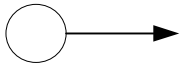
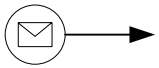
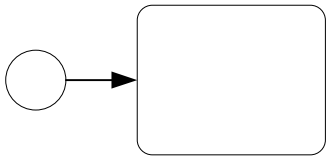
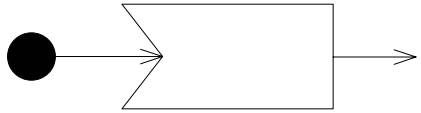
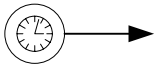
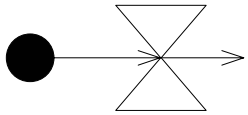
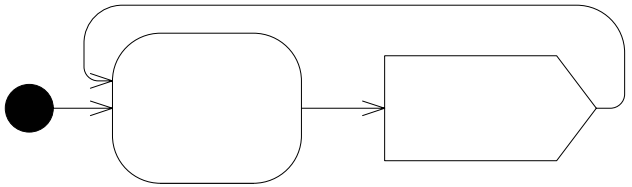
Appendix D: Mapping BPMN to BPD Profile

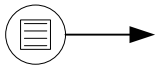
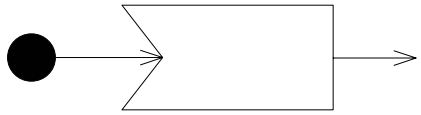
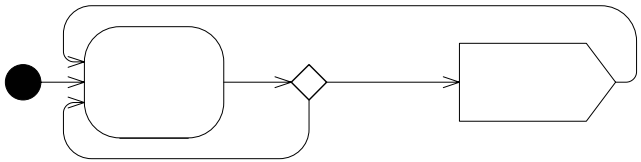
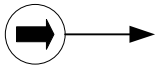
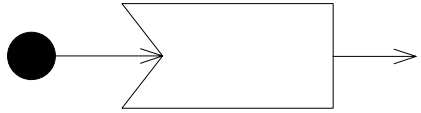
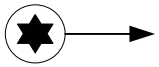
Members of bpmi.org and the OMG are interested in the unification of the UML 2.0 and BPMN notation for the support of the business user. This draft mapping is in support of this dialog and will be completed in a revised submission. Note, it may be the case that provision of a new, business-level notation will require changes to the UML 2.0 metamodel; these will be documented in any case where they arise.

Mapping Overview

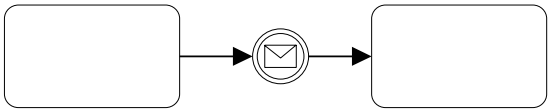
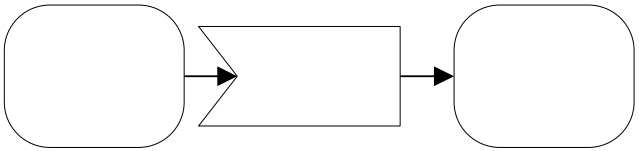
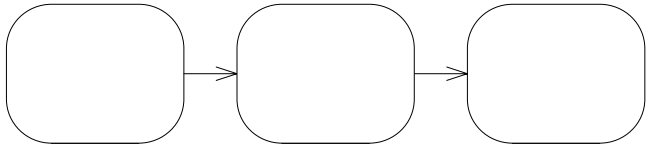
The following table outlines the mapping defined from the BPMN notation to the BPD meta model in detail.

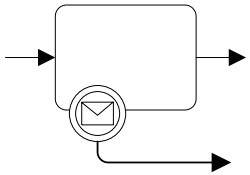
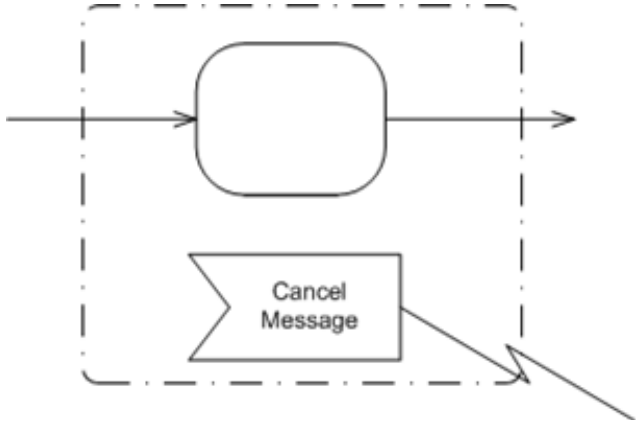
Element/Concept	Description	BPMN Notation	Activity Diagram Notation
Event	An event is something that "happens" during the course of a business process. These events affect the flow of the process and usually have a cause (trigger) or an impact (result). There are three types of Events, based on when they affect the flow: Start, Intermediate, and End.		Start Node, End Node, Signal, Connector, and various design patterns. See the rows that follow that deal with Events.
Start (None, Message, Timer, Rule, Link, Multiple)	As the name implies, the Start Event indicates where a particular process will start. Start Events can have "Triggers" that define the cause for the event. There are multiple ways that these events can be triggered.		The Start Event maps to the Start Node plus some design patterns. See next five rows…  It should be noted that if a receipt of a signal is used in the mapping, then the start node, followed by a control flow is not required.

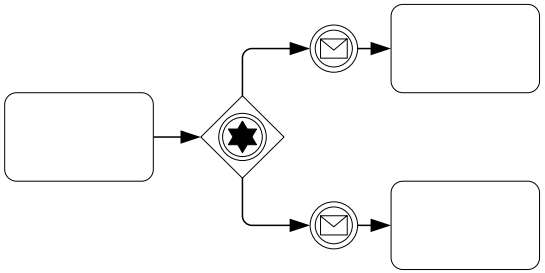
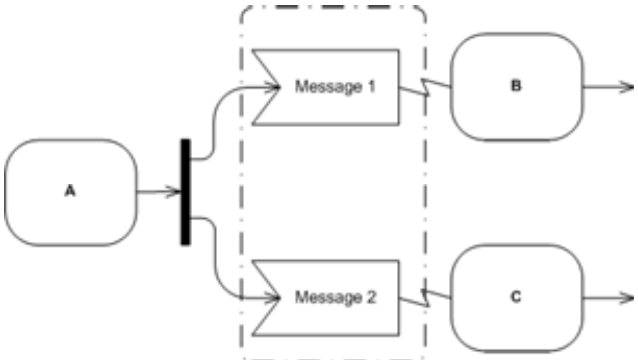
Message Trigger	The message is generally sent from an outside source and triggers the start of the process. The receipt of the message needs to be configured the same way as a receive type of activity.	 In BPMN, this is also equivalent to a None Start Event followed by a Receive Task. 	Start Node followed by a Signal Receipt AcceptEventAction Initial node is optional (If no incoming transitions, the action starts when the enclosing activity starts). 
Timer Trigger	The process is started whenever a timer reaches a specific time/date or a recurring time/date.		The Timer Start Event is mapped to Start Node followed by a AcceptEventAction where the trigger is time event.  To support this signal, a separate Process must be created that runs infinitely and has a timer to send the Signal. A no action activity implemented as a timer to go off on a specific time and date or on a cycle. 

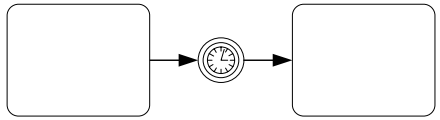
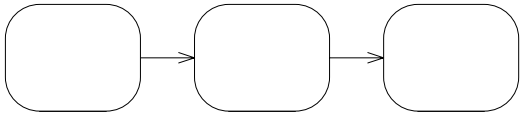
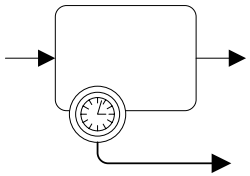
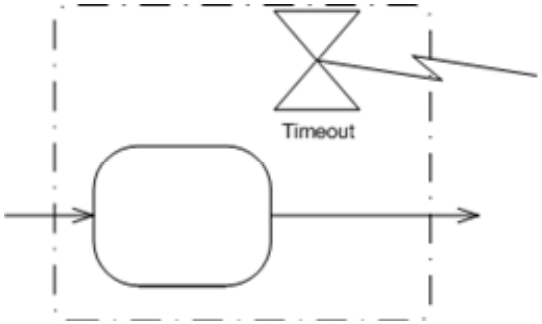
Rule Trigger	This indicates that a Signal is sent from another Process, based on the satisfaction of a Rule, should start the current process. There will be a corresponding End Event in another Process.		The Rule Start Event is mapped to Start Node followed by a receive Signal.  To support this signal, a separate Process must be created that runs infinitely and has an activity that checks data and a decision as to whether or not to send a signal. 
Link Trigger	This indicates that a Signal is sent from another Process should start the current process. There will be a corresponding End Event in another Process.	This indicates that a Signal is sent from another Process to start the current process. There will be a corresponding End Event in another Process. 	The Link Start Event is mapped to Start Node followed by a receive Signal. 
Multiple Trigger	Any number or combination of the above types of Start Events. Any one of the specified Triggers will start the Process.		Combinations of above, depending Event settings

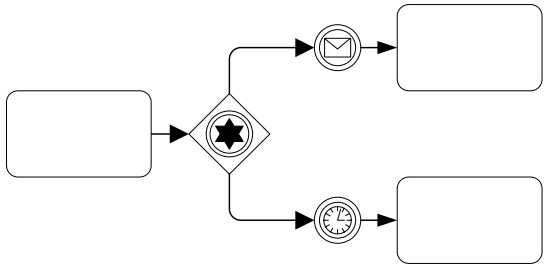
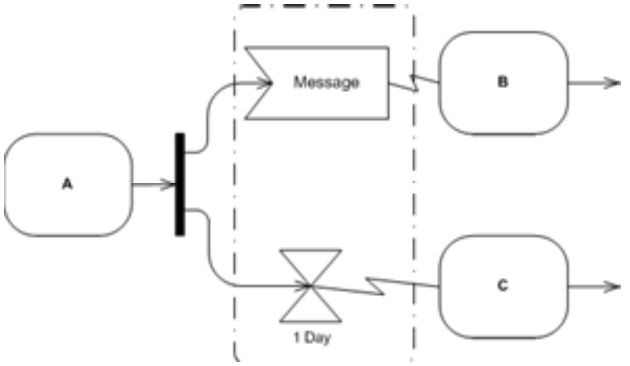
Intermediate (None, Message, Timer, Exception, Cancel, Compensation, Rule, Link, Multiple, Branching)	Intermediate Events occur between a Start Event and an End Event. It will affect the flow of the process, but will not start or (directly) terminate the process. Intermediate Events have "Triggers" that define the cause for the event. There are multiple ways that these events can be triggered.		Signals, Connectors, and various design patterns. See next 26 rows…
Message Trigger	The trigger is the receipt of a message from an outside source (outside of the Process).		See next three rows…

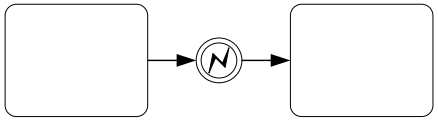
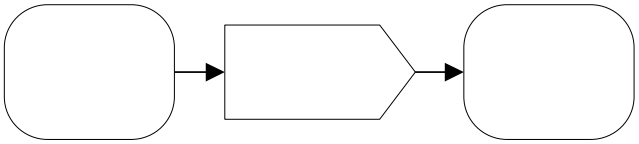
Within Normal Flow	When used in this form, the Message Intermediate Event acts as a place where a message is expected from an outside source. The Process (or that particular path) will wait for the message to arrive before it continues.	 In BPMN, this is also equivalent to a Receive Task. 	The Message Intermediate Event in flow can map to a Receive Signal.  Alternatively, an activity that receives the message can be used. 

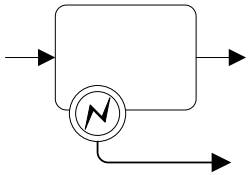
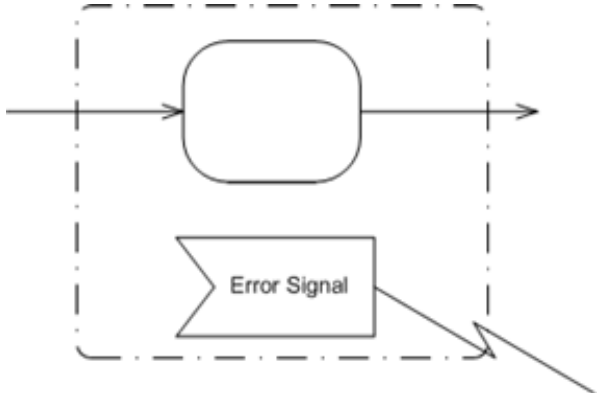
Attached to Activity Boundary	When used in this form, the Message Intermediate Event acts as a mechanism to interrupt an activity. If the activity is running and the particular message is received, then the activity will be stopped and the flow will be diverted to the Sequence Flow that exits the Intermediate Event. The configuration of the receipt of the message is basically the same as configuring an activity that receives a message.	The Intermediate Event is physically attached to the boundary of the activity. 	To create this behavior, an interruptible region is created to surround the activity. Also, a Receive Signal is placed within the region. An interrupting edge is used to exit the region (and the activity) with the signal arrives. 

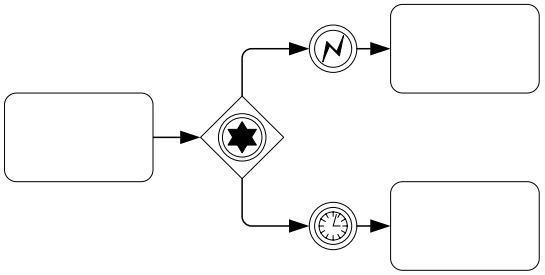
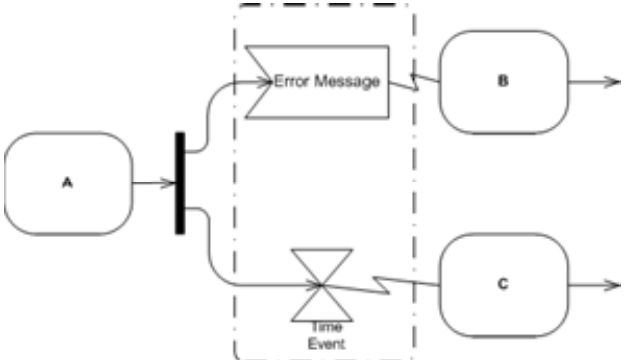
Within Event-Based Decision	When used in this form the Message Intermediate Event is a part of decision pattern that is based on Events that occur during the process. In this case, the first message that arrives will determine that path that is chosen. The occurrence of one message will exclude the other path(s).		The Message Intermediate Events will map to the receipt of separate Signals. All the signals to be received by the pattern must be available at the same time, so they follow a fork node. But only one will be used—to the exclusion of all others. To do this, the signal receipts are all placed within an interruptible region. When a signal is received, this will lead to an interrupting edge that will cause the closing of the region, thus excluding the other signal receipts. 
Timer Trigger	This event is triggered when the process time reaches the time specified in the attributes of the Event. The time may be specified as a recurring time/date or a relative time/date.		See next three rows…

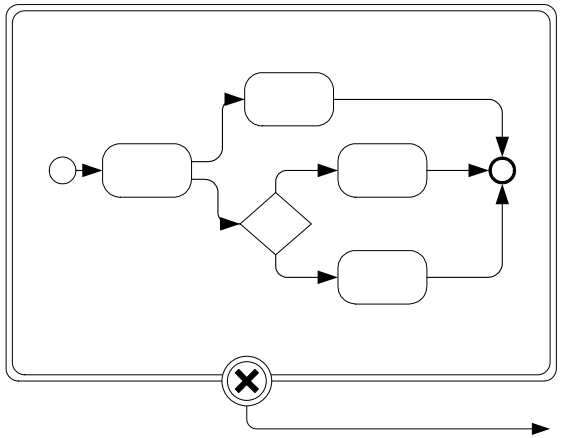
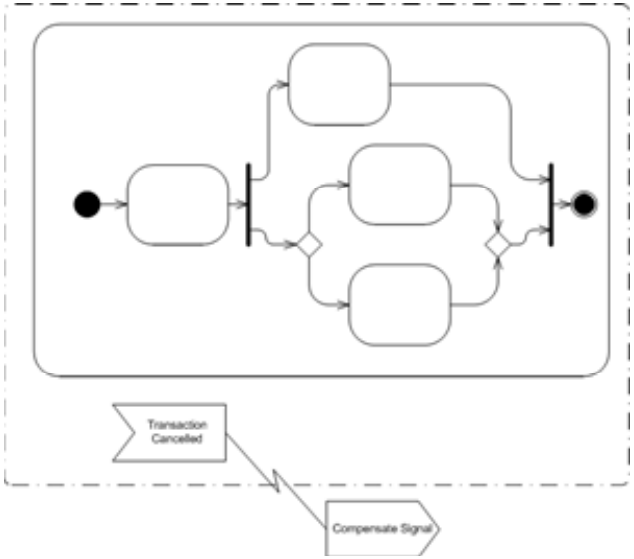
Within Normal Flow	When used in this form, the Timer Intermediate Event acts as a place a delay is expected. The Process (or that particular path) will wait until the process time reaches the time specified in the attributes of the Event.		The Timer Intermediate Event is mapped to an activity that is implemented as a Timer. [Add a delay stereotype to metamodel.] 
Attached to Activity Boundary	When used in this form, the Timer Intermediate Event acts as a mechanism to interrupt an activity. If the activity is running and the specified time is reached, then the activity will be stopped and the flow will be diverted to the Sequence Flow that exits the Intermediate Event.	The Intermediate Event is physically attached to the boundary of the activity. 	To create this behavior, an interruptible region is created to surround the activity. Also, an AcceptEventAction where the trigger is time event is placed within the region. An interrupting edge is used to exit the region (and the activity) with the signal arrives. 

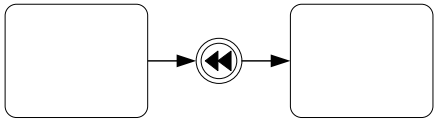
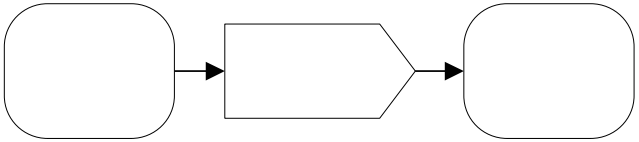
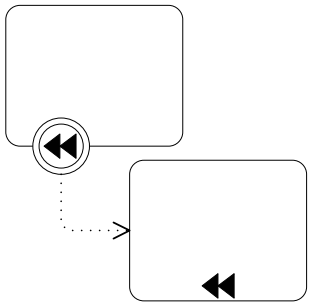
Within Event-Based Decision	When used in this form the Timer Intermediate Event is a part of decision pattern that is based on Events that occur during the process. In this case, the first event, either the Message or the time event, that arrives will determine that path that is chosen. The occurrence of one event will exclude the other path(s).		The Timer Intermediate Event will map to the receipt of a Signal that is an AcceptEventAction where the trigger is time event. All the signals to be received by the pattern must be available at the same time, so they follow a fork node. But only one will be used—to the exclusion of all others. To do this, the signal receipts are all placed within an interruptible region. When a signal is received, this will lead to a interrupting edge that will cause the closing of the region, thus excluding the other signal receipts. 
Exception Trigger	This Event either reacts to or generates an exception, depending on the use of the Event in the Process.		See next three rows…

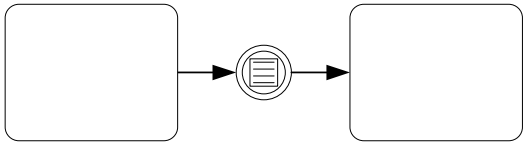
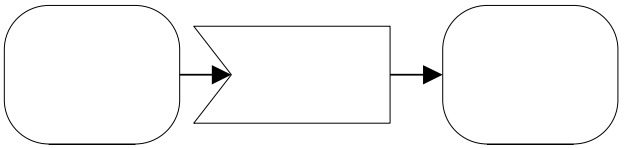
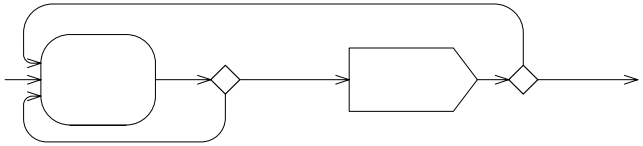
Within Normal Flow	When used in this form, the Exception Intermediate Event acts as a place a where an exception should be triggered (e.g., the exception is thrown). The Process will trigger the exception and then continue to the next activity.		The Exception Intermediate Event is mapped to a Send Signal of type Exception. RaiseExceptionAction <Review all EventActions.> 

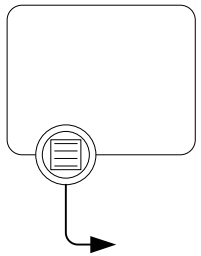
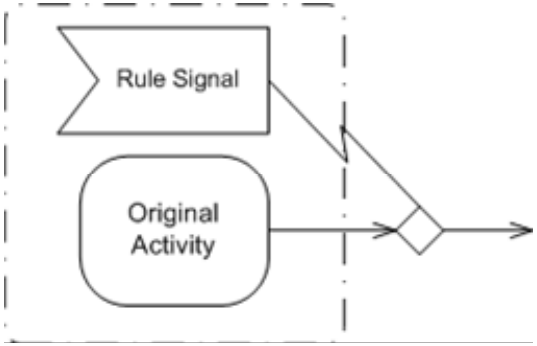
Attached to Activity Boundary	When used in this form, the Exception Intermediate Event acts as a mechanism to interrupt an activity. If the activity is running and the specified exception has been triggered, either through an application message or through another Intermediate Event that "throws" the exception, then the activity will be stopped and the flow will be diverted to the Sequence Flow that exits the Intermediate Event.	The Intermediate Event is physically attached to the boundary of the activity. 	To create this behavior, an interruptible region is created to surround the activity. Within the region, a signal receipt is added to catch the signal that was created elsewhere. An interrupting edge that exits the signal is used to exit the region (and the activity) when the signal arrives.  Note: UML also provides an exception handling mechanism that creates protected nodes and specific activities to be used when an exception occurs. This mechanism is more specific than the patterns created by a BPMN model. BPMN uses a more general capability of diverting flow when an exception occurs.

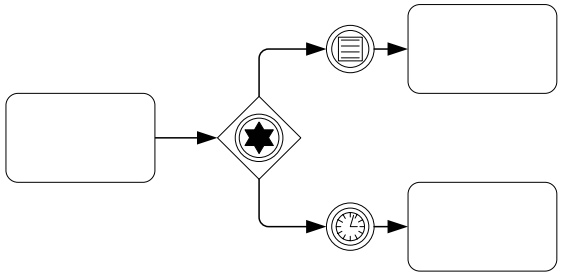
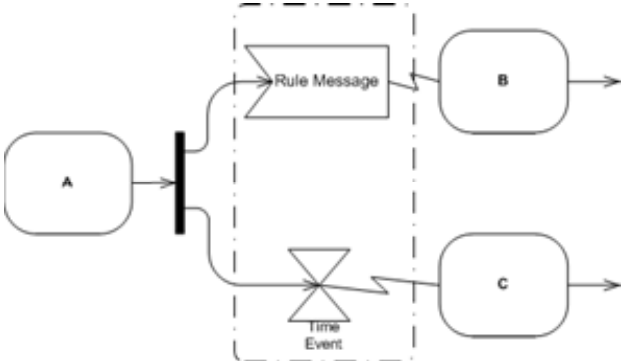
Within Event-Based Decision	When used in this form the Exception Intermediate Event is a part of decision pattern that is based on Events that occur during the process. In this case, the first event, either the exception or the time event, that arrives will determine that path that is chosen. The occurrence of one event will exclude the other path(s).		The Exception Intermediate Event will map to the receipt of a Signal. All the signals to be received by the pattern must be available at the same time, so they follow a fork node. But only one will be used—to the exclusion of all others. To do this, the signal receipts are all placed within an interruptible region. When a signal is received, this will lead to a interrupting edge that will cause the closing of the region, thus excluding the other signal receipts. 
Cancel Trigger	This Event only applies to Sub-Processes that are defined as Transactions.		See next row…

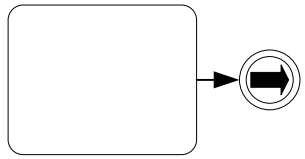
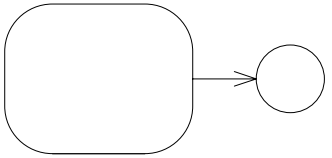
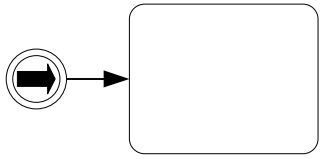
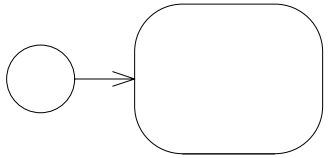
Attached to Activity Boundary	This is the only form that this Event can be used. When attached to the boundary of a Transaction Sub-Process, the Cancel Intermediate Event acts as a mechanism to interrupt an activity. If the activity is running and the specified cancel notification has been triggered, either through an application message, transaction protocol, or through a Cancel End Event within the Transaction, then the activity will be stopped and the flow will be diverted to the Sequence Flow that exits the Intermediate Event.		To use the Cancel Intermediate Event, an interruptible region encloses a Sub-Process. A receive signal is used to interrupt the region. If some mechanism for roll-back, including compensation is required, then a Compensate Signal should be sent after the Transaction has been interrupted. 
Compensation Trigger	This Event either reacts to or generates a compensation, depending on the use of the Event in the Process.		See next two rows…

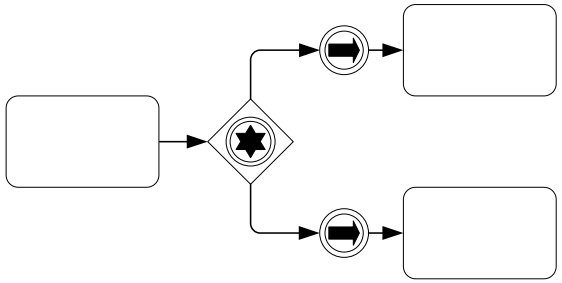
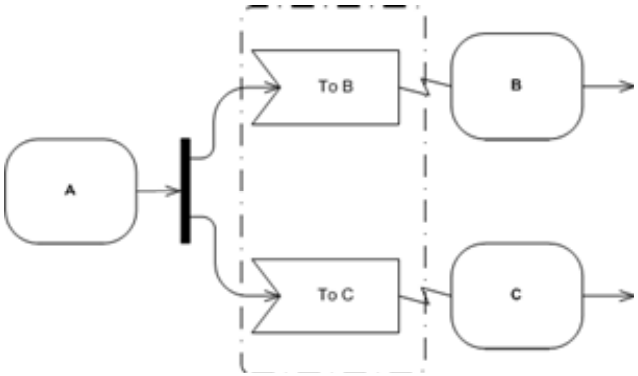
Within Normal Flow	When used in this form, the Compensation Intermediate Event acts as a place a where a compensation should be triggered (e.g., the compensation is thrown). The Process will trigger the compensation and then continue to the next activity.		This could be a signal. The receipt of the signal should be in a separate process that is ready to receive it. 
Attached to Activity Boundary	When used in this form, the Compensation Intermediate Event acts as a mechanism to identify the activity that will be used to compensate for the original performance of the activity. If the activity *has been completed* and the specified compensation has been triggered, then the (compensation) activity that is Associated with the original activity will be performed.		See Compensation Association.

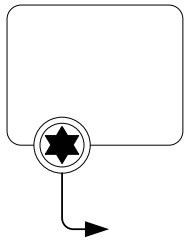
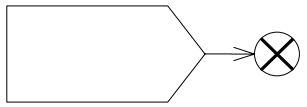
Rule Trigger	This event is triggered when a business rule becomes true during the performance of the process.		See next three rows…
Within Normal Flow	When used in this form, the Rule Intermediate Event acts as a place where a delay will occur until a specific rule has been satisfied. The Process (or that particular path) will wait for the rule to be true before it continues.		This map to a signal that arrives when the rule is determined to be true. Could be guard – a guard will wait. <Check>  To support this signal, a separate Process must be created that runs in parallel and has an activity that checks data and a decision as to whether or not to send a signal. 

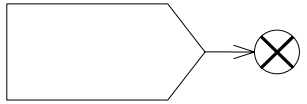
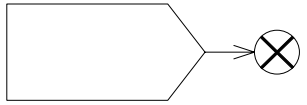
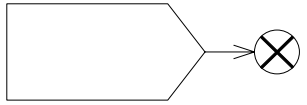
Attached to Activity Boundary	When used in this form, the Rule Intermediate Event acts as a mechanism to interrupt an activity. If the activity is running and the particular rule is satisfied, then the activity will be stopped and the flow will be diverted to the Sequence Flow that exits the Intermediate Event.		To create this behavior, an interruptible region is created to surround the activity. Also, a Receive Signal is placed within the region. An Interrupting edge is used to exit the region (and the activity) with the signal arrives. The Interrupting Edge and the normal edge that exits the activity are merged since only one of them will happen.  To support this signal, a separate Process must be created that runs in parallel and has an activity that checks data and a decision as to whether or not to send a signal (see row above).

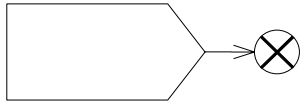
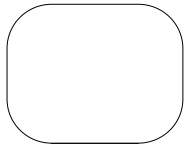
Within Event-Based Decision	When used in this form the Rule Intermediate Event is a part of decision pattern that is based on Events that occur during the process. In this case, the first event, either the rule being satisfied or the time event, that arrives will determine that path that is chosen. The occurrence of one event will exclude the other path(s).		The Rule Intermediate Event will map to the receipt of a Signal. All the signals to be received by the pattern must be available at the same time, so they follow a fork node. But only one will be used—to the exclusion of all others. To do this, the signal receipts are all placed within an interruptible region. When a signal is received, this will lead to a interrupting edge that will cause the closing of the region, thus excluding the other signal receipts. 
Link Trigger	This Event either acts as a "Go To" object or receives a signal generated by another Process, depending on the use of the Event in the Process.		See next two rows…

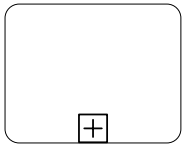
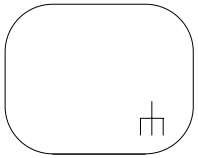
Within Normal Flow			This maps to an Activity Edge Connector. 
Within Normal Flow			This maps to an Activity Edge Connector. 

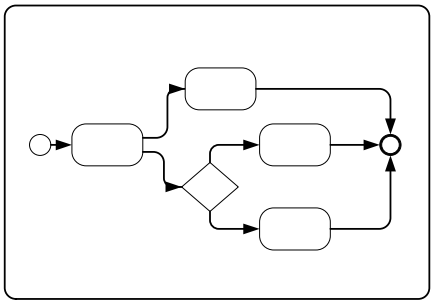
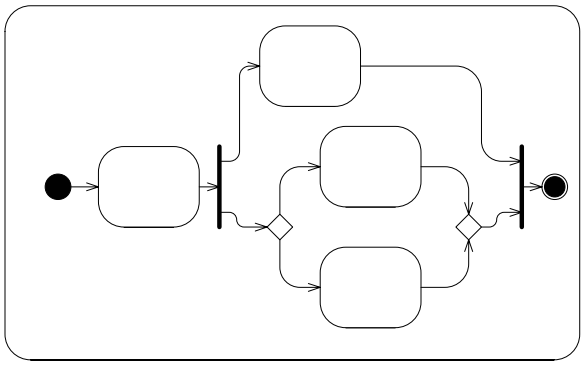
Within Event-Based Decision	When used in this form the Link Intermediate Event is a part of decision pattern that is based on Events that occur during the process. In this case, the first event that arrives will determine that path that is chosen. The occurrence of one Event will exclude the other path(s).		The Link Intermediate Event will map to the receipt of a Signal. All the signals to be received by the pattern must be available at the same time, so they follow a fork node. But only one will be used—to the exclusion of all others. To do this, the signal receipts are all placed within an interruptible region. When a signal is received, this will lead to a interrupting edge that will cause the closing of the region, thus excluding the other signal receipts. 
Multiple Trigger	This Event can only be used as a mechanism to interrupt activities.		See next row…

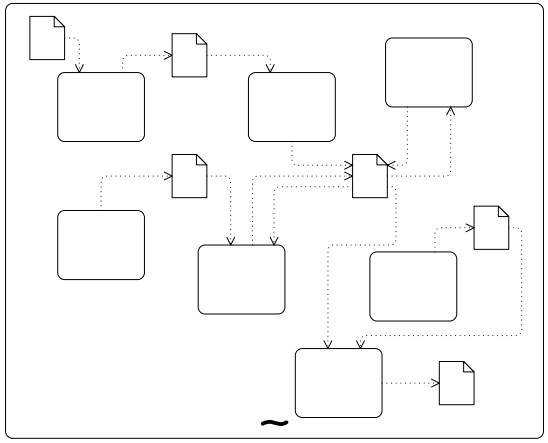
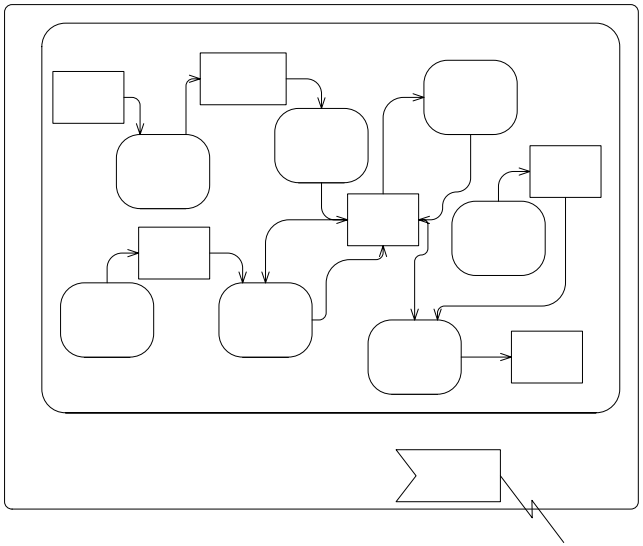
Attached to Activity Boundary	This is the only form that this Intermediate Event can be used. When attached to the boundary of an activity, the Multiple Intermediate Event acts as a mechanism to interrupt an activity. If the activity is running and *any one* of the specified triggers occurs, then the activity will be stopped and the flow will be diverted to the Sequence Flow that exits the Intermediate Event.		Combinations of above, depending Event settings
End (None, Message, Exception, Cancel, Compensation, Link, Terminate, Multiple)	As the name implies, the End Event indicates where a process will end. End Events may define a "Result" that is a consequence of a Sequence Flow ending. These Results may require some processing before the Process can be completed.		Activity Final, Flow Final. See next seven rows… 
Message Result	A message is sent before the Process (or path) is completed.		This maps to a Signal that is followed by a flow final node. 

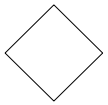
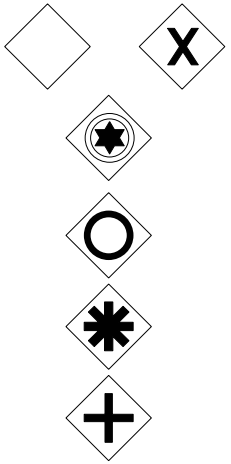
Exception Result	An exception is "thrown" before the Process (or path) is completed.		This maps to a signal that is followed by a flow final node. It will be paired with a signal receipt that is with an interruptible region. See Exception Trigger. 
Cancel Result	This is only used within a Sub-Process that is set as a Transaction. A cancel notification is "thrown" and the transaction will be stopped and rolled back.		This maps to a signal that cancels a transaction that is followed by a flow final node. It will be paired with a signal receipt that is contained an interruptible region. See Cancel Trigger. 
Compensate Result	A Compensation notice is "thrown" before the Process (or path) is completed.		This maps to a compensation signal that is followed by a flow final node. It will be paired with a signal receipt for this signal. This receipt will likely start another activity used for compensation. See Compensation Association. 

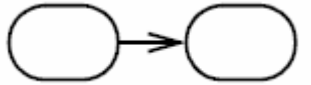
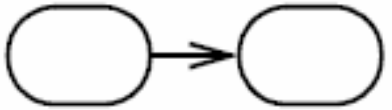
Link Result	This indicates that a Signal will be sent to another Process. There will be a corresponding Start Event in another Process.		This maps to a Signal that is followed by a flow final node. This will be paired with a corresponding receipt signal that will start another activity. See Link Trigger. 
Multiple Result	All of the above Results (except Cancel) may be used in combination and any quantity. All specified results will be processed before the Process (or path) is completed.		Combinations of the above End Events, depending Event settings
Terminate Result	This End Event will signal the end of the Path and will cause the termination (without compensation) of any other path that may still be active.		This maps to the Activity Final 
Task (Atomic)	A Task is an atomic activity that is included within a Process. A Task is used when the work in the Process is not broken down to a finer level of Process Model detail.		This maps to an Action for all TaskTypes except Referenced or an Activity with a CallBehaviorAction for the TaskType Referenced. 

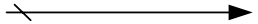
Process/Sub-Process (non-atomic)	A Sub-Process is a compound activity that is included within a Process. It is compound in that it can be broken down into a finer level of detail (a Process) through a set of sub-activities.	See Next Two Figures	See next to rows…
Collapsed Sub-Process	The details of the Sub-Process are not visible in the Diagram. A "plus" sign in the lower-center of the shape indicates that the activity is a Sub-Process and has a lower-level of detail.		This maps to an Activity with a CallBehaviorAction (for referenced—SubProcessType is Independent) or StructuredActivityNode (for nested—SubProcessType is Embedded). 

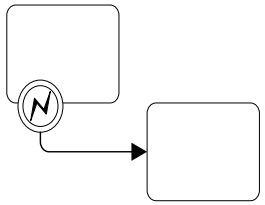
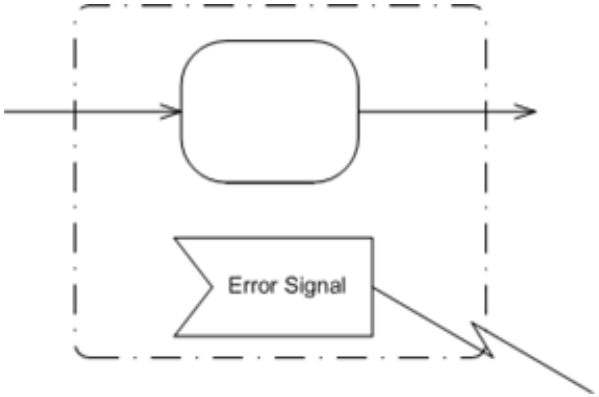
Expanded Sub-Process	The boundary of the Sub-Process is expanded and the details (a Process) are visible within its boundary. Note that Sequence Flow cannot cross the boundary of a Sub-Process.		This maps to an Activity with a CallBehaviorAction (for referenced—SubProcessType is Independent) or StructuredActivityNode (for nested—SubProcessType is Embedded). 

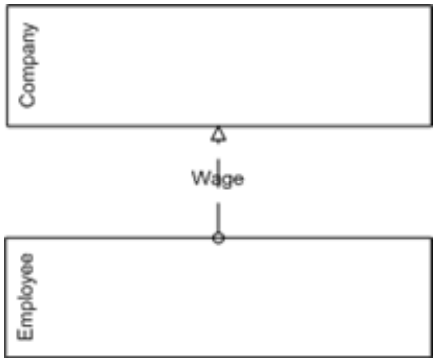
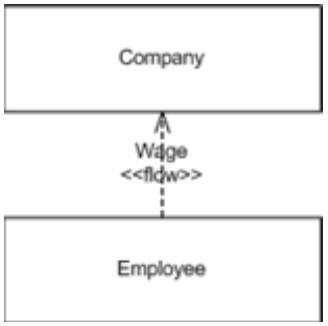
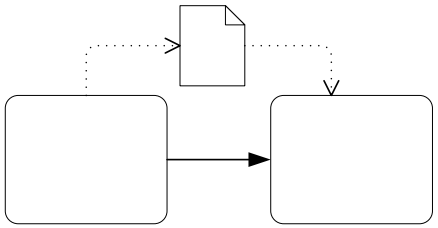
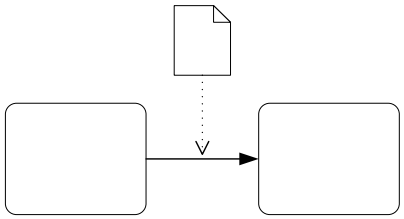
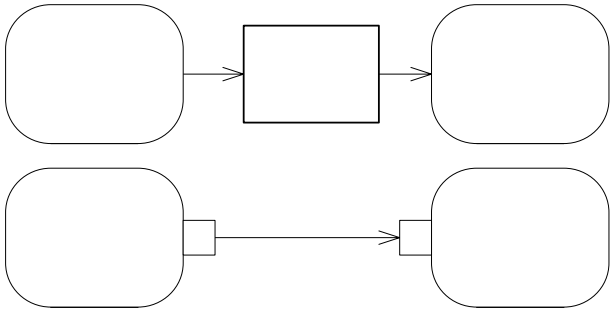
Ad-Hoc Process	This type of Process is a collection of activities that has no specific sequence or multiplicity. Presumably, the performers of the activities will determine when and how often the activities will occur. In addition, there is some condition that will determine that the Process as completed.	The activities in this process can be done in almost any order. But there are some dependencies as shown by the input and output data objects. But the “data flow” does not imply the sequence of performance. For example, The “Write Text” Task produces a text draft, but the next activity may be the “Generate Graphics” or “Organize References” Task. 	This maps to an Activity that has Sub-Activities. The sub-activities are not organized in with any control flow. However, data flow may be used. [Unfortunately (for Ad-Hoc) Data flow also contains control flow, which creates unwanted implications of control]  This particular type of activity does not map to an execution level, such as BPEL. This needs to be supported by an activity somewhere that will monitor the condition of the Ad-Hoc Process and then send a signal that will be used to interrupt the interruptible region.

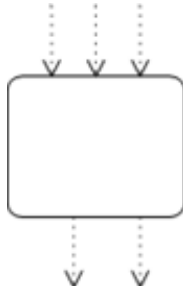
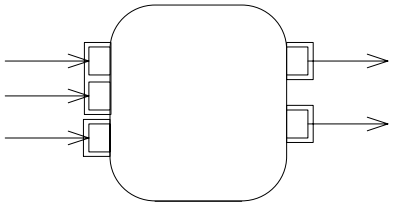
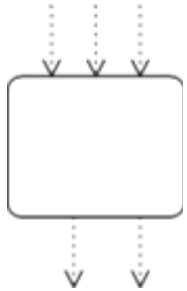
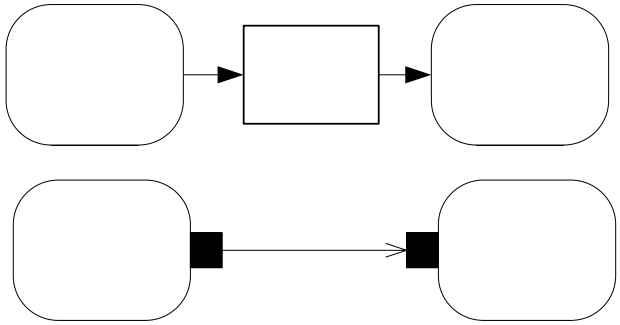
Gateway	A Gateway is used to control the divergence and convergence of multiple Sequence Flow. Thus, it will determine branching, forking, merging, and joining of paths.		
Gateway Control Types	Icons within the diamond shape will indicate the type of flow control behavior. The types of control include: • XOR -- exclusive decision and merging. Both Data-Based and Event-Based. Data-Based can be shown with or without the "X" marker. • OR -- inclusive decision • Complex -- complex conditions and situations (e.g., 3 out of 5) • AND -- forking and joining Each type of control affects both the incoming and outgoing Flow.		See the sections on Decisions, Merges, Forks, and Joins
Sequence Flow	A Sequence Flow is used to show the order that activities will be performed in a Process.	See next seven figures	

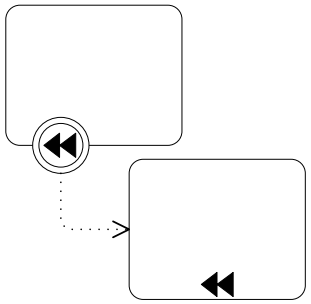
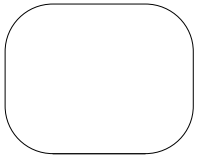
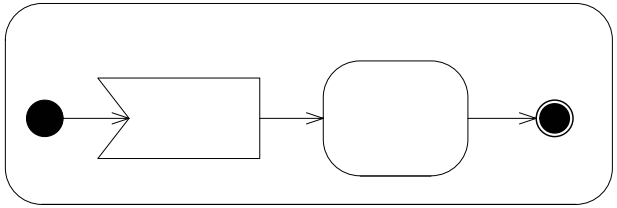
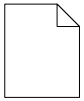
Normal flow	Normal Sequence Flow refers to the flow that originates from a Start Event and continues through activities via alternative and parallel paths until it ends at an End Event.		Control Flow 
Uncontrolled flow	Uncontrolled flow refers to flow that is not affected by any conditions or does not pass through a Gateway. The simplest example of this is a single Sequence Flow connecting two activities. This can also apply to multiple Sequence Flow that converge on or diverge from an activity. For each uncontrolled Sequence Flow a "Token" will flow from the source object to the target object.		Control Flow 

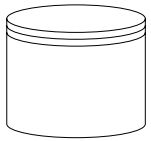
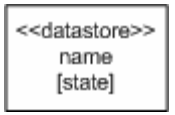
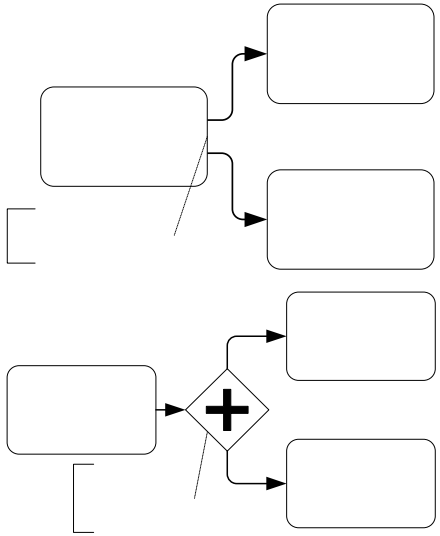
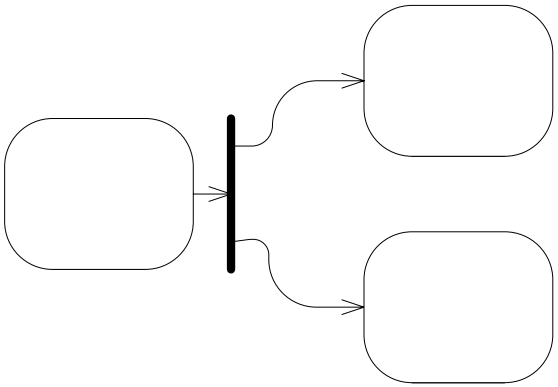
Conditional flow	Sequence Flow can have condition expressions that are evaluated at runtime to determine whether or not the flow will be used. If the conditional flow is outgoing from an activity, then the Sequence Flow will have a mini-diamond at the beginning of the line (see figure to the right). If the conditional flow is outgoing from a Gateway, then the line will not have a mini-diamond (see figure in the row above).		Control Flow (Activity Edge with Guard) 
Default flow	For Data-Based Exclusive Decisions, one type of flow is the Default condition flow. This flow will be used only if all the other outgoing conditional flow is not true at runtime. These Sequence Flow will have a diagonal slash will be added to the beginning of the line (see the figure to the right). Note that it is an Open Issue whether Default Conditions will be used for Inclusive Decision situations.		Control Flow (Activity Edge with Guard set to “else”) 

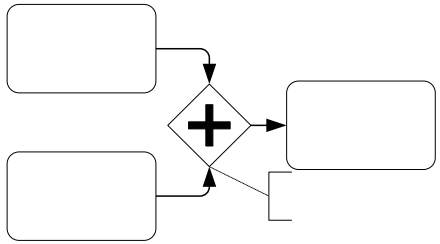
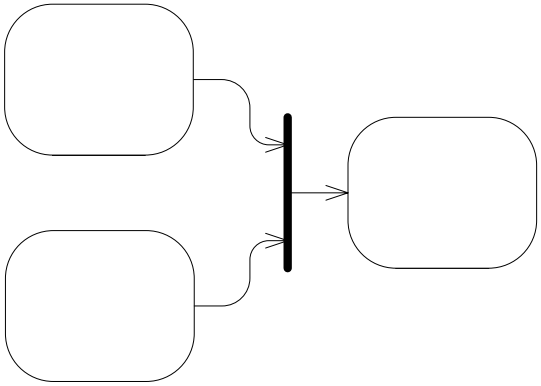
Exception flow	Exception flow occurs outside the normal flow of the Process and is based upon an Intermediate Event that occurs during the performance of the Process. All the Intermediate Event Triggers, except Compensation, can create Exception flow (see Message Trigger, Timer Trigger, Exception Trigger, Cancel Trigger, Rule Trigger, and Link Trigger)		This maps to an Interruptible Region. The details depends on the type of Intermediate Event Trigger. See the mappings shown in the different Trigger types as referenced in the second column. The example shown below is for an Exception Trigger. An interruptible region is created to surround the activity. Within the region, a signal receipt is added to catch the signal that was created elsewhere. An interrupting edge that exits the signal is used to exit the region (and the activity) when the signal arrives. 

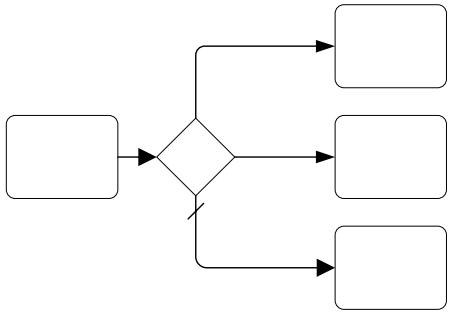
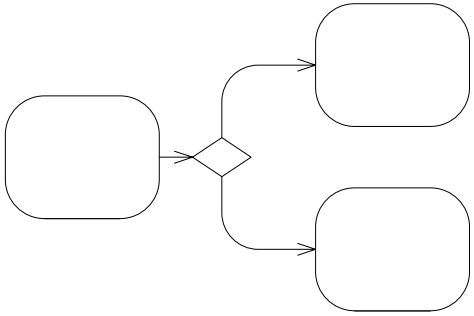
Message Flow	A Message Flow is used to show the flow of messages between two entities that are prepared to send and receive them. In BPMN, two separate Pools in the Diagram will represent the two entities.		Information Flow to an entity. This will be paired with BPMN pools that will be mapped to partitions and entities. 
Object Flow	In BPMN there is no specific creation of "Object Flow." However, the data/document input and output requirements are shown through Data Object Artifacts and Associations. This creates a complete decoupling of data and control flow. However, UML object flow MUST follow control flow. This creates some BPMN to UML mapping issues (see Ad-Hoc Process)	 Note that Sequence Flow is not required between the two activates in BPMN to show the flow of the object. 	This maps to Object Flow in both forms.  Note that a control flow is always paired with object flow.

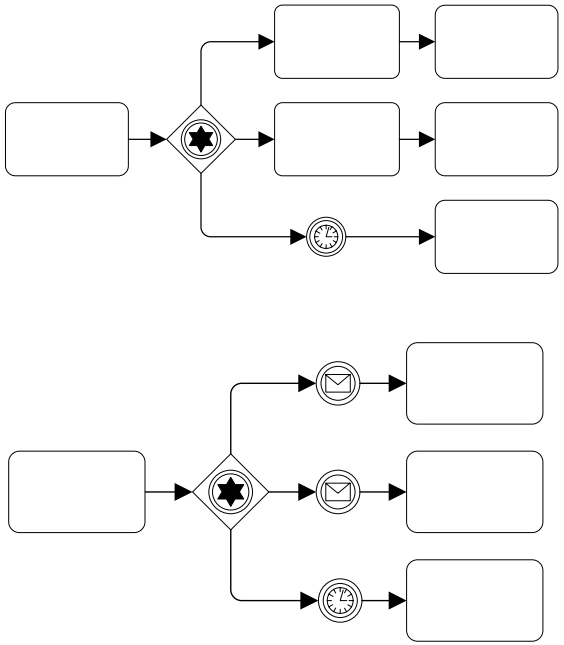
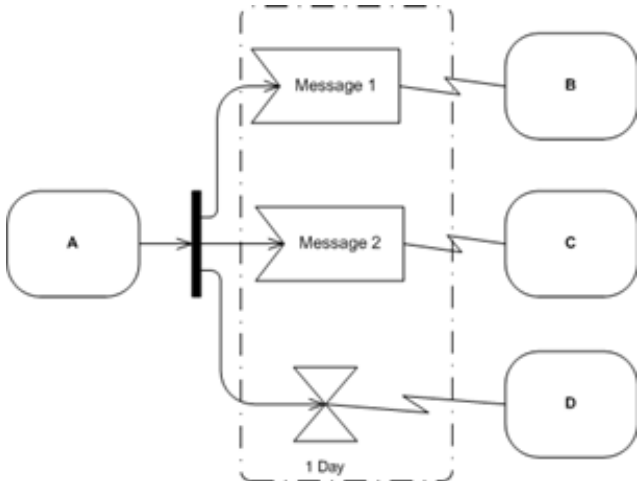
Complex Input/Output Requirements		This information is supported by activity Attributes. 	This maps to Parameter Sets attached to the activity to show AND and OR input/output requirements. 
Streaming Input/Output Requirements		The current version of BPMN does not graphically distinguish between types of inputs and outputs. This information is supported by activity Attributes. 	This maps to streaming style object flow or streaming pins. 

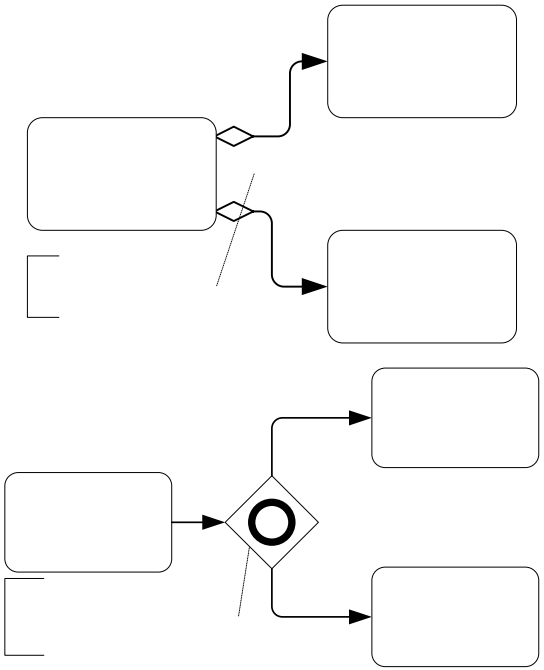
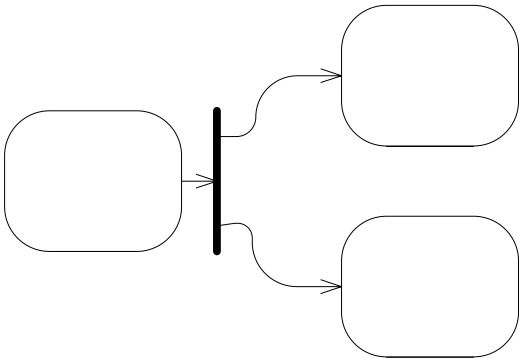
Compensation Association	Compensation Association occurs outside the normal flow of the Process and is based upon an event (a Cancel Intermediate Event) that is triggered through the failure of a Transaction or a Compensate Event. The target of the Association must be marked as a Compensation Activity.		There is no direct association with the original activity and the activity that is needed for compensation. Thus, the original activity is modeled by itself.  The activity used for compensation is set aside in a separate Process that is only activated if the Compensate signal is sent through some part of the original process. 
Data Object	Data Objects are considered artifacts because they do not have any direct affect on the Sequence Flow or Message Flow of the Process, but they do provide information about what the Process does.		This maps to an ObjectNode 

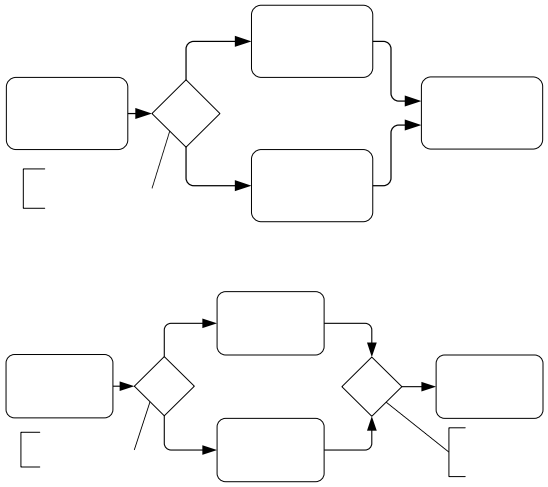
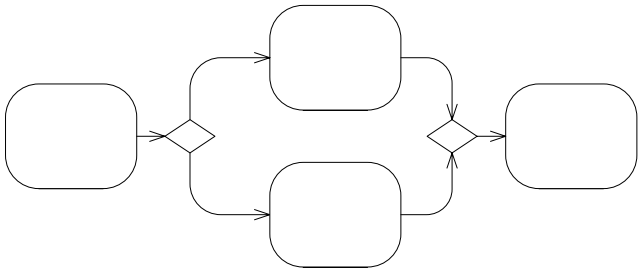
Database	Databases are considered artifacts because they do not have any direct affect on the Sequence Flow or Message Flow of the Process, but they do provide information about what the Process does.	This object is currently an extension to BPMN (through the extensibility of BPMN Artifacts). 	This maps to a DataStoreNode 
Fork (AND-Split)	BPMN uses the term "fork" to refer to the dividing of a path into two or more parallel paths (also known as an AND-Split). It is a place in the Process where activities can be performed concurrently, rather than serially. There are two options: Multiple Outgoing Sequence Flow can be used (see figure top-right). This represents "uncontrolled" flow is the preferred method for most situations. A Parallel (AND) Gateway can be used (see figure bottom-right). This will be used rarely, usually in combination with other Gateways.		This maps to a fork node. 

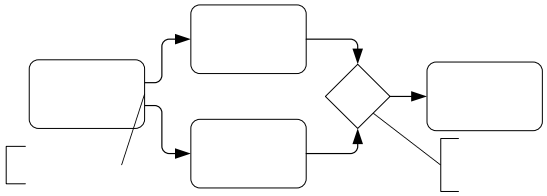
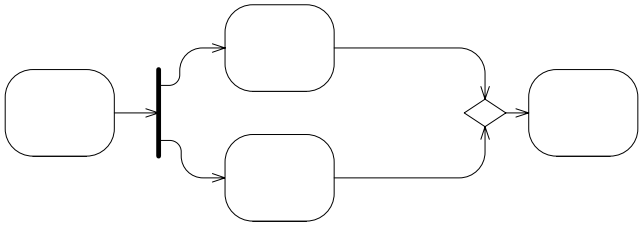
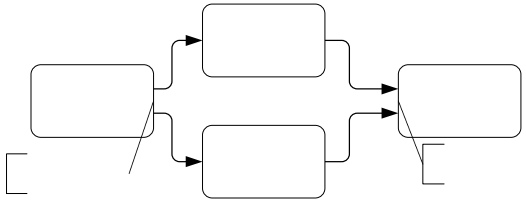
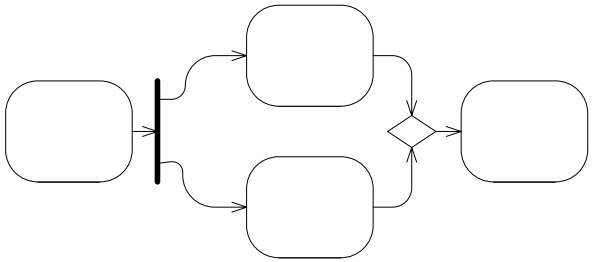
Join (AND-Join)	BPMN uses the term "join" to refer to the combining of two or more parallel paths into one path (also known as an AND-Join or synchronization). A Parallel (AND) Gateway is used to show the joining of multiple flows.		This maps to a Join Node 
Decision, Branching Point; (OR-Split)	Decisions are Gateways within a business process where the flow of control can take one or more alternative paths.	See next five rows.	
Exclusive	An Exclusive Gateway (XOR) restricts the flow such that only one of a set of alternatives may be chosen during runtime. There are two types of Exclusive Gateways: Data-based and Event-based.	See next two rows	See next two rows

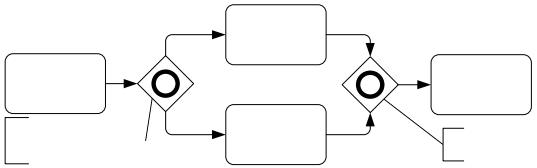
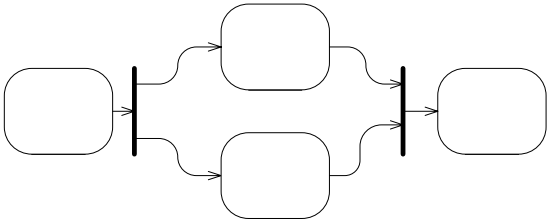
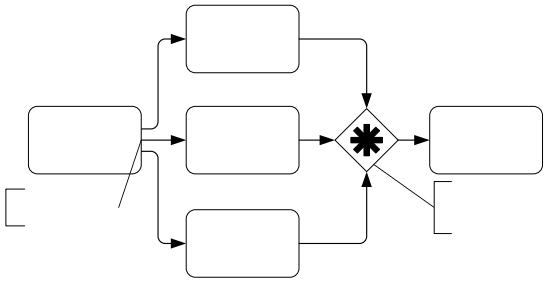
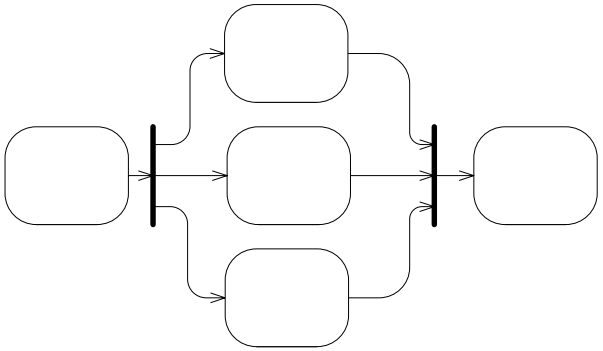
Data-Based	This Decision represents a branching point where Alternatives are based on conditional expressions contained within the outgoing Sequence Flow. Only one of the Alternatives will be chosen.		

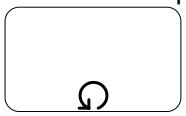
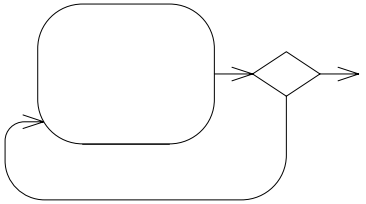
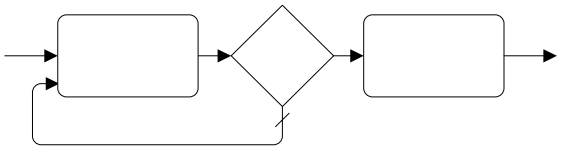
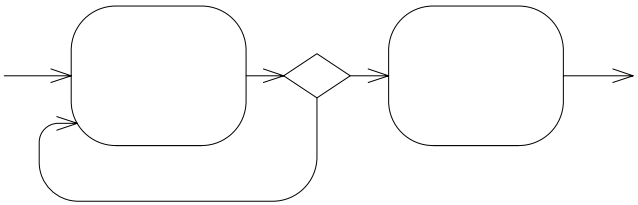
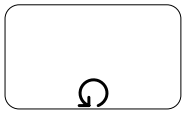
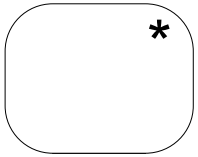
Event-Based	This Decision represents a branching point where Alternatives are based on an Event that occurs at that point in the Process. The specific Event, usually the receipt of a Message, determines which of the paths will be taken. Other types of Events can be used, such as Timer. Only one of the Alternatives will be chosen. There are two options for receiving Messages: Tasks of Type Receive can be used (see figure top-right). Intermediate Events of Type Message can be used (see figure bottom-right). The details of the mapping pattern will depend on the specific Intermediate Events that are used (see Message Trigger, Timer Trigger, Exception Trigger, Rule Trigger, and Link Trigger).		There is not a specific UML element to which this behavior will map, rather, it is a specific pattern of UML elements. In general, there will be a set of signals that will be used to determine the appropriate path to take. All the signals to be received by the pattern must be available at the same time, so they follow a fork node. But only one will be used—to the exclusion of all others. To do this, the signal receipts are all placed within an interruptible region. When a signal is received, this will lead to an interrupting edge that will cause the closing of the region, thus excluding the other signal receipts. 

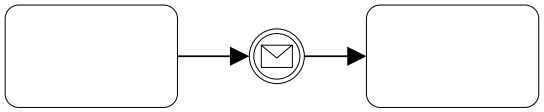
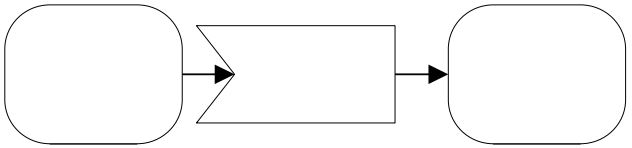
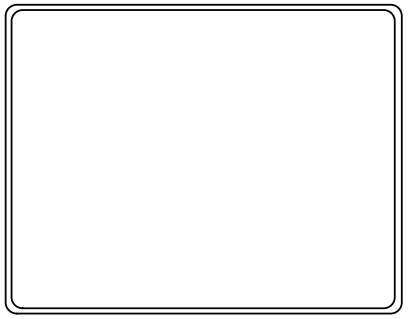
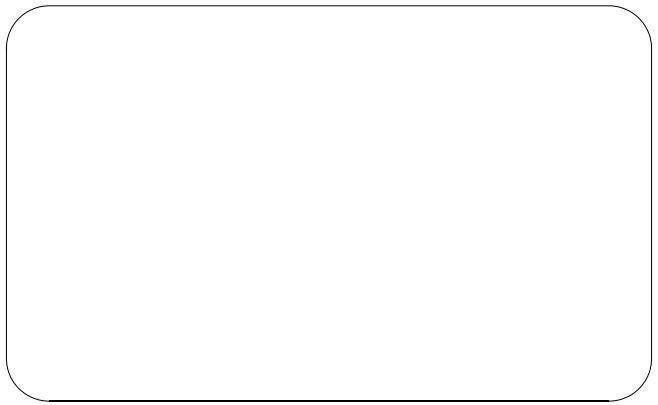
Inclusive	This Decision represents a branching point where Alternatives are based on conditional expressions contained within the outgoing Sequence Flow. In some sense it is a grouping of related independent Binary (Yes/No) Decisions. Since each path is independent, all combinations of the paths may be taken, from zero to all. However, it should be designed so that at least one path is taken. There are two versions of this type of Decision: The first uses a collection of conditional Sequence Flow, marked with mini-diamonds (see top-right figure). The second uses an OR Gateway, usually in combination with other Gateways (see bottom-right picture).		This maps to a fork node with outgoing control flow with guards. 

Merging (OR-Join)		See next five rows…	See next five rows…
Simple Merge			This maps to a merge node. 

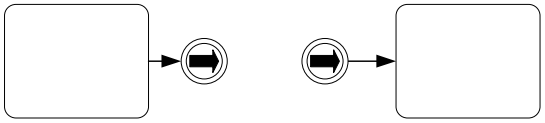
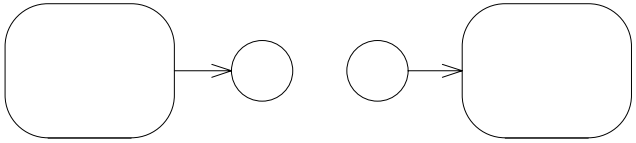
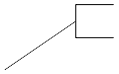
Discriminator	This creates a situation were there are multiple Tokens merging, but only the first Token is allowed to continue.	The Exclusive Gateway will accept the first Token and then exclude any others. 	This maps to a merge node. However, the edges leading to the merge node have guards that prohibit the flow if the other edge(s) leading to the same merge node have not yet occurred. Thus, the exclusive behavior is handled by the guards. 
Multiple Merge	This allows multiple Tokens to pass through the merge.	Uncontrolled (no Gateway) Sequence Flow into Task “D” will result in a separate instance of the Task for each Token that arrives. 	This maps to a merge node that is preceded (upstream) by a fork node. 

Synchronizing Merge	This synchs up the appropriate number of Tokens.	Either one or two Tokens will be merged. Activity “D” will occur once. 	This maps to a join node with a condition. 
N out of M Join	This is a join situation where a number (greater than 1), but less than the total number of Tokens is required for the process to continue. A condition can be specified, which will determine the appropriate number.	A Complex Gateway can be used to define the conditions required at the join. 	This maps to a condition for a Join node. 
Looping	BPMN provides 2 (two) mechanisms for looping within a Process.	See Next Two Figures	

Activity Looping	The properties of Tasks and Sub-Processes will determine if they are repeated or performed once. There are two types of loops: Standard and Multi-Instance. A small looping indicator will be displayed at the bottom-center of the activity.	Standard Loop 	The condition within the activity is extracted and a Decision is created. Then flow is connected back to the activity for the false path out of the Decision. 
Flow Looping	Loops can 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 is an incoming Sequence Flow for the original object.	Loop with Sequence Flow 	This maps to a Loop with control flow (or data flow). 
Multiple Instances	The attributes of Tasks and Sub-Processes will determine if they are repeated or performed once. A small parallel indicator will be displayed at the bottom-center of the activity.	Multiple Instance in Parallel  Multiple Instances in Sequence 	This maps to an Expansion Region. 

Process Break (something out of the control of the process makes the process pause)	A Process Break is a location in the Process that shows where an expected delay will occur within a Process. An Intermediate Event is used to show the actual behavior (see top-right figure). In addition, a Process Break artifact, as designed by a modeler or modeling tool, can be associated with the Event to highlight the location of the delay within the flow.		This maps to a Signal that occurs during the middle of the process. 
Transaction	A transaction is a Sub-Process that is supported by special protocol that insures that all parties involved have complete agreement that the activity should be completed or cancelled. The attributes of the activity will determine if the activity is a transaction. A double-lined boundary indicates that the activity is a Transaction.		Activity stereotyped as Transaction. [We could add the double-lined boundary as part of the stereotype.] 

Nested Sub-Process (Inline Block)	A nested Sub-Process is an activity that shares the same set of data as its parent process. This is opposed to a Sub-Process that is independent, re-usable, and referenced from the parent process. Data needs to be passed to the referenced Sub-Process, but not to the nested Sub-Process.	There is no special indicator for nested Sub-Processes, it is supported by Sub-Process attributes.	This maps to a StructuredActivityNode
Group (a box around a group of objects for documentation purposes)	A grouping of activities that does not affect the Sequence Flow. The grouping can be used for documentation or analysis purposes. Groups can also be used to identify the activities of a distributed transaction that is shown across Pools.		Interruptible Region (without an Interruption Edge)? Partition 
Off-Page Connector	Generally used for printing, this object will show where the Sequence Flow leaves one page and then restarts on the next page. A Link Intermediate Event can be used as an Off-Page Connector.		This maps to an Activity Edge Connector 

Go To object	Related to Off-Page Connector. Creates a virtual connection between two locations of a diagram.	The exact use of Link Intermediate Events is an open issue for BPMN. 	This maps to an Activity Edge Connector 
Association	An Association is used to associate information with flow objects. Text and graphical non-flow objects can be associated with the flow objects.		Not sure. Could be related to data flow, but also relates to other Artifacts besides Data Objects. Data Flow
Text Annotation (attached with an Association)	Text Annotations are a mechanism for a modeler to provide additional information for the reader of a BPMN Diagram.		This maps to a Note 
Pool	A Pool is a "swimlane" and a graphical container for partitioning a set of activities from other Pools, usually in the context of B2B situations.		Partition with no parent partition. It is also an Entity so that Information Flow can be shown between two Pools. 

Lanes	A Lane is a sub-partition within a Pool and will extend the entire length of the Pool, either vertically or horizontally. Lanes are used to organize and categorize activities within a Pool.		This maps to a Partition. 

Table 2 BPD Complete Element Set

Examples & Discussion

Comment: Steve White to provide some examples.