Chapter 4 - Advanced Process Modeling

More on Rework and Repetition

Loop activity: is notation for a **structured** cycle - a repetition block delimited by a single entry point to the cycle and a single exit point from the cycle.

- Captures **sequential repetition** meaning instances of the loop activity are executed one after the other.
- Note how we use an annotation "Until responde approved" to specify the loop condition.
- We may or may not specify the content of a loop sub-process but if we do, **don't forget to put a decision activity as the last activity in the sub-process**, otherwise we don't know when to repeat the sub-process.

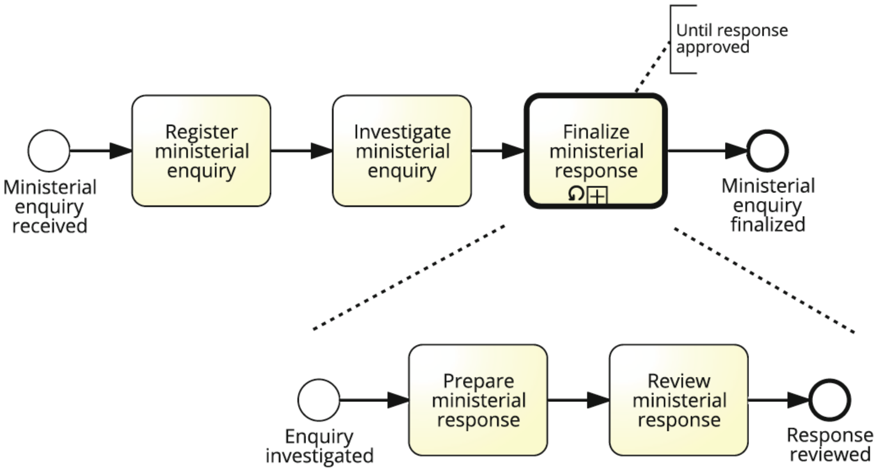

Until response approved
Ministerial enquiry received
Register ministerial enquiry
Investigate ministerial enquiry
Finalize ministerial response
Ministerial enquiry finalized
Enquiry investigated
Prepare ministerial response
Review ministerial response
Response reviewed

Fig. 4.1 The process model for addressing ministerial correspondence of Figure 3.13 simplified using a loop activity

Cycle: it is unstructured - there is more than one entry and/or exit point or the entry/exit point might be inside the repetition block.

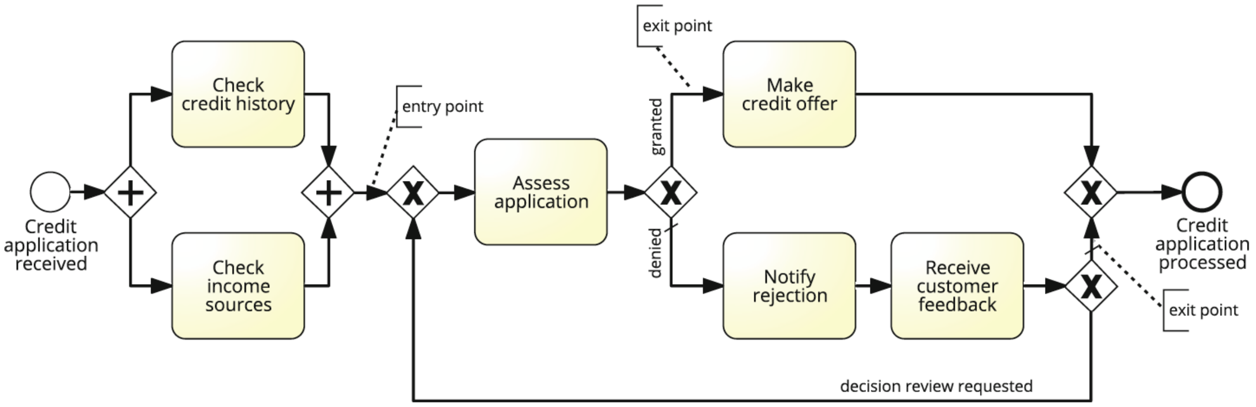
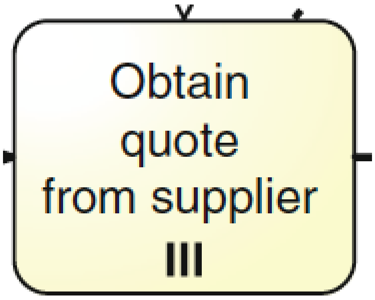


Fig. 4.2 An example of unstructured cycle

This cycle has one entry point and two exit points, cycles like this one cannot be rewritten as a cycle with only one exit point, unless additional conditions are used to specify the conditions in which the cycle can be exited.

Parallel Repetition

- Sequential repetition is not enough, sometimes we need to execute multiple instances of the same activity at the same time.
- To do this we use a **multi-instance activity**.
- **Multi-instance activity**: indicates an activity (task or sub-process) that is executed multiple times **concurrently** (potentially in parallel).
    - Useful when the same activity is executed for multiple entities or data items.
    - Represented as an activity marked with 3 small vertical lines at the bottom.



- In this example, we want to obtain a quote from five suppliers, after all quotes are received, they are evaluated and the best quote is selected. We use and AND-split to model the five tasks in parallel. However, this solution has two **issues**.
    1. The larger the number of suppliers, the larger this model will be because we need 1 task per supplier.
    2. We need to revise the model every time the number of suppliers changes (we should use an organisational **database** that keeps an updated list of suppliers).

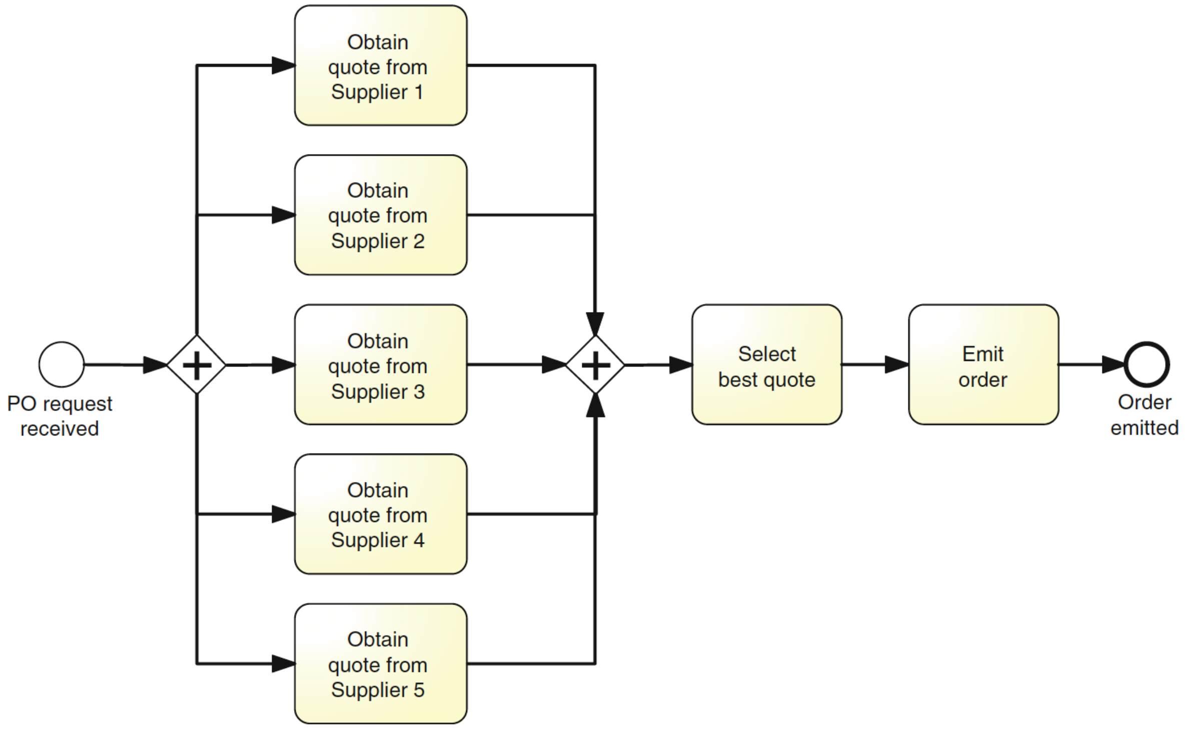

Obtain quote from Supplier 1
Obtain quote from Supplier 2
PO request received
Obtain quote from Supplier 3
Select best quote
Emit order
Order emitted
Obtain quote from Supplier 4
Obtain quote from Supplier 5

Fig. 4.3 Obtaining quotes from five suppliers

- Note that this model is smaller and it also works with a dynamic list of suppliers, which may change on an instance-by instance basis. To do this:
    - We need a **task to retrieve the list** of suppliers and **pass this list to the multi-instance activity**.
    - The data object also has a multi-instance symbol, this represents a **collection** - a list of data objects (in this case suppliers) and is used as input to a multi-instance activity. Note that t**he number of items in the collection is going to determine the number of activities instances to be created**. In alternative, we can specify the number of instances to be created via an annotation on the multi-instance activity (ex: "15 suppliers" or "as per suppliers database").
    - Suppose we have 20 suppliers in our database and we do not want to wait for all of them to attend our request for a quote. To do this, simply **annotate the multi-instance activity with the minimum number of instances that need to complete** before passing control to the outgoing arc ("complete when 5 quotes obtained").
    - When the multi-instance is triggered, 20 tokens are generated but as soon as the 5 instances complete all the other instances are canceled (tokens destroyed) and one token is sent to the

output to signal completion.

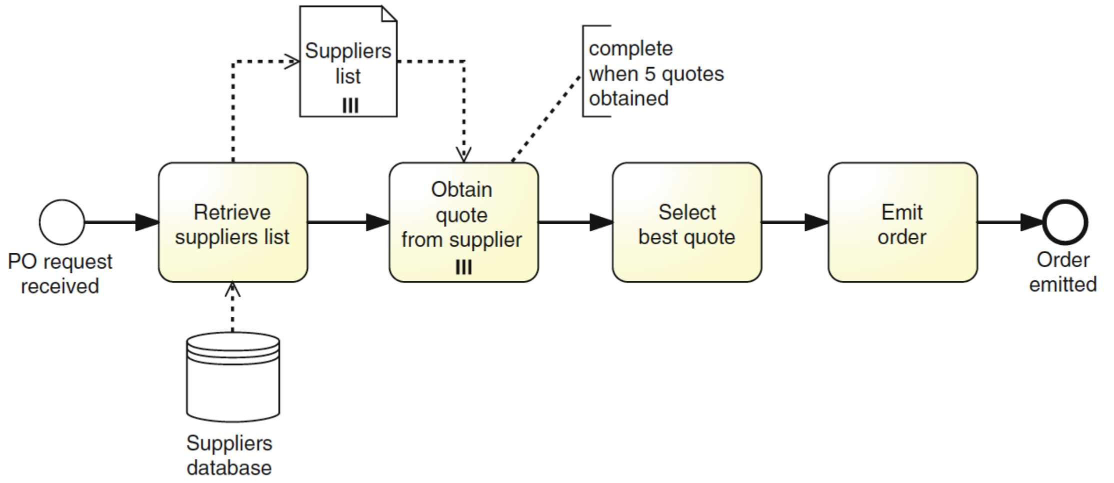


Fig. 4.4 Obtaining quotes from a number of suppliers determined on-the-fly

- No need to have pools "Supplier 1" and "Supplier 2", merge them into Supplier as **multi-instance pool** - represents a set of resources classes with similar characteristics.

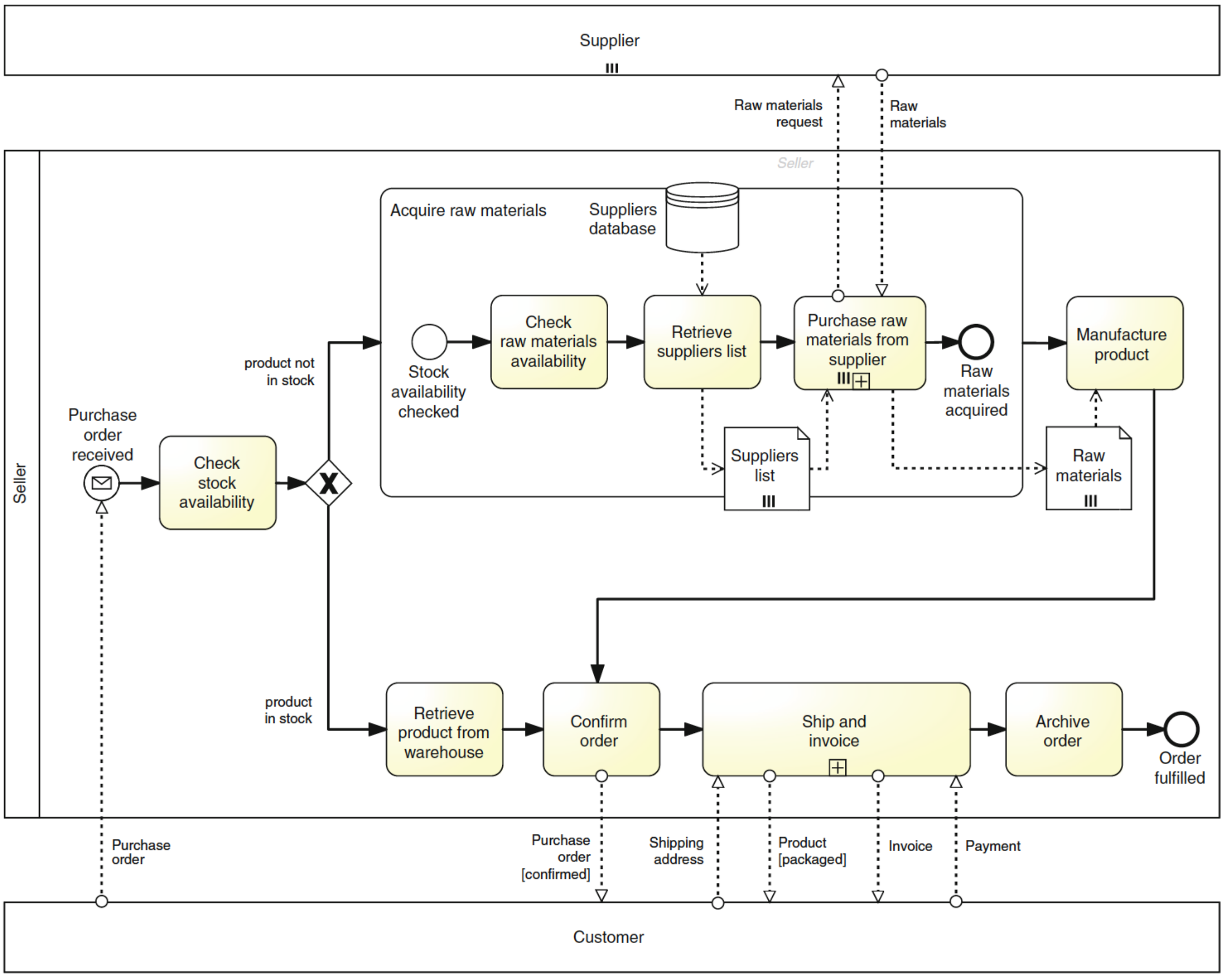


Fig. 4.5 Using a multi-instance pool to represent multiple suppliers

Uncontrolled Repetition

Ad hoc sub-process: represents an activity that can be repeated a number of times without a specific order until a condition is met. Represented by a tilde symbol (~) at the bottom of the sub-process box.

We can't represent start and end events in a ad hoc sub-process).

- For example: a customer may want to check the progress of his order, the customer may do this at any time after we submits the purchase and as often as he desires, the customer may attempt to cancel the order or update personal details before the order has been fulfilled. These activities are **uncontrolled**. They may be repeated multiple times with no specific order or not occur at all until a condition is met.

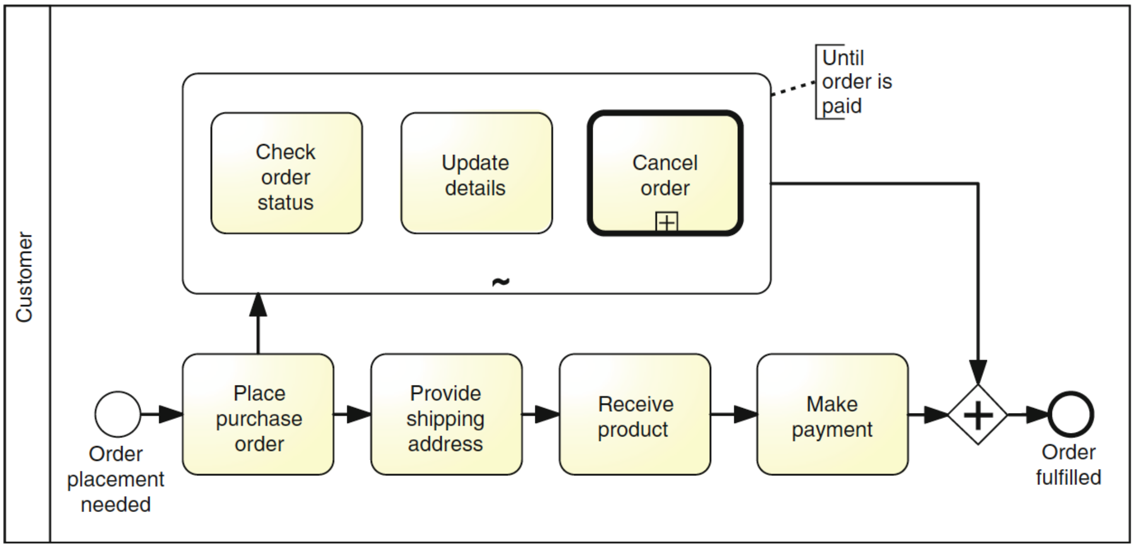


Fig. 4.6 Using an ad hoc sub-process to model uncontrolled repetition

Handling events

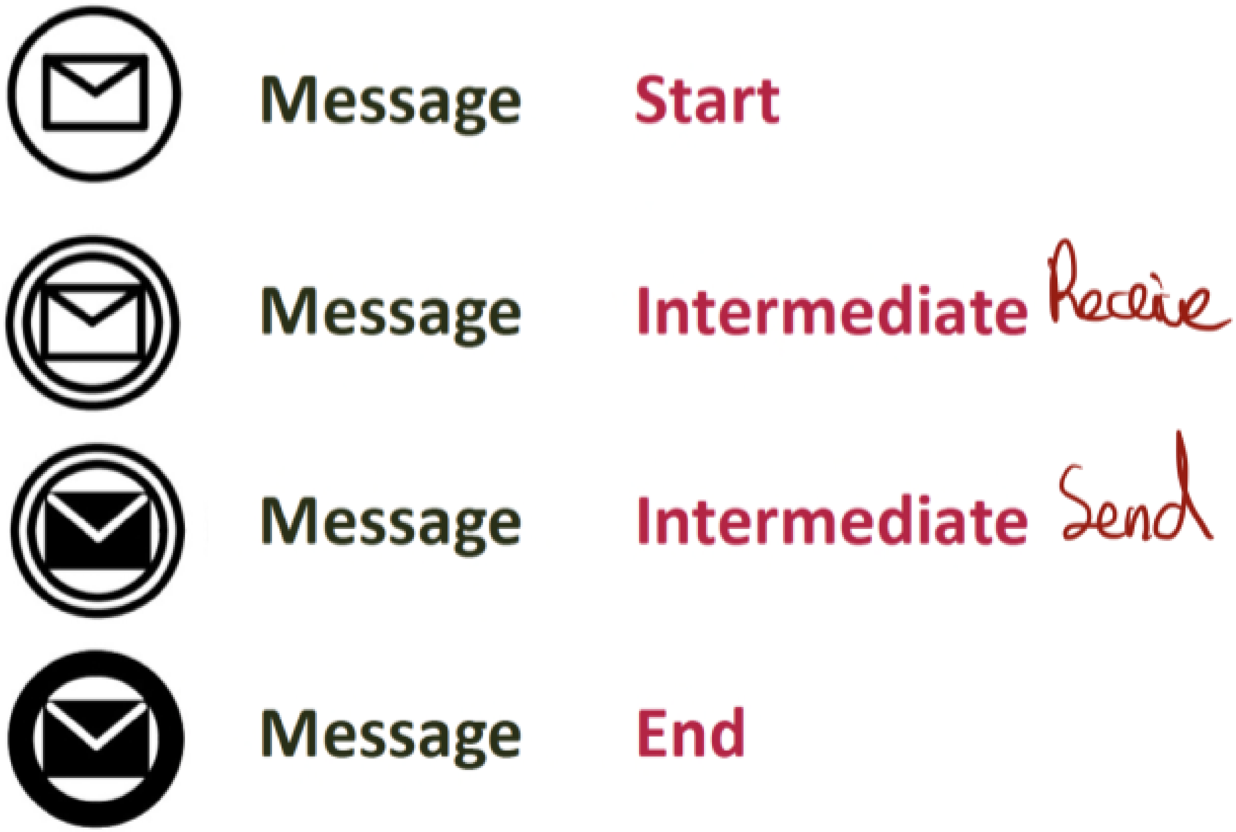
- **Start Events**: signal how process instances start, tokens created.
- **End Events**: signal how process instances complete, tokens destroyed.
- **Intermediate Events**: events that occur during a process, the token is trapped in the incoming sequence flow of an intermediate until an event occurs, after it occurs the token traverses the event in an instant - events do not retain tokens. Ex: an order confirmation is received after sending an order out to the customer.
  - Represented as a circle with a double border.

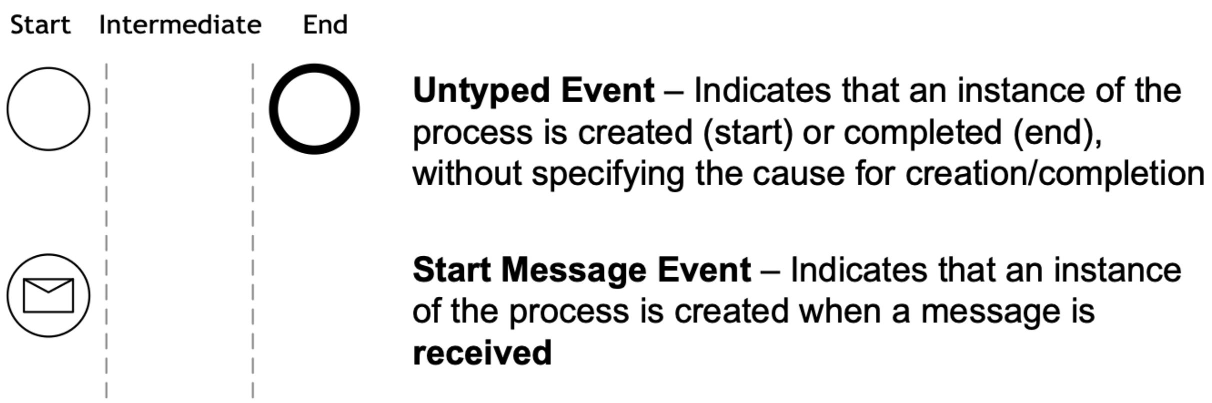
Message Events

Types of message events

- **Start message event**
- **End message event**: means that a process concludes upon sending a message.
- **Intermediate (send/receive) message event**: means that a message was received or that a message was sent (alternative to activities that are used to send or receive a message because these activities aren't units of work but in fact just the mechanical sending or receiving of a message).

- Ex: "Return application to applicant" and "Receive updated application"





Untyped Event – Indicates that an instance of the process is created (start) or completed (end), without specifying the cause for creation/completion

Start Message Event – Indicates that an instance of the process is created when a message is **received**

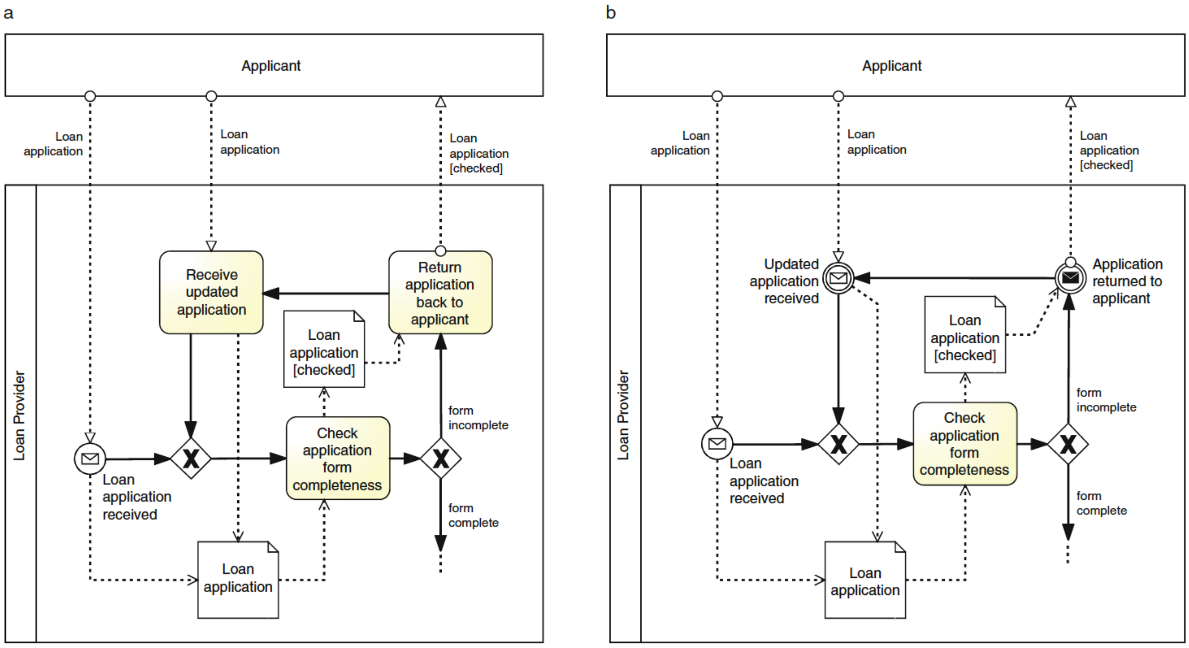

a
Applicant
Loan application
Loan application
Loan application [checked]
Loan Provider
Receive updated application
Return application back to applicant
Loan application [checked]
Check application form completeness
form incomplete
form complete
Loan application received
Loan application
b
Applicant
Loan application
Loan application
Loan application [checked]
Loan Provider
Updated application received
Application returned to applicant
Loan application [checked]
Check application form completeness
form incomplete
form complete
Loan application received
Loan application

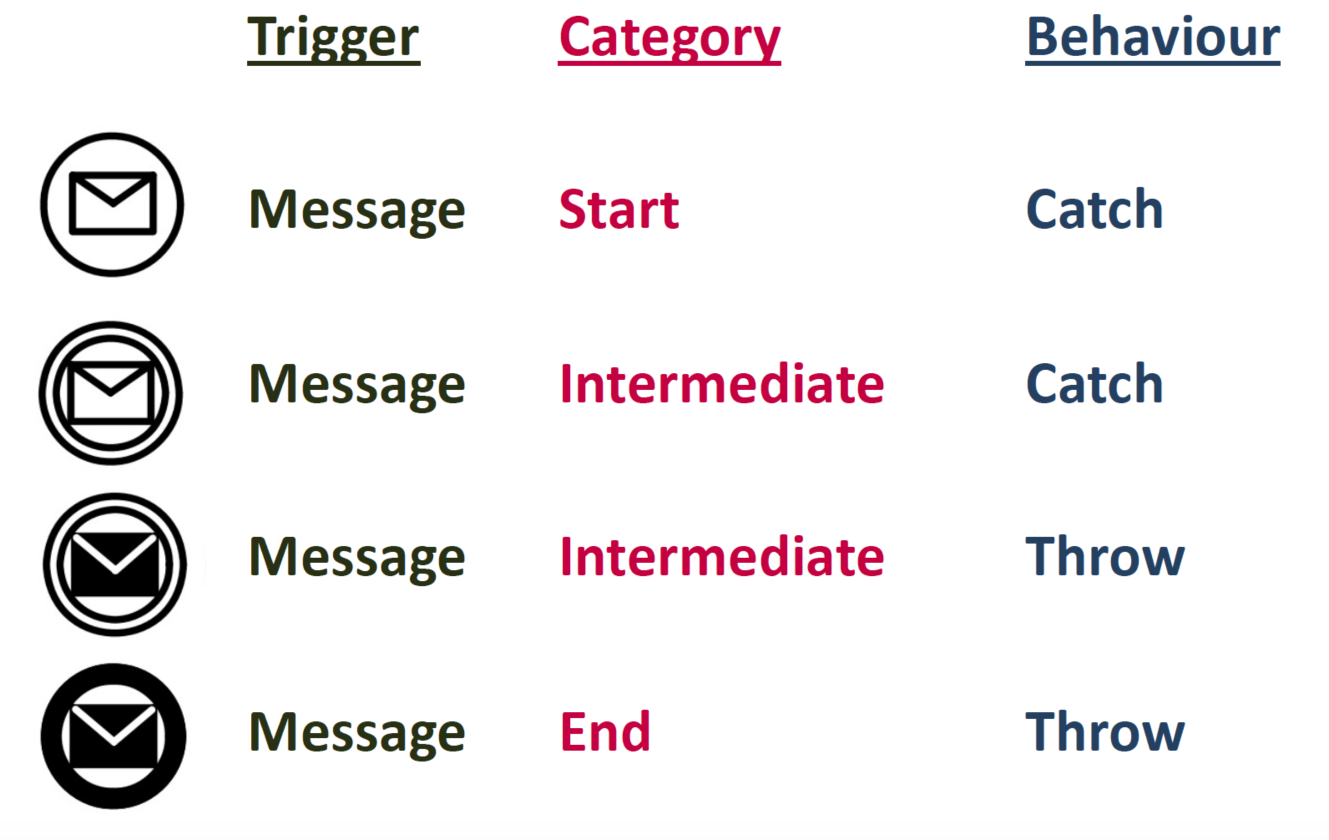
Fig. 4.7 Replacing activities that only send or receive messages (**a**) with message events (**b**)

- **Modeling Tip**:
  - if send activity followed by untyped event, replace with an **end message event**.
- **Beware**:
  - a start message event is not alternative to an untyped start event followed by a received activity.
    - a process can start by receiving a message
    - a process may start at any time after which the first activity requires a message to be received

Catching/Throwing events

- **Catching Events**: a marker with no fill, like the message start event.

- **Throwing Events**: a marker with a dark fill, like the message end event.



Trigger	Category	Behaviour
Message	Start	Catch
Message	Intermediate	Catch
Message	Intermediate	Throw
Message	End	Throw

Temporal Events

- **Timer event**: represents the start of a process upon the occurrence of a specific temporal event. Ex: every Friday morning, every working day of the month...
- It may be used as an intermediate event to capture that a temporal interval needs to elapse before the process instance can proceed.
- Timer events are **catching events only** because the process does not generate the timer, it reacts to it.

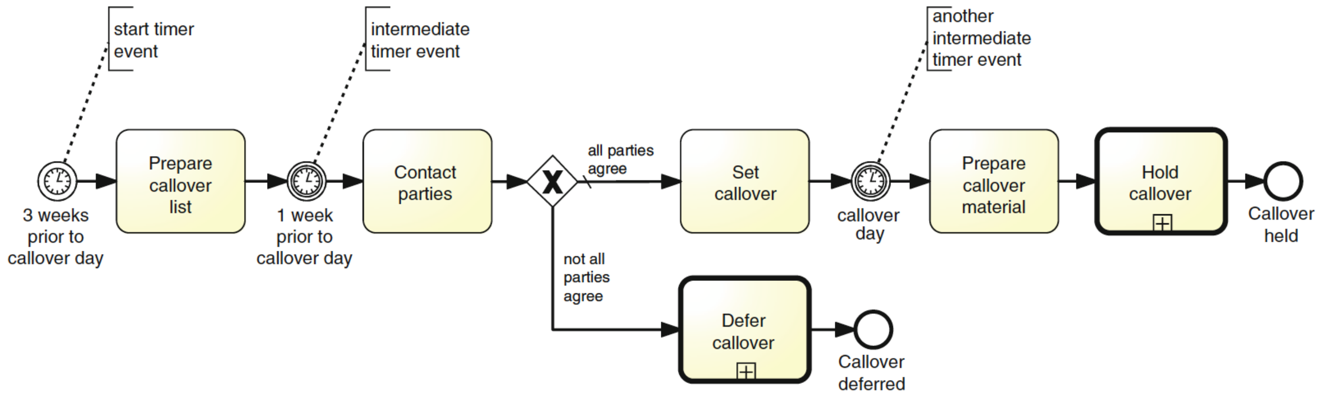


Fig. 4.8 Using timer events to drive the various activities of a business process

- A token is generated 3 weeks prior to the callover date, once the "Prepare callover list" activity is completed, the token is sent through the incoming arc of the intermediate timer event "1 week prior to callover day", thus the event becomes **enabled**, the token remains trapped in the incoming arc of this event until the temporal event occurs (when it's 1 week prior), when this happens the token instantly traverses the event symbol and moves to the outgoing arc. This is why events are instant, they do not retain tokens as opposed to activities that retain tokens for the duration of their execution.

Racing Events

- **Event-based exclusive (XOR) split/Deferred choice**: used when two external events race against each other, the first of the two events that occurs will determine the continuation of the process.
- In the example, the execution stops until either the message event or the timer event occurs, whichever event occurs first will determine which way the execution will proceed.

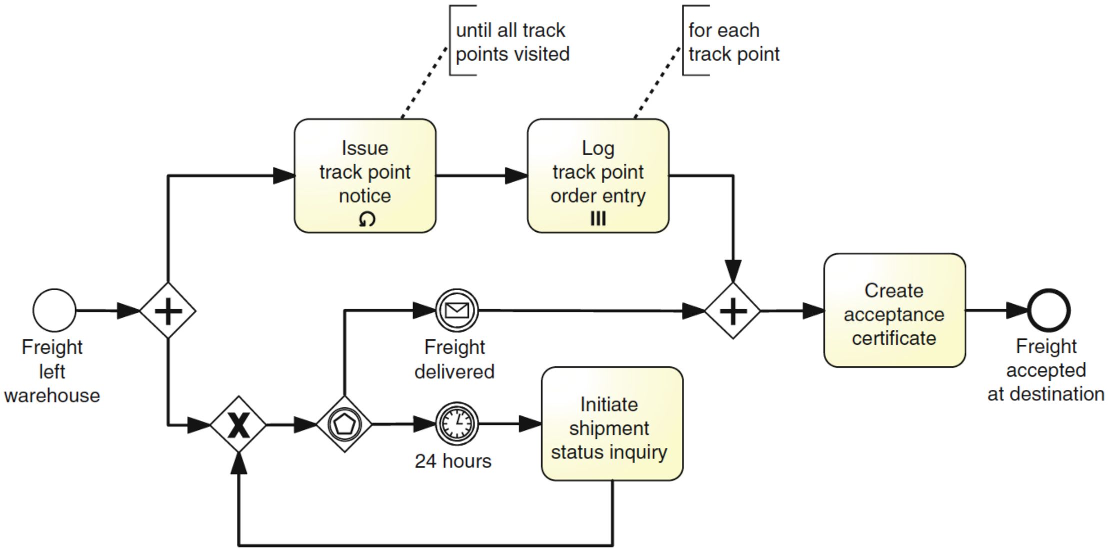

until all track points visited
for each track point
Issue track point notice
Log track point order entry
Freight left warehouse
Freight delivered
Create acceptance certificate
Freight accepted at destination
24 hours
Initiate shipment status inquiry

Fig. 4.9 A race condition between an incoming message and a timer

- The difference between the XOR-split is that models an internal choice determined by the outcome of a decision activity , whereas this one models an internal choice that is determined by the environment of the process.
- **The event-based split can only be followed by intermediate catching events like a timer or a message event or by receiving activities.**
- **There is no event-based XOR-join so the branches emanating from an event-based split are merged with a normal XOR-join.**

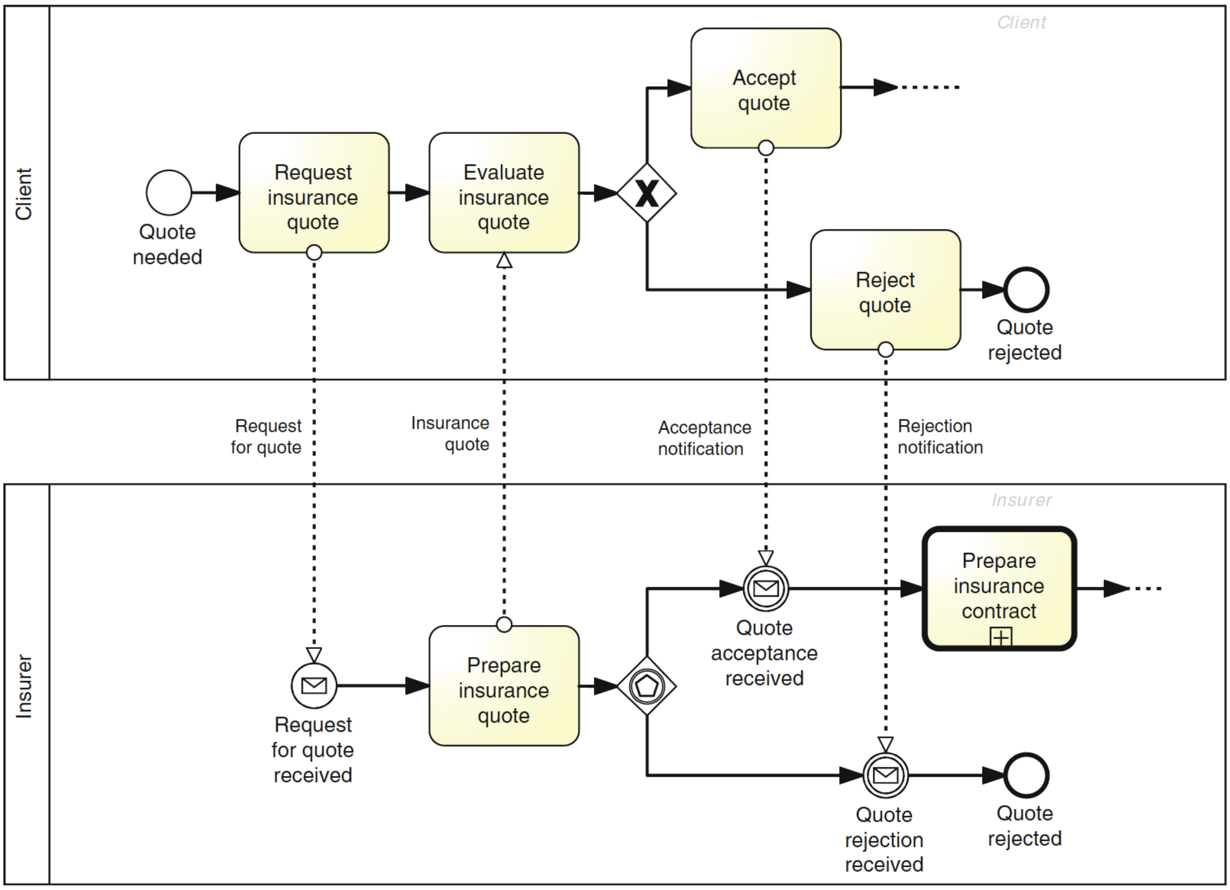


Fig. 4.10 Matching an internal choice in one party with an event-based choice in the other party

- Can be used to avoid deadlocks.

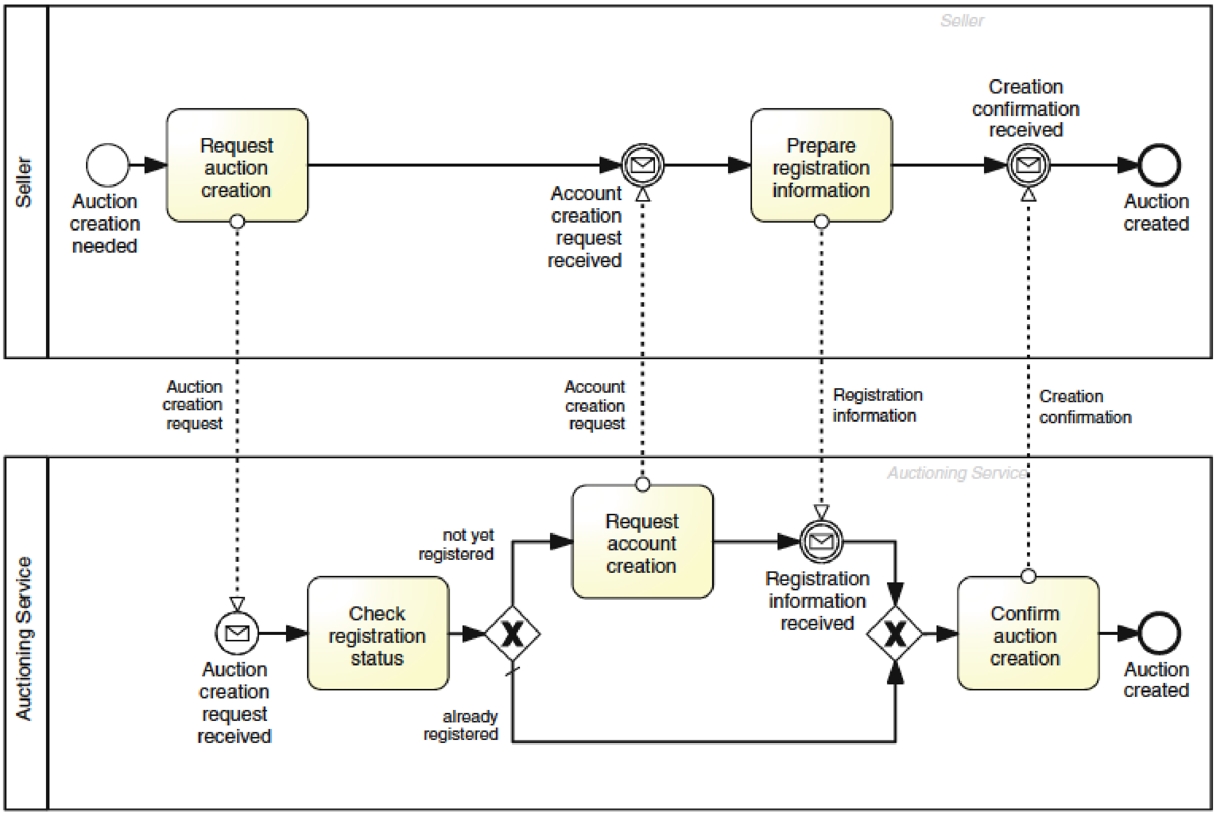


Fig. 4.11 An example of collaboration that can deadlock if the decision is made for “already registered”

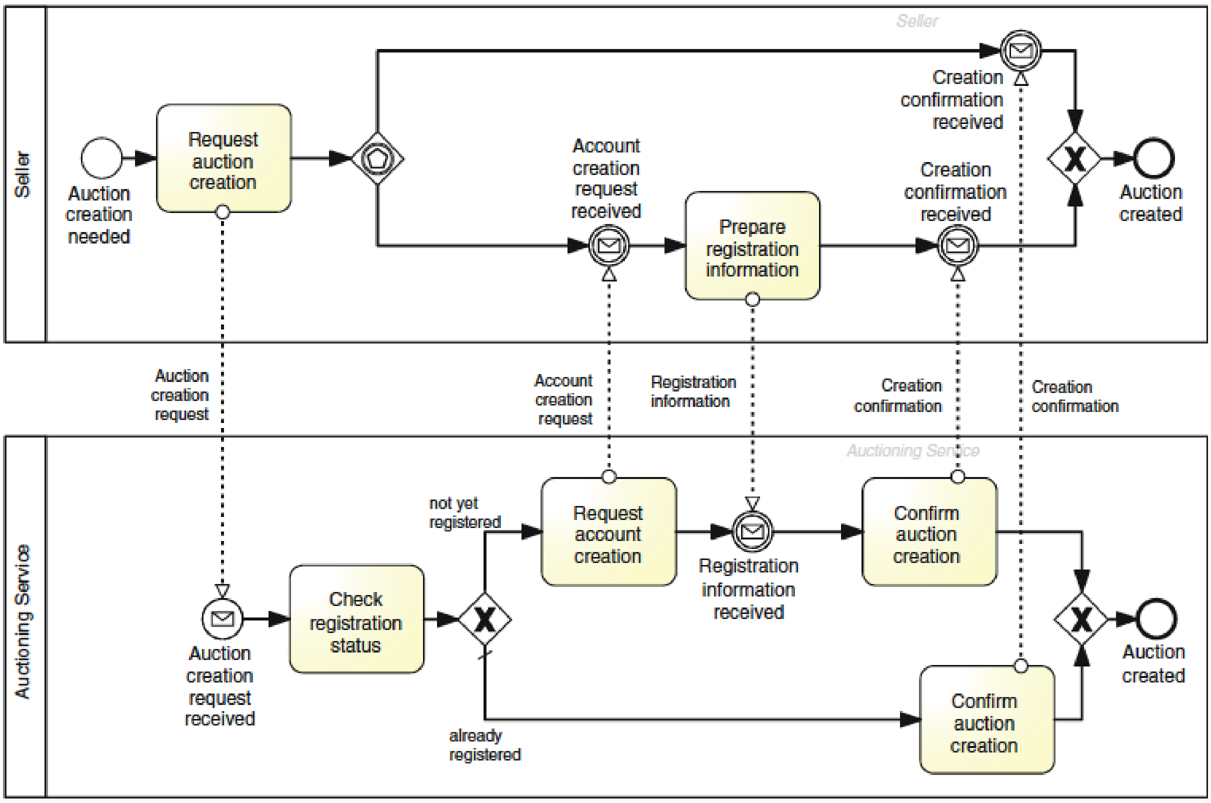


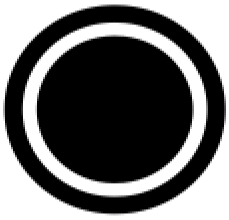
Fig. 4.12 Using an event-based gateway to fix the problem of a potential deadlock in the collaboration of Figure 4.11

Handling Exceptions

- **Exceptions**: events that deviated a process from its normal course (from the sunny day scenario, the rainy day scenario). The objective is to identify all causes of problems in a given process. Ex: business faults (like out of stock) and technology faults (like a database crash).

Process Abortion

- **End terminate event**: aborts the running process and signals an abnormal process termination. This event destroys all tokens in the process model and any sub-process.



- In the example, a home loan is rejected and the process is aborted if the applicant has high debts or high liability.

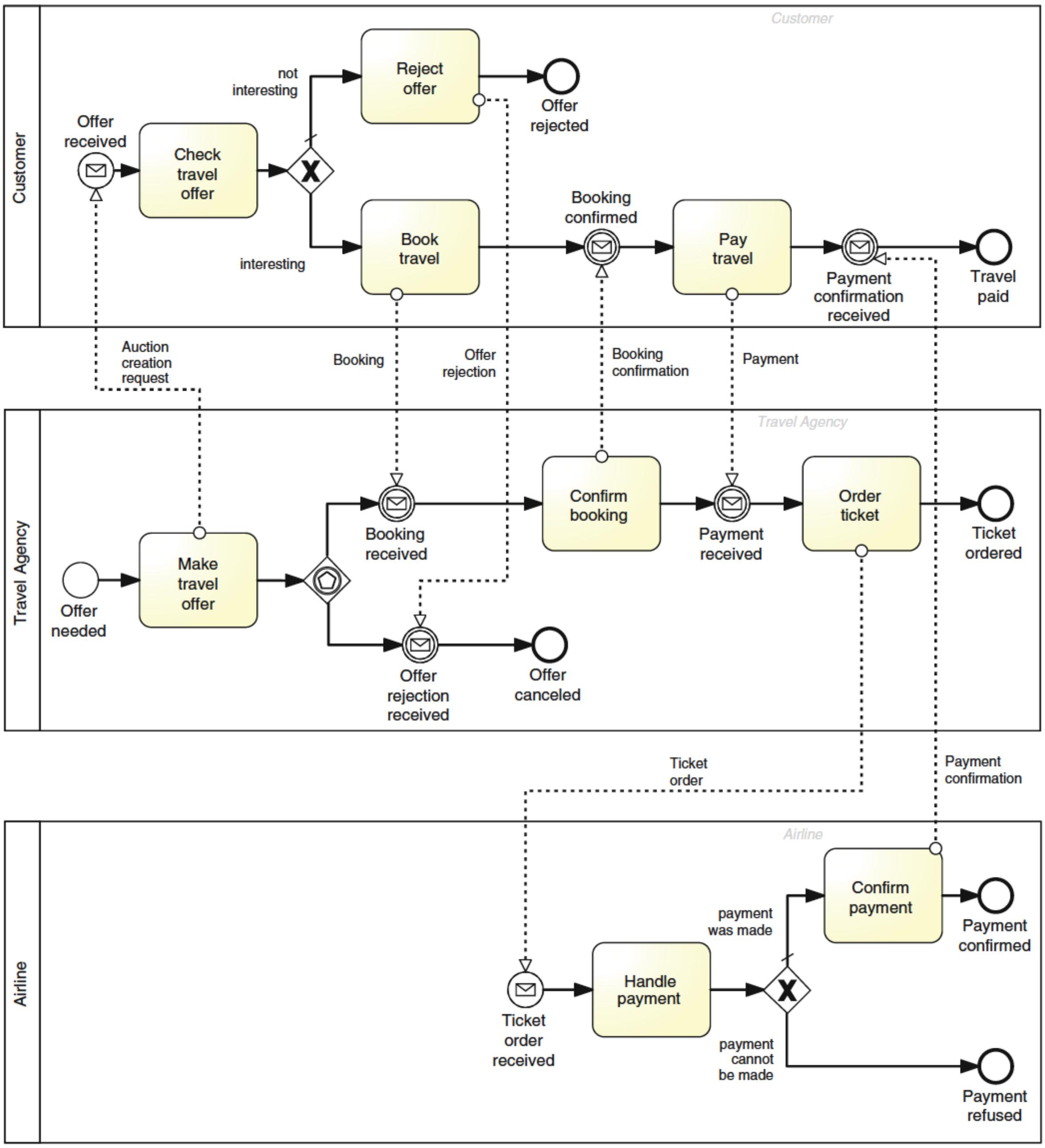
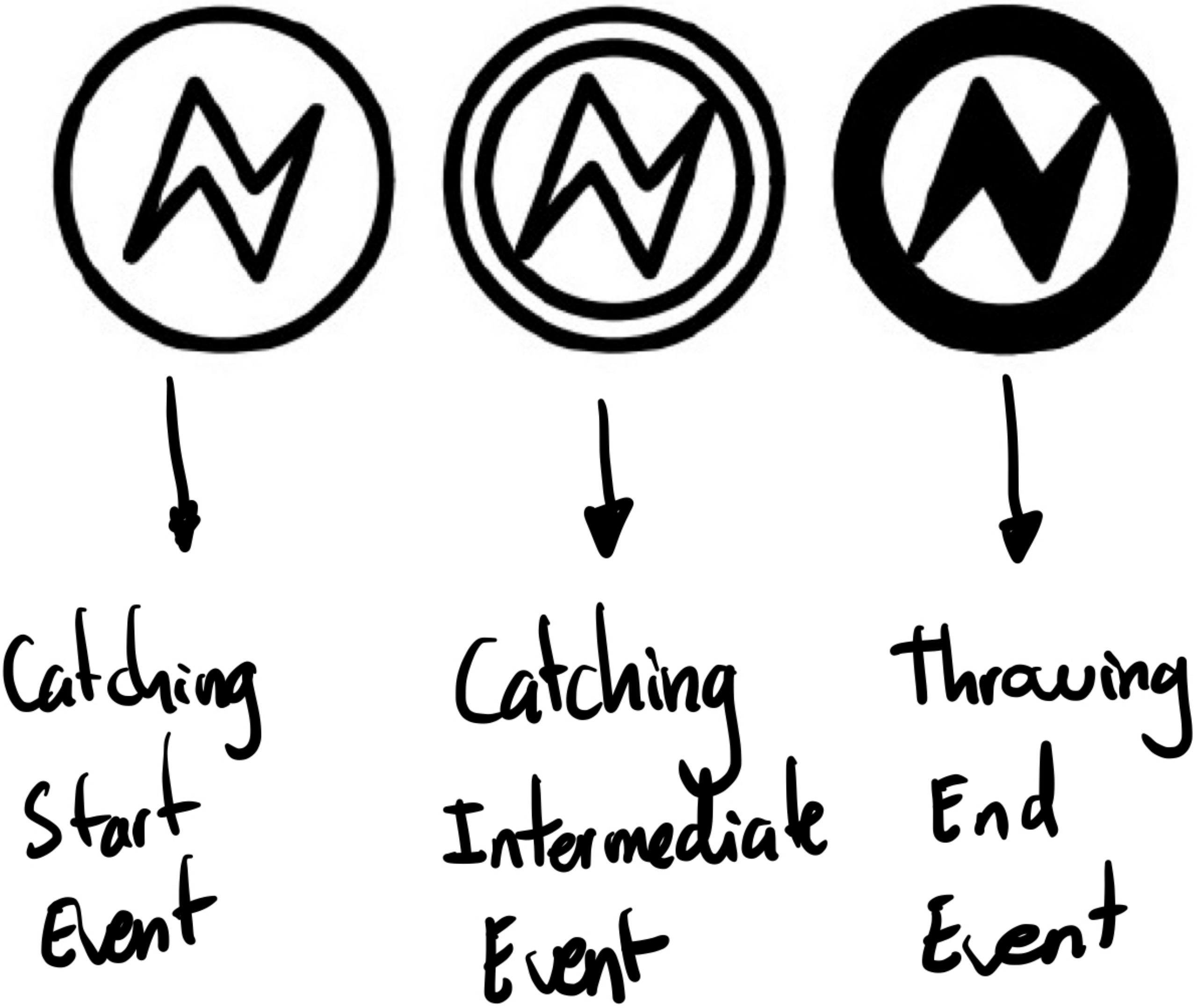


Fig. 4.13 A collaboration diagram between a customer, a travel agency, and an airline

Internal Exceptions

- **Error Event**: interrupts the specific activity that has caused the exception, then we start a recovery procedure to bring the process back to a consistent state and continue its execution, if it isn't possible we abort the process altogether. This event interrupts the enclosing sub-process and throws an exception which is then caught by an intermediate catching error event which is attached to the boundary of the same sub-process. This **boundary event** triggers the recovery procedure through an outgoing branch, called **exception flow**.


Catching Start Event
Catching Intermediate Event
Throwing End Event

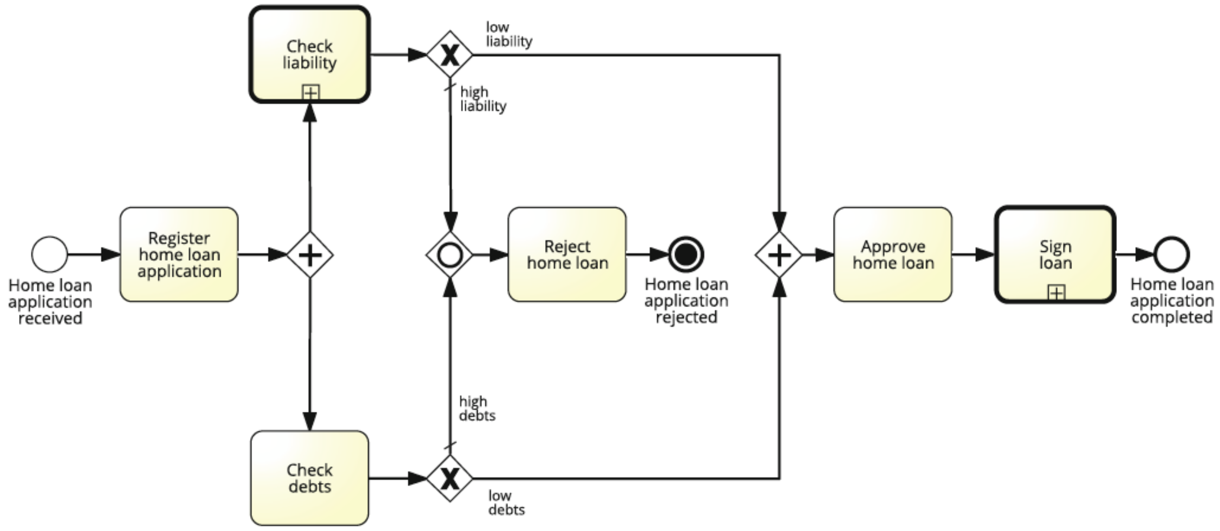

Check liability
low liability
high liability
Home loan application received
Register home loan application
Reject home loan
Home loan application rejected
Approve home loan
Sign loan
Home loan application completed
high debts
Check debts
low debts

Fig. 4.14 Using a terminate event to signal abnormal process termination

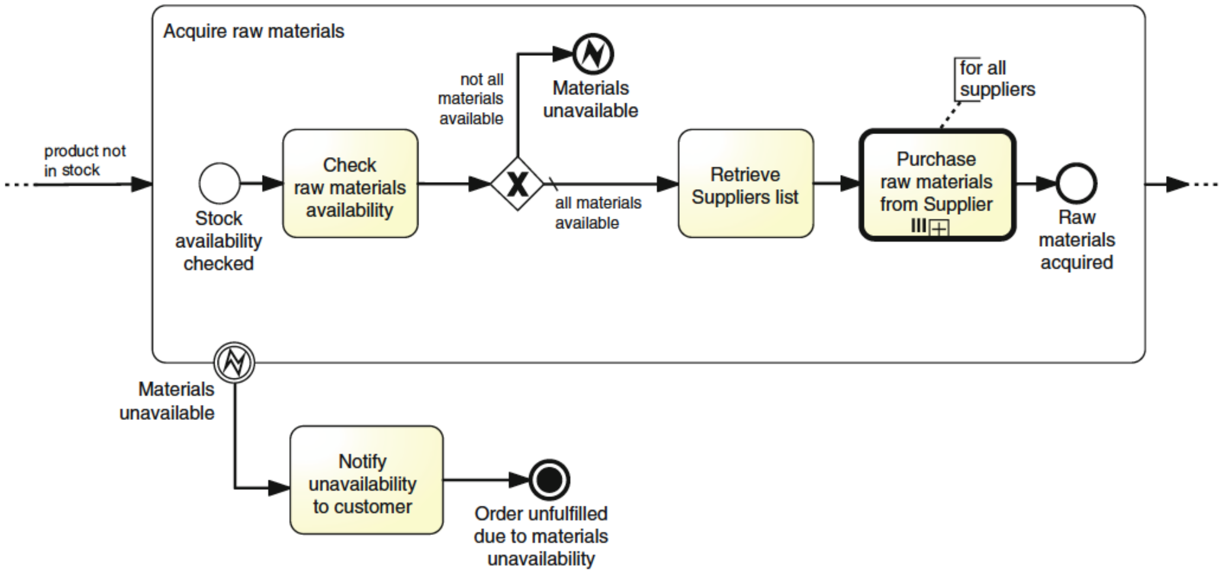

Acquire raw materials
product not in stock
Stock availability checked
Check raw materials availability
not all materials available
Materials unavailable
all materials available
Retrieve Suppliers list
for all suppliers
Purchase raw materials from Supplier
Raw materials acquired
Materials unavailable
Notify unavailability to customer
Order unfulfilled due to materials unavailability

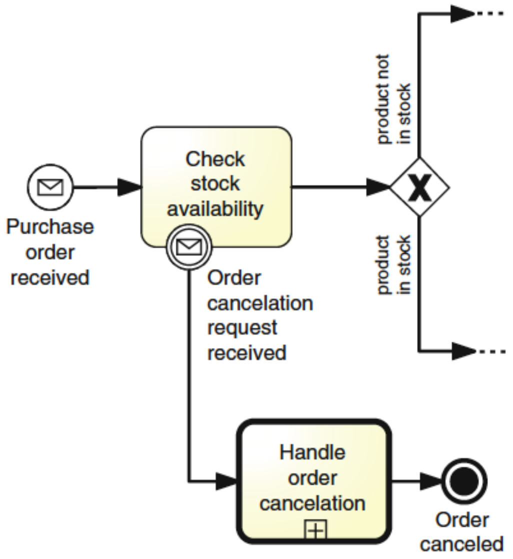
Fig. 4.15 Error events model internal exceptions

External Exceptions

- An **unsolicited exception** (originate externally to the process) may be caused by an **external event** occurring during an activity.
- They are captured by a **catching intermediate message event to an activity's boundary**.
- When the intermediate event is triggered, the token is removed from the activity (causing interruption) and sent via the exception flow emanating from the boundary event to perform the recovery procedure.
- For example, while checking for stock availability for a product in an order, the seller may receive and order cancelation from the customer. The seller should interrupt the stock availability check and handle the order cancelation.
- Before using the boundary event, we need to identify the **scope** within which the process should be receptive of this event.

- For example, order cancelation requests can only be handled during the execution of task "Check stock availability", thus the scope is made up by this one task, the scope can however include multiple activities, in this case we encapsulate the interested activities into a sub-process and attach. the event to the boundary of the sub-process.

Fig. 4.16 Boundary events catch external events that can occur during an activity



Activity Timeouts

- Interruption of an activity that is taking too long and must be completed within a given timeframe. Ex: approval must be completed within 24h.
- To model this, we attach an **intermediate timer event to the boundary of the activity**.
    - The timer is activated when the enclosing activity starts.
    - If the timer expires before the activity completes, it interrupts the activity.
- **In other words, a timer event works as a timeout when attached to an activity boundary.**

Non-Interrupting Events and Complex Exceptions

- There are cases where an external event occurring during an activity should just trigger a procedure without interrupting the activity itself.
- To denote that the boundary event is **non-interrupting** we use a dashed double border.
- Example: The customer may send a request to update details during the stock availability check, the details should be updated in the database without interrupting the stock check.

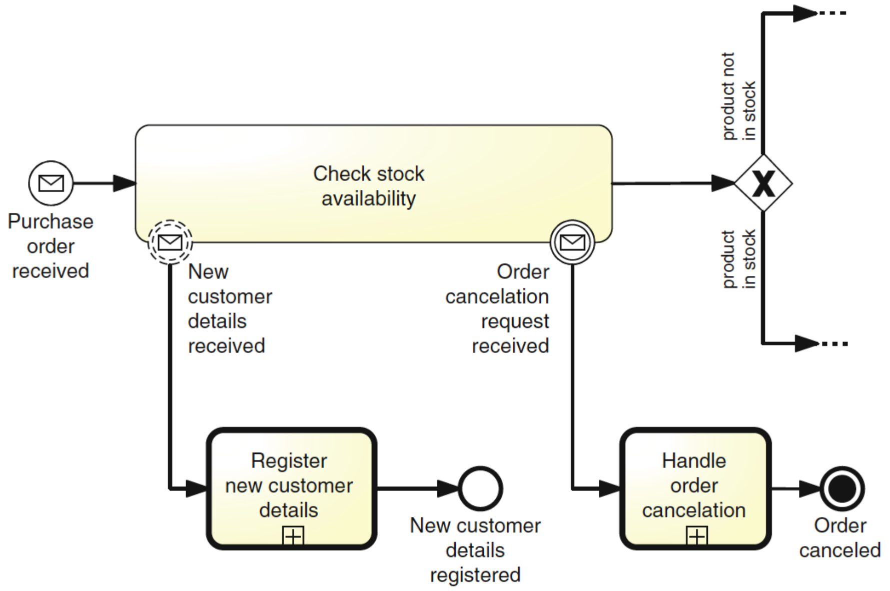

Purchase order received
Check stock availability
New customer details received
Order cancelation request received
product not in stock
product in stock
Register new customer details
New customer details registered
Handle order cancelation
Order canceled

Fig. 4.17 Non-interrupting boundary events catch external events that occur during an activity and trigger a parallel procedure without interrupting the enclosing activity

- **End Signal Events**: broadcasts a signal defined by the event label and is caught by all catching signals events bearing the same label.

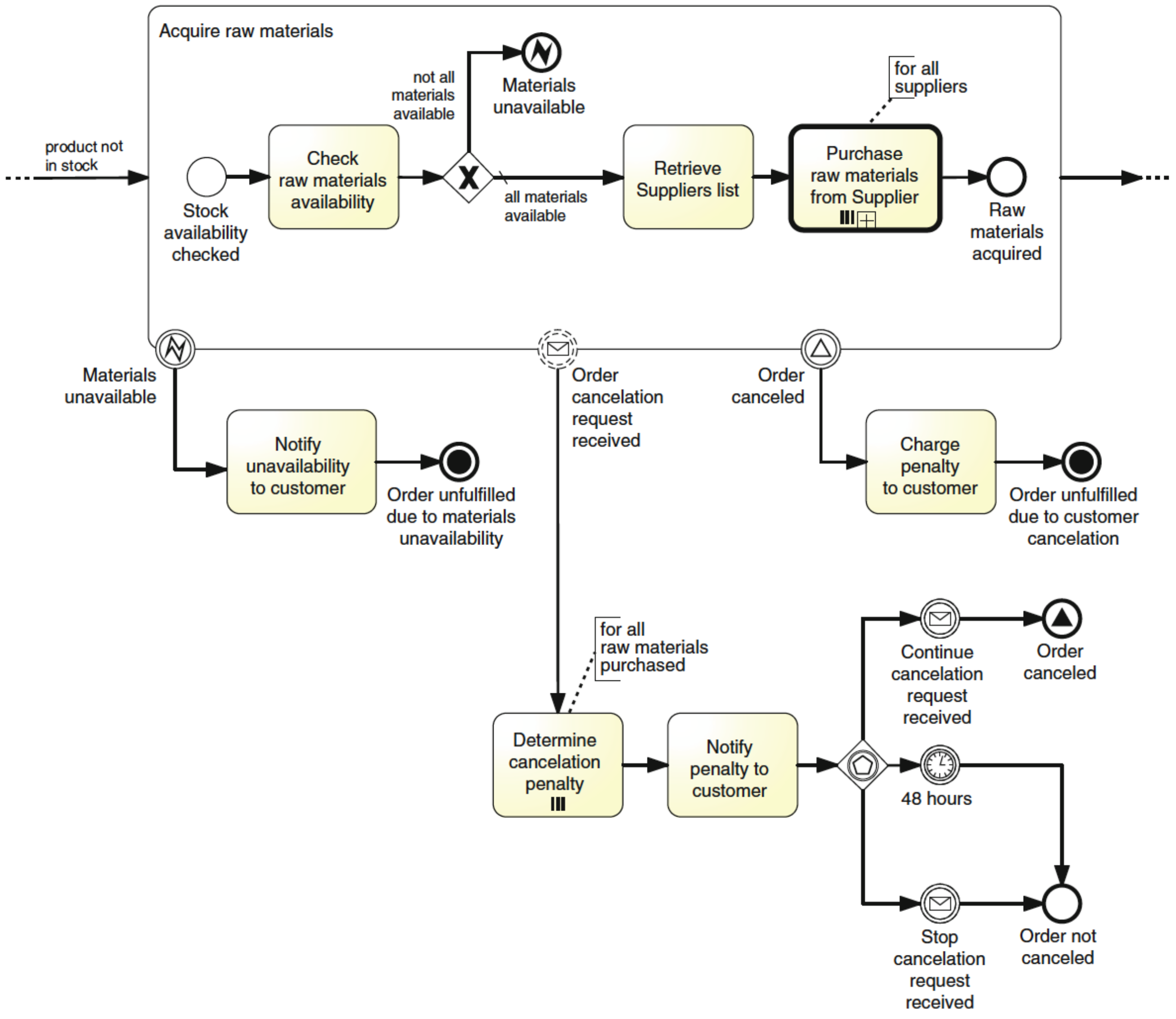

Acquire raw materials
product not in stock
Stock availability checked
Check raw materials availability
not all materials available
Materials unavailable
all materials available
Retrieve Suppliers list
for all suppliers
Purchase raw materials from Supplier
Raw materials acquired
Materials unavailable
Notify unavailability to customer
Order unfulfilled due to materials unavailability
Order cancelation request received
Order canceled
Charge penalty to customer
Order unfulfilled due to customer cancelation
for all raw materials purchased
Determine cancelation penalty
Notify penalty to customer
Continue cancelation request received
Order canceled
48 hours
Stop cancelation request received
Order not canceled

Fig. 4.18 Non-interrupting events can be used in combination with signal events to model complex exception handling scenarios

Event Sub-processes

- **Event sub-process**: alternative notation to boundary events, it is started by an event (the one that would be attached to the boundary) and encloses the procedure that would be triggered by the. boundary event.
- Can model events that occur during the execution of the whole process. Ex: a customer may send an inquiry about the order status anytime during the process.

- If the start event is non-interrupting, it's represented with a dashed single border.

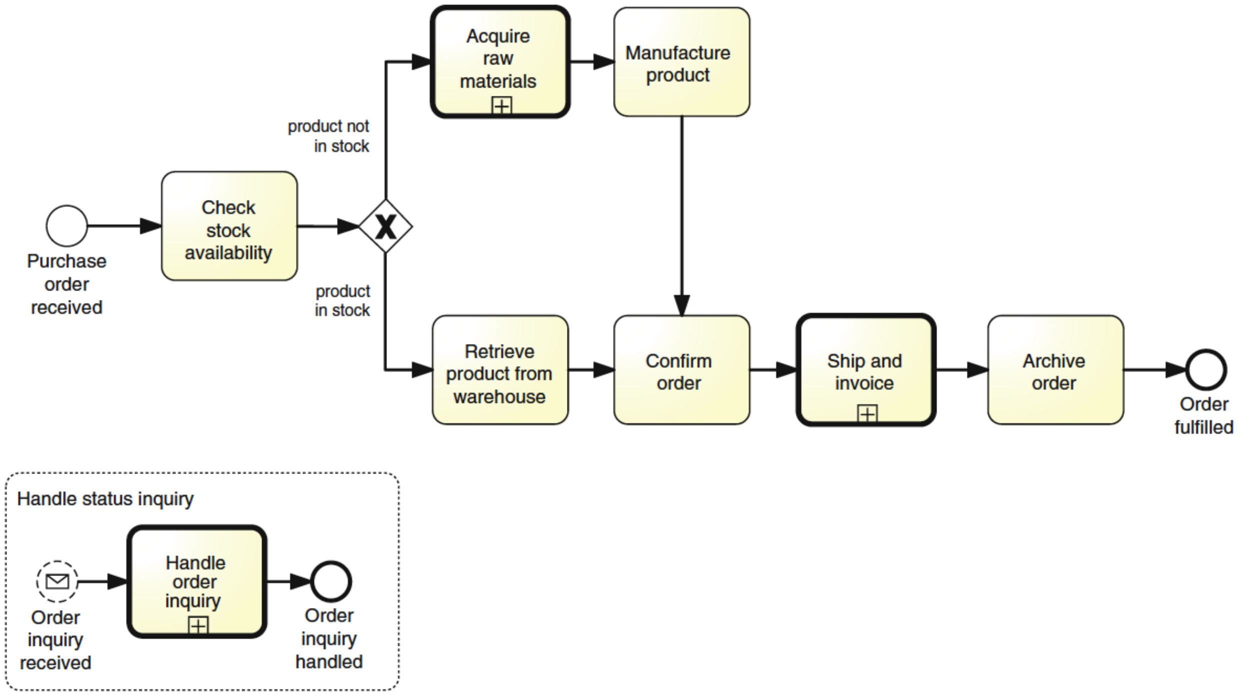


Fig. 4.19 Event sub-processes can be used in place of boundary events and to catch events thrown from outside the scope of a particular sub-process

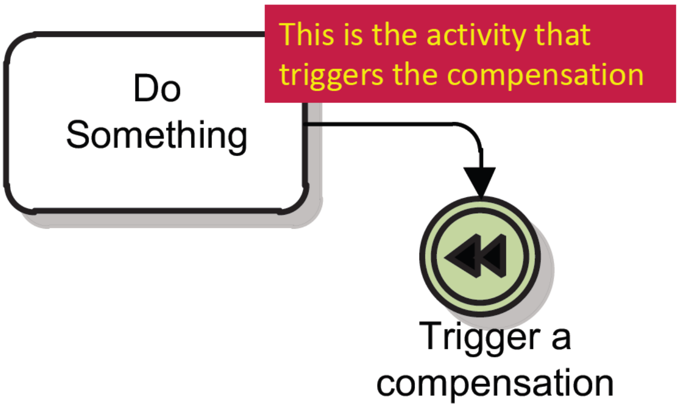
Event sub-processes or boundary events?

- Event sub-processes must conclude with an end event - disadvantage - the procedure in an event sub-process cannot be wired back to the rest of the sequence flow.
- Event sub-processes can be defined globally and can be reused in other models for the same organisation -advantage.
- Event sub-processes are defined at the level of the entire process while boundary. events must refer to a specific activity.
- **Use event sub-processes**: when the event that needs to be handled may occur anytime during the process or when we need to capture a reusable procedure .
- **Use boundary events**: for all other cases, because the procedure triggered can be wired back to the rest of the flow.

Activity Compensation

- We may need to **undo** one or more steps that have already been completed as part of the recovery procedure used in handling an exception.
- **Compensation**: tries to restore the process to a business state close to the one before starting the sub-process that was interrupted.
- A compensation handler is made of a **throwing compensate event (in the recovery procedure)**, a **catching intermediate event (attached to the activities that need to be compensated)** and a **compensation activity**.
- **Compensation association**: dotted arrow with open arrowhead.

- Compensation is only effective if the attached activity has completed, once all activities have been compensated, the process resumes after the throwing compensate even, unless this is an end event.
- If the compensation is for the entire process, use an event sub-process with a start compensate event in place of the boundary event.



This is the activity that triggers the compensation

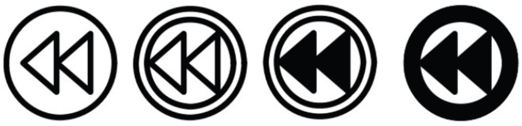
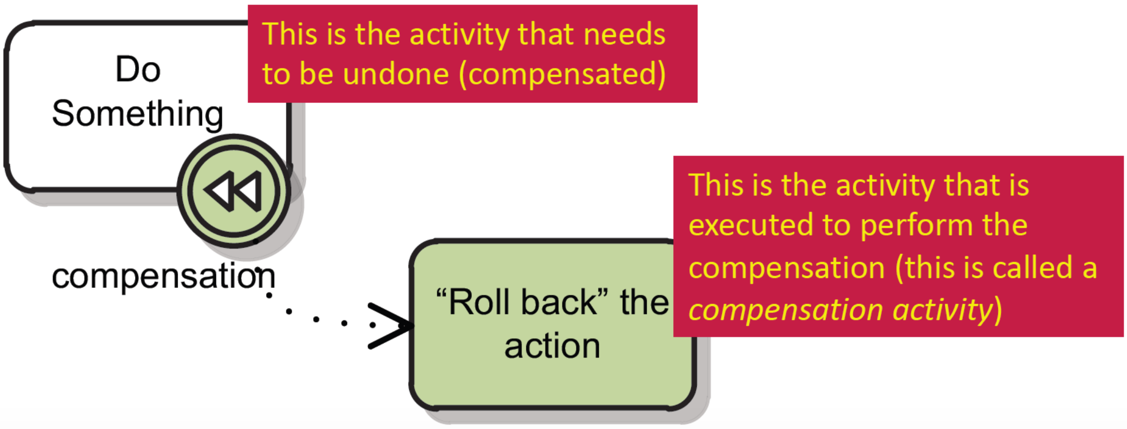


Figure 10.75 – Compensation Events



This is the activity that needs to be undone (compensated)

This is the activity that is executed to perform the compensation (this is called a *compensation activity*)

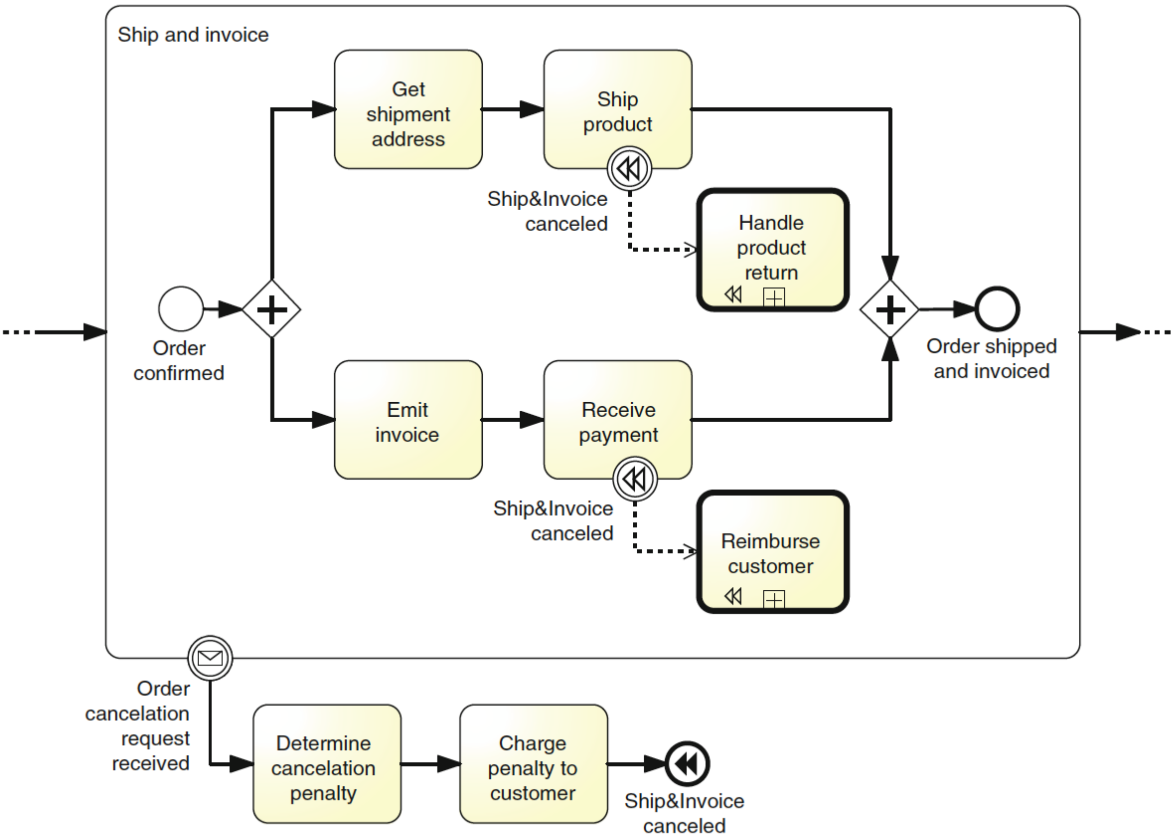


Fig. 4.20 Compensating for the shipment and for the payment

Processes and Business Rules

- A business rule implements an organisational policy or practice.
- Ex: in an online shop, platinum customers have a 20% discount for each purchase above 250€.
- Business rules can appear in process models in the forms of:
  - Decision Activities
  - Condition of a flow coming out an (X)OR split
  - Conditional Events
- **Conditional Events**: causes the activation of its flow when the business rule is fulfilled, can be used as start or intermediate events (after an event-based gateway or attached to an activity boundary).
  - **Difference between intermediate conditional events and a condition on a flow**
    - A flow is only tested once, if it is not satisfied the flow is not taken (another or the default will be taken).
    - The conditional event is tested until the rule is satisfied, the token remains trapped before the event until the rule is satisfied.

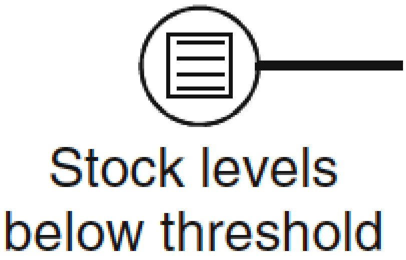
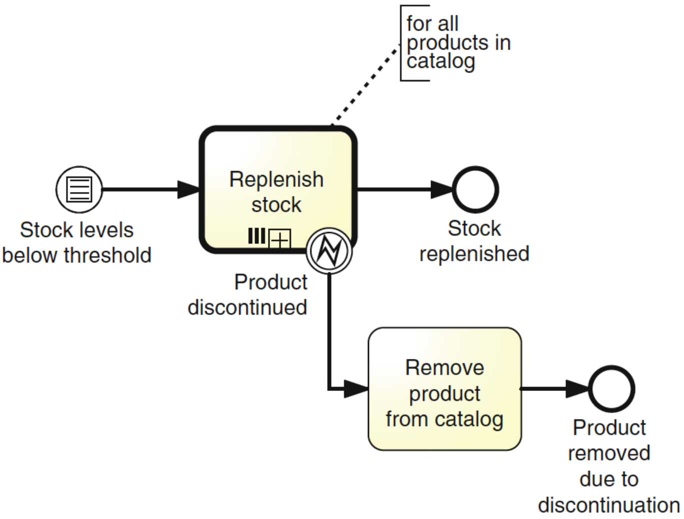


Fig. 4.21 A replenishment order is triggered every time the stock levels drop below a threshold


for all products in catalog
Stock levels below threshold
Replenish stock
Stock replenished
Product discontinued
Remove product from catalog
Product removed due to discontinuation