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 learn how to model complex business processes with BPMN. The constructs presented in this chapter build on top of the knowledge acquired in Chap. 3. In particular, we will expand on activities, events and gateways. We will learn how to use activities to model sub-processes and how to reuse these sub-processes across different processes. We will also extend activities to model more sophisticated forms of rework and repetition. As per events, we will expand on message events, present temporal events and show how race conditions can be modeled among these event types via a new type of gateway. We will also learn how to use events to handle business process exceptions. Finally, we will show how a collaboration diagram can be abstracted into a choreography diagram that only focuses on the interactions between the involved business parties.

4.1 Process Decomposition

When capturing complex business processes, the resulting process model may be too large to be understood at once. Take the order fulfillment process model in Fig. 3.12. While the scenario at hand is still relatively simple, this model already contains 14 activities, six gateways and two events. As we add data objects and message flows, the model gets larger and so harder to understand (see e.g. Fig. 3.16). To improve its readability, we can simplify the process by hiding certain parts within a *sub-process*. A sub-process represents a self-contained, composite activity that can be broken down into smaller units of work. Conversely, an atomic activity, also called *task*, is an activity capturing a unit of work that cannot be further broken down.

In order to use a sub-process, we first need to identify groups of related activities, i.e. those activities which together achieve a particular goal or generate a particular outcome in the process model under analysis. In our order fulfillment example, we can see that the activities "Check raw materials availability" and "Purchase raw

M. Dumas et al., *Fundamentals of Business Process Management*,
DOI 10.1007/978-3-642-33143-5_4, © Springer-Verlag Berlin Heidelberg 2013

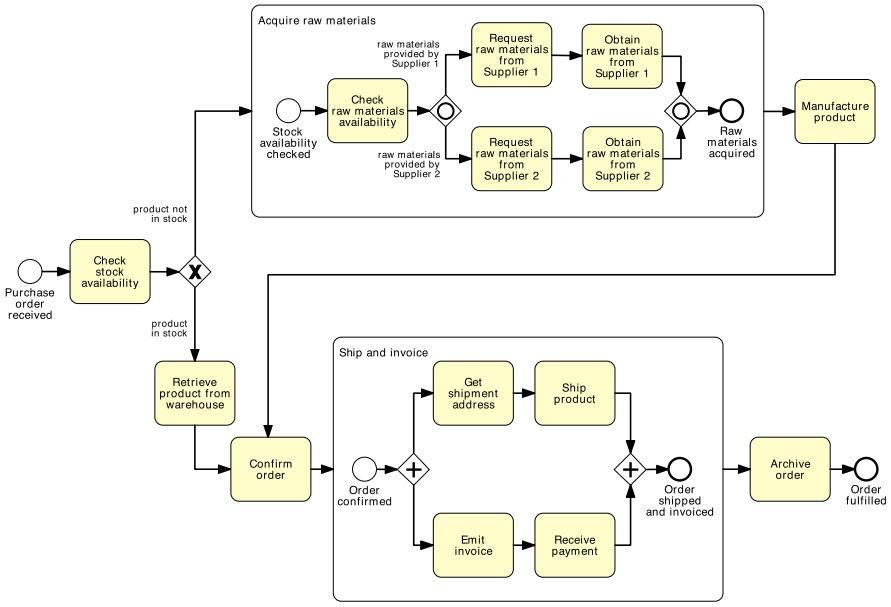


Fig. 4.1 Identifying sub-processes in the order fulfillment process of Fig. 3.12

materials from Supplier 1(2)", lead together to the acquisition of raw materials. Thus these activities, and their connecting gateways, can be encapsulated in a sub-process. In other words, they can be seen as the internal steps of a macro-activity called "Acquire raw materials". Similarly, the two parallel branches for shipping and invoicing the order can be grouped under another sub-process activity called "Ship and invoice". Figure 4.1 illustrates the resulting model, where the above activities have been enclosed in two sub-process activities. We represent such activities with a large rounded box which encloses the internal steps. As we can observe from Fig. 4.1, we also added a start event and an end event inside each sub-process activity, to explicitly indicate when the sub-process starts and completes.

Recall that our initial objective was to simplify a process model. Once we have identified the boundaries of the sub-processes, we can simplify the model by hiding the content of its sub-processes, as shown in Fig. 4.2. This is done by replacing the macro-activity representing the sub-process with a standard-size activity. We indicate that this activity hides a sub-process by marking it with a small square with a plus sign (+) inside (like if we could expand the content of that activity by pressing the plus button). This operation is called collapsing a sub-process. By collapsing a sub-process we reduce the total number of activities (the order fulfillment process has only six activities now), thus improving the model readability. In BPMN, a sub-process which hides its internal steps is called *collapsed sub-process*, as opposed to an *expanded sub-process* which shows its internal steps (as in Fig. 4.1).

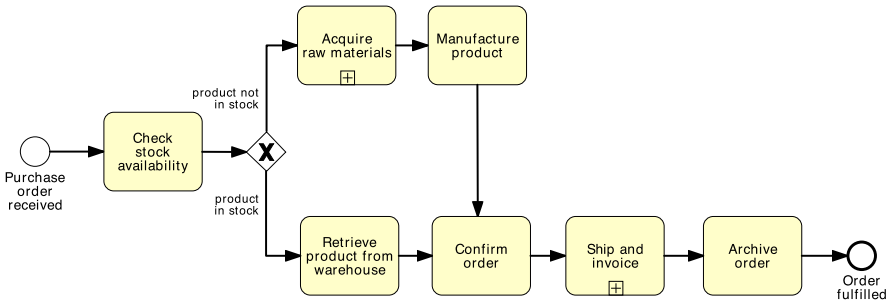


Fig. 4.2 A simplified version of the order fulfillment process after hiding the content of its sub-processes

Exercise 4.1 Identify suitable sub-processes in the process for assessing loan applications modeled in Exercise 3.5.

Hint Use the building blocks that you created throughout Exercises 3.1–3.4.

Collapsing a sub-process does not imply losing its content. The sub-process is still there, just defined at an abstraction level below. In fact, we can nest sub-processes in multiple levels, so as to decompose a process model hierarchically. An example is shown in Fig. 4.3, which models a business process for disbursing home loans. In the first level we identified two sub-processes: one for checking the applicant's liability, the other for signing the loan. In the second level, we factored out the scheduling of the loan disbursement within the process for signing loans into a separate sub-process.

As we go down the hierarchical decomposition of a process model, we can add more details. For example, we may establish a convention that at the top level we only model core business activities, at the second level we add decision points, and so on all the way down to modeling exceptions and details that are only relevant for process automation.

Question When should we decompose a process model into sub-processes?

We should use sub-processes whenever a model becomes too large that is hard to understand. While it is hard to precisely define when a process model is "too large", since understandability is subjective, it has been shown that using more than approximately 30 flow objects (i.e. activities, events, gateways) leads to an increased probability of making mistakes in a process model (e.g. introducing behavioral issues). Thus, we suggest to use as few elements as possible per each process model level, and in particular to decompose a process model if this has more than 30 flow objects.

Reducing the size of a process model, for example by collapsing its sub-processes, is one of the most effective ways of improving a process model's readability. Other structural aspects that affect the readability include the density of the

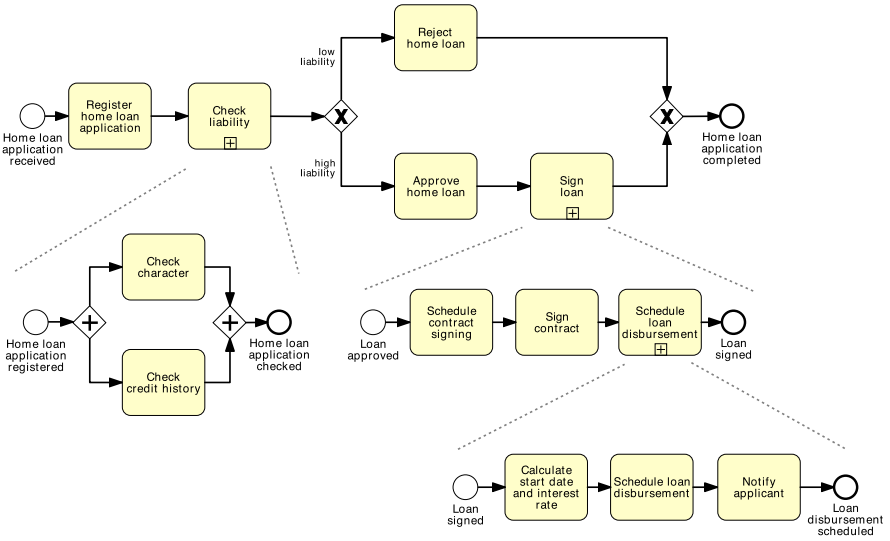


Fig. 4.3 A process model for disbursing home loans, laid down over three hierarchical levels via the use of sub-processes

process model connections, the number of parallel branches, the longest path from a start to an end event, as well as cosmetic aspects such as the layout, the labels style (e.g. always use a verb-noun style), the colors palette, the lines thickness, etc. More information on establishing process modeling guidelines can be found in Chap. 5.

We have shown that we can simplify a process model by first identifying the content of a sub-process, and then hiding this content by collapsing the sub-process activity. Sometimes, we may wish to proceed in the opposite direction, meaning that when modeling a process we already identify activities that can be broken down in smaller steps, but we intentionally under-specify their content. In other words, we do not link the sub-process activity to a process model at a lower level capturing its content (like if by pressing the plus button nothing would happen). The reason for doing this is to tell the reader that some activities are made up of sub-steps, but that disclosing the details of these is not relevant. This could be the case of activity "Ship product" in the order fulfillment example, for which modeling the distinction between its internal steps for packaging and for shipping is not relevant.

4.2 Process Reuse

By default a sub-process is embedded within its parent process model, and as such it can only be invoked from within that process model. Often, when modeling a business process we may need to reuse parts of other process models of the same organization. For example, a loan provider may reuse the sub-process for signing

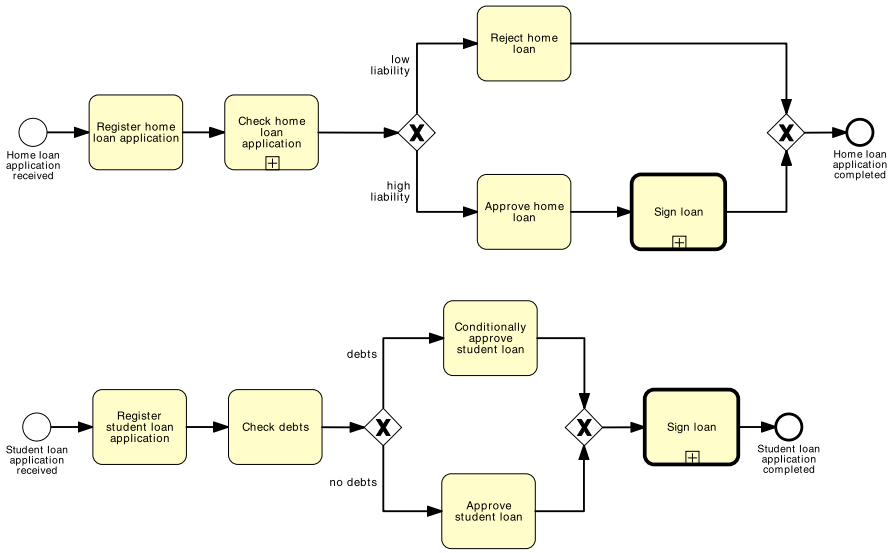


Fig. 4.4 The process model for disbursing student loans invokes the same model for signing loans used by the process for disbursing home loans, via a call activity

loans contained in the home loan disbursement for other types of loan, such as a process for disbursing student loans or motor loans.

In BPMN, we can define the content of a sub-process outside its parent process, by defining the sub-process as a *global* process model. A global process model is a process model that is not embedded within any process model, and as such can be invoked by other process models within the same process model collection. To indicate that the sub-process being invoked is a global process model, we use the collapsed sub-process activity with a thicker border (this activity type is called *call activity* in BPMN). Coming back to the loan disbursement example of Fig. 4.3, we can factor out the sub-process for signing loans and define it as a global process model, so that it can also be invoked by a process model for disbursing student loans (see Fig. 4.4).

Question Embedded or global sub-process?

Our default choice should be to define sub-processes as global process models so as to maximize their reusability within our process model collection. Supporting processes such as payment, invoicing, HR, printing, are good candidates for being defined as global process models, since they are typically shared by various business processes within an organization. Besides reusability, another advantage of using global process models is that any change made to these models will be automatically propagated to all process models that invoke them. In some cases, however, we may want to keep the changes internal to a specific process. For example, an invoicing process used for corporate orders settlement would typically be different

from the invoicing process for private orders. In this case, we should keep two model variants of the invoice sub-process, each embedded within its parent process model: corporate and private order settlement.

Example 4.1 Let us consider the procurement process of a pharmaceutical company.

> A pharmaceutical company has different business units within its manufacturing department, each producing a specific type of medicine. For example, there is a business unit looking after inhaled medications, and another one producing vaccines. The various business units make use of a direct procurement process for ordering chemicals, and of an indirect procurement process for ordering spare parts for their equipment.

The direct procurement process depends on the raw materials that are required to produce a specific type of medicine. For example, vaccines typically include adjuvants that help improve the vaccine's effectiveness, which are not contained in inhaled medications. Similarly, inhaled medications contain a chemical propellant to push the medicine out of the inhaler, which is not required for vaccines. Since this procurement process is specific to each business unit, we need to model it as an embedded sub-process within the manufacturing process model of each unit. On the other hand, the process for ordering spare parts to the equipment for synthesizing chemicals can be shared across all units, since all units make use of the same equipment. Thus, we will model this process with a global process model.

Before concluding our discussion on sub-processes, we need to point to some syntactical rules for using this element. A sub-process is a regular process model. It should start with at least one start event, and complete with at least one end event. If there are multiple start events, the sub-process will be triggered by the first such an event that occurs. If there are multiple end events, the sub-process will return control to its parent process only when each token flowing in this model reaches an end event. Moreover, we cannot cross the boundary of a sub-process with a sequence flow. To pass control to a sub-process, or receive control from a sub-process, we should always use start and end events. On the other hand, message flows can cross the boundaries of a sub-process to indicate messages that emanate from, or are directed to, internal activities or events of the sub-process.

Exercise 4.2 Identify suitable sub-processes in the business process of Exercise 1.7. Among these sub-processes, identify those that are specific to this business process versus those that can potentially be shared with other business processes of the same company.

4.3 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

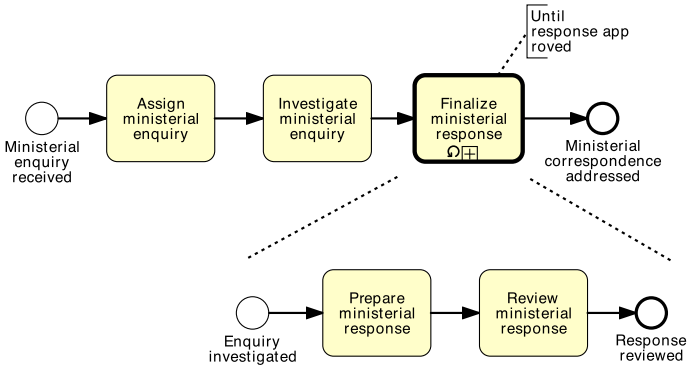


Fig. 4.5 The process model for addressing ministerial correspondence of Fig. 3.13 simplified using a loop activity

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 Fig. 4.5. 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 so, 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.

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 above. Sometimes there might be more than one entry and/or exit point, or the entry/exit point might be inside the repetition block. Consider for example the model in Fig. 4.6. Here the repetition block is made up of activities "Assess application", "Notify rejection" and "Receive customer feedback"; the cycle has an entry point and two exit points, of which one inside the repetition block. When an unstructured cycle has multiple exit points, like in this case, a loop activity cannot be used, unless additional conditions are used to specify the situations in which the cycle can be exited, which will render the model more complex.

Exercise 4.3

1. Identify the entry and exit points that delimit the unstructured cycles in the process models shown in Solution 3.4 and in Exercise 3.9. What are the repetition blocks?

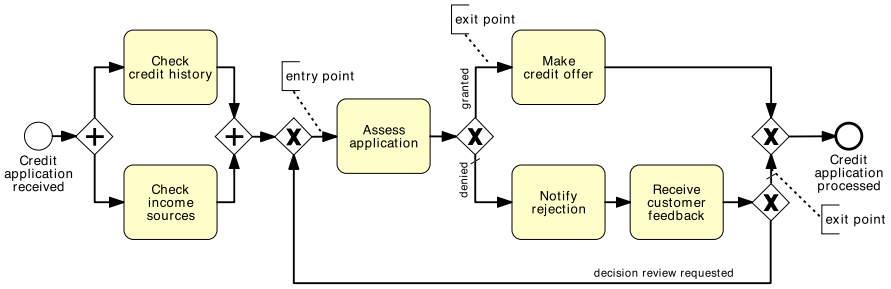


Fig. 4.6 An example of unstructured cycle

2. Model the business process of Solution 3.4 using a loop activity.

4.3.1 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 4.2 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 five preferred suppliers exist. Then we can use an AND-split to model five tasks in parallel, each to obtain a quote from one supplier, as shown in Fig. 4.7. However, there are several problems with this solution. First, the larger the number of suppliers, the larger the resulting model will be, since we need to use 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 obviate these problems, BPMN provides a construct called *multi-instance* activity. A multi-instance activity indicates an activity (being it a task or a sub-process) that is executed multiple times concurrently. Such a construct is useful when the same activity needs to be executed for multiple entities or data items, like 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.

A multi-instance activity is depicted as an activity marked with three small vertical lines at the bottom. Figure 4.8 shows a revised version of the procurement process model in Fig. 4.7. 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.

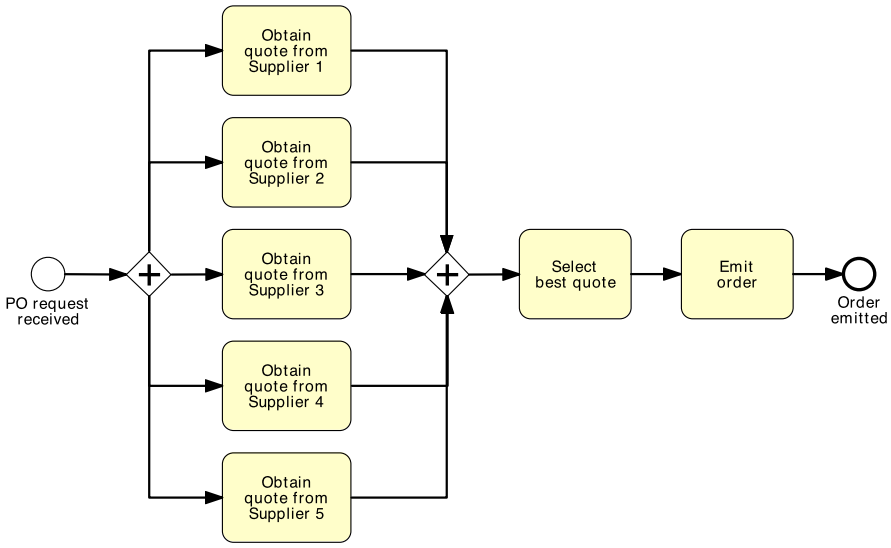


Fig. 4.7 Obtaining quotes from five suppliers

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 would 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 reply back to our request for 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 five quotes obtained", as shown in Fig. 4.8). 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 fulfillment example in Fig. 4.2, 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

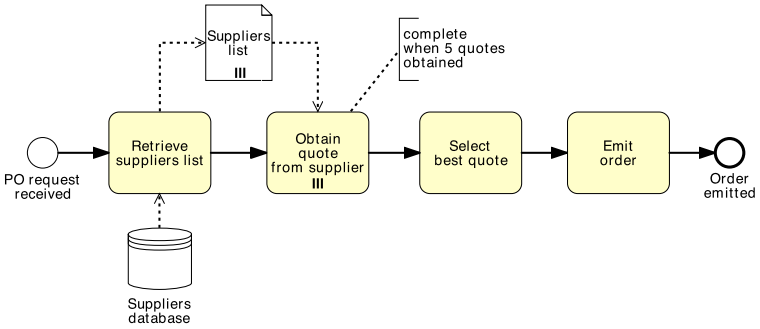


Fig. 4.8 Obtaining quotes from multiple suppliers, whose number is not known a priori

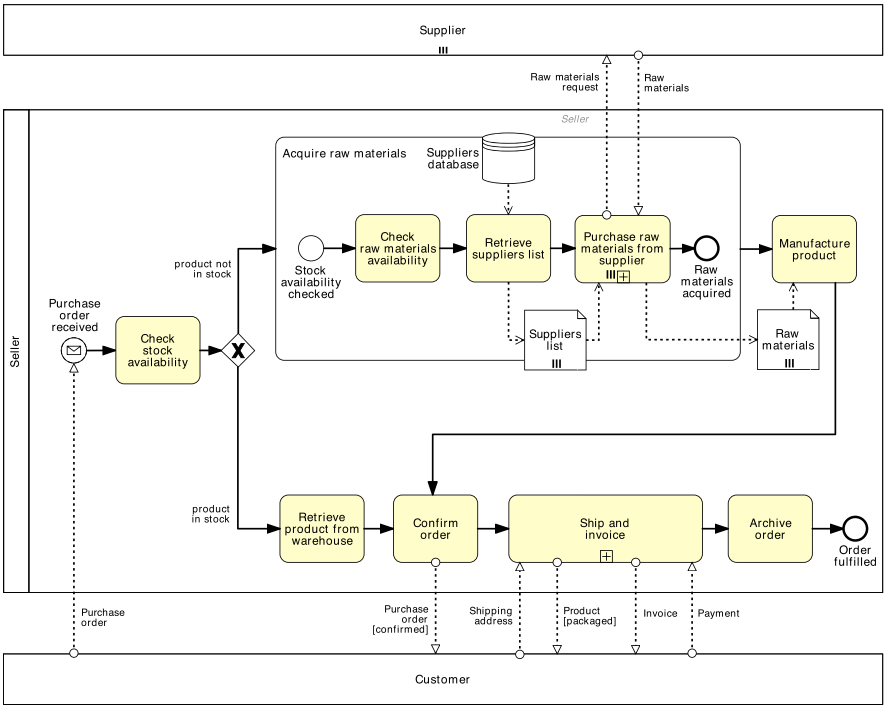


Fig. 4.9 Using a multi-instance pool to represent multiple suppliers

on the fly from a suppliers database (the updated model is shown in Fig. 4.9). By the same principle, we replace the two pools "Supplier 1" and "Supplier 2" with a single pool, namely "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 Sect. 3.4 is not enforced.

Exercise 4.4 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.

4.3.2 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 fulfillment process needs to inquire about the progress of their order. The customer may do so simply by sending an e-mail 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 their 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 4.10 shows the example of the customer's 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 4.5 Model the following process snippet.

A typical army recruitment process starts by shortlisting all candidates' applications. Those shortlisted are then called to sit 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 hours. The candidates that pass all tests are asked to sit a mental exam and a physical exam, followed by an interview. Only those that also pass these two exams and perform well in the interview can be recruited in the army.

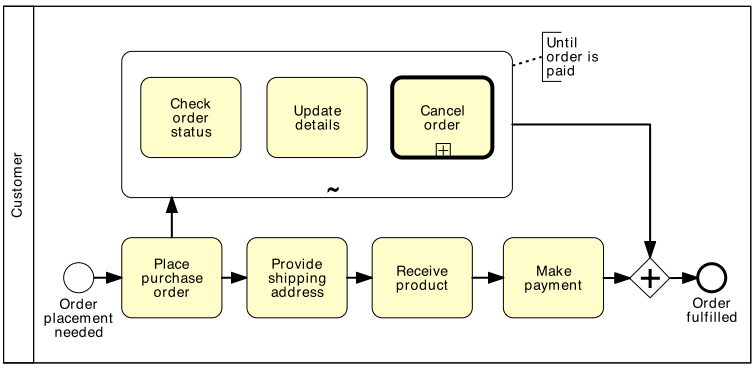


Fig. 4.10 Using an ad-hoc sub-process to model uncontrolled repetition

4.4 Handling Events

As we pointed out in Chap. 3, 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, for example 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. Then the token traverses the event instantaneously, i.e. events cannot retain tokens. An intermediate event is represented as a circle with a double border.

4.4.1 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 (cf. Fig. 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 event 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 just 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 applications" in Fig. 4.11a, which is an extract of the loan assessment model of Solution 3.7. 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 Fig. 4.11b, since these activities

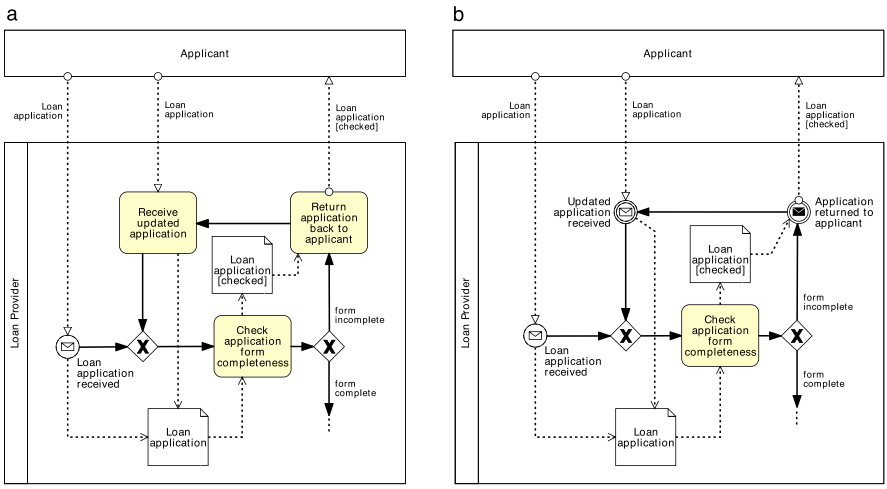


Fig. 4.11 Replacing activities that only send or receive messages (**a**) with message events (**b**)

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 darkened.

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 performed.

Question Typed or untyped event?

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

Exercise 4.6 Is there any other activity in the loan assessment model of Solution 3.7 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*

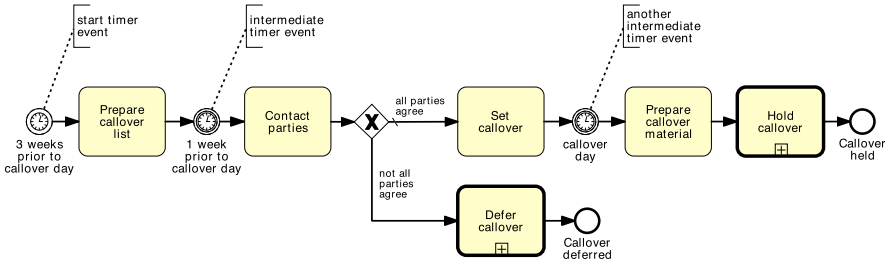


Fig. 4.12 Using timer events to drive the various activities of a business process

event, i.e. an event that throws a trigger from within the process. An intermediate message event has both flavors since it can be used both as a catching event (the message is received from another pool) or as a throwing event (the message is sent to another pool).

4.4.2 Temporal Events

Besides the message event, there are other triggers that can be specified for a start event. One worth of notice 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 7am.

A timer event may also be used as intermediate event, to model a temporal interval that needs to elapse before the process instance can proceed. To indicate a timer event, we mark the event symbol with a light 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.

Example 4.3 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 Fig. 4.12. 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*, since they cannot retain tokens as opposed to activities, which retain tokens for the duration of their execution (recall that activities consume time).

Exercise 4.7 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 their 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 on Day 30 the payment is still outstanding, the account is closed and a disconnection fee is applied. A debt-recovery procedure is then started.

4.4.3 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 by an empty pentagon enclosed in a double-line circle. Figure 4.13 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 that we saw in Chap. 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

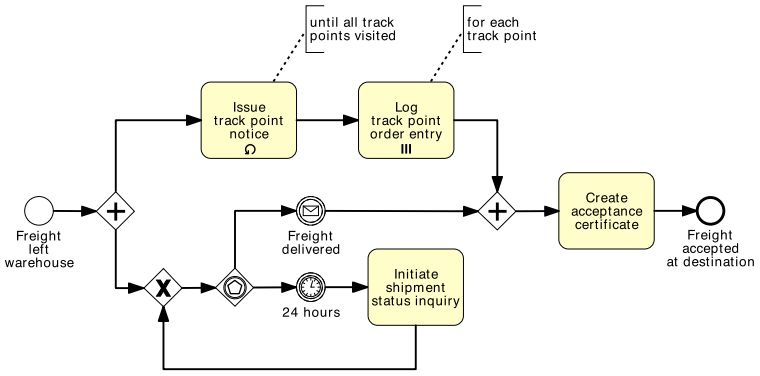


Fig. 4.13 A race condition between an incoming message and a timer

by the process environment.[1] 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 *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 4.8 Model the following process.

> A restaurant chain submits a purchase order (PO) to replenish its warehouses every Thursday. The restaurant chain's procurement system expects to receive 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 split can be used as the counterpart of an internal decision on a collaborating party. For example, a choice made from within the Client pool to send either an acceptance message or a rejection message to an Insurer, needs to be matched by an event-driven decision on the insurer pool to *react* to the choice made by the client. This example is illustrated in Fig. 4.14.

Event-based gateways can be used to avoid behavioral anomalies in the communication between pools. Take for example the collaboration diagram between the auctioning service and the seller in Fig. 4.15. This collaboration may deadlock if the seller is already registered, as this party will wait for the account creation request message which in that case will 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 Fig. 4.16.

[1] Specifically, the XOR-split of Chap. 3 is called *data-based XOR-split* since the branch to take is based on the evaluation of two or more conditions on data that are produced by a decision activity.

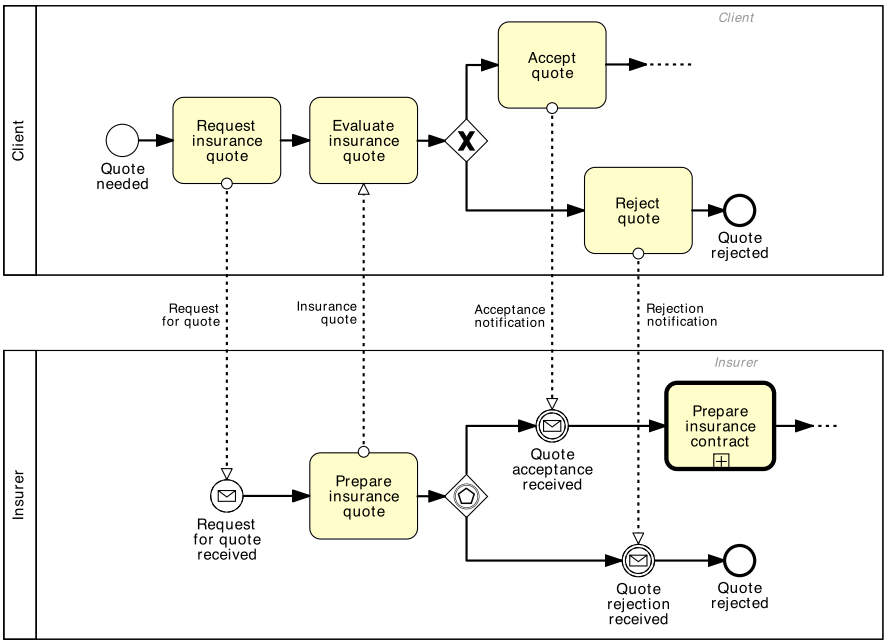


Fig. 4.14 Matching an internal choice in one party with an event-based choice in the other party

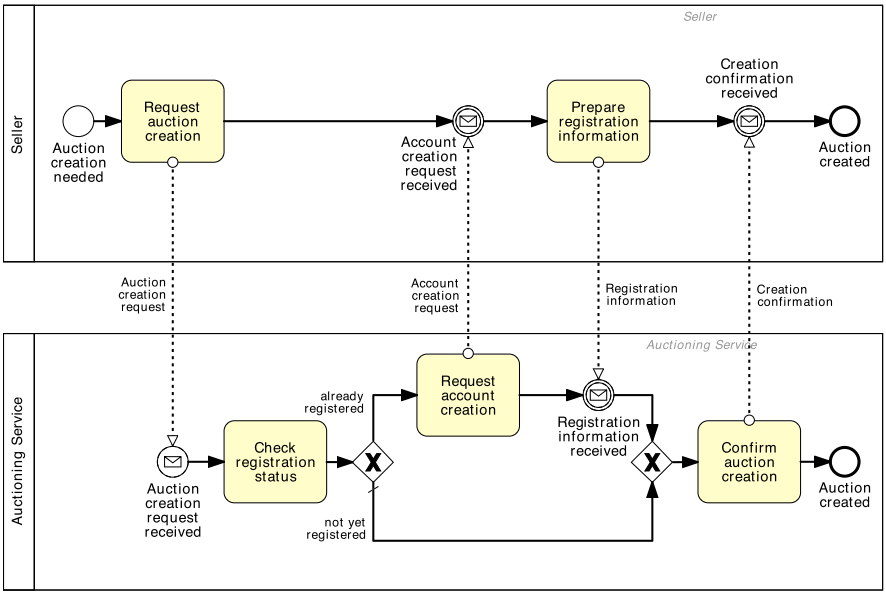


Fig. 4.15 An example of deadlocking collaboration between two pools

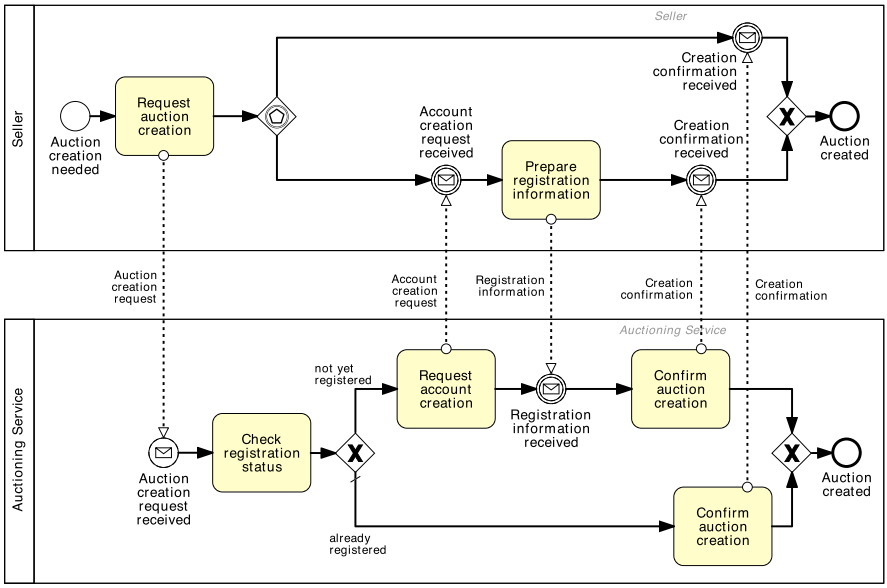


Fig. 4.16 Using an event-based gateway to fix the deadlocking collaboration of Fig. 4.15

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 in one party needs to be matched by an event-based decision in 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.

Exercise 4.9 Fix the collaboration diagram in Fig. 4.17.

Acknowledgement This exercise is partly inspired by: *Niels Lohmann: "Correcting Deadlocking Service Choreographies Using a Simulation-Based Graph Edit Distance". LNCS 5240, Springer, 2008.*

4.5 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. These "rainy-day" situations 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* like an exception due to an out-of-stock or discontinued product, and *technology faults* like a database crash, a network outage or a program logic violation. They deviate the normal process course since they cause the interruption

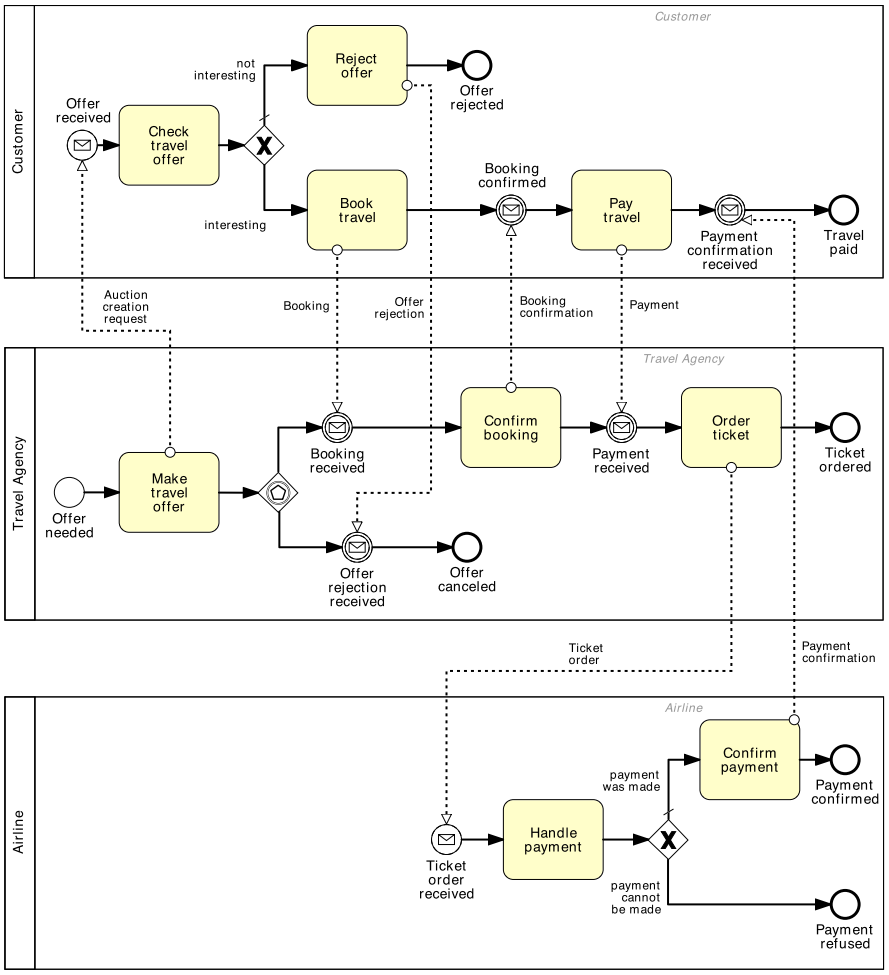


Fig. 4.17 A collaboration diagram between a client, a travel agency and an airline

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.

4.5.1 Process Abortion

The simplest way of handling an exception is to abort the running process and signal an improper process termination. This can be done by using an *end terminate event*, as shown in Fig. 4.18. An end terminate event (depicted as an end event marked

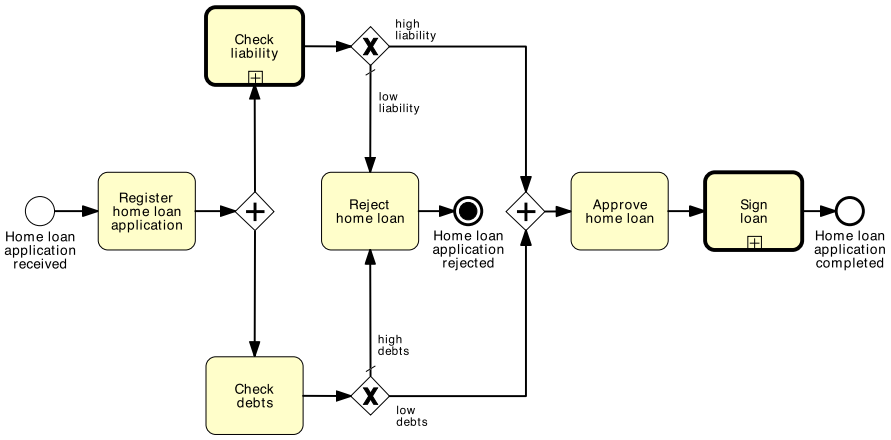


Fig. 4.18 Using a terminate event to signal improper process termination

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 Fig. 4.18—a variant of the home loan that we already saw in Fig. 4.3—a home loan is rejected and the process is aborted if the applicant has debts and/or low liability. From a token semantics, 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 debts or low liability and no 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 4.10 Revise the examples presented so far in this chapter, by using the terminate event appropriately.

4.5.2 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 scenario. 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*.

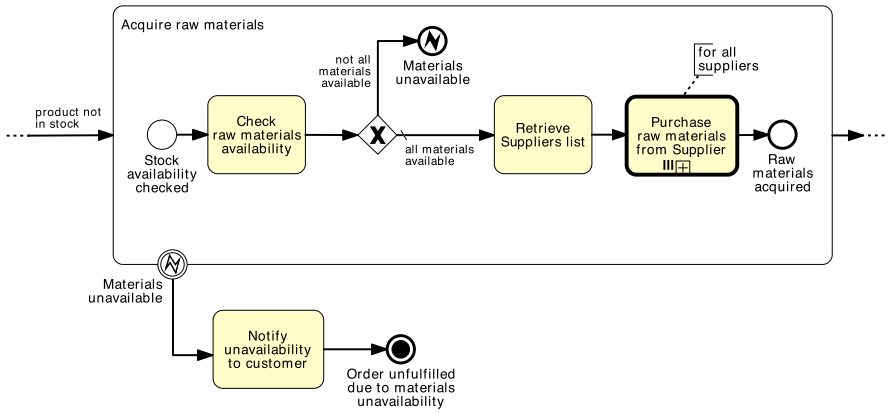


Fig. 4.19 Error events model internal exceptions

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 Fig. 4.19 in the context of our order fulfillment process. If there is an out of stock exception, the acquisition of raw materials is interrupted and the 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 through 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.

4.5.3 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 cancellation from the customer. Upon this request, the Seller should interrupt the stock availability check and handle the order cancellation. 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 Fig. 4.20. From a token semantics, when the intermediate message event is triggered, the token is removed from the enclosing activity, consequently causing the activity interruption, and sent through the exception flow emanating from the boundary event, to perform the recovery procedure.

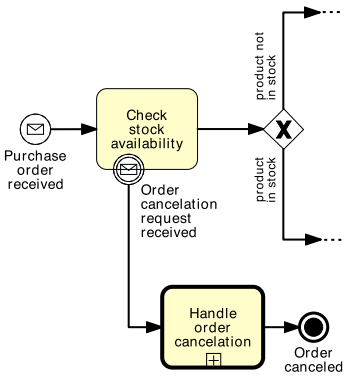


Fig. 4.20 Boundary events catch external events that can occur during an activity

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 fulfillment example, order cancellation 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 sub-process's boundary.

Exercise 4.11 Model the following routine for logging into an Internet bank account.

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.

4.5.4 Activity Timeouts

Another type of exception is that provoked by 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 hours), we can attach an intermediate timer event to the boundary of the activity: the timer is activated when the enclosing activity starts, and if it fires before the activity completes, provokes the activity's interruption. In other words, a timer event works as a timeout when attached to an activity's boundary.

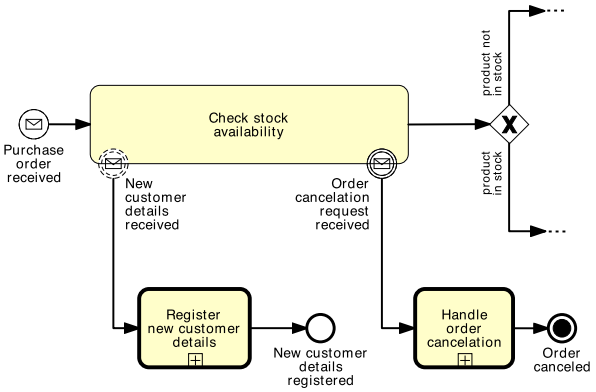


Fig. 4.21 Non-interrupting boundary events catch external events that occur during an activity, and trigger a parallel procedure without interrupting the enclosing activity

Exercise 4.12 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 four days from the receipt of the order. This implies that transportation quotes have to be prepared within 48 hours from the receipt of the order to remain within the terms of the contract.

4.5.5 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 fulfillment process, the customer may send a request to update their details during the stock availability check. The details should be updated in the customer database without interrupting the stock check. In order to denote that the boundary event is *non-interrupting*, we use a dashed double border, as shown in Fig. 4.21.

Exercise 4.13 Extend the process for assessing loan applications of Solution 3.7 as follows.

> An applicant who has decided not to combine their loan with a home insurance plan may change their 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 Fig. 4.19 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

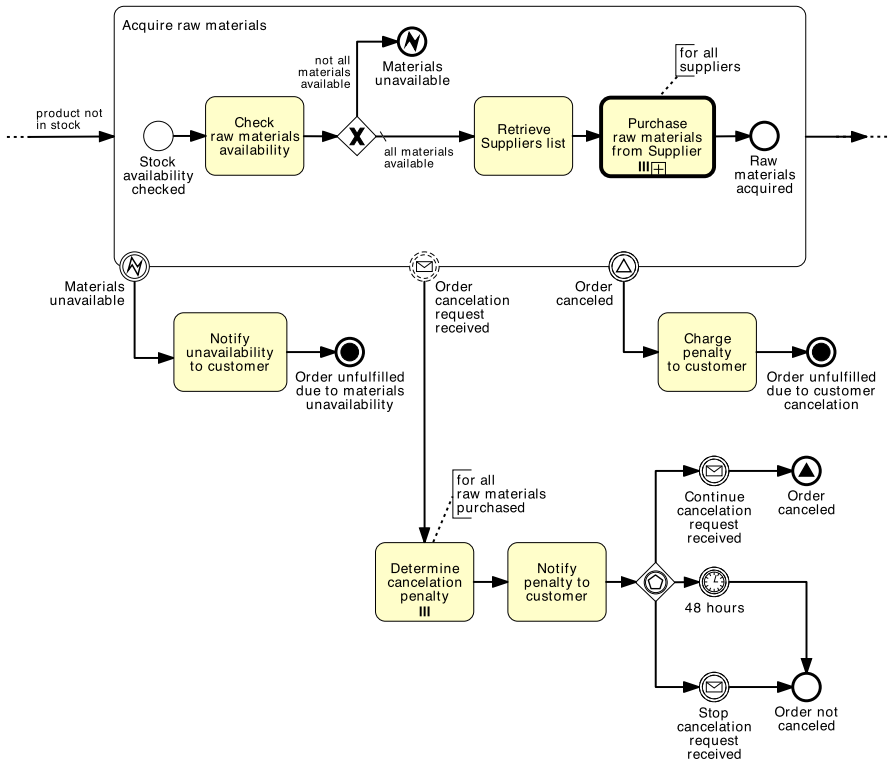


Fig. 4.22 Non-interrupting events can be used in combination with signal events to model complex exception handling scenarios

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 hours to either stop the cancellation, in which case nothing is done, or go on with it (see Fig. 4.22). In the latter case, we throw an end *signal event*. This event, depicted with a triangle marker, broadcasts a signal defined by the event's label, which can be caught by all catching signal events bearing the 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 from 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.

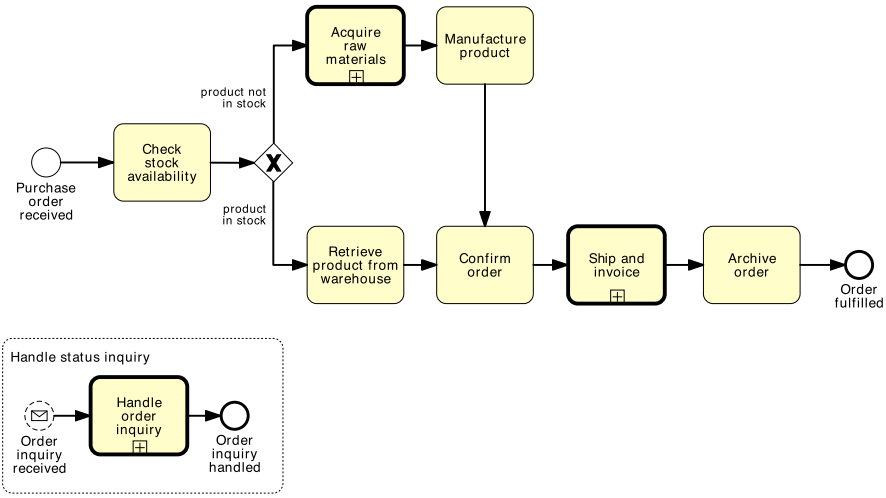


Fig. 4.23 Event sub-processes can be used in place of boundary events, and to catch events thrown from outside the scope of a particular sub-process

4.5.6 Interlude: 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 fulfillment 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 Fig. 4.23.

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. Similar 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 to use event sub-processes when the event that needs to be handled may occur during the entire 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 4.14 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. 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 re-submit the expense report from scratch.

4.5.7 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.

Let us delve into the sub-process for shipment and invoice handling of the order fulfillment example and assume that also this activity can be interrupted upon the receipt of an order cancellation request (see Fig. 4.24). After communicating the cancellation 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 either 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

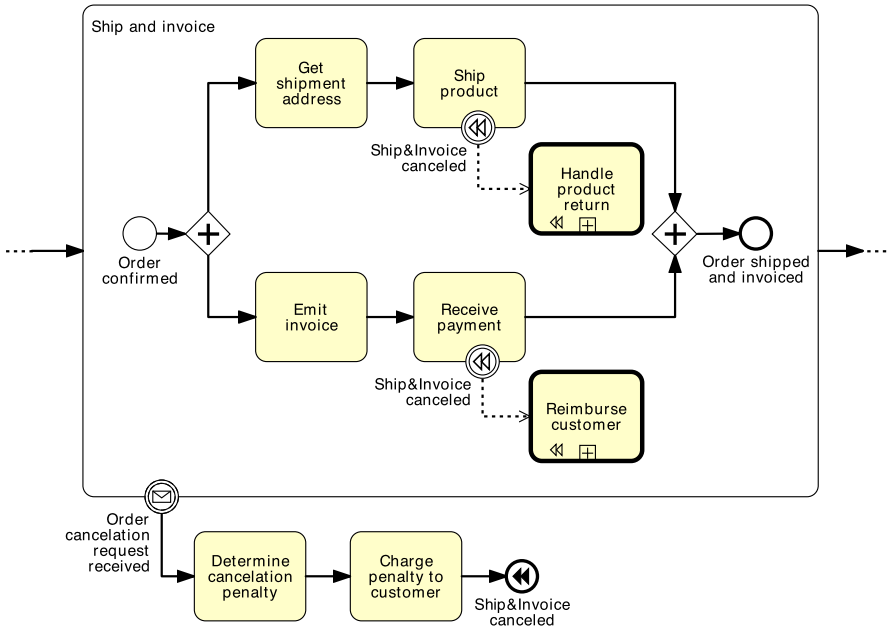


Fig. 4.24 Compensating for the shipment and for the payment

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, and 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.

In this section we have seen various ways to handle exceptions in business process, from simple process abortion to complex exception handling. Before adding exceptions it is important to understand the sunny-day scenario well. So start by modeling that. 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.

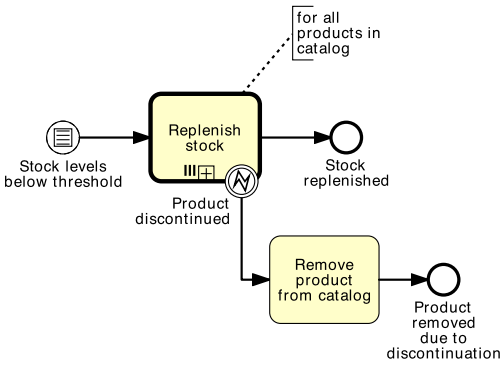


Fig. 4.25 A replenishment order is triggered every time the stock levels drop below a threshold

Exercise 4.15 Modify the model that you created in Exercise 4.14 as follows.

If the report is not handled within 30 days, the process is stopped, the employee receives a cancellation notice email and must re-submit 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 cancellation notice email.

4.6 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 3.5 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 attached to an activity's boundary. An example of conditional event is shown in Fig. 4.25.

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 Fig. 4.25, 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.

Exercise 4.16 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.

4.7 Process Choreographies

Sometimes it might be hard to frame a business collaboration between two or more parties, e.g. two organizations, by working directly at the level of the collaboration diagram. First, the collaboration diagram is typically too low-level and if the terms of the interactions between the two parties are not clear yet, it might be confusing to mix communication aspects with internal activities. Second, a party may not be willing to expose their internal activities to other parties (e.g. the logic behind a claim approval should remain private). Thus, it might be opportune to first focus on the interactions that have to occur among all involved parties, and on the order in which these interactions can take place. In BPMN, this information is captured by a *choreography diagram*. A choreography diagram is the process model of the interactions occurring between two or more parties. This high-level view on a collaboration acts as a contract among all involved parties. Once this contract has been crafted, each party can take it and refine it into their private processes, or alternatively, all parties can work together to refine the choreography into a collaboration diagram.

Figure 4.26 shows the choreography for the order fulfillment collaboration of Fig. 4.9. As we can see, a choreography is indeed a process model: it is started by one or more start events and concluded by one or more end events, activities are connected via sequence flows and gateways are used for branching and merging. The key characteristic is, however, that an activity represents an *interaction* between two parties, rather than a unit of work. An interaction can be one-way (one message is exchanged) or *two-way* (a message is followed by a return message in the opposite direction). Each interaction has an *initiator* or sender (the party sending the message), and a *recipient* or receiver (the party receiving the message, who may reply with a return message). For example, the first activity of Fig. 4.26, "Submit purchase order" takes place between the Customer, who sends the purchase order, and the Seller, who receives it.

A choreography activity is depicted as a box with rounded corners where two bands, one at the top, the other at the bottom of the box, represent the two parties involved in the interaction captured by the activity. A light band is used for the initiator whilst a darkened band is used for the recipient. The position of each band with respect to the box is left to the modeler, so long as the two bands are on opposite sides. An envelope attached to a band via a dashed line represents the message sent by that party. This envelop is darkened if it is the return message of a two-way interaction.

A precedence relation between two interactions can only be established if the initiator of the second interaction is involved in the preceding interaction (either as

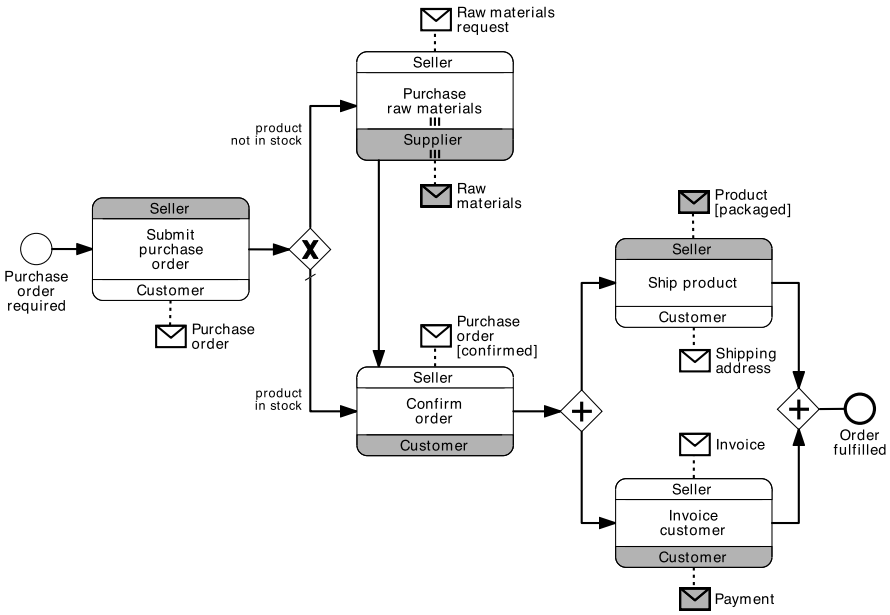


Fig. 4.26 The choreography diagram for the collaboration diagram in Fig. 4.9

a sender or as a receiver), except for the first interaction. In this way the sender of the second interaction 'knows' when this can take place. On the other hand, if there are no order dependencies between two or more interactions, we can link these interactions via an AND-split, as shown in Fig. 4.26. Make sure, however, that the sender of each interaction following the split is involved in the interaction preceding the split.

An (X)OR-split models the outcomes of a an internal decision that is taken by one party. This imposes that the data upon which the decision is taken are made available to that party via an interaction prior to the split. In our example, the data required by the XOR-split are extrapolated from the purchase order, which is sent to the seller in the interaction just before the split. Furthermore, all interactions immediately following the split must be initiated by the party who took the decision. In our example, these are done by the seller. In fact, it makes no sense that a decision taken by one party results in an interaction initiated by another party—the latter would not be able to know the results of the decision at that stage.

The event-based XOR-split is used when the data to make a choice are not exposed through an interaction before the split. Thus, the parties not involved in the decision will only know about this with the receipt of a message. This imposes that the interactions following an event-based split must either have the same sender or the same receiver. For example, we use an event-based split to model a situation where an applicant waits for a confirmation message that may either arrive from a broker or directly from the insurer (the decision of which party to interact with

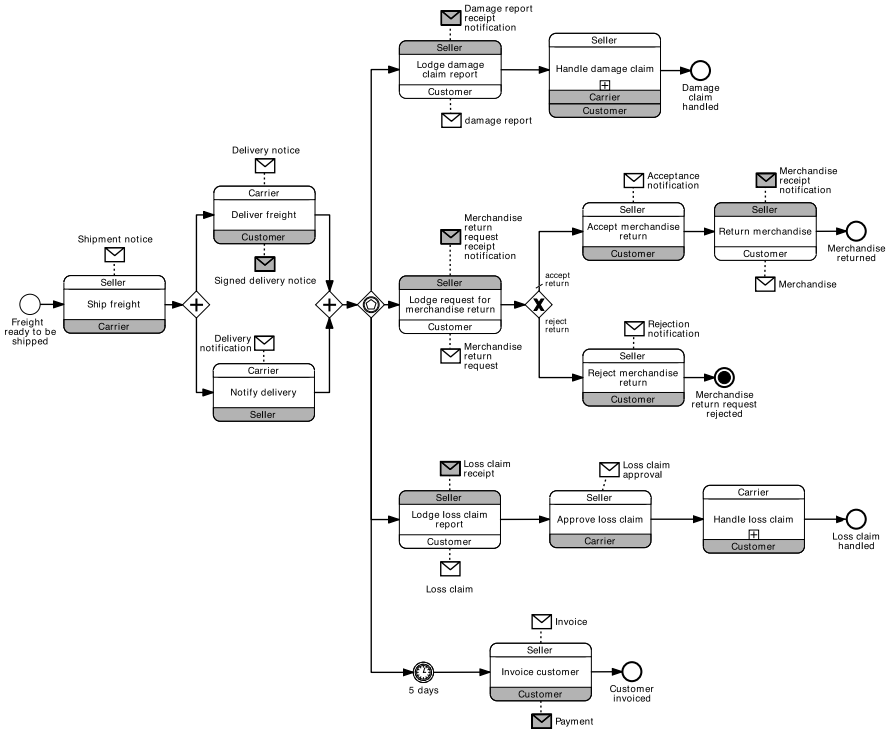


Fig. 4.27 The choreography diagram between a seller, a customer and a carrier

the applicant is taken by the broker together with the insurer). Figure 4.27 shows another example: here a seller waits for one of three possible messages from a customer, representing three different types of complaint. The decision is taken by the customer and the seller is not aware of this until the specific complain is received. The interactions following an event-based split can be constrained by a timer. In this example, if the seller does not receive any message after five days, they will trigger an interaction to invoice the customer. In this case, all parties in the interactions following the split must be involved in the interaction preceding the split in order to be aware of the timer.

Exercise 4.17 The choreography below illustrates the interactions that may occur among a seller, a customer and a carrier after the freight has been delivered by the carrier to the client. Use this diagram as a template to build the corresponding collaboration diagram. Observe the use of the terminate event in this example. In a choreography this event can only be used to denote a negative outcome and not to forcefully terminate the choreography, since the parties not involved in the interaction preceding the terminate event would not know that the terminate event has been reached.

Complex interactions involving more than one business party are modeled via a sub-choreography activity. This activity is represented with the plus symbol (as a sub-process) and may have multiple bands representing all roles involved. For example, the "Handle damage claim" interaction in Fig. 4.27 occurs between the seller, the carrier and the customer. The messages involved in a sub-choreography are only visible when expanding the content of the sub-choreography where also their order becomes apparent.

Artifacts cannot be explicitly expressed in a choreography via data objects or data stores. This is because a choreography does not have a central control mechanism to maintain data.

Exercise 4.18 Model the choreography and collaboration diagrams 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 may either deal with the application themselves, if the amount of the mortgage loan is within the mandate the broker has been given by BestLoans, or forward the application to BestLoans. If the broker deals with the application themselves, this results in either a rejection or an approval letter being sent back to the client. 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 their application with the mortgage department or with the broker, depending on which entity is dealing with the client. Moreover, the client may request the cancellation 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 two days with a cancellation confirmation, in which case the process is canceled, or with a cancellation 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 Recap

This chapter provided us with the means to model complex business processes. We first learned how to structure complex process models in hierarchical levels via sub-process activities. Sub-processes represent activities that can be broken down in a number of internal steps, as compared to tasks, which capture single units of work. An interesting aspect of sub-processes is that they can be collapsed to hide details. We also discussed how to maximize reuse by defining global sub-processes within a process model collection, and invoking them via call activities. A global sub-process is modeled once and shared by different process models in a repository.

We then 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. Further, we saw how the concept of multi-instantiation can be related to data collections and extended to pools. We also discussed ad-hoc sub-processes for capturing unstructured repetition.

Next, we expanded on various types of event. 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 events external to the process, using an event-based split followed by intermediate catching events.

Afterwards, we showed how to handle exceptions. Exceptions are situations that deviate the process from its normal course, due to technology or business faults. 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 configured not to interrupt the attached activity. In this case the event is called non-interrupting. These events are convenient to model procedures that have to be launched in parallel to an activity's execution, when an event occurs. 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.

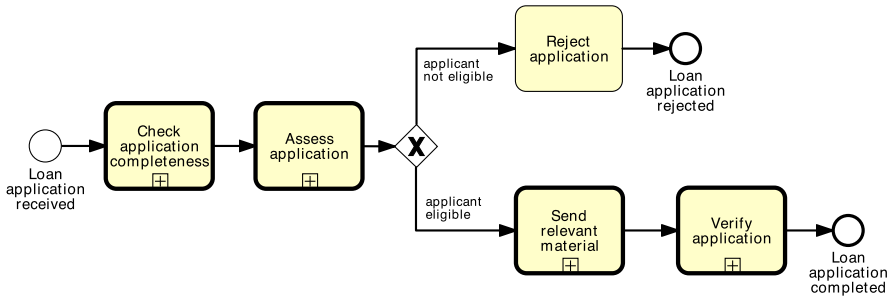
We then saw how business rules can be defined in process models via conditional events. A conditional event, available as a start and catching intermediate event, allows a process instance to start, or progress, only when the corresponding (boolean) business rule evaluates to true.

We concluded this chapter on advanced process modeling by introducing choreography diagrams. A choreography diagram models the interactions that happen between the various business parties partaking in a business process. Each activity in

the choreography captures an interaction between a sender and a receiver. To establish an order dependency between two interactions, the sender of the second activity must be involved in the first one, otherwise this party will not be able to determine when to send the message pertaining to the second interaction. We also discussed the rules for using gateways in a choreography where a decision is made by a specific party based on data or events. Sub-choreographies can be used to model complex interactions involving more than two parties, in a similar vein to sub-processes in a collaboration.

4.9 Solutions to Exercises

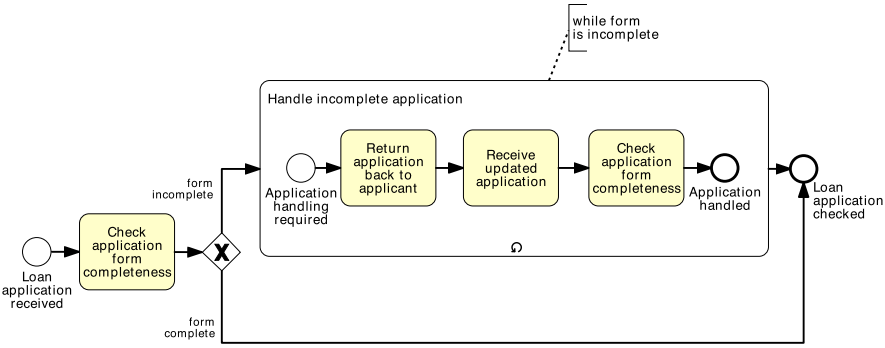
Solution 4.1



Solution 4.2 Possible sub-processes are "Request purchase", "Issue purchase order", "Receive goods" and "Handle invoice". Of these, "Handle invoice" could be shared with other procure-to-pay processes of the same company, e.g. with that described in Example 1.1 for BuildIT. The first three sub-processes are internal to this procure-to-pay process, because they are specific to the enterprise system that supports this process.

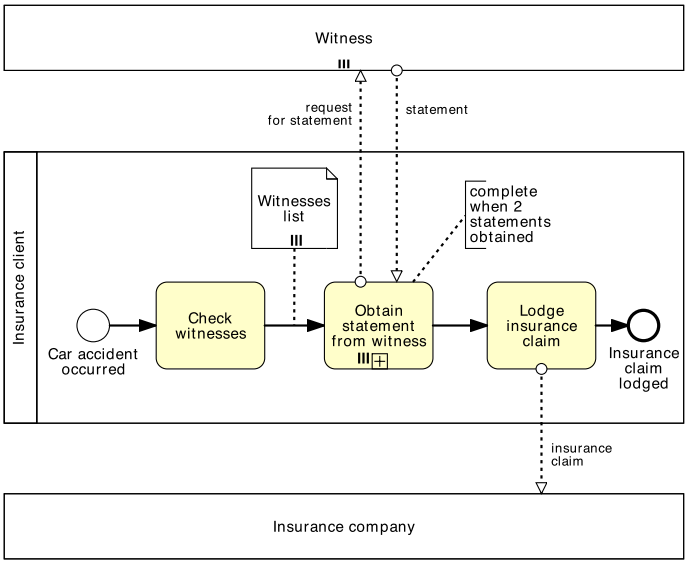
Solution 4.3

1. In Exercise 3.9 the repetition block goes from activity "Record claim" to activity "Review claim rejection". The entry point to the cycle is the input arc of activity "Record claim"; the exit points are arcs "claim to be accepted" and "claim rejection accepted", the former being inside the repetition block.
2. In Solution 3.4 the repetition block is made up of 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-split, while the exit point is the arc "form complete" which is inside 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.

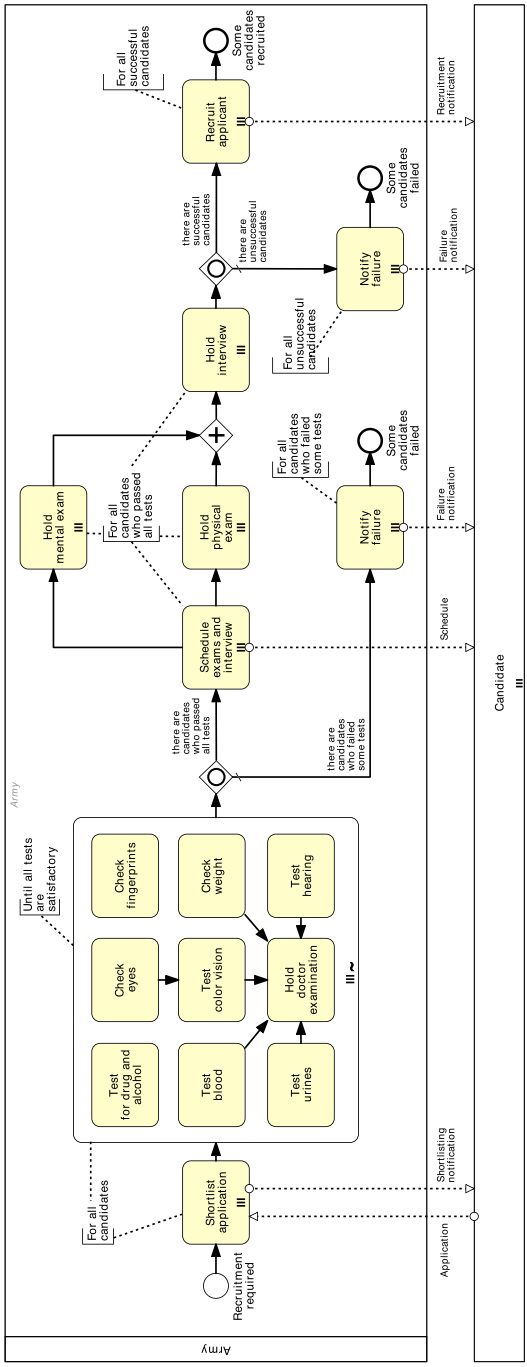


In this case using a loop activity is still advantageous, since we reduce the size of the original model if we collapse the sub-process.

Solution 4.4

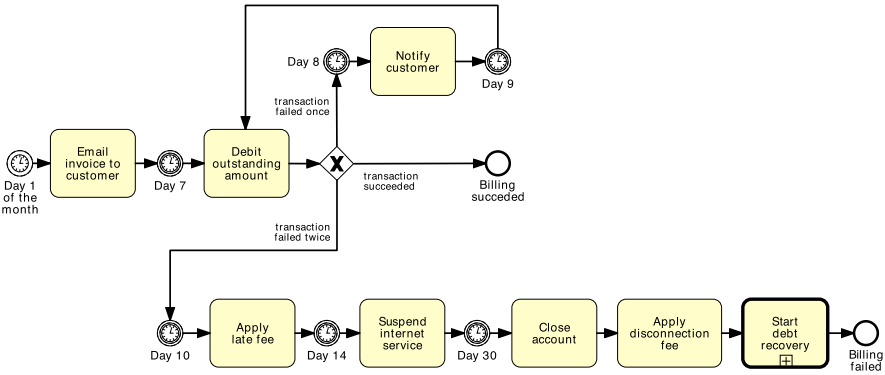


Solution 4.5

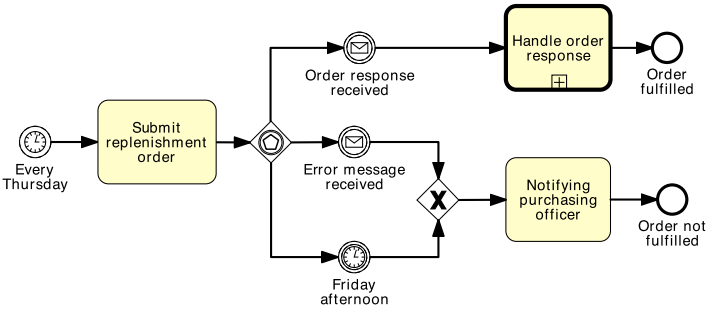

Army
Recruitment required
For all candidates
Shortlist application
Until all tests are satisfactory
Test for drug and alcohol
Test blood
Test urines
Check eyes
Test color vision
Check fingerprints
Check weight
Test hearing
Hold doctor examination
there are candidates who passed all tests
there are candidates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

Solution 4.6 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 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.

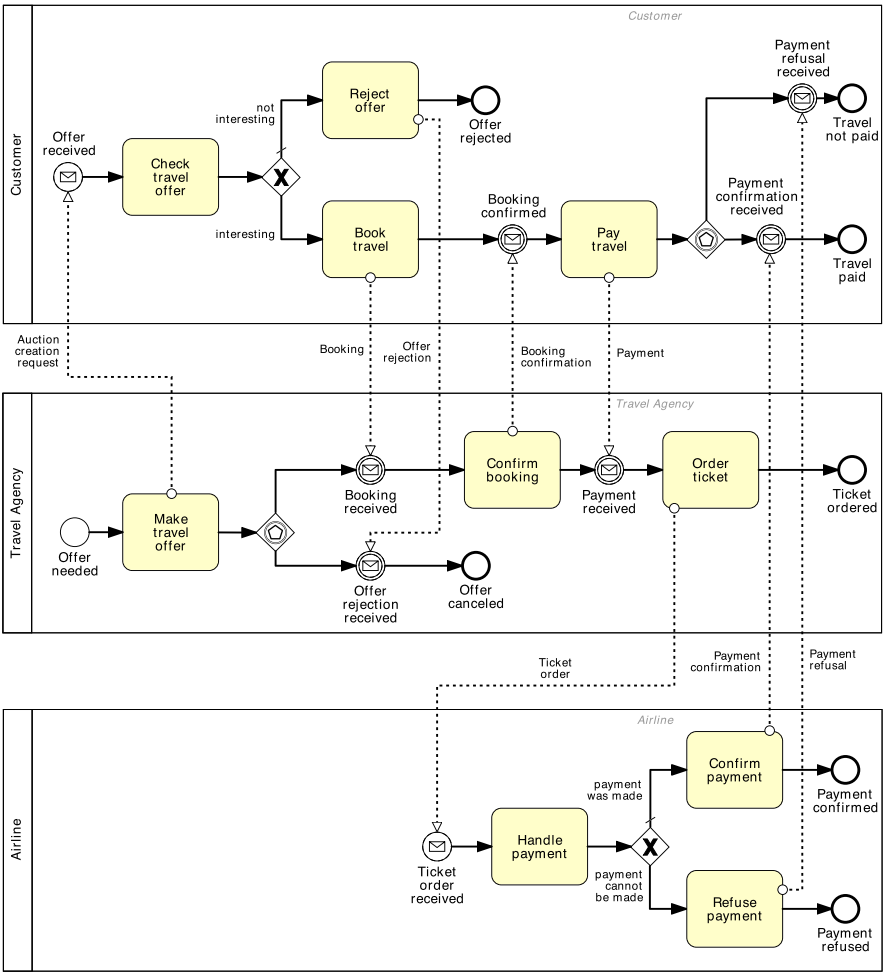
Solution 4.7



Solution 4.8

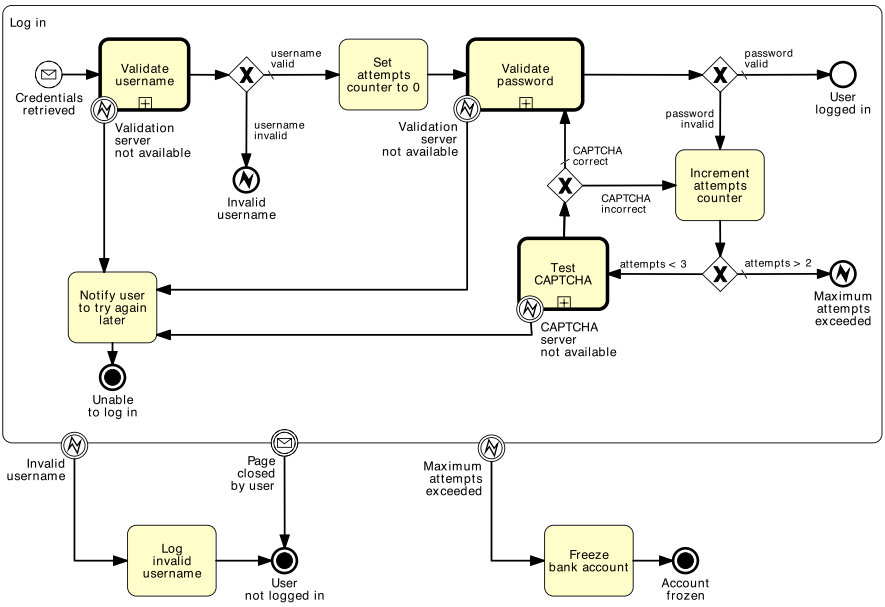


Solution 4.9

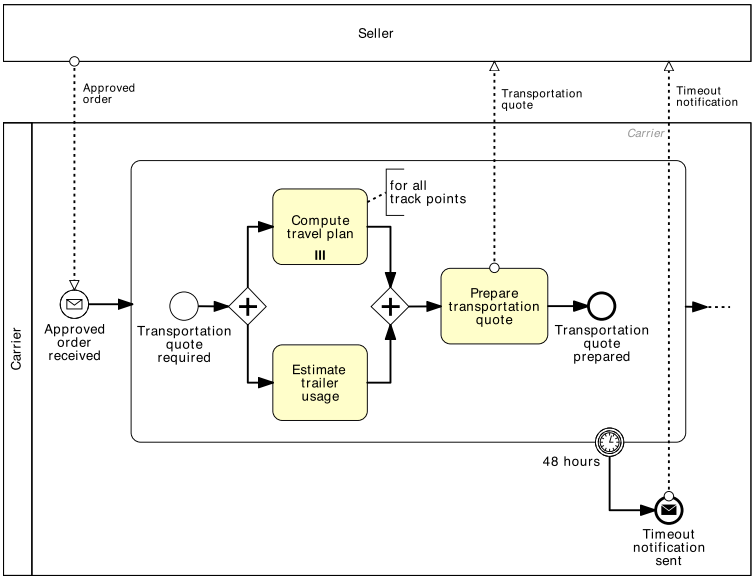


Solution 4.10 The following end events should be terminate events: Fig. 4.12—"callover deferred", Fig. 4.14—"Quote rejected" in the Client and Insurer pools, Fig. 4.18—"Offer rejected" in the Customer pool, "Offer canceled" in the Travel Agency pool and "Payment refused" in the Airline pool.

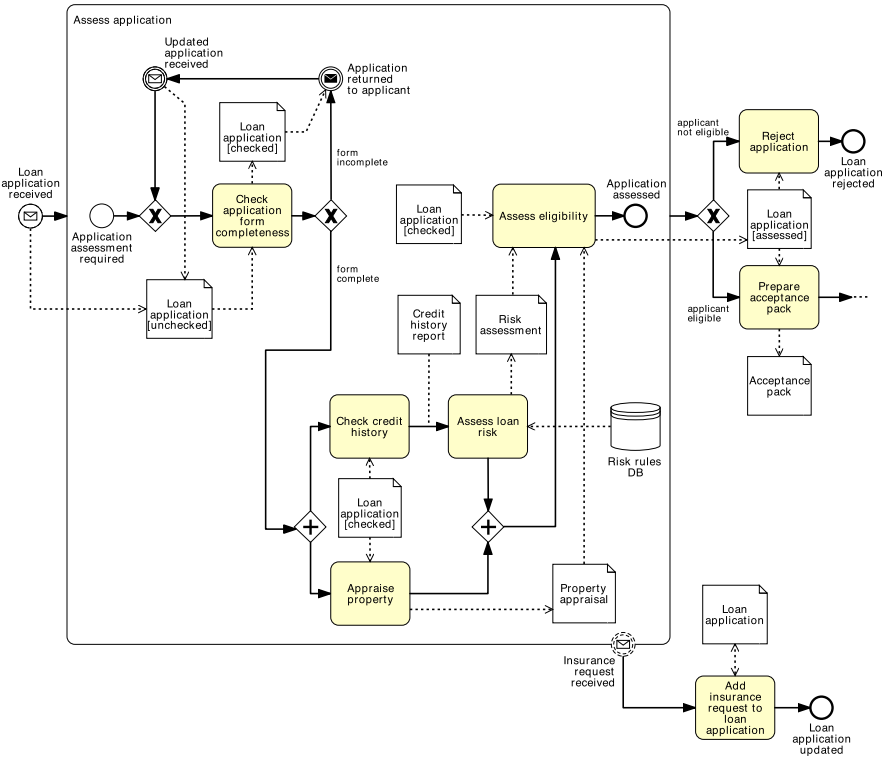
Solution 4.11



Solution 4.12

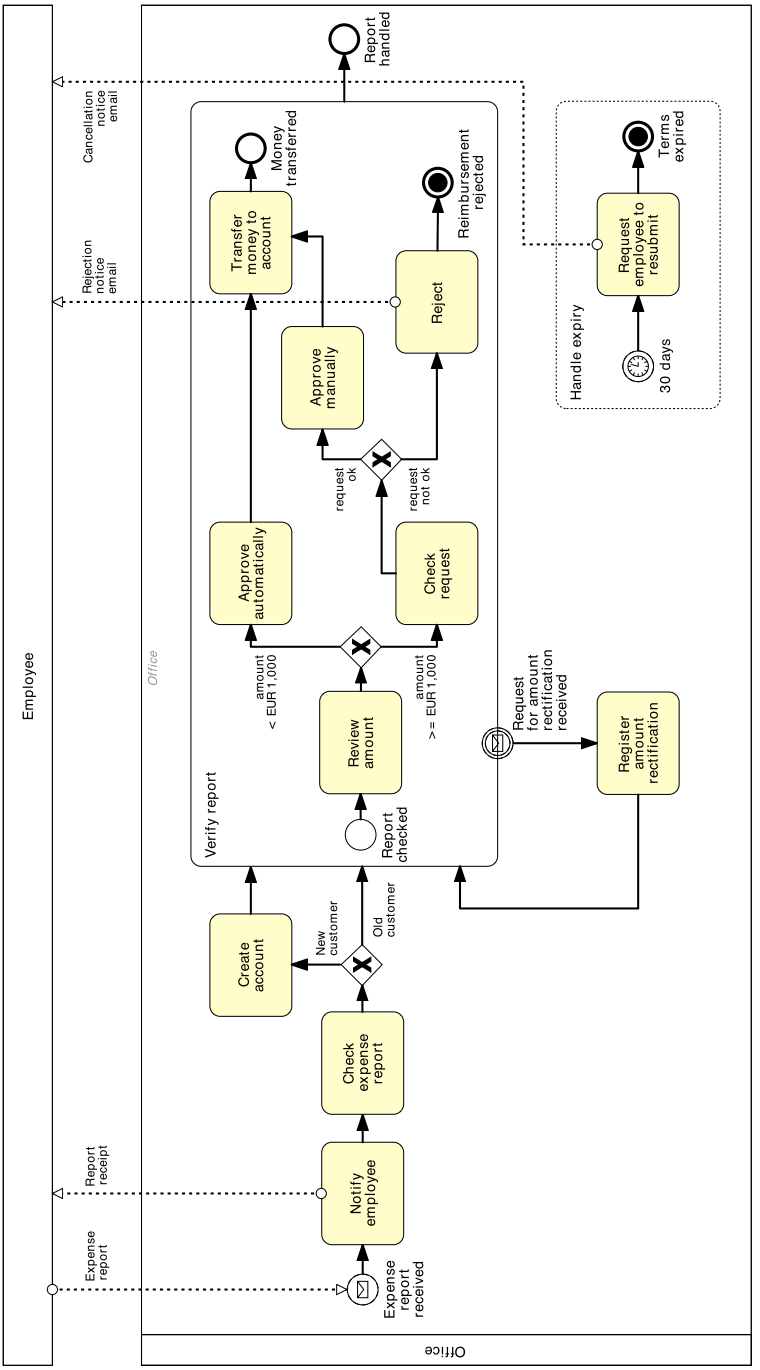


Solution 4.13

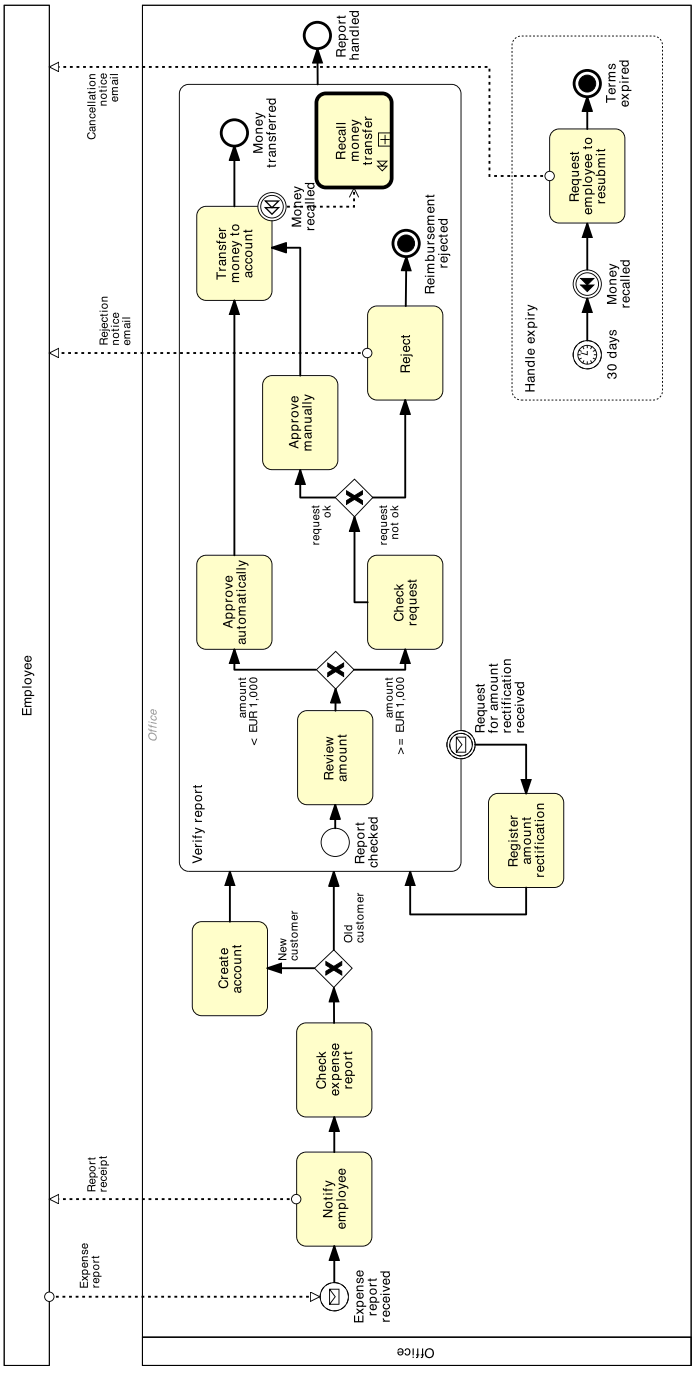


Observe that in the “Assess application” sub-process, the Loan application can have two possible states: “checked” or “unchecked”. In order to use the Loan application in any such state as input of activity “Add insurance request to loan application”, we do not specify any state for this data object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

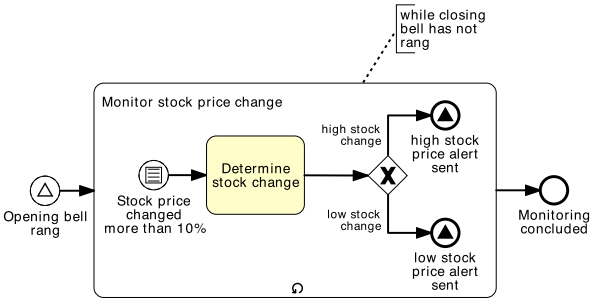
Solution 4.14



Solution 4.15



Solution 4.16



In this solution 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.

Solution 4.17

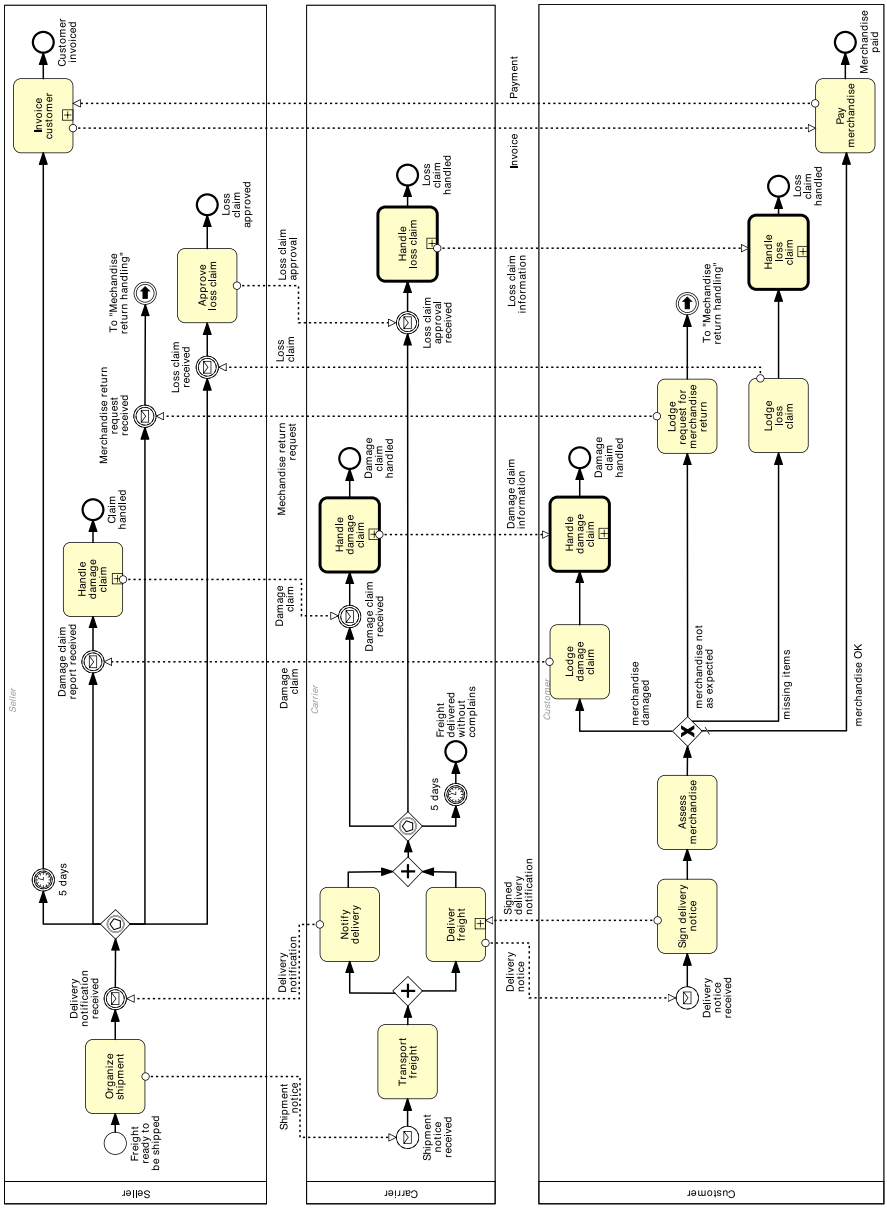


Fig. 4.28 Collaboration diagram—part 1/2 (Freight shipment fragment)

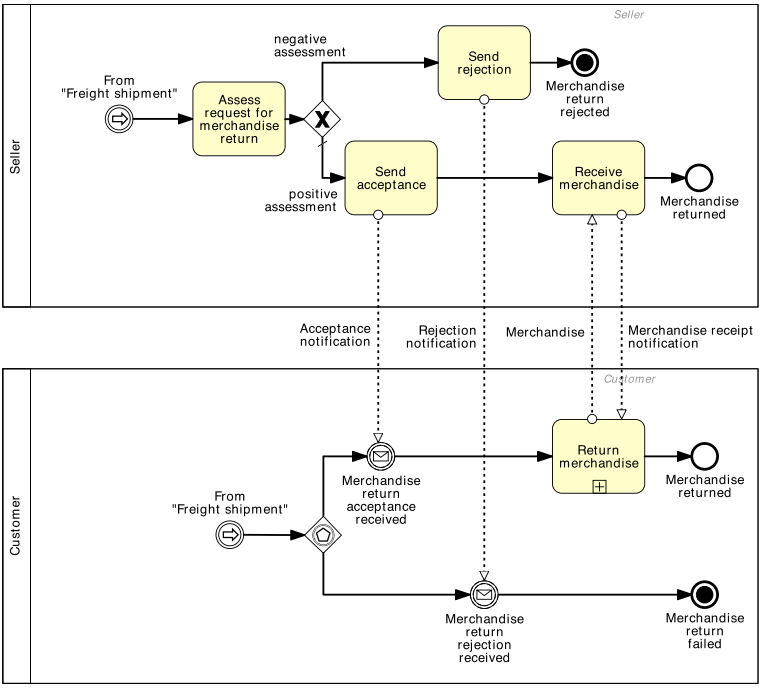


Fig. 4.29 Collaboration diagram—part 2/2 (Merchandise return handling fragment)

In Solution 4.17 we used the *link event* to lay the diagram over two pages, since the model was too large to fit in one page. The link event does not have any semantics: it is purely a notational expedient to break a diagram over multiple pages. An intermediate throwing link event (marked with a full arrow) breaks the process flow and provides a link to the diagram where the flow continues; an intermediate catching link event (marked with an empty arrow) resumes the flow and indicates the diagram where this flow is resumed from.

Solution 4.18

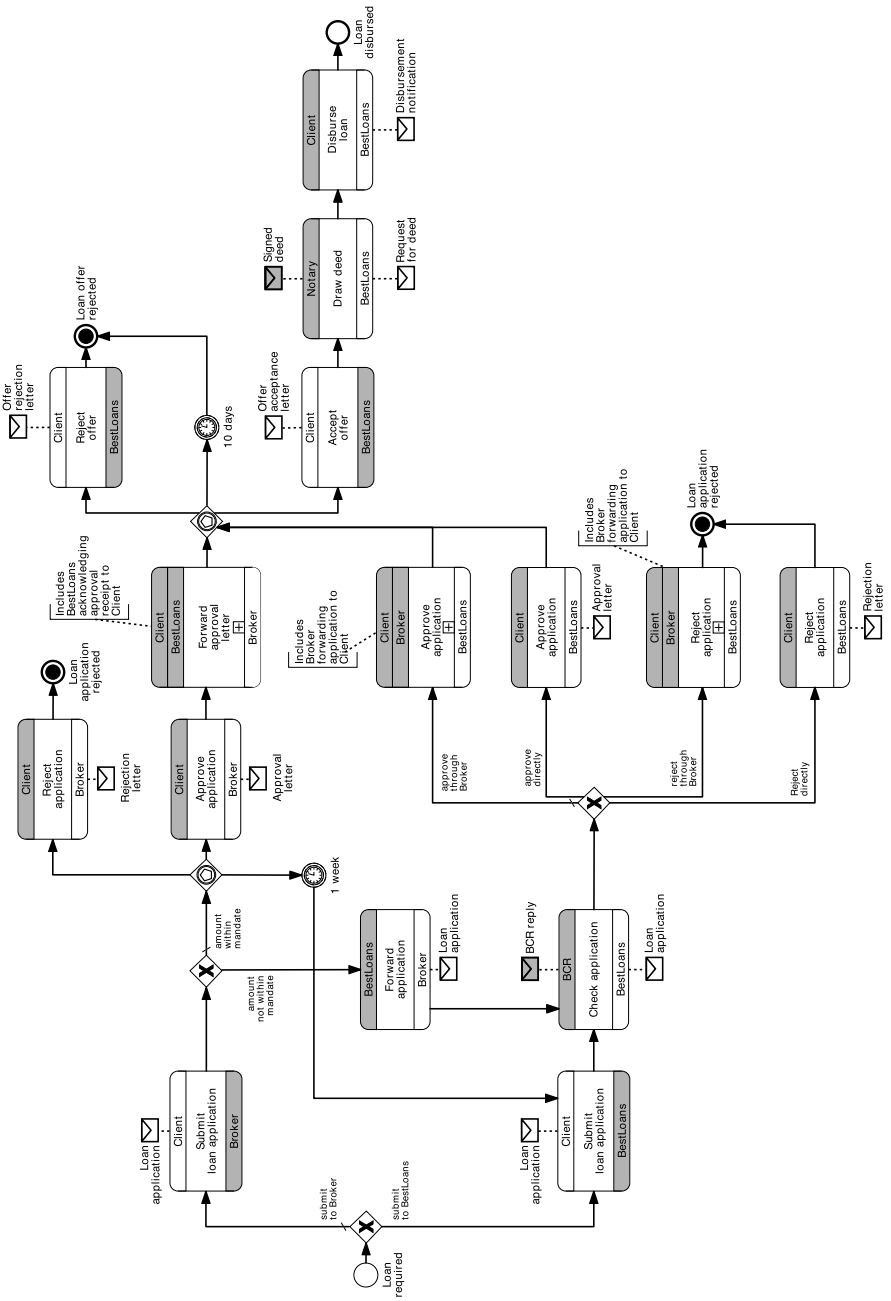


Fig. 4.30 Choreography diagram—part 1/2

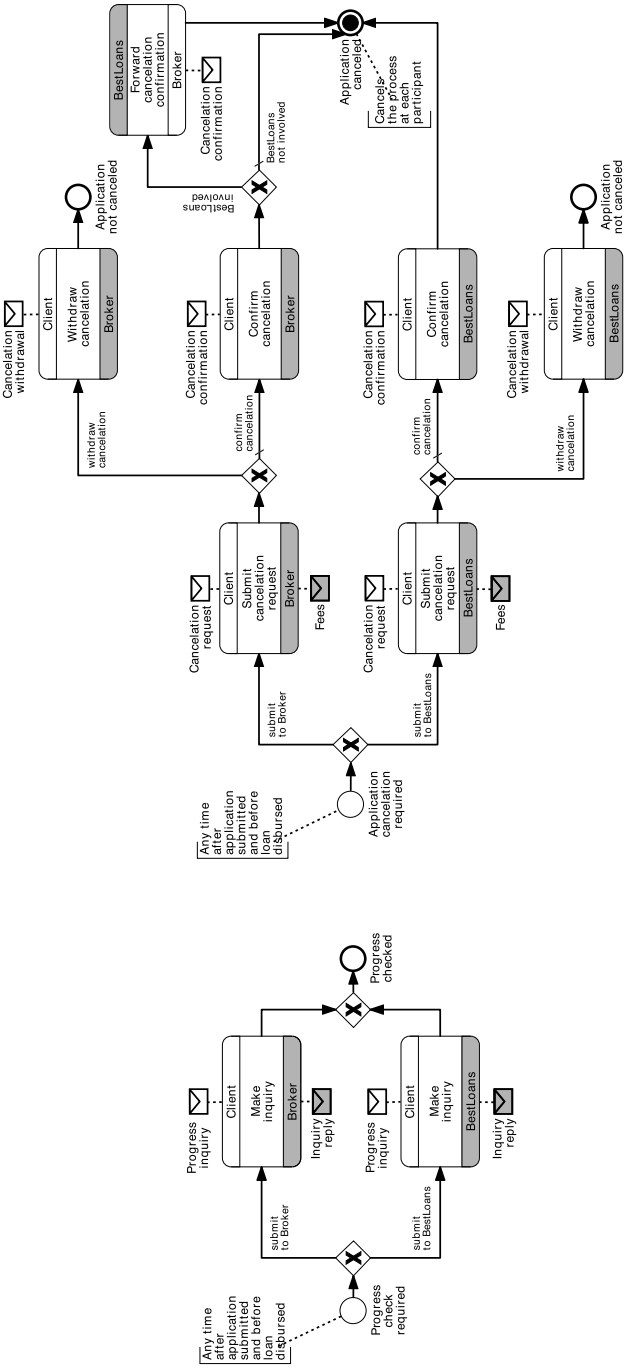


Fig. 4.31 Choreography diagram—part 2/2

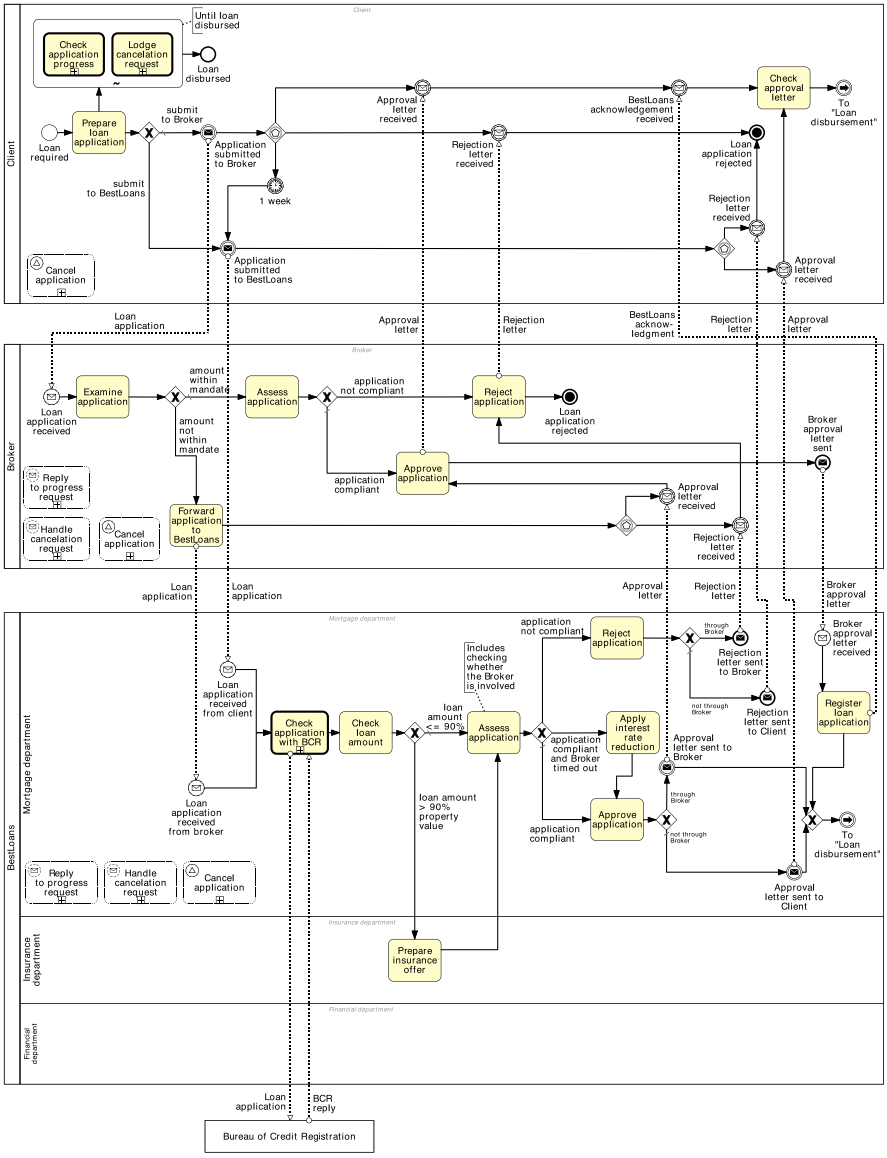


Fig. 4.32 Collaboration diagram—part 1/3 (Loan establishment fragment)

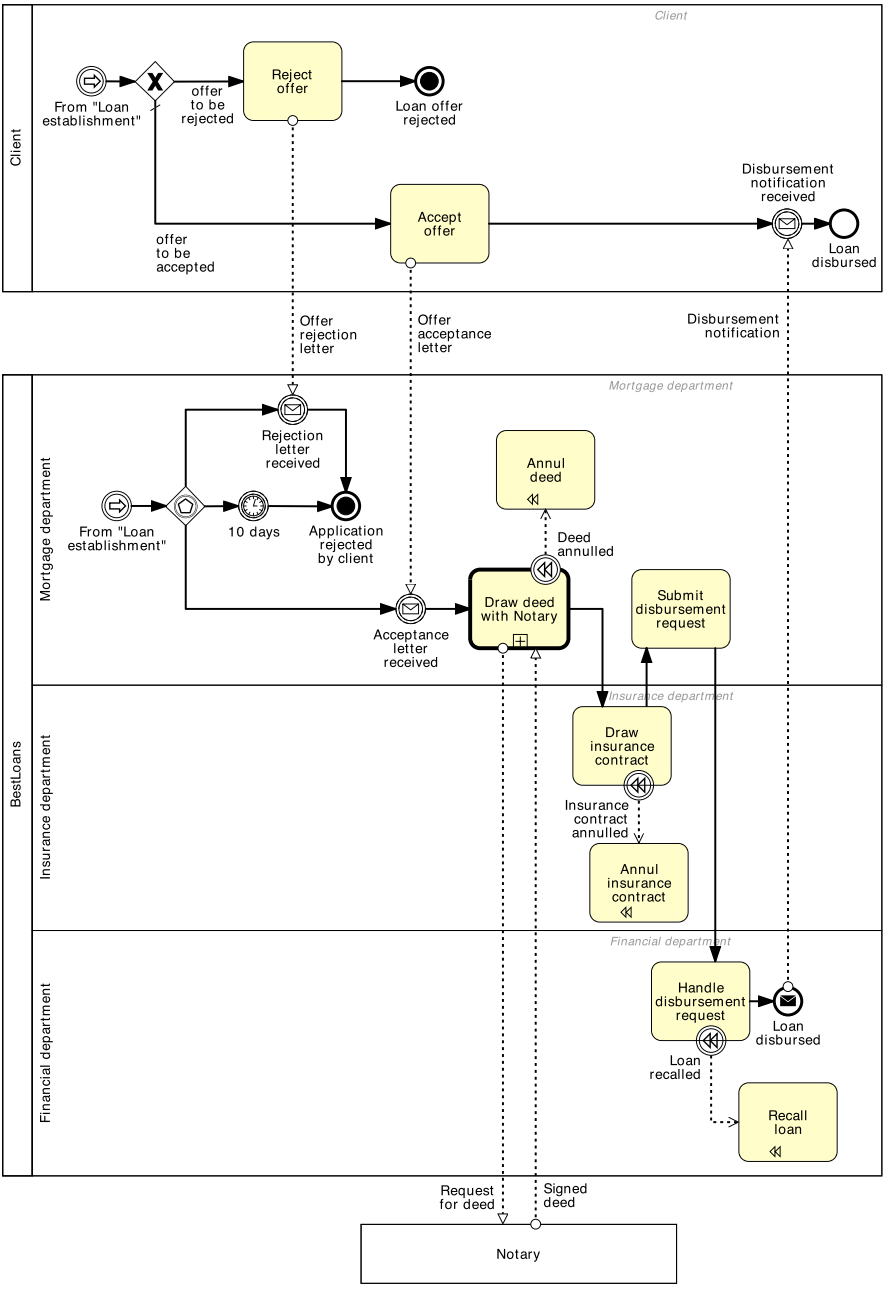


Fig. 4.33 Collaboration diagram—part 2/3 (Loan disbursement fragment)

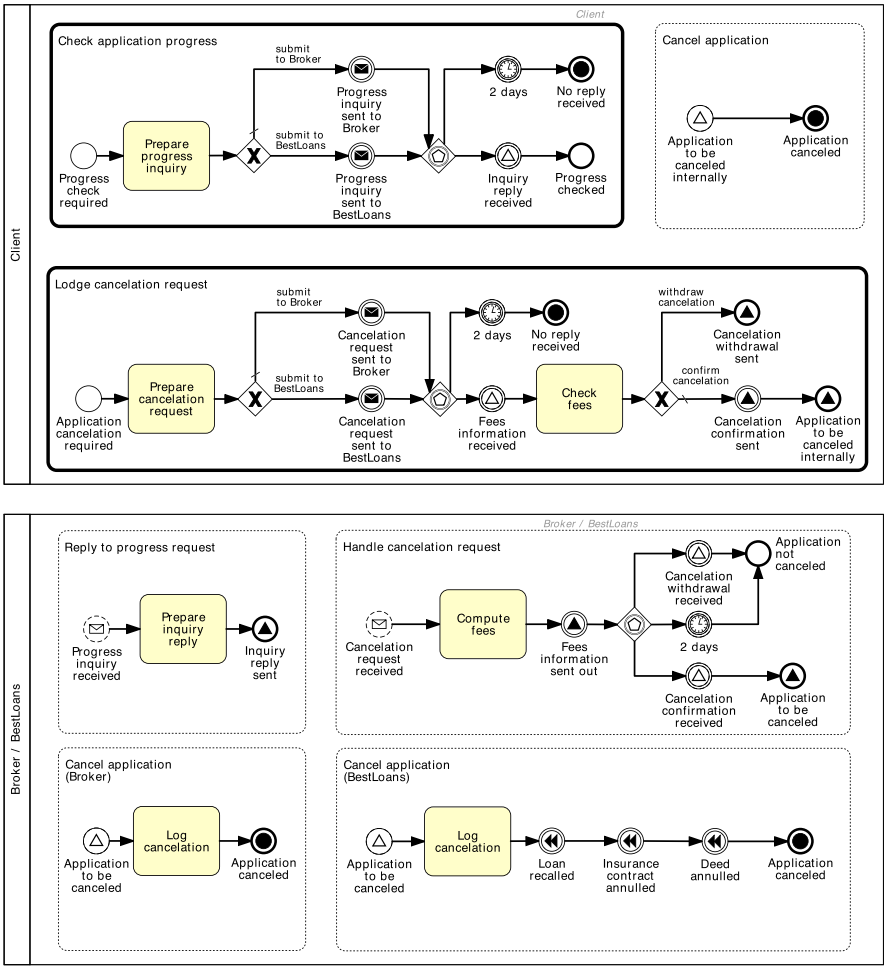


Fig. 4.34 Collaboration diagram—part 3/3 (sub-processes)

4.10 Further Exercises

Exercise 4.19

1. Model the prescription fulfillment process described in Exercise 1.6. Use sub-processes where required, and nest them appropriately.
2. Is there any sub-process that can potentially be shared with other business processes of the same pharmacy, or of other pharmacies?

Exercise 4.20 Model the business process described in Exercise 3.12 using a loop activity.

Exercise 4.21

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.

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

Exercise 4.22 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 4.23 Model the following process for selecting Nobel prize laureates for chemistry.

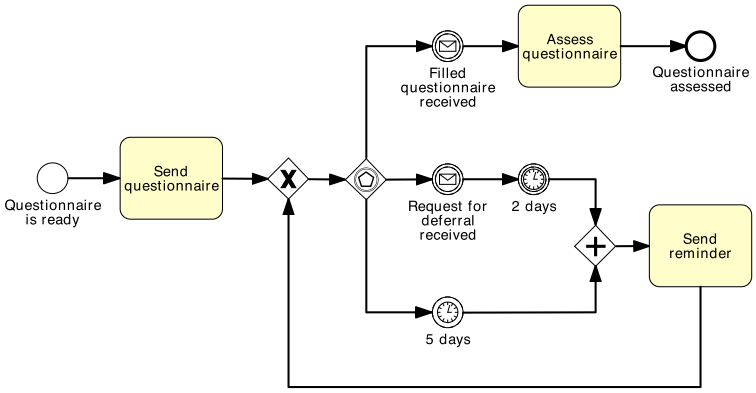
> September: nomination forms are sent out. The Nobel committee sends out confidential forms to around 3,000 people—selected professors at universities around the world, Nobel laureates in physics and chemistry, and members of the Royal Swedish Academy of Sciences, among others.
> February: deadline 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 and diploma, and a document confirming the prize amount.

Acknowledgement This exercise is taken from "Nomination and Selection of Chemistry Laureates", Nobelprize.org. 29 Feb 2012 (http://www.nobelprize.org/nobel_prizes/chemistry/nomination).

Exercise 4.24

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.

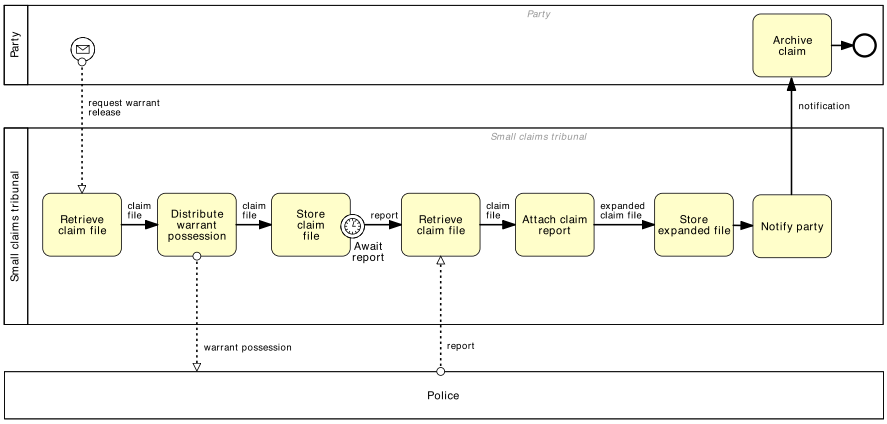
Exercise 4.25 What is wrong with the following model?



Exercise 4.26 Extend the billing process model seen in Exercise 4.7 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 4.27 What is wrong with the following model?



Exercise 4.28 Model the following business process at a supplier.

After a supplier notifies a retailer of the approval of a purchase order, the supplier can either receive an order confirmation, an order change or an order cancellation from the retailer. It may happen that no response is received at all. If no response is received after 48 hours, or if an order cancellation is received, the supplier will cancel the order. If an order confirmation is received within 48 hours, the supplier will process the order normally. If an order change is received within 48 hours, 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 4.29 Revise the model in Exercise 3.9 by using the terminate event.

Exercise 4.30 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 4.31 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 any item is not 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 cancellation 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 4.32

1. What is the difference between a collaboration and a choreography diagram? What are the respective modeling objectives?
2. Model the choreography diagram for the collaboration diagram that you modeled in Exercise 3.7.

Exercise 4.33 Model the choreography and collaboration diagrams 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 that will incur to process the application. These costs depend on the type of development plan (for residential or commercial purposes), and on the permit/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 re-submit 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.

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 4.34 Model the choreography and collaboration diagrams 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 labor 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 five 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 three 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 cancellation 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 4.35 Model the choreography and collaboration diagrams for the following business process at MetalWorks. Keep in mind that the purpose of this BPMN diagram is to serve as a means of communication between the business stakeholders and the IT team who has to build a software system to automate this process.

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 either accept it, reject it, or ask again the customer to make further changes.

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 typically 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. If the material has to be ordered, the procurement officer selects a suitable supplier for the raw material and sends a PO to the selected supplier. This "PO for a raw material" is called a "material PO", and it 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 cancellation has been successful. If the customer needs to pay a penalty, the sales officer first asks the customer if they accept to pay the cancellation penalty. If the customer accepts to pay the cancellation penalty, the order is canceled and all work related to the order is stopped. Otherwise, the work related to the order continues.

4.11 Further Reading

In this chapter we showed how sub-processes can be used to reduce the complexity of a process model by reducing the overall process model *size*. Size is a metric strongly related to the understandability of a process model. Intuitively, the smaller the size, the more understandable will the model be. There are other metrics that can be measured from a process model to assess its understandability, for instance the *degree of structuredness*, the *diameter*, and the *coefficient of connectivity*. A comprehensive discussion on process model metrics is available in [50]. The advantages of modularizing process models into sub-processes and automatic techniques are covered in [75], while the correlation between number of flow objects and error probability in process models is studied in [54, 55].

BPMN 2.0 provides various other event types besides the main ones presented in this chapter. For example, the *link event* can be used to modularize a process sequentially (useful when a process model does not fit on a single paper and has to be divided over multiple papers). An example of link event is shown in Figs. 4.28 and 4.29. The *multiple event* can be used to catch one of a set of events or throw a set of

events. Moreover, BPMN 2.0 provides a further diagram type besides collaborations and choreographies. This diagram type is called *conversation diagram*, and focuses on the messages exchanged by two or more process parties, abstracting from the precise order in which the messages occur. All these constructs are described in detail in the BPMN 2.0 specification by OMG [61]. There are also various books that present BPMN 2.0 by example, among others [2, 87, 107].

This chapter concludes our coverage of the BPMN 2.0 language. For further information on this language, we point to the BPMN web-site: www.bpmn.org, where the official specification can be downloaded from. This site also provides a link to a handy BPMN poster, a quick reference guide on all BPMN elements and includes a comprehensive list of books on the subject, as well as tool implementations that support this standard.