Chapter 4
Advanced Process Modeling

The sciences do not try to explain, they hardly even try to interpret, they mainly make models.
John von Neumann (1903–1957)

In this chapter we will delve more into how to model complex business processes with BPMN. The constructs presented here build on top of the knowledge acquired in Chapter 3. In particular, we will expand on activities, events, and gateways. We will extend activities to model more sophisticated forms of rework and repetition. We will also discuss more specific types of events, including message events, temporal events, and cancelation. These can be used to model race conditions together with a new type of gateway. Finally, we will also learn how to use events to handle business process exceptions.

4.1 More on Rework and Repetition

In the previous chapter, we described how to model rework and repetition via the XOR gateways. Expanded sub-processes offer an alternative way to model parts of a process that can be repeated. Let us consider again the process for addressing ministerial correspondence of Example 3.7. To make this model simpler, we can take the fragment identified by the XOR-join and the XOR-split (which includes the repetition block and the loopback branch) and replace it with a sub-process containing the activities in the repetition block. To identify that this sub-process may be repeated (if the response is not approved), we mark the sub-process activity with a loop symbol, as shown in Figure 4.1. We can use an annotation to specify the loop condition, e.g., “until response approved”.

As for any sub-process, you may decide not to specify the content of a loop sub-process. However if you do specify the content, do not forget to put a decision activity as the last activity inside the sub-process, otherwise there is no way to determine when to repeat the sub-process.

M. Dumas et al., *Fundamentals of Business Process Management*,
https://doi.org/10.1007/978-3-662-56509-4_4

Question Loop activity or cycle?

The loop activity is a shorthand notation for a structured cycle, i.e., a repetition block delimited by a single entry point to the cycle and a single exit point from the cycle, like in the example in Figure 4.1. Sometimes there might be more than one entry and exit point, or the entry or exit point might be inside the repetition block. Consider for example the model in Figure 4.2. Here, there is a cycle consisting of activities "Assess application", "Notify rejection", and "Receive customer feedback". This cycle has one entry point and two exit points, and therefore it is unstructured. A cycle with multiple exit points, like this one, cannot be rewritten as a cycle with only one exit point, unless additional conditions are used to specify the situations in which the cycle can be exited.

Exercise 4.1

1. Identify the entry and exit points that delimit the unstructured cycles in the process models shown in Exercise 3.12 (page 112) and in Solution 3.4 (page 109). What are the repetition blocks?
2. Model the business process of Solution 3.4 (page 109) using a loop activity.

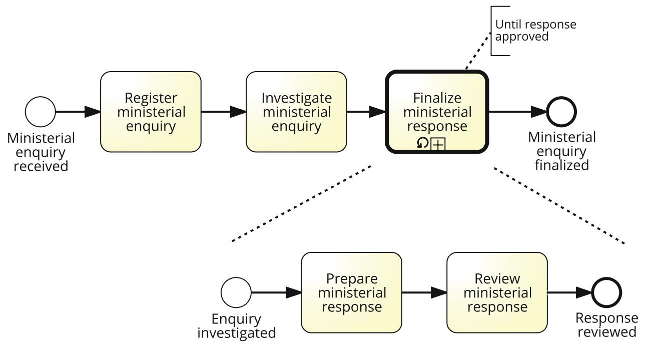

Until response approved
Ministerial enquiry received
Register ministerial enquiry
Investigate ministerial enquiry
Finalize ministerial response
Ministerial enquiry finalized
Enquiry investigated
Prepare ministerial response
Review ministerial response
Response reviewed

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

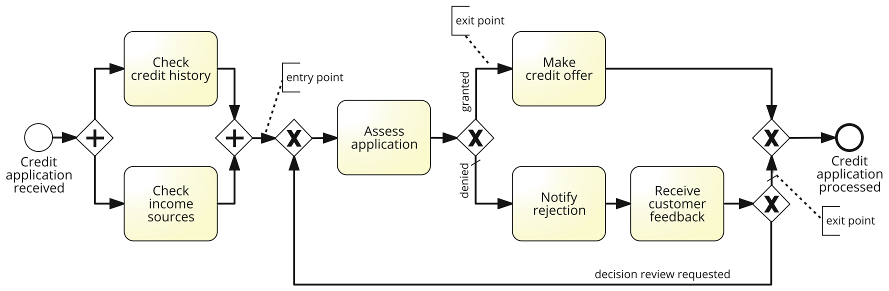


Fig. 4.2 An example of unstructured cycle

4.1.1 Parallel Repetition

The loop activity allows us to capture sequential repetition, meaning that instances of the loop activity are executed one after the other. Sometimes though, we may need to execute multiple instances of the same activity at the same time, like in the following example.

Example 4.1 In a procurement process, a quote is to be obtained from all preferred suppliers. After all quotes are received, they are evaluated and the best quote is selected. A corresponding purchase order is then placed.

Let us assume there are five preferred suppliers. We can use an AND-split to model five tasks in parallel, each to obtain a quote from one supplier, as shown in Figure 4.3. However, there are two issues with this solution. First, the larger the number of suppliers, the larger the resulting model will be, because we need one task per supplier. Second, we need to revise the model every time the number of suppliers changes. In fact, it is often the case in reality that an updated list of suppliers is kept in an organizational database which is queried before contacting the suppliers.

To avoid these problems, BPMN provides a construct called *multi-instance* activity. A multi-instance activity indicates an activity (a task or a sub-process) that is executed multiple times *concurrently*, i.e., potentially in parallel. This construct is useful when the same activity is executed for multiple entities or data items, as for example to request quotes from multiple suppliers (as in our example), to check the availability of each line item in an order separately, to send and gather questionnaires for multiple witnesses in the context of an insurance claim, etc.

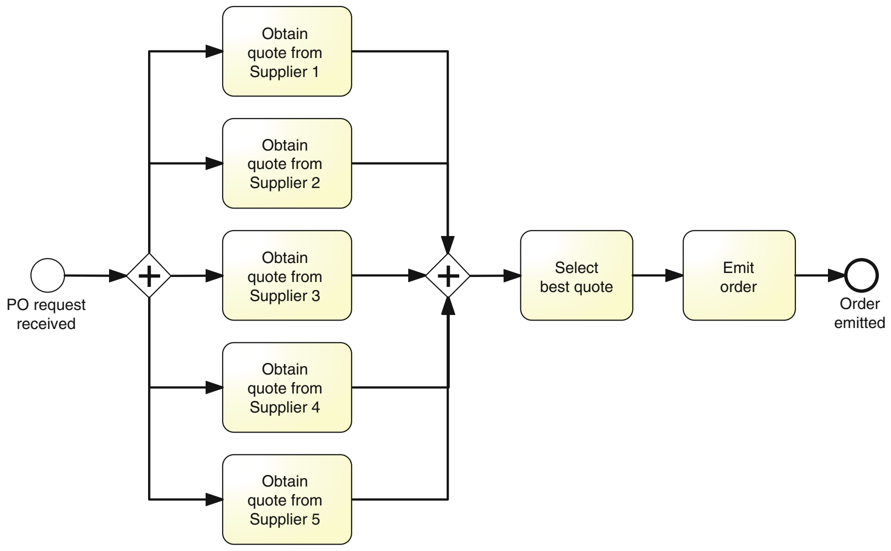

Obtain quote from Supplier 1
Obtain quote from Supplier 2
Obtain quote from Supplier 3
Obtain quote from Supplier 4
Obtain quote from Supplier 5
PO request received
Select best quote
Emit order
Order emitted

Fig. 4.3 Obtaining quotes from five suppliers

A multi-instance activity is depicted as an activity marked with three small vertical lines at the bottom. Figure 4.4 shows a revised version of the procurement process model in Figure 4.3. Not only is this model smaller, but it can also work with a dynamic list of suppliers, which may change on an instance-by-instance basis. To do so, we added a task to retrieve the list of suppliers and passed this list to a multi-instance task, which contacts the various suppliers. You may have noticed that in this example we have also marked the data object Suppliers list with the multi-instance symbol. This is used to indicate a *collection* of similar data objects, like a list of order items or a list of customers. When a collection is used as input to a multi-instance activity, the number of items in the collection determines the number of activity instances to be created. Alternatively, we can specify the number of instances to be created via an annotation on the multi-instance activity (e.g., "15 suppliers" or "as per suppliers database").

Let us come back to our example. Assume the list of suppliers has become quite large over time, say there are 20 suppliers in the database. As per our organizational policies, however, five quotes from five different suppliers are enough to make a decision. Thus, we do not want to wait for all 20 suppliers to attend to our request for a quote. To do so, we can annotate the multi-instance activity with the minimum number of instances that need to complete before passing control to the outgoing arc (e.g., "complete when 5 quotes obtained" as shown in Figure 4.4). When the multi-instance activity is triggered, 20 tokens are generated, each marking the progress of one of the 20 instances. Then, as soon as the first five instances complete, all the other instances are canceled (the respective tokens are destroyed) and one token is sent to the output arc to signal completion. □

Let us take the order-to-cash example in Figure 3.18, and expand the content of the sub-process for acquiring raw materials. To make this model more realistic, we can use a multi-instance sub-process in place of the structure delimited by the two OR gateways, assuming that the list of suppliers to be contacted will be determined on the fly from a suppliers database (the updated model is shown in Figure 4.5). By the same principle, we replace the two pools "Supplier 1" and "Supplier 2" with a

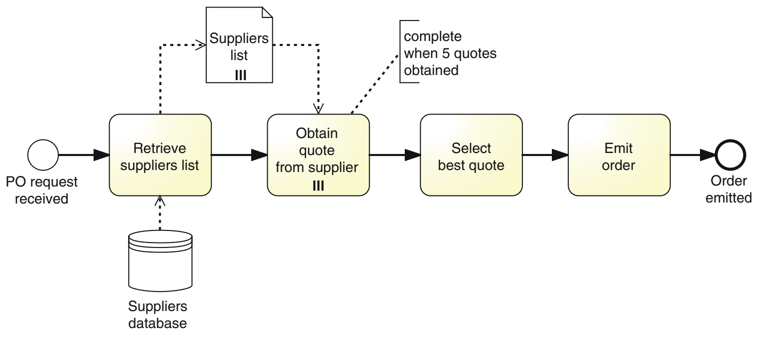


Fig. 4.4 Obtaining quotes from a number of suppliers determined on-the-fly

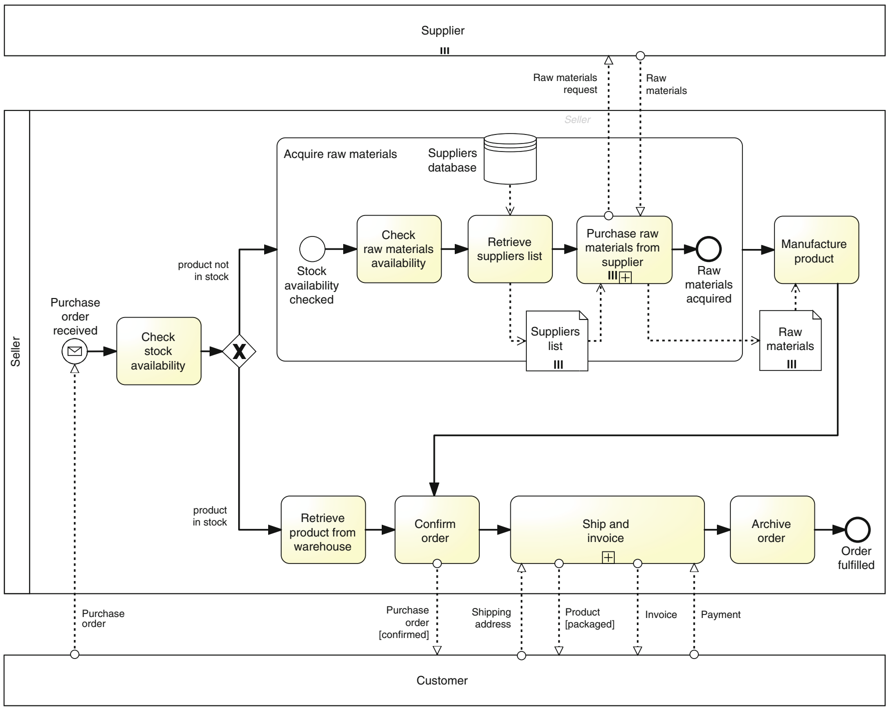


Fig. 4.5 Using a multi-instance pool to represent multiple suppliers

single pool called "Supplier", which we also mark with the multi-instance symbol—a multi-instance pool represents a set of resource classes or resources having similar characteristics.

From this figure we note that there are four message flows connected to the sub-process "Ship and invoice" as a result of collapsing the content of this activity. The order in which these messages are exchanged is determined by the activities inside this sub-process that receive and send them. In other words, when it comes to a collapsed sub-process activity, the message semantics for tasks described in Section 3.4 is not enforced.

Exercise 4.2 Model the following process fragment.

> After a car accident, a statement is sought from two witnesses out of the five that were present in order to lodge the insurance claim. As soon as the first two statements are received, the claim can be lodged with the insurance company without waiting for the other statements.

4.1.2 Uncontrolled Repetition

Sometimes we may need to model that one or more activities can be repeated a number of times without a specific order until a condition is met. For example, let us assume that the customer of our order-to-cash process needs to inquire about the progress of its order. The customer may do so simply by sending an email to the seller. This may be done any time after the customer has submitted the purchase order and as often as the customer desires. Similarly, the customer may attempt to cancel the order or update the personal details before the order has been fulfilled. These activities are *uncontrolled* in the sense that they may be repeated multiple times with no specific order or not occur at all until a condition is met—in our case the order being fulfilled.

To model a set of uncontrolled activities, we can use an *ad hoc sub-process*. Figure 4.6 shows the example of the customer process, where the completion condition ("until order is fulfilled") has been specified via an annotation. The ad hoc sub-process is marked with a tilde symbol at the bottom of the sub-process box.

A partial order may be established among the activities of an ad hoc sub-process via the sequence flow. However, we cannot represent start and end events in an ad hoc sub-process.

Exercise 4.3 Model the following process snippet.

> A typical army recruitment process starts by shortlisting all candidates' applications. Those shortlisted are then called to take the following tests: drug and alcohol, eye, color vision, hearing, blood, urine, weight, fingerprinting, and doctor examination. The color vision can only be done after the eye test, while the doctor examination can only be done after color vision, hearing, blood, urine, and weight have been tested. Moreover, it may be required for some candidates to repeat some of these tests multiple times in order to get a correct assessment, e.g., the blood test may need to be repeated if the candidate has taken too much sugar in the previous 24 h. The candidates who pass all tests are asked to take a mental exam and a physical exam, followed by an interview. Only those who pass all these exams and perform well in the interview can be recruited in the army.

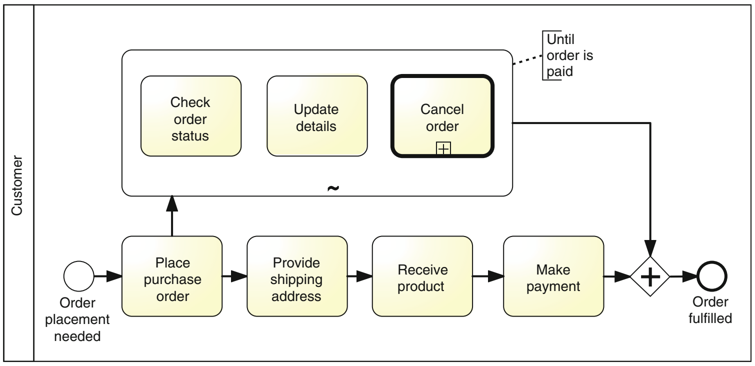


Fig. 4.6 Using an ad hoc sub-process to model uncontrolled repetition

4.2 Handling Events

We have learnt in Chapter 3 that events are used to model something that happens instantaneously in a process. We saw start events, which signal how process instances start (tokens are created), and end events, which signal when process instances complete (tokens are destroyed). When an event occurs during a process, e.g., an order confirmation is received after sending an order out to the customer and before proceeding with the shipment, the event is called *intermediate*. A token remains *trapped* in the incoming sequence flow of an intermediate event until the event occurs. Once the event occurs, the token traverses the event instantaneously, i.e., events cannot retain tokens. An intermediate event is represented as a circle with a double border.

4.2.1 Message Events

In the previous chapter, we showed that we can mark a start event with an empty envelope to specify that new process instances are triggered by the receipt of a message (see Figure 3.16). Besides the start message event, we can also mark an end event and an intermediate event with an envelope to capture the interaction between our process and another party. These types of events are collectively called *message events*. An end message event signals that a process concludes upon sending a message. An intermediate message event signals the receipt of a message or that a message has been sent during the execution of the process. Intermediate and end message events represent an alternative notation to those activities that are solely used to send or receive a message. Take for example activities "Return application to applicant" and "Receive updated application" in Figure 4.7a, which is an extract of the loan assessment model of Solution 3.8. It is more meaningful to replace the former activity with an intermediate send message event and the latter activity with an intermediate receive message event, as illustrated in Figure 4.7b, since these activities do not really represent units of work, but rather the mechanical sending or receiving of a message. An intermediate message event that receives a message is depicted as a start message event, but with a double border. If the intermediate event signals a message being sent, the envelope is dark.

Further, if the send activity is immediately followed by an untyped end event, we can replace this with an end message event, since again, this activity is merely used to send a message after which the process concludes. An end message event is depicted as an end event marked with a darkened envelope. Beware that a start message event is not an alternative notation for an untyped start event followed by a receive activity: these two constructs are not interchangeable. In the former case, process instances start upon the receipt of a specific message; in the latter case, process instances may start at any time, after which the first activity requires a message to be received.

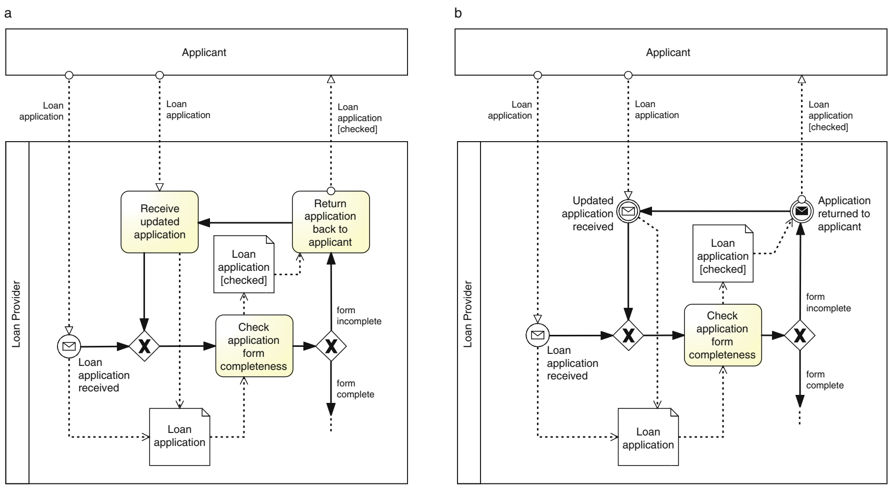

a
Applicant
Loan application
Loan application
Loan application [checked]
Receive updated application
Return application back to applicant
Loan application [checked]
Loan Provider
Check application form completeness
form incomplete
form complete
Loan application received
Loan application
b
Applicant
Loan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

Question Typed or untyped event?

We suggest specifying the type of an event whenever this is known, since it helps the reader to better understand the process model.

Exercise 4.4 Is there any other activity in the loan assessment model of Solution 3.8 (page 111) that can be replaced by a message event?

In BPMN, events come in two flavors based on the filling of their marker. A marker with no fill, like that on the start message event, denotes a *catching event*, i.e., an event that catches a trigger, typically originating from outside the process. A marker with a dark fill like that on the end message event denotes a *throwing event*, i.e., an event that throws a trigger from within the process. An intermediate message event can be used in both as a catching event (the message is received from another pool) or as a throwing event (the message is sent to another pool).

4.2.2 Temporal Events

Besides the message event, there are other triggers that can be specified for a start event. One of them is the *timer event*. This event type indicates that process instances start upon the occurrence of a specific temporal event, e.g., every Friday morning, every working day of the month, every morning at 7 a.m.

A timer event may also be used as an intermediate event to capture that a temporal interval needs to elapse before the process instance can proceed. To indicate a timer event, we mark the event symbol with a watch inside the circle. Timer events are catching events only since a timer is a trigger outside the control of the process. In other words, the process does not generate the timer, but rather reacts to this.

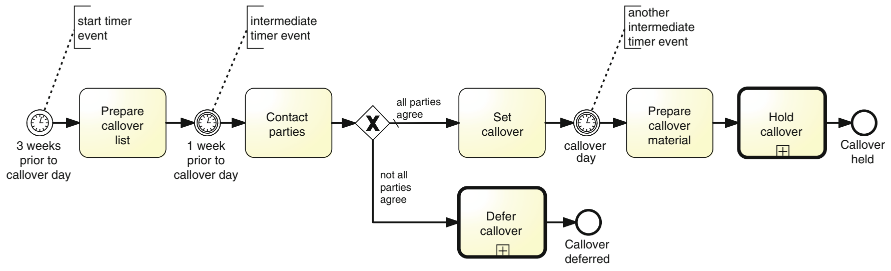


Fig. 4.8 Using timer events to drive the various activities of a business process

Example 4.2 Let us consider the following process at a small claims tribunal.

> In a small claims tribunal, callovers occur once a month to set down the matter for the upcoming trials. The process for setting up a callover starts three weeks prior to the callover day with the preparation of the callover list containing information such as contact details of the involved parties and estimated hearing date. One week prior to the callover, the involved parties are contacted to determine if they are all ready to go to trial. If this is the case, the callover is set, otherwise it is deferred to the next available slot. Finally on the callover day, the callover material is prepared and the callover is held.

This process is driven by three temporal events: it starts three weeks prior to the callover date, continues one week prior to the callover date, and concludes on the day of the callover. To model these temporal events we need one start and two intermediate timer events, as shown in Figure 4.8. Let us see how this process works from a token semantics point of view. A token capturing a new instance is generated every time it is three weeks prior to the callover date (we assume this date has been scheduled by another process). Once the first activity "Prepare callover list" has been completed, the token is sent through the incoming arc of the following intermediate timer event, namely "1 week prior to callover day". The event thus becomes *enabled*. The token remains trapped in the incoming arc of this event until the temporal event itself occurs, i.e., only when it is one week prior to the callover day. Once this is the case, the token instantaneously traverses the event symbol and moves to the outgoing arc. This is why events are said to be instantaneous: they cannot retain tokens as opposed to activities, which retain tokens for the duration of their execution (recall that activities consume time). □

Exercise 4.5 Model the billing process of an Internet Service Provider (ISP).

> The ISP sends an invoice by email to the customer on the first working day of each month (Day 1). On Day 7, the customer has the full outstanding amount automatically debited from its bank account. If an automatic transaction fails for any reason, the customer is notified on Day 8. On Day 9, the transaction that failed on Day 7 is re-attempted. If it fails again, on Day 10 a late fee is charged to the customer's bank account. At this stage, the automatic payment is no longer attempted. On Day 14, the Internet service is suspended until payment is received. If the payment is still outstanding on Day 30, the account is closed and a disconnection fee is applied. A debt-recovery procedure is then started.

4.2.3 Racing Events

A typical scenario encountered when modeling processes with events is the one where two external events *race* against one another. The first of the two events that occurs determines the continuation of the process. For example, after an insurance quote has been sent to a client, the client may reply either with an acceptance message, in which case an insurance contract will be made, or with a rejection, in which case the quote will be discarded.

This race between external events is captured by means of the *event-based exclusive (XOR)* split. An event-based exclusive split is represented by a gateway marked with an empty pentagon enclosed in a double-line circle. Figure 4.9 features an event-based exclusive split. When the execution of the process arrives at this point (in other words, when a token arrives at this gateway), 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. If the timer event occurs first, a shipment status inquiry will be initiated and the execution flow will come back to the event-based exclusive gateway. If the message signaling the freight delivery is received first, the execution flow will proceed along the sequence flow that leads to the AND-join.

The difference between the XOR-split, which we saw in Chapter 3, and the event-based XOR-split is that the former models an internal choice that is determined by the outcome of a decision activity, whereas the latter models a choice that is determined by the environment of the process. For this reason, the XOR-split of Chapter 3 is called *data-based XOR-split*, because the branch to be taken is determined based on the evaluation of two or more conditions on data produced by a decision activity. An internal choice is determined by the outcome of a decision activity. Thus, the event-based XOR-split can only be followed by intermediate catching events like a timer or a message event, or by receiving activities. Since the choice is delayed until an event happens, the event-based split is also known as

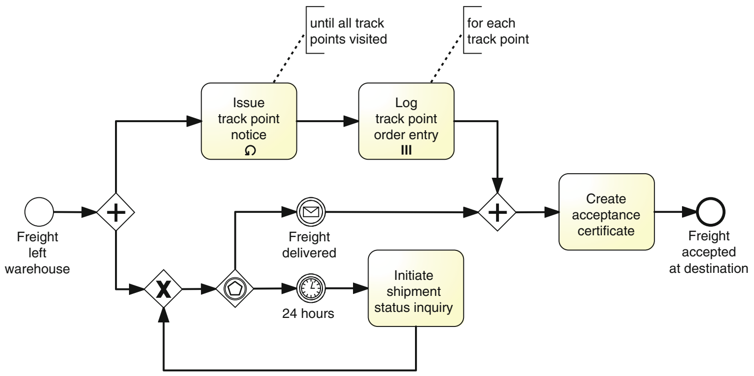

until all track points visited
for each track point
Issue track point notice
Log track point order entry
Freight left warehouse
Freight delivered
24 hours
Initiate shipment status inquiry
Create acceptance certificate
Freight accepted at destination

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

deferred choice. There is no event-based XOR-join, so the branches emanating from an event-based split are merged with a normal XOR-join.

Exercise 4.6 Model the following process.

> A restaurant chain submits a purchase order (PO) to replenish its warehouses every Thursday. The restaurant chain's procurement system assumes the receipt of either a "PO Response" or an error message. However, it may also happen that no response is received at all due to system errors or due to delays in handling the PO on the supplier's side. If no response is received by Friday afternoon or if an error message is received, a purchasing officer at the restaurant chain's headquarters should be notified. Otherwise, the PO Response is processed normally.

The event-based XOR-split can be used as the counterpart of an internal decision on a collaborating party. For example, consider Figure 4.10. The choice made from within the client pool to send either an acceptance message or a rejection message to an insurer has to be matched by an event-driven decision on the insurer pool to *react* to the choice made by the client.

Event-based gateways can be used to avoid behavioral anomalies in the communication between pools. Take for example the collaboration diagram of the auctioning service and the seller in Figure 4.11. This collaboration may deadlock if the seller is already registered, because this party will wait for the account creation request message that can never arrive. To fix this issue, we need to allow the seller to receive the creation confirmation message straightaway in case the seller is already registered, as shown in Figure 4.12.

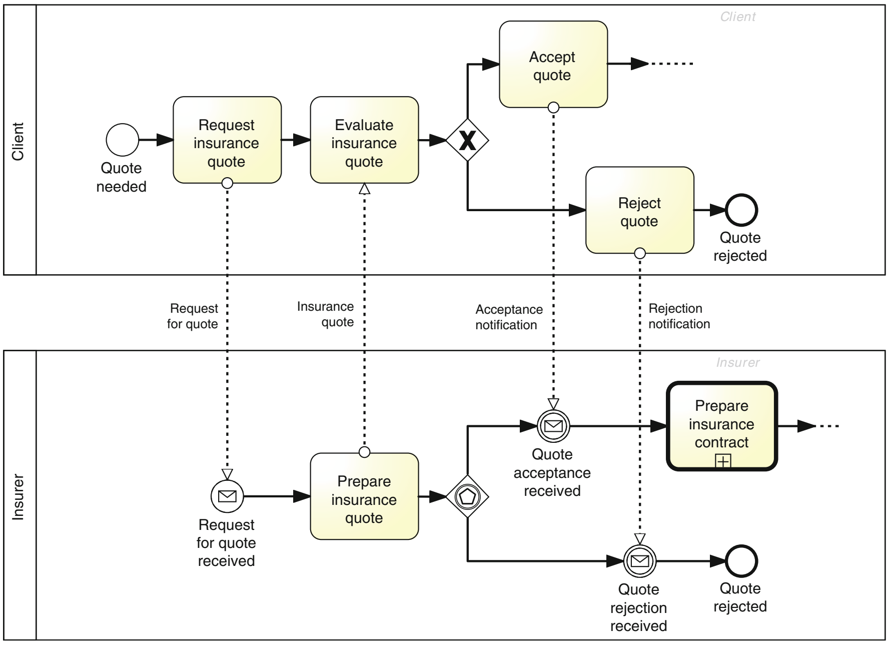


Fig. 4.10 Matching an internal choice in one party with an event-based choice in the other party

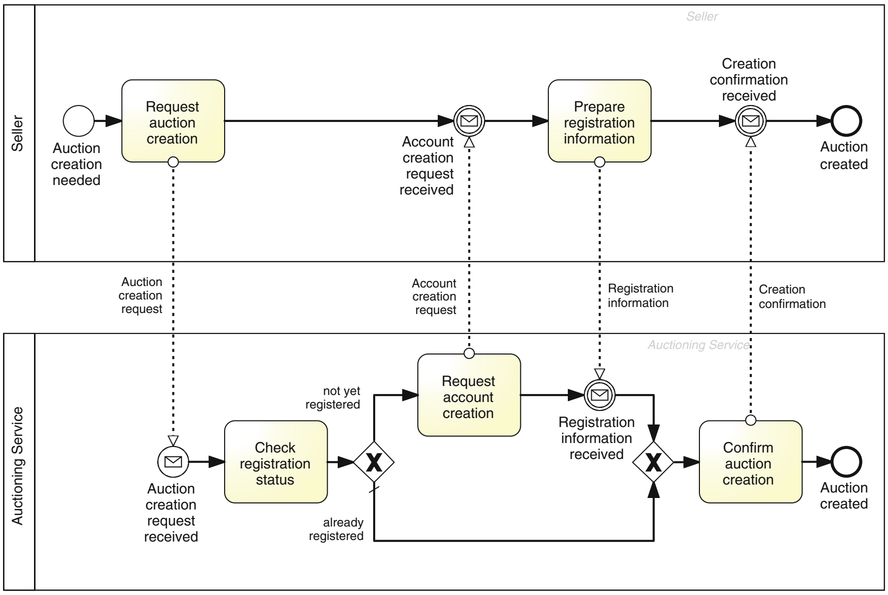


Fig. 4.11 An example of collaboration that can deadlock if the decision is made for "already registered"

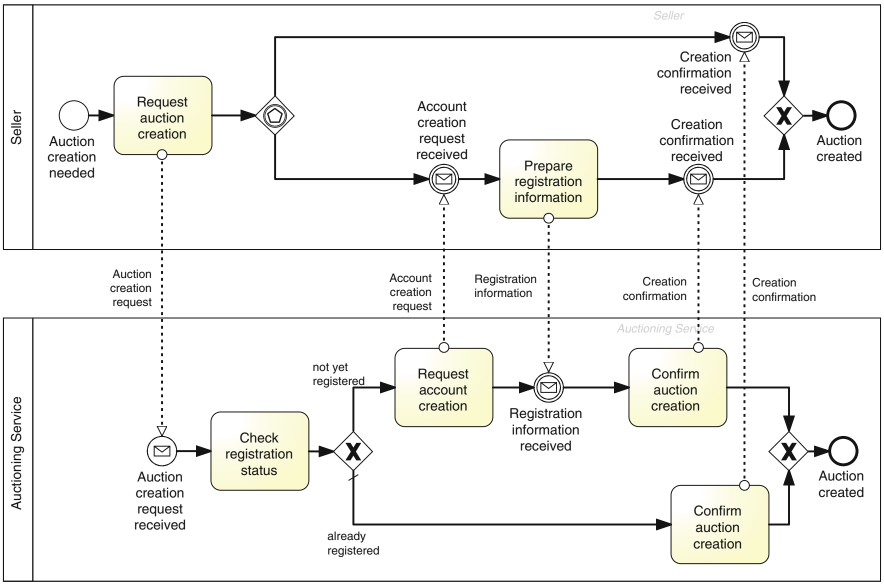


Fig. 4.12 Using an event-based gateway to fix the problem of a potential deadlock in the collaboration of Figure 4.11

When connecting pools with each other via message flows, make sure you check the order of these connections so as to avoid deadlocks. Recall, in particular, that an internal decision of one party needs to be matched by an event-based decision of the other party and that an activity with an outgoing message flow will send that message upon activity completion, whereas an activity with an incoming message flow will wait for that message to start.

4.3 Handling Exceptions

Exceptions are events that deviate a process from its normal course, i.e., from what is commonly known as the "sunny-day" scenario. "Rainy-day" scenarios happen frequently in reality and as such they should be modeled when the objective is to identify all possible causes of problems in a given process. Exceptions include *business faults* such as an out-of-stock or discontinued product and *technology faults* like a database crash, a network outage or a program logic violation. They cause the interruption or abortion of the running process. For example, in case of an out-of-stock product, an order-to-cash process may need to be interrupted to order the product from a supplier, or aborted altogether if the product cannot be supplied within a given timeframe.

Exercise 4.7 Fix the collaboration diagram in Figure 4.13.

Acknowledgment This exercise is partly inspired by [92].

4.3.1 Process Abortion

The simplest way of handling an exception is to abort the running process and signal an abnormal process termination. This can be done by using an *end terminate event* as shown in Figure 4.14. An end terminate event (depicted as an end event marked with a full circle inside) causes the immediate cessation of the process instance at its current level and for any sub-process.

In the example of Figure 4.14—a variant of the home loan that we already saw in Figure 3.19—a home loan is rejected and the process is aborted if the applicant has high debts or high liability. The terminate event destroys all tokens in the process model and in any sub-process. In our example, this is needed to avoid the process to deadlock at the AND-join, since a token may remain trapped before the AND-join if there is high liability and low debts or low liability and high debts.

Observe that if a terminate event is triggered from within a sub-process, it will not cause the abortion of the parent process but only that of the sub-process, i.e., the terminate event is only propagated downwards in a process hierarchy.

Exercise 4.8 Revise the examples presented so far in this chapter by using the terminate event appropriately.

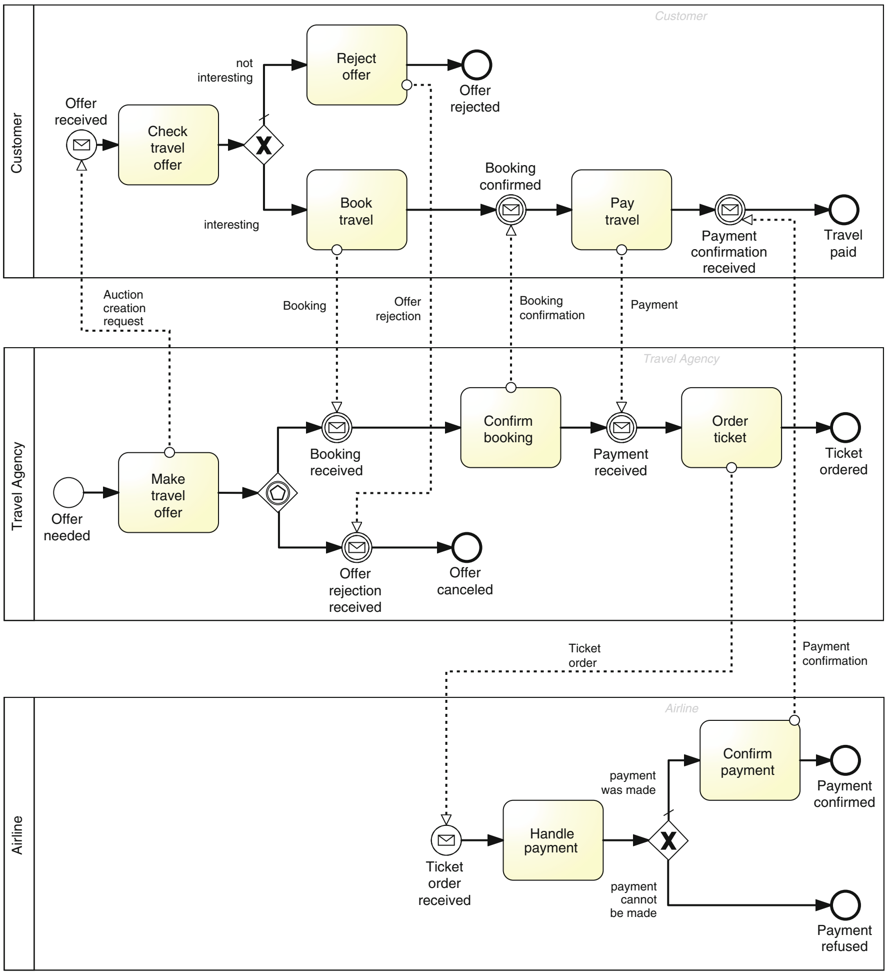


Fig. 4.13 A collaboration diagram between a customer, a travel agency, and an airline

4.3.2 Internal Exceptions

Instead of aborting the whole process, we can handle an exception by interrupting the specific activity that has caused the exception. Next, we can start a recovery procedure to bring the process back to a consistent state and continue its execution, and if this is not possible, only then, abort the process altogether. BPMN provides the *error event* to capture these types of scenarios. An end error event is used to interrupt the enclosing sub-process and throw an exception. This exception is then caught by an intermediate catching error event which is attached to the boundary of the same sub-process. In turn, this *boundary event* triggers the recovery procedure through an outgoing branch, which is called *exception flow*.

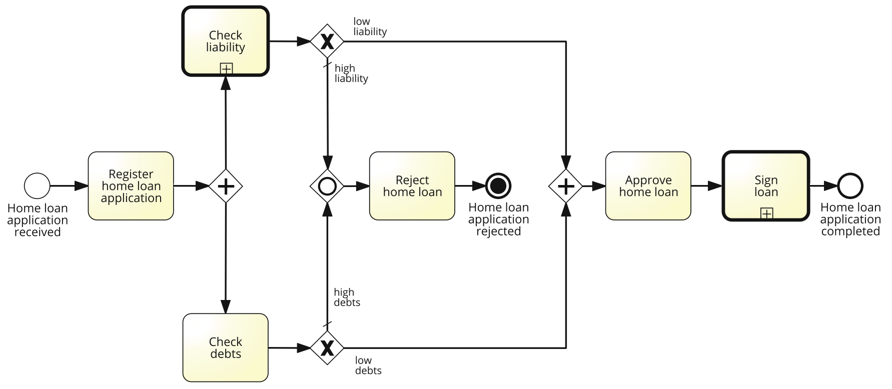

Check liability
low liability
high liability
Home loan application received
Register home loan application
Reject home loan
Home loan application rejected
Approve home loan
Sign loan
Home loan application completed
high debts
Check debts
low debts

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

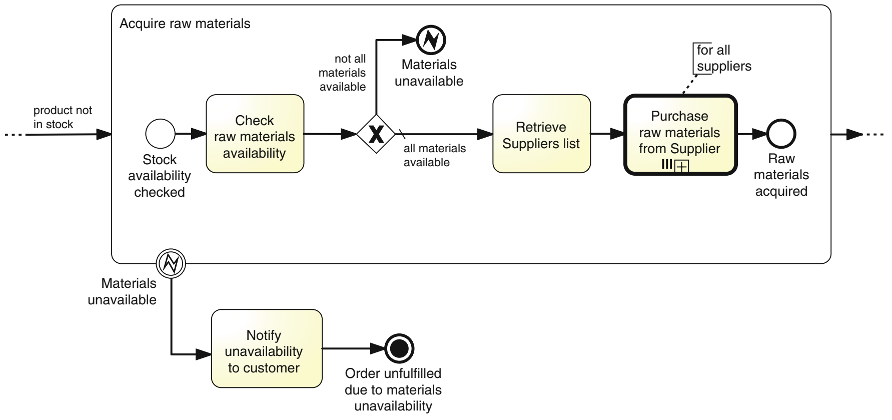

Acquire raw materials
product not in stock
Stock availability checked
Check raw materials availability
not all materials available
Materials unavailable
all materials available
Retrieve Suppliers list
for all suppliers
Purchase raw materials from Supplier
Raw materials acquired
Materials unavailable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

Fig. 4.15 Error events model internal exceptions

The error event is depicted as an event with a lightning marker. Following the BPMN conventions for throwing and catching events, the lightning is empty for the catching intermediate event and full for the end throwing event.

An example of error events is shown in Figure 4.15 in the context of our order-to-cash process. If there is an out-of-stock exception, the acquisition of raw materials is interrupted and a recovery procedure is triggered, which in this case simply consists of a task to notify the customer before aborting the process. In terms of token semantics, upon throwing an end error event, all tokens are removed from the enclosing sub-process (causing its interruption) and one token is sent via the exception flow emanating from the boundary error event. There is no restriction on the modeling elements we can put in the exception flow to model the recovery procedure. Typically, we would complete the exception flow with an end terminate event to abort the process, or wire this flow back to the normal sequence flow if the exception has been properly handled.

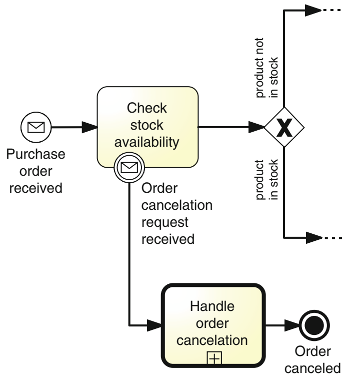


Fig. 4.16 Boundary events catch external events that can occur during an activity

4.3.3 *External Exceptions*

An exception may also be caused by an external event occurring during an activity. For example, while checking the stock availability for the product in a purchase order, the seller may receive an order cancelation from the customer. Upon this request, the seller should interrupt the stock availability check and handle the order cancelation. Scenarios like the above are called *unsolicited exceptions* since they originate externally to the process. They can be captured by attaching a catching intermediate message event to an activity's boundary as shown in Figure 4.16. From a token semantics, when the intermediate message event is triggered, the token is removed from the enclosing activity, causing the activity interruption, and sent via the exception flow emanating from the boundary event to perform the recovery procedure.

Before using a boundary event we need to identify the *scope* within which the process should be receptive of this event. For example, in the order-to-cash example, order cancelation requests can only be handled during the execution of task "Check stock availability". Thus, the scope for being receptive to this event is made up by this single task. Sometimes the scope should include multiple activities. In these cases, we can encapsulate the interested activities into a sub-process and attach the event to the boundary of the sub-process.

Exercise 4.9 Model the following routine for accessing an Internet bank service.

> The routine for logging into an Internet bank account starts once the credentials entered from the user have been retrieved. First, the username is validated. If the username is not valid, the routine is interrupted and the invalid username is logged. If the username is valid, the number of password trials is set to zero. Then, the password is validated. If this is not valid, the counter for the number of trials is incremented and if lower than three, the user is asked to enter the password again, this time together with a CAPTCHA test to increase the security level. If the number of failed attempts reaches three times, the routine is interrupted and the account is frozen. Moreover, the username and password validation may be interrupted should the validation server not be available. Similarly, the server to test the CAPTCHA may not be available at the time of log in. In these cases, the procedure is interrupted after notifying the user to try again later. At any time during the log in routine, the customer may close the web page, resulting in the interruption of the routine.

4.3.4 Activity Timeouts

Another type of exception is the interruption of an activity which is taking too long to complete. To model that an activity must be completed within a given timeframe (e.g., an approval must be completed within 24 h), we can attach an intermediate timer event to the boundary of the activity: the timer is activated when the enclosing activity starts. If it fires before the activity completes, it provokes the interruption of the activity. In other words, a timer event works as a timeout when attached to an activity boundary.

Exercise 4.10 Model the following process fragment.

> Once a wholesale order has been confirmed, the supplier transmits this order to the carrier for the preparation of the transportation quote. In order to prepare the quote, the carrier needs to compute the route plan (including all track points that need to be traversed during the travel) and estimate the trailer usage (e.g., whether it is a full track load, half track load or a single package). By contract, wholesale orders have to be dispatched within 4 days from the receipt of the order. This implies that transportation quotes have to be prepared within 48 h from the receipt of the order to remain within the terms of the contract.

4.3.5 Non-Interrupting Events and Complex Exceptions

There are situations where an external event occurring during an activity should just trigger a procedure without interrupting the activity itself. For example, in the order-to-cash process, the customer may send a request to update its details during the stock availability check. The details should be updated in the customer database without interrupting the stock check. To denote that the boundary event is *non-interrupting*, we use a dashed double border as shown in Figure 4.17.

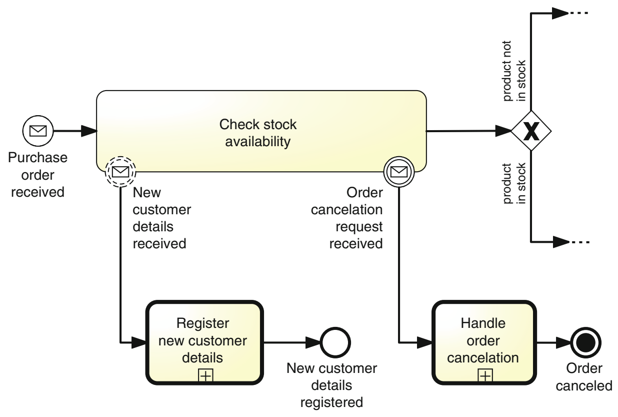

Purchase order received
Check stock availability
product not in stock
product in stock
New customer details received
Order cancelation request received
Register new customer details
New customer details registered
Handle order cancelation
Order canceled

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

Exercise 4.11 Extend the process for assessing loan applications from Solution 3.8 (page 111) as follows.

> An applicant who has decided not to combine the loan with a home insurance plan may change its mind any time before the eligibility assessment has been completed. If a request for adding an insurance plan is received during this period, the loan provider will simply update the loan application with this request.

Non-interrupting events can be used to model more complex exception handling scenarios. Consider again the example in Figure 4.15 and assume that the customer sends a request to cancel the order during the acquisition of raw materials. We catch this request with a non-interrupting boundary message event, and first determine the penalty that the customer will need to incur based on the raw materials that have already been ordered. We forward this information to the customer who then may decide within 48 h to either stop the cancelation, in which case nothing is done, or go on with it (see Figure 4.18). In the latter case, we throw an end *signal event*. This event, depicted with a triangle marker, broadcasts a signal defined by the event label. This signal can be caught by all catching signal events bearing the

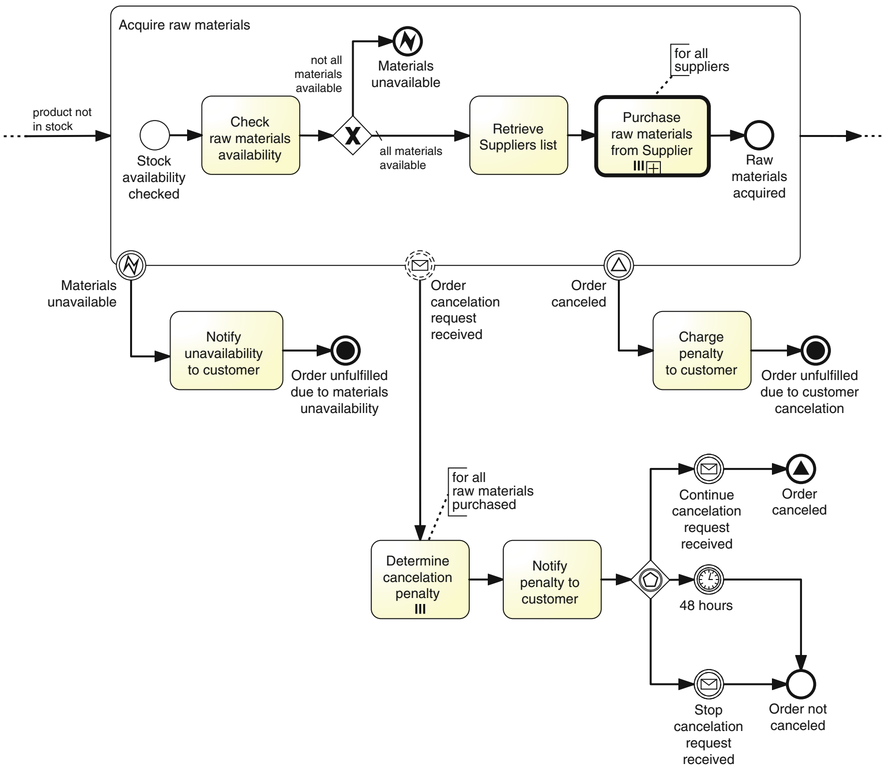

Acquire raw materials
product not in stock
Stock availability checked
Check raw materials availability
not all materials available
Materials unavailable
all materials available
Retrieve Suppliers list
for all suppliers
Purchase raw materials from Supplier
Raw materials acquired
Materials unavailable
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

same label. In our case, we throw an "Order canceled" signal and catch this with a matching intermediate signal event on the boundary of the sub-process for acquiring raw materials. This event causes the enclosing sub-process to be interrupted and then triggers a recovery procedure to charge the customer, after which the process is aborted. We observe that in this scenario the activity interruption is triggered from within the process, but outside the activity itself.

Observe that the signal event is different than the message event, since it has a source but no specific target, whilst a message has both a specific source and a specific target. Like messages, signals may also originate from a process modeled in a separate diagram.

4.3.6 Event Sub-processes

An alternative notation to boundary events is the *event sub-process*. An event sub-process is started by the event, which would otherwise be attached to the boundary of an activity, and encloses the procedure that would be triggered by the boundary event. An important difference with boundary events is that event sub-processes do not need to refer to a specific activity, but can model events that occur during the execution of the whole process. For example, any time during the order-to-cash process the customer may send an inquiry about the order status. To handle this request, which is not specific to a particular activity of this process, we can use an event sub-process as shown in Figure 4.19.

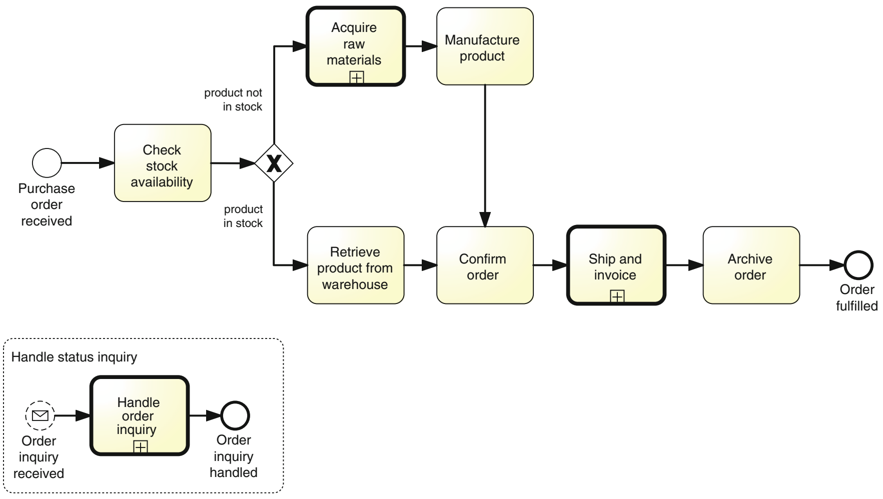


Fig. 4.19 Event sub-processes can be used in place of boundary events and to catch events thrown from outside the scope of a particular sub-process

The event sub-process is depicted within a dotted rectangle with rounded corners, which is placed into an expanded sub-process or into the top-level process. Similarly to boundary events, an event sub-process may or may not interrupt the enclosing process depending on whether its start event is interrupting or not. If its start event is non-interrupting, this is depicted with a dashed (single) border.

All syntactical rules for a sub-process apply to the event sub-process, except for boundary events, which cannot be defined on event sub-processes. For example, the event sub-process can also be represented as a collapsed sub-process. In this case, the start event is depicted on the top-left corner of the collapsed event sub-process rectangle to indicate how this event sub-process is triggered.

Question Event sub-processes or boundary events?

Event sub-processes are self-contained, meaning that they must conclude with an end event. This has the disadvantage that the procedure captured inside an event sub-process cannot be wired back to the rest of the sequence flow. The advantage is that an event sub-process can also be defined as a global process model and thus be reused in other process models of the same organization. Another advantage is that event sub-processes can be defined at the level of an entire process whereas boundary events must refer to a specific activity. Thus, we suggest using 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. For all other cases, boundary events are more appropriate since the procedure triggered by these events can be wired back to the rest of the flow.

Exercise 4.12 Model the following business process for reimbursing expenses.

> After an expense report is received from an employee, the employee is notified of the receipt of the report. Next, a new account must be created if the employee does not already have one. The report is then reviewed for automatic approval. Amounts under € 1,000 are automatically approved while amounts equal to or over € 1,000 require manual approval. In case of rejection, the employee must receive a rejection notice by email. In case of approval, the reimbursement is deposited directly to the employee's bank account and an approval notice is sent to the employee via email, with the details of the money transfer. At any time during the review, the employee can send a request for amount rectification. In that case the rectification is registered and the report needs to be reviewed again. Moreover, if the report is not handled within 30 days, the process is stopped and the employee receives a cancelation notice email so that he can resubmit the expense report from scratch.

4.3.7 Activity Compensation

As part of a recovery procedure, we may need to *undo* one or more steps that have already been completed, due to an exception that occurred in the enclosing sub-process. In fact, the results of these steps, and possibly their side effects, may no longer be desired and for this reason they should be reversed. This operation is called *compensation* and tries to restore the process to a business state close to the one before starting the sub-process that was interrupted.

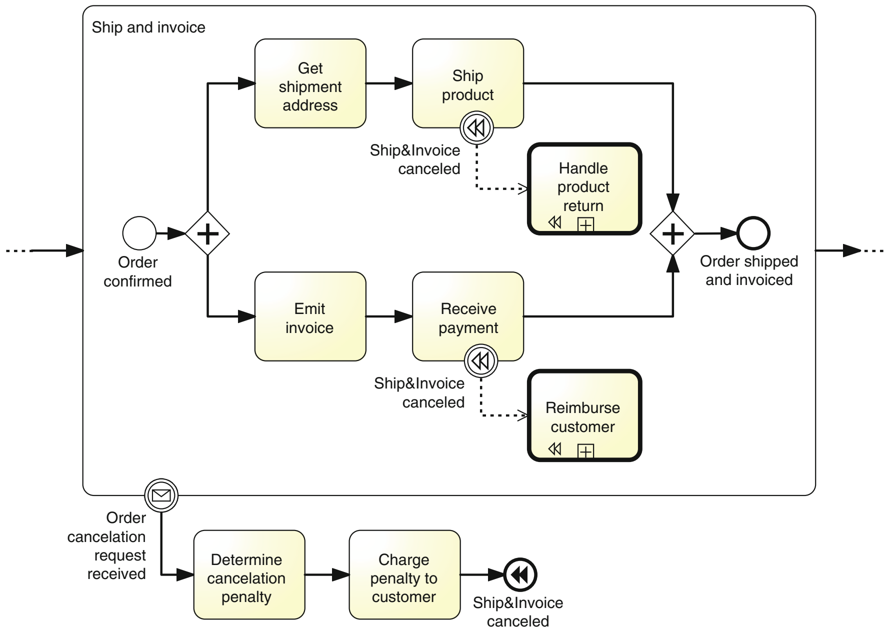


Fig. 4.20 Compensating for the shipment and for the payment

Let us delve into the sub-process for shipment and invoice handling of the order-to-cash example and assume that also this activity can be interrupted upon the receipt of an order cancelation request (see Figure 4.20). After communicating the cancelation penalty to the customer, we need to revert the effects of the shipment and of the payment. Specifically, if the shipment has already been made, we need to handle the product return, whereas if the payment has already been made, we need to reimburse the customer. These compensations can be modeled via a *compensation handler*. A compensation handler is made up of a throwing *compensate event* (an event marked with a rewind symbol), a catching intermediate compensate event, and a compensation activity. The throwing compensate event is used inside the recovery procedure of an exception to start the compensation and can be an intermediate or an end event (in the latter case, the recovery procedure concludes with the compensation). The catching intermediate compensation event is attached to those activities that need to be compensated—in our example "Ship product" and "Receive payment". These boundary events catch the compensation request and trigger a *compensation activity* specific to the activity to be compensated. For example the compensation activity for "Receive payment" is "Reimburse customer". The boundary event is connected to the compensation activity via a dotted arrow with an open arrowhead, called *compensation association* (whose notation is the same as that of the data association). This activity is marked with the compensate symbol to indicate its purpose. It must not have any outgoing flow. In case the compensation procedure is complex, this activity can be a sub-process.

Compensation is only effective if the attached activity has completed. Once all activities that could be compensated are compensated, the process resumes from after the throwing compensation event, unless this is an end event. If the compensation is for the entire process, we can use an event sub-process with a start compensate event in place of the boundary event.

4.3.8 Summary

In this section we saw various ways to handle exceptions in a business process, ranging from simple process abortion to complex error event and compensation handling. Before adding exceptions it is important to understand the sunny-day scenario well. So we recommend you to start by modeling the simple sunny-day scenario first. Then, think of all possible situations that can go wrong. For each of these exceptions, identify what type of exception handling mechanism needs to be used. First, determine the cause of the exception: internal or external. Next, decide if aborting the process is enough or if a recovery procedure needs to be triggered. Finally, evaluate whether the interrupted activity needs to be compensated as part of the recovery procedure.

Exercise 4.13 Modify the model that you created in Exercise 4.12 as follows.

> If the report is not handled within 30 days, the process is stopped, the employee receives a cancelation notice email, and must resubmit the expense report. However, if the reimbursement for the employee's expenses had already been made, a money recall needs to be made to get the money back from the employee before sending the cancelation notice email.

4.4 Processes and Business Rules

A business rule implements an organizational policy or practice. For example, in an online shop, platinum customers have a 20% discount for each purchase above € 250. Business rules can appear in different forms in a process model. We have seen them modeled in a decision activity and in the condition of a flow coming out of an (X)OR-split (see Exercise 3.6 on page 96 for some examples). A third option is to use a dedicated BPMN event called *conditional event*. A conditional event causes the activation of its outgoing flow when the respective business rule is fulfilled. Conditional events, identified by a lined page marker, can be used as start or intermediate catching events, including after an event-based gateway, or they can be attached to an activity boundary. An example of conditional event is shown in Figure 4.21.

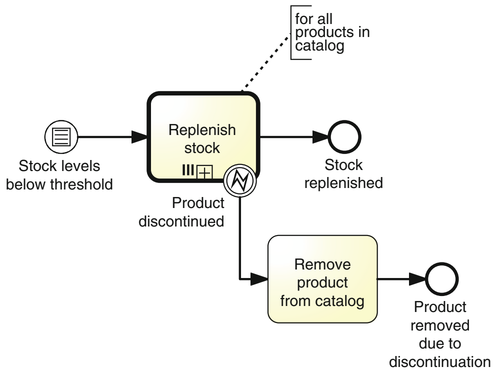

for all products in catalog
Replenish stock
Stock levels below threshold
Stock replenished
Product discontinued
Remove product from catalog
Product removed due to discontinuation

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

The difference between an intermediate conditional event and a condition on a flow is that the latter is only tested once, and if it is not satisfied the corresponding flow is not taken (another flow or the default flow will be taken instead). The conditional event, on the other hand, is tested until the associated rule is satisfied. In other words, the token remains trapped before the event until the rule is satisfied.

In the example of Figure 4.21, observe the use of the error event on the boundary of a multi-instance activity. This event only interrupts the activity instance that refers to the particular product being discontinued, i.e., the instance from which the error event is thrown. All other interrupting boundary events, i.e., message, timer, signal and conditional, interrupt all instances of a multi-instance activity.

Chapter 10 will illustrate how business rules can be implemented using decision tables specified using the Decision Model and Notation (DMN) language.

Exercise 4.14 Model the following business process snippet.

> In a stock exchange, stock price variations are continuously monitored during the day. A day starts when the opening bell rings and concludes when the closing bell rings. Between the two bells, every time the stock price changes by more than 10%, the entity of the change is first determined. Next, if the change is high, a "high stock price" alert is sent, otherwise a "low stock price" alert is sent.

4.5 Recap

This chapter provided us with the means to model complex business processes. First, we expanded on the topic of rework and repetition. We illustrated how structured loops can be modeled using a loop activity. Furthermore, we presented the multi-instance activity as a way to model an activity that needs to be executed multiple times without knowing the number of occurrences beforehand. We also saw how the concept of multi-instantiation can be related to data collections and extended to pools, and discussed ad hoc sub-processes for capturing unstructured repetition.

Next, we expanded on various types of events. We explained the difference between catching and throwing events and distinguished between start, end, and intermediate events. We saw how message exchange between pools can be framed by message events and how timer events can be used to model temporal triggers to the process or delays during the process. We then showed how to capture racing conditions between external events via the event-based XOR-split.

Afterwards, we showed how to handle exceptions. The simplest way to react to an exception is to abort the process via a terminate end event. Exceptions can be handled by using a catching intermediate event on the boundary of an activity. If the event is caught during the activity's execution, the activity is interrupted and a recovery procedure may be launched. Another type of exception is the activity timeout. This occurs when an activity does not complete within a given timeframe. A boundary event can also be non-interrupting, to model procedures that have to be launched in parallel to an activity's execution. Related to exception handling is the notion of activity compensation. Compensation is required to revert the effects of an activity that has been completed, if these effects are no longer desired due to an exception that has occurred.

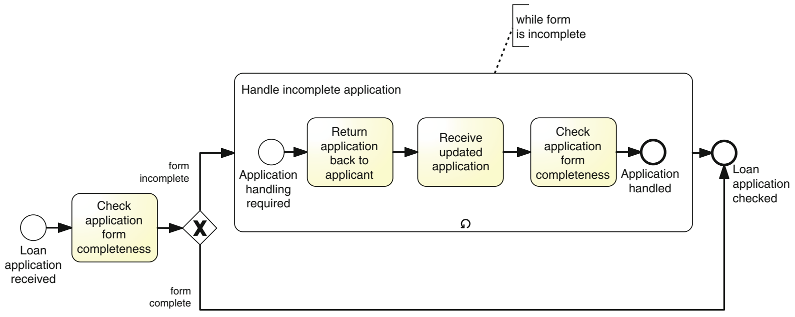
We also saw how business rules can be defined in process models via conditional events. A conditional event allows a process instance to start or progress only when the corresponding business rule evaluates to true.

4.6 Solutions to Exercises

Solution 4.1

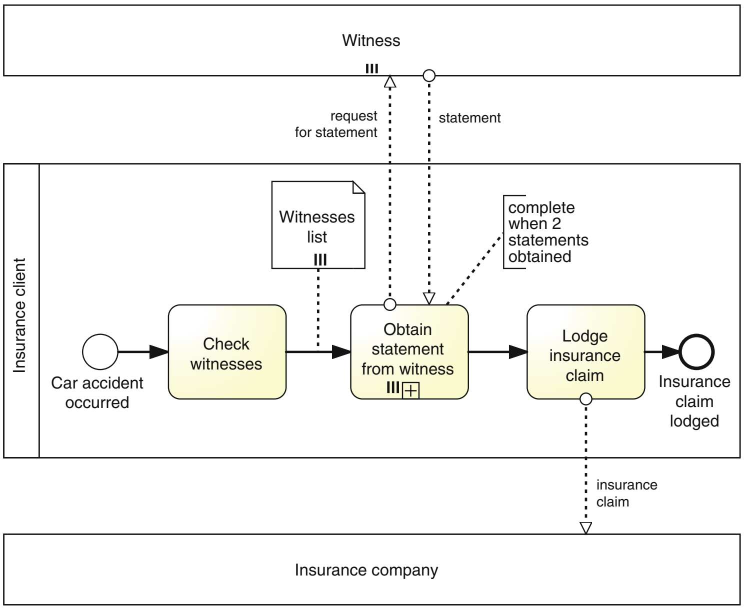
1. In Exercise 3.12 the repetition block goes from activity "Record claim" to activity "Review claim rejection". The entry point to the cycle is the arc from activity "Create claim" to the subsequent XOR-join. The exit points are the arcs "claim to be accepted" and "claim rejection accepted", the former emanating from within the repetition block.
2. In Solution 3.4 the repetition block is made up of the activities "Check application form completeness", "Return application back to applicant", and "Receive updated application". The entry point to the cycle is the outgoing arc of the XOR-join, while the exit point is the arc "form complete" which emanates from

within the repetition block. To model this cycle with a loop activity, we need to repeat activity "Check application form completeness" outside the loop activity, as shown below.

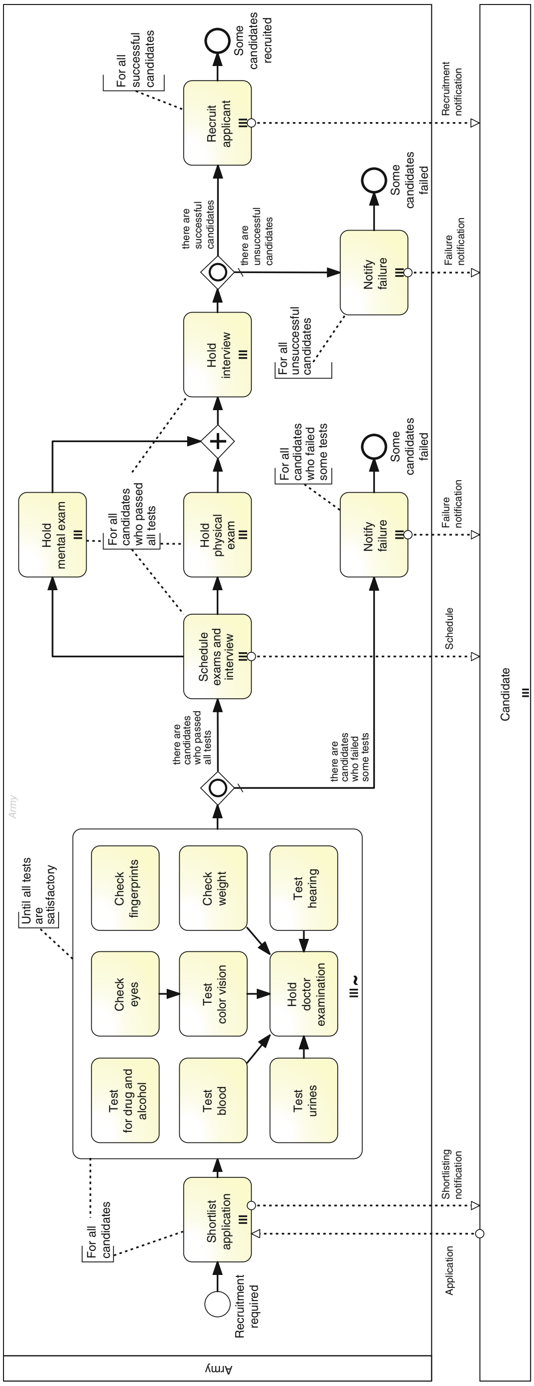

while form is incomplete
Handle incomplete application
Return application back to applicant
Receive updated application
Check application form completeness
Application handling required
Application handled
Loan application checked
form incomplete
form complete
Check application form completeness
Loan application received

In this case using a loop activity is still advantageous, since we reduce the size of the original model if we collapse the sub-process.

Solution 4.2

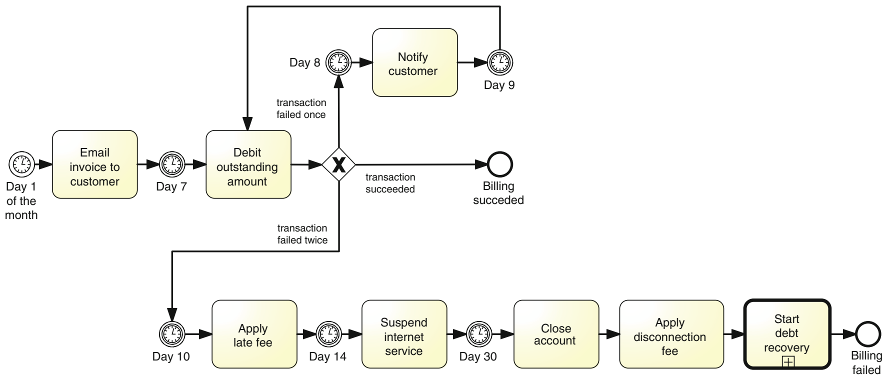


Solution 4.3

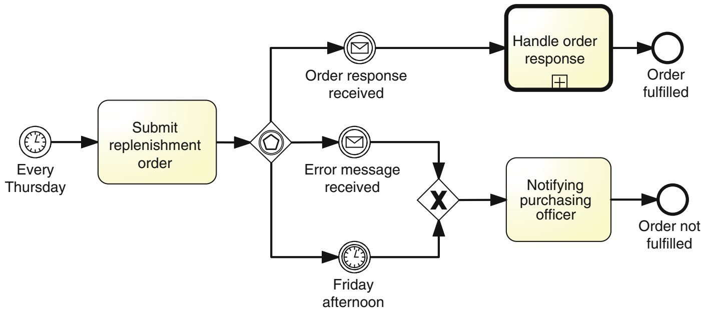

Army
Candidate
Recruitment required
Shortlist application
For all candidates
Application
Shortlisting notification
Until all tests are satisfactory
Test for drug and alcohol
Check eyes
Check fingerprints
Test blood
Test color vision
Check weight
Test urines
Hold doctor examination
Test hearing
there are candidates who passed all tests
there are candidates who failed some tests
Schedule exams and interview
Schedule
Hold mental exam
Hold physical exam
For all candidates who passed all tests
Notify failure
For all candidates who failed some tests
Failure notification
Some candidates failed
Hold interview
there are successful candidates
there are unsuccessful candidates
For all unsuccessful candidates
Notify failure
Failure notification
Some candidates failed
Recruit applicant
For all successful candidates
Recruitment notification
Some candidates recruited

Solution 4.4 The activity “Send acceptance pack” can be replaced by an intermediate send message event; activities “Notify cancelation” and “Notify approval” can each be replaced by an end message event, thus removing the last XOR-join and the untyped end event altogether. Note that the activity “Send home insurance quote” cannot be replaced by a message event since it subsumes the preparation of the quote. In fact, a more appropriate label for this activity would be “Prepare home insurance quote”. Similarly, we cannot get rid of activity “Reject application” as this activity changes the status of the application before sending the latter out.

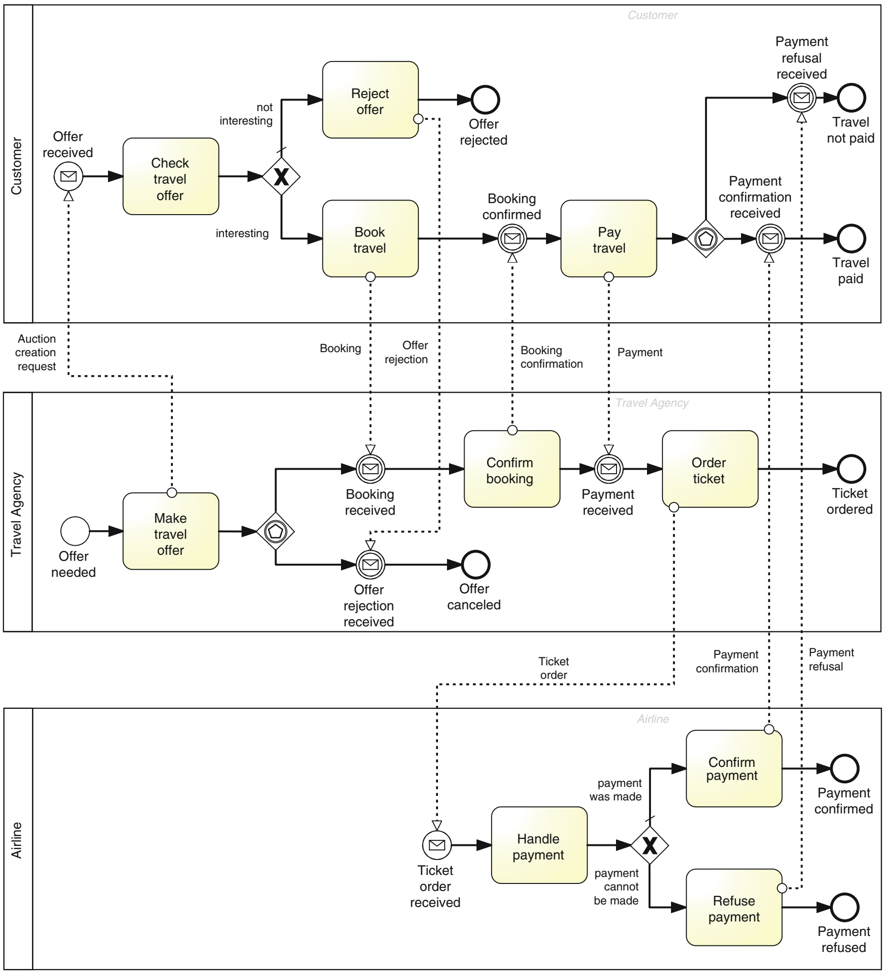
Solution 4.5



Solution 4.6

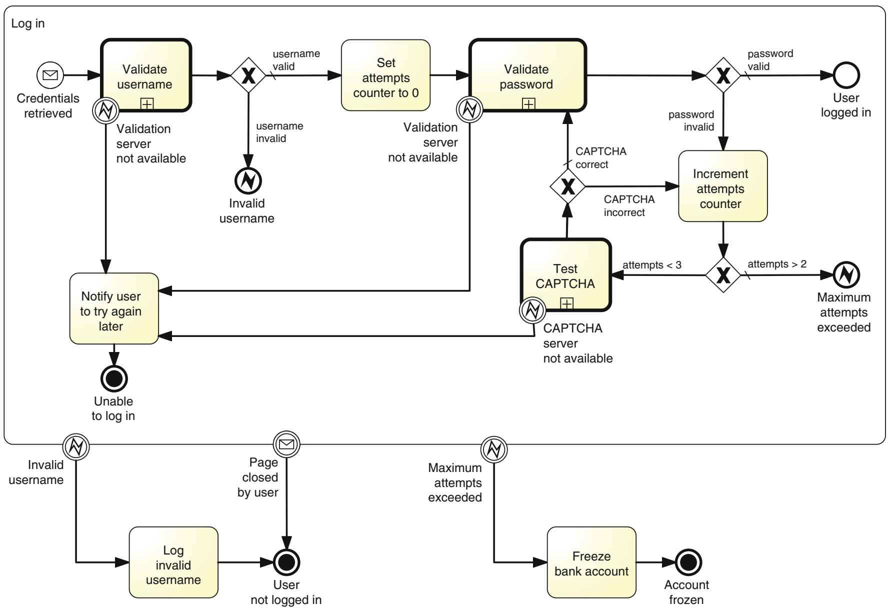


Solution 4.7

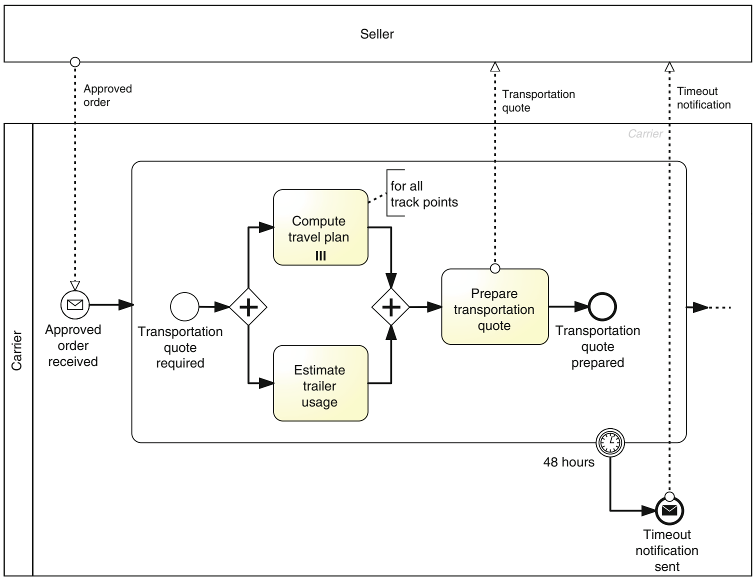


Solution 4.8 The following end events should be terminate events. Figure 4.8: “callover deferred”; Figure 4.10: “Quote rejected” in the client and insurer pools; Figure 4.14: “Offer rejected” in the customer pool, “Offer canceled” in the travel agency pool, and “Payment refused” in the airline pool.

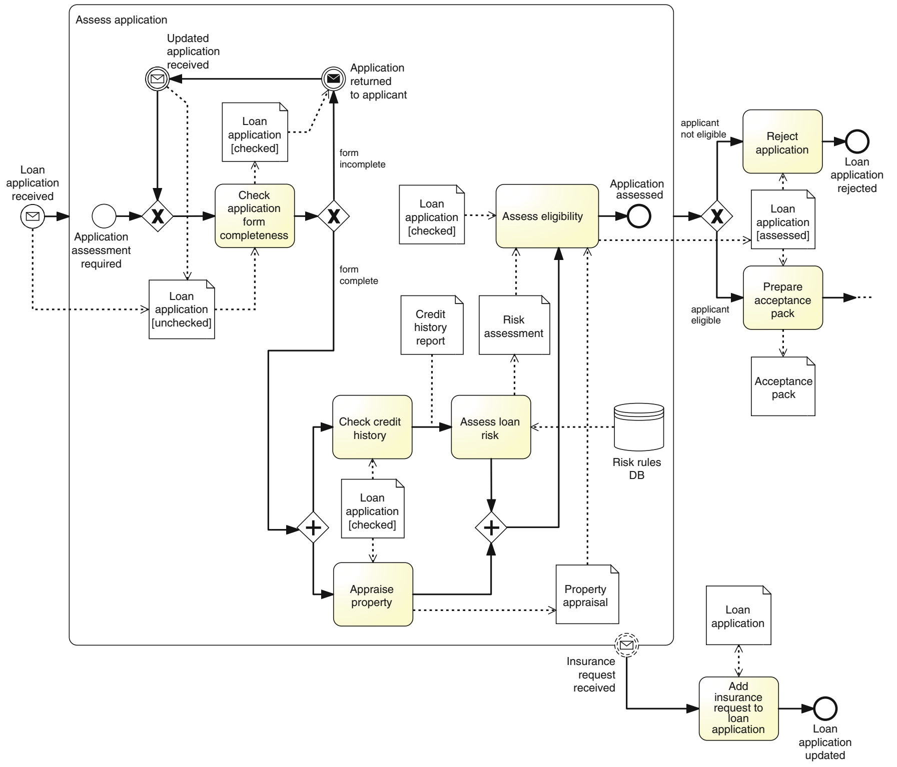
Solution 4.9



Solution 4.10

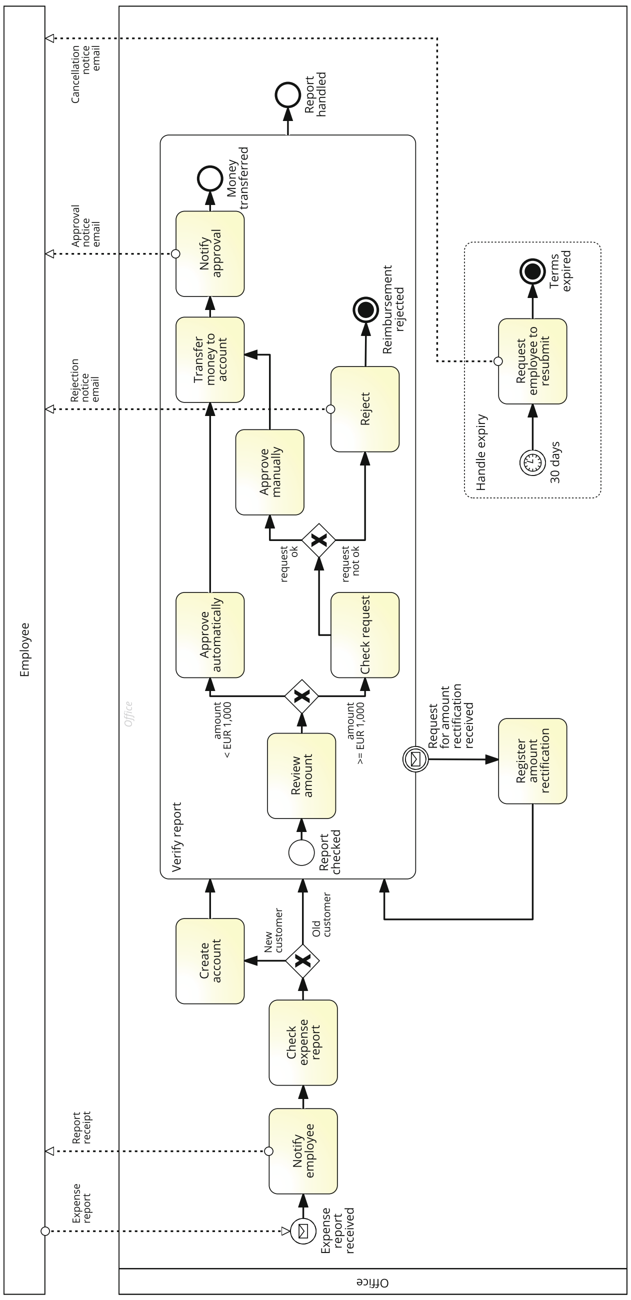


Solution 4.11

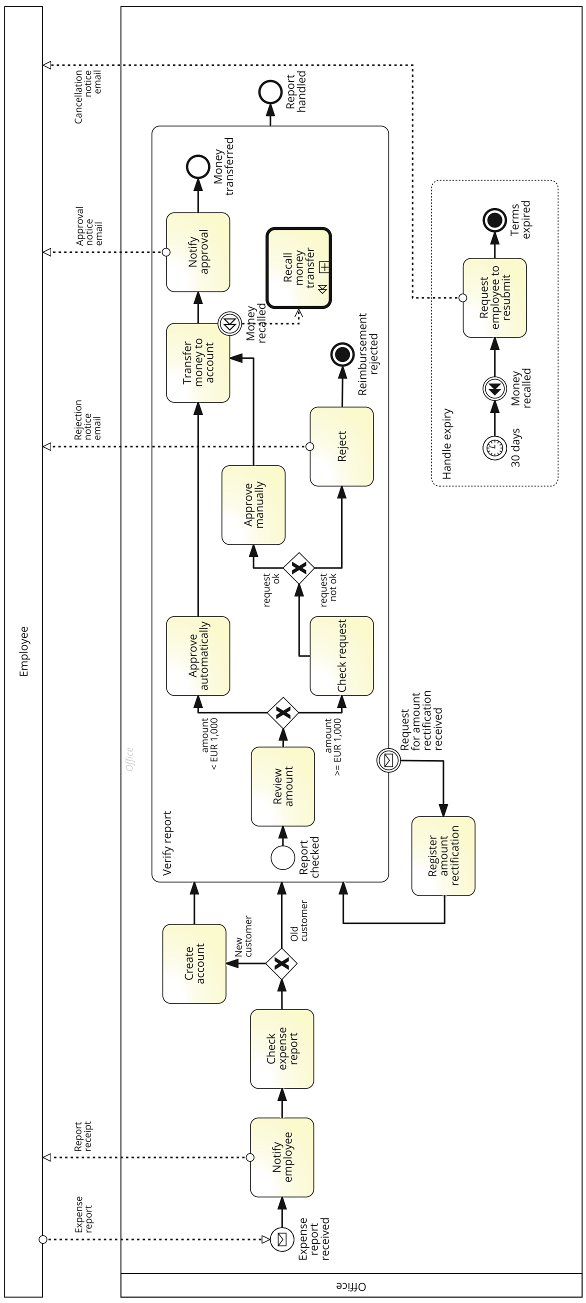

Assess application
Updated application received
Application returned to applicant
Loan application [checked]
form incomplete
Loan application received
Application assessment required
Check application form completeness
form complete
Loan application [unchecked]
Loan application [checked]
Assess eligibility
Application assessed
applicant not eligible
Reject application
Loan application rejected
Loan application [assessed]
Prepare acceptance pack
applicant eligible
Acceptance pack
Credit history report
Risk assessment
Check credit history
Assess loan risk
Risk rules DB
Loan application [checked]
Appraise property
Property appraisal
Loan application
Insurance request received
Add insurance request to loan application
Loan application updated

Observe that in the "Assess application" sub-process, the loan application can have two possible states: "checked" or "unchecked". In order to use the loan application in any such state as input of activity "Add insurance request to loan application", we do not specify any state for this data object in the above model.

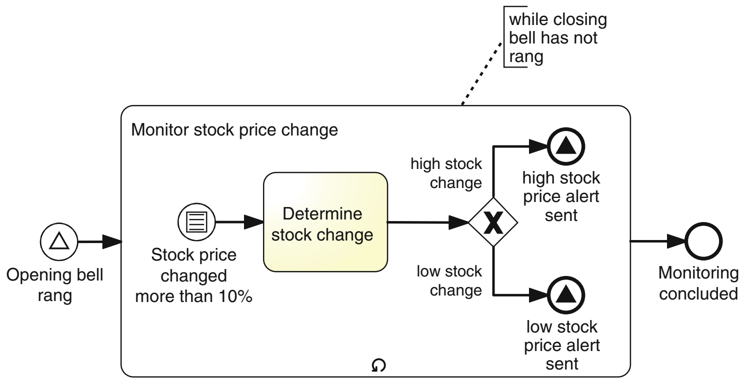
Solution 4.12



Solution 4.13



Solution 4.14


while closing bell has not rang
Monitor stock price change
Opening bell rang
Stock price changed more than 10%
Determine stock change
high stock change
low stock change
high stock price alert sent
low stock price alert sent
Monitoring concluded

Here, we did not use a boundary event to stop the sub-process for monitoring stock price changes since this way, the sub-process would only stop because of an exception. Rather, we used the loop condition to allow the sub-process to complete normally, i.e., without being interrupted.

4.7 Further Exercises

Exercise 4.15 Model the business process described in Exercise 3.15 (page 113) using a loop activity.

Exercise 4.16 Answer the following questions.

1. What is the limitation of using a loop activity to model repetition instead of using unstructured cycles?
2. What is the requirement for a sub-process to be used as a loop activity?
3. Model the procure-to-pay process described in Example 1.1 (page 3).

Hint. Use the model in Figure 1.6 (page 19) as a starting point for item (3).

Exercise 4.17 Model the following business process.

> Mail from the party is collected on a daily basis by the mail processing unit. Within this unit, the mail clerk sorts the unopened mail into the various business areas. The mail is then distributed. When the mail is received by the registry, it is opened and sorted into groups for distribution, and thus registered in a mail register. Afterwards, the assistant registry manager within the registry performs a quality check. If the mail is not compliant, a list of requisitions explaining the reasons for rejection is compiled and sent back to the party. Otherwise,

the matter details are captured and provided to the cashier, who takes the applicable fees attached to the mail. At this point, the assistant registry manager puts the receipt and copied documents into an envelope and posts it to the party. Meantime, the cashier captures the party details and prints the physical court file.

Exercise 4.18 Model the following process for selecting Nobel Prize laureates for chemistry.

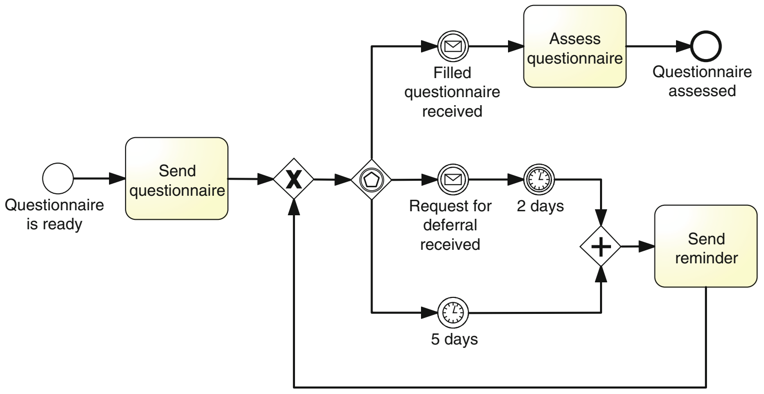
September: nomination forms are sent out. The Nobel committee sends out confidential forms to around 3,000 people—selected professors at universities around the world, Nobel laureates in physics and chemistry, and members of the Royal Swedish Academy of Sciences, among others.
February: deadline for submission. The completed nomination forms must reach the Nobel Committee no later than 31 January of the following year. The committee screens the nominations and selects the preliminary candidates. About 250–350 names are nominated as several nominators often submit the same name.
March–May: consultation with experts. The Nobel committee sends the list of the preliminary candidates to specially appointed experts for their assessment of the work of the candidates.
June–August: writing of the report. The Nobel committee puts together the report with recommendations to be submitted to the Academy. The report is signed by all members of the committee.
September: committee submits recommendations. The Nobel committee submits its report with recommendations on the final candidates to the members of the Academy. The report is discussed at two meetings of the chemistry section of the Academy.
October: Nobel laureates are chosen. In early October, the Academy selects the Nobel laureates in chemistry through a majority vote. The decision is final and without appeal. The names of the Nobel laureates are then announced.
December: Nobel laureates receive their prize. The Nobel Prize award ceremony takes place on 10 December in Stockholm, where the Nobel laureates receive their Nobel Prize, which consists of a Nobel medal, a diploma, and a document confirming the prize amount.

Acknowledgment This exercise is taken from "Nomination and Selection of Chemistry Laureates", Nobelprize.org. 9 Oct 2017 (http://www.nobelprize.org/nobel_prizes/chemistry/nomination).

Exercise 4.19

1. What is the difference between throwing and catching events?
2. What is the meaning of an event attached to an activity's boundary and what events can be attached to an activity's boundary?
3. What is the difference between the untyped end event and the terminate end event?

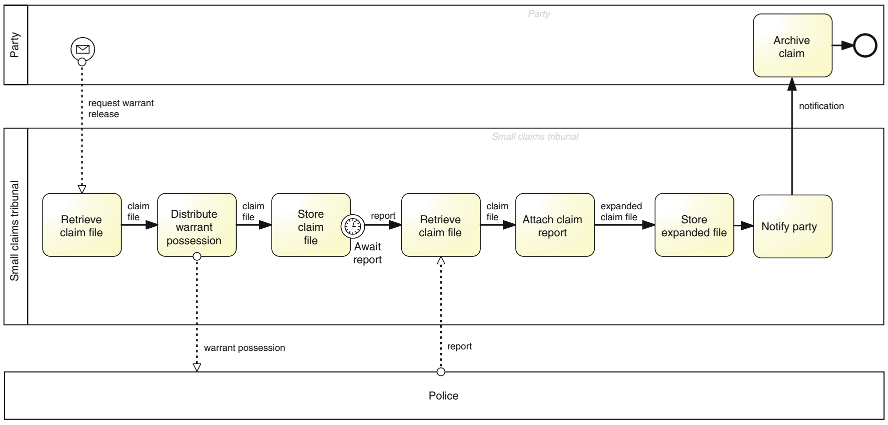
Exercise 4.20 What is wrong with the following model?



Exercise 4.21 Extend the billing process model in Exercise 4.5 (page 125) as follows.

> Any time after the first transaction has failed, the customer may pay the invoice directly to the ISP. If so, the billing process is interrupted and the payment is registered. This direct payment must also cover the late fees based on the number of days passed since Day 7 (the last day to avoid incurring late fees). If the direct payment does not include late fees, the ISP sends a notification to the customer that the fees will be charged in the next invoice, before concluding the process.

Exercise 4.22 What is wrong with the following model?



Exercise 4.23 Model the following business process at a supplier.

After a supplier notifies a retailer of the approval of a purchase order, the supplier can receive an order confirmation, an order change, or an order cancelation from the retailer. It may happen that no response is received at all. If no response is received after 48 h, or if an order cancelation is received, the supplier will cancel the order. If an order confirmation is received within 48 h, the supplier will process the order normally. If an order change is received within 48 h, the supplier will update the order and ask again the retailer for confirmation. The retailer is allowed to change an order at most three times. Afterwards, the supplier will automatically cancel the order.

Exercise 4.24 Revise the model in Exercise 3.12 (page 112) by using the terminate event.

Exercise 4.25 Model the following business process.

When a claim is received, it is first registered. After registration, the claim is classified leading to two possible outcomes: simple or complex. If the claim is simple, the insurance policy is checked. For complex claims, both the policy and the damage are checked independently. A possible outcome of the policy check is that the claim is invalid. In this case, any processing is canceled and a letter is sent to the customer. In the case of a complex claim, this implies that the damage checking is canceled if it has not been completed yet. After the check(s), an assessment is performed which may lead to two possible outcomes: positive or negative. If the assessment is positive, the garage is phoned to authorize the repairs and the payment is scheduled (in this order). In any case (whether the outcome is positive or negative), a letter is sent to the customer and the process ends. At any moment after the registration and before the end of the process, the customer may call to modify the details of the claim. If a modification occurs before the payment is scheduled, the claim is classified again (simple or complex) and the process is repeated. If a request to modify the claim is received after the payment is scheduled, the request is rejected.

Exercise 4.26 Model the following business process.

An order handling process starts when an order is received. The order is first registered. If the current date is not a working day, the process waits until the following working day before proceeding. Otherwise, an availability check is performed and a purchase order response is sent back to the customer. If none of the items is available, any processing related to the order must be stopped. Thereafter, the client needs to be notified that the purchase order cannot be further processed. Anytime during the process, the customer may send a purchase order cancel request. When such a request is received, the purchase order handling process is interrupted and the cancelation is processed. The customer may also send a "Customer address change request" during the order handling process. When such a request is received, it is just registered without further action.

Exercise 4.27 Model the order-to-cash process of the equipment rental company described in Exercise 2.14 (page 70). The process starts with the receipt of a purchase order, and ends when the payment of the invoice is received or the invoice is put into debt collection (the debt collection itself should be left out of scope).

Exercise 4.28 Draw a collaboration diagram for the following business process for electronic land development applications.

The Smart Electronic Development Assessment System (Smart eDA) is a Queensland Government initiative aimed to provide an intuitive service for preparing, lodging, and assessing land development applications. The land development business process starts with the receipt of a land development application from an applicant. Upon the receipt of a land development application, the assessment manager interacts with the cadastre to retrieve geographical information on the designated development area. This information is used to get an initial validation of the development proposal from the city council. If the plan is valid, the assessment manager sends the applicant a quote of the costs of processing the application. These costs depend on the type of development plan (for residential or commercial purposes) and on the permit or license that will be required for the plan to be approved. If the applicant accepts the quote, the assessment can start.

The assessment consists of a detailed analysis of the development plan. First, the assessment manager interacts with the Department of Main Roads (DMR) to check for conflicts with planned road development works. If there are conflicts, the application cannot proceed and must be rejected. In this case, the applicant is notified by the assessment manager. The applicant may wish to modify the development plan and resubmit it for assessment. In this case, the process is resumed from where it was interrupted.

If the development plan includes modifications to the natural environment, the assessment manager needs to request a land alteration permit to the Department of Natural Resources and Water (NRW). If the plan is for commercial purposes, additional fees will be applied to obtain this permit. Once the permit is granted, this is sent by NRW directly to the applicant. Likewise, if the designated development area is regulated by special environment protection laws, the assessment manager needs to request an environmental license to the Environmental Protection Agency (EPA). Similarly, once the license is granted, this is sent by EPA directly to the applicant. Once the required permit and/or license have been obtained, the assessment manager notifies the applicant of the final approval.

At any time during this process, the applicant can track the progress of their application by interacting directly with the assessment manager. Moreover, they can cancel the application should they wish to do so. In that case, all involved parties need to be notified and any license or permit needs to be revoked.

Assessment manager, cadastre, DMR, NRW, and EPA are all Queensland Government entities. In particular, NRW and EPA are part of the Department of Environment and Resource Management within the Queensland Government.

Exercise 4.29 Draw a collaboration diagram for the following business process for ordering maintenance activities at Sparks.

The ordering business process starts with the receipt of a request for work order from a customer. Upon the receipt of this request, the ordering department of Sparks estimates the expected usage of supplies, parts and labour and prepares a quote with the estimated total cost for the maintenance activity. If the customer's vehicle is insured, the ordering department interacts with the insurance department to retrieve the details of the customer's insurance plan so that these can be attached to the quote. The ordering department then sends the quote to the customer, who can either accept or reject the quote by notifying the ordering department within 5 days. If the customer accepts the quote, the ordering department contacts the warehouse department to check if the required parts are in stock before scheduling an appointment with the customer. If some parts are not in stock, the ordering department orders the required parts by interacting with a certified reseller and waits for an order confirmation from the reseller to be received within 3 days. If it is not received, the order department orders the parts again from a second reseller. If no reply is received from the second reseller too, the order department notifies the customer that the parts are not available and the process terminates. If the required parts are in stock or have been ordered, the ordering department interacts with an external garage to book a suitably-equipped service bay and a suitably-qualified mechanic to perform the work. A confirmation of the appointment is then sent by the garage to the order department which forwards the confirmation to the customer. The customer has one week to pay Sparks, otherwise the ordering department cancels the work order by sending a cancelation notice to both the service bay and the mechanic that have been booked for this order. If the customer pays in time, the work order is performed.

Exercise 4.30 Draw a collaboration diagram for the following business process at MetalWorks.

A build-to-order (BTO) process, also known as make-to-order process, is an order-to-cash process where the products to be sold are manufactured on the basis of a confirmed purchase order. In other words, the manufacturer does not maintain any ready-to-ship products in their stock. Instead, the products are manufactured on demand when the customer orders them. This approach is used in the context of customized products, such as metallurgical products, where customers often submit orders for products with very specific requirements.

We consider a BTO process at a company called MetalWorks. The process starts when MetalWorks receives a purchase order (PO) from one of its customers. This PO is called the "customer PO". The customer PO may contain one or multiple line items. Each line item refers to a different product.

Upon receiving a customer PO, a sales officer checks the PO to determine if all the line items in the order can be produced within the timeframes indicated in the PO. As a result of this check, the sales officer may either confirm the customer PO or ask the customer to revise the terms of the PO (for example: change the delivery date to a later date). In some extreme cases, the sales officer may reject the PO, but this happens very rarely. If the customer is asked to revise the PO, the BTO process will be put in "stand-by" until the customer submits a revised PO. The sales officer will then check the revised PO and accept it, reject it, or ask again the customer to make further changes. However, the sales officer has been instructed to accept changes to the PO up to three times, after which the PO must be escalated to a senior sales officer, who can either accept the further changes one more time, or reject the PO altogether.

Once a PO is confirmed, the sales officer creates one "work order" for each line item in the customer PO. In other words, one customer PO gives place to multiple work orders (one per line item). The work order is a document that allows employees at MetalWorks to keep track of the manufacturing of a product requested by a customer.

In order to manufacture a product, multiple raw materials are required. Some of these raw materials are maintained in stock in the warehouse of MetalWorks, but others need to be sourced from one or multiple suppliers. Accordingly, each work order is examined by a production engineer. The production engineer determines which raw materials are required in order to fulfill the work order. The production engineer annotates the work order with a list of required raw materials. Each raw material listed in the work order is later checked by a procurement officer. The procurement officer determines whether the required raw material is available in stock or it has to be ordered by accessing the specific catalog for that product line. If the material has to be ordered, the procurement officer consults the suppliers database, selects one or more suitable suppliers for the raw material and sends a request for quote to the selected suppliers. If more than one supplier is identified, the procurement officer selects the best quote out of the first three quotes received from the suppliers (the other quotes, if they arrive, are discarded), and emits a "material PO" for the selected supplier. This material PO is a PO for a raw material and is different from the customer PO. A material PO is a PO sent by MetalWorks to one of its suppliers, whereas a customer PO is a PO received by MetalWorks from one of its customers.

Once all materials required to fulfill a work order are available, the production can start. The responsibility for the production of a work order is assigned to the same production engineer who previously examined the work order. The production engineer is responsible for scheduling the production. Once the product has been manufactured, it is checked by a quality inspector. Sometimes, the quality inspector finds a defect in the product and reports it to the production engineer. The production engineer then decides whether: (i) the product should undergo a minor fix; or (ii) the product should be discarded and manufactured again.

Once the production has completed, the product is shipped to the customer. There is no need to wait until all the line items requested in a customer PO are ready before shipping them. As soon as a product is ready, it can be shipped to the corresponding customer.
At any point in time before the shipment of the product, the customer may send a "cancel order" message for a given PO. When this happens, the sales officer determines if the order can still be canceled, and if so, whether or not the customer should pay a penalty. If the order can be canceled without penalty, all the work related to that order is stopped and the customer is notified that the cancelation has been successful. If the customer needs to pay a penalty, the sales officer first asks the customer if they accept to pay the cancelation penalty. If the customer accepts to pay the cancelation penalty, the order is canceled and all work related to the order is stopped. Otherwise, the work related to the order continues.

Exercise 4.31 Draw a collaboration diagram for the following booking-to-cash process.

Fotof provides photography services in the fields of family photography, personal event photography (e.g. weddings and party photography) and commercial photography (e.g., corporate events). One of the core processes of Fotof is its booking-to-cash process, which goes all the way from the moment a customer makes a booking for a photo shooting session, through the order placement, to the moment the customer pays and obtains the ordered pictures. In the last year, Fotof received 10K orders from commercial customers, and 80K orders from private customers.
The booking-to-cash process starts when a customer makes a booking for a shooting session at a photo studio. A booking can be done via phone or via email addressed directly to a specific photo studio. The request is handled by a customer service representative at the photo studio. Each studio employs two customer service representatives: a senior one, who is also manager of the studio, and a junior one. The customer service representative enters the details of the booking into the photo studio information system.
The booking is assigned to one of the photographers of the studio. After a photo shooting session, the photographer uploads the pictures to a file server. Eventually, a technician cleans up the pictures by deleting duplicates and failed shots. Later the technician edits the remaining shots and arranges them into a photo gallery using a dedicated photo studio software tool. Once the gallery is completed, the customer is notified by email. The notification includes a URL of the gallery.
Customers can view the gallery, select the pictures they wish to order in print (and how many copies) and those they wish to get in digital copy (full resolution). Customers can also annotate a selected picture in order to ask for additional editing (special requests). When placing their order, customers can specify whether they will pickup the printed copies at the studio or have them delivered by post. In the latter case, a shipment fee is added to the order. Once the customer has submitted the order, a technician performs additional editing (if required by the customer). In the case of special requests, the technician may need to communicate with the customer by email or phone to clarify the request and to determine how to fulfill it, and whether the special request will entail an additional fee and how much. If printouts are required, the technician prints them out, puts them in an envelope, and drops them at the studio's counter.
Pictures from a shooting session are kept in the corresponding gallery for up to 30 days (a reminder is sent to the customer 5 days before the expiry date). If a customer has not placed an order past this period, an invoice is sent for the minimum billing amount (see below). Invoices are payable within 7 days of their issue. A customer service representative sends a reminder when an invoice is overdue.
Once the pictures are ready, a customer service representative determines the amount to be invoiced (including additional fees for special requests). The customer service representative then produces an invoice and sends it to the customer. Once the invoice has been paid, the customer service representative packs and sends the printouts for postal

delivery (if the customer ordered printouts) and sends a URL to the customer where the customer can find the full-resolution digital pictures they ordered. The matching of incoming payments to invoices is done automatically by an accountancy system (the same system that is used to issue invoices).

Booking or order cancelations can occur in three ways: (i) prior to the shooting session (booking cancelation); (ii) in case of no-show (the customer did not show up to the shooting session and did not reschedule it); or (iii) after the shooting, if the customer does not order any pictures within 30 days. Cancelations prior to the photo shooting session do not incur a fee. Cancelations due to no-shows do not attract a fee if they are in-studio; they attract a fee of € 50 if they are "on location". In case of a no-show, the customer may reschedule the booking to a later day but the no-show fee for on-location shootings is charged to the customer in any case. If a customer does not order any picture after a shooting session, the customer is invoiced a photo shooting fee of € 100 for in-studio sessions (€ 150 for on-location ones).

Exercise 4.32 Draw a collaboration diagram for the following mortgage application process at BestLoans.

The mortgage application process starts with the receipt of a mortgage application from a client. When an application is sent in by the client to the broker, the broker either examines the application, if the amount of the mortgage loan is within the mandate the broker has been given by BestLoans, or forwards the application to BestLoans.

If the application is examined by the broker, the broker must send either a rejection or an approval letter to the client within one week. If the broker sends an approval letter, then it forwards the details of this application to BestLoans so that from there on the client can interact directly with BestLoans for the sake of disbursing the loan. In this case, BestLoans registers the application and sends an acknowledgment to the client.

The broker can only handle a given number of clients at a time. If the broker is not able to reply within one week, the client must contact BestLoans directly. In this case, a reduction on the interest rate is applied should the application be approved.

If BestLoans deals with the application directly, its mortgage department checks the credit of the client with the Bureau of Credit Registration. Moreover, if the loan amount is more than 90% of the total cost of the house being purchased by the client, the mortgage department must request a mortgage insurance offer from the insurance department. After these interactions BestLoans either sends an approval letter or a rejection to the broker, which the broker then forwards to the client (this interaction may also happen directly between the mortgage department and the client if no broker is involved).

After an approval letter has been submitted to the client, the client may either accept or reject the offer by notifying this directly to the mortgage department. If the mortgage department receives an acceptance notification, it writes a deed and sends it to an external notary for signature. The notary sends a copy of the signed deed to the mortgage department. Next, the insurance department starts an insurance contract for the mortgage. Finally, the mortgage department submits a disbursement request to the financial department. When this request has been handled, the financial department notifies the client directly.

Any time during the application process, the client may inquire about the status of the application with the mortgage department or with the broker, depending on which entity is dealing with the client. Moreover, the client may request the cancelation of the application. In this case the mortgage department or the broker computes the application processing fees, which depend on how far the application process is, and communicates these to the client. The client may reply within 2 days with a cancelation confirmation, in which case the process is canceled, or with a cancelation withdrawal, in which case the process continues. If the process has to be canceled, BestLoans may need to first recall the loan (if the disbursement has been done), then annul the insurance contract (if an insurance contract has been drawn), and finally annul the deed (if a deed has been drawn).

4.8 Further Readings

We have seen single-pool business process diagrams as well as multi-pool collaboration diagrams. There are two other types of diagrams in BPMN, namely *choreography diagrams* and *conversation diagrams*. Choreography diagrams allow us to capture interactions between parties (as opposed to tasks) and the order of these interactions. Conversation diagrams allow us to capture interactions only, without any ordering. Choreography and conversation diagrams in BPMN originate from a language for modeling Web service interactions called Let's Dance [197].

For further information on the BPMN language we point to the BPMN website.[1] This site also provides a link to handy BPMN material including a quick reference guide to all BPMN elements and a comprehensive list of books on the subject. Among the many books dedicated to BPMN, we can cite those by Silver [163], Allweyer [7], and Freund & Rücker [49]. A compact BPMN poster, available in 15 languages can be downloaded from the BPM Offensive Berlin website.[2]

[1] http://www.bpmn.org.

[2] http://www.bpmb.de/index.php/BPMNPoster.