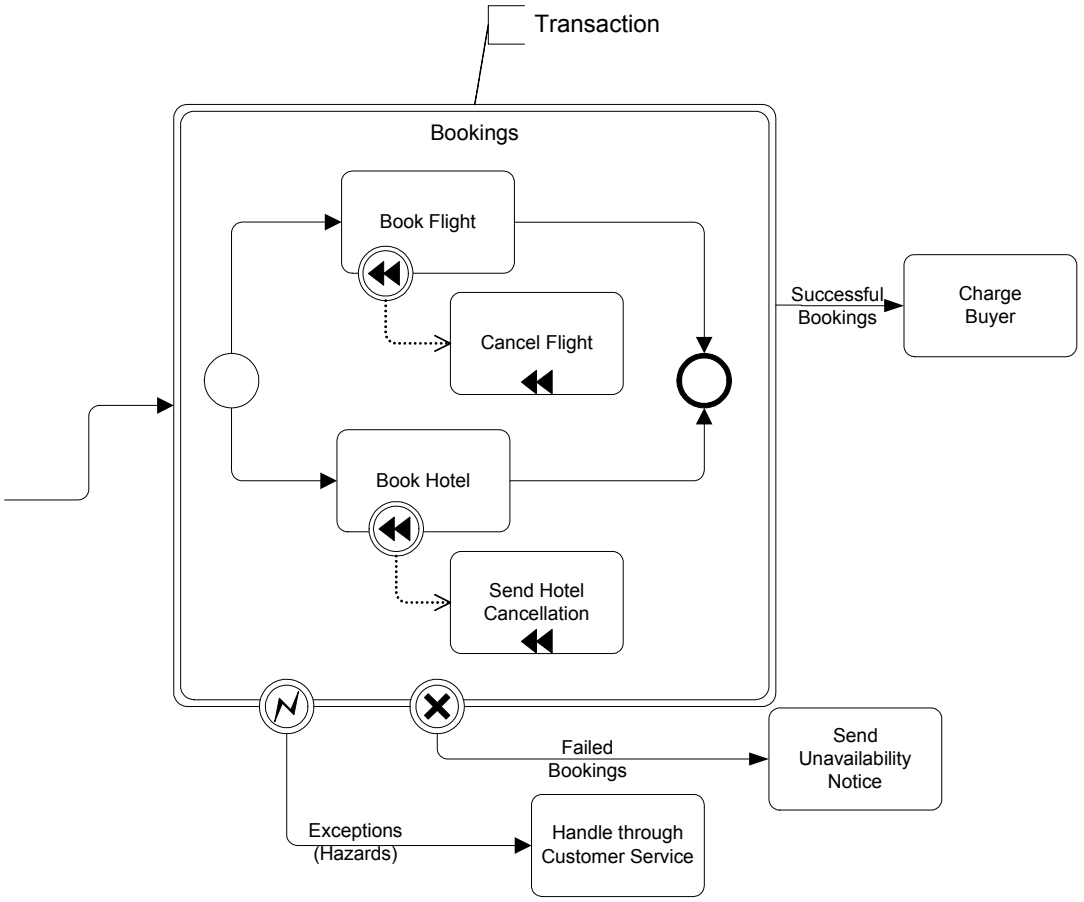

Figure 98 Compensation Shown in the context of a Transaction