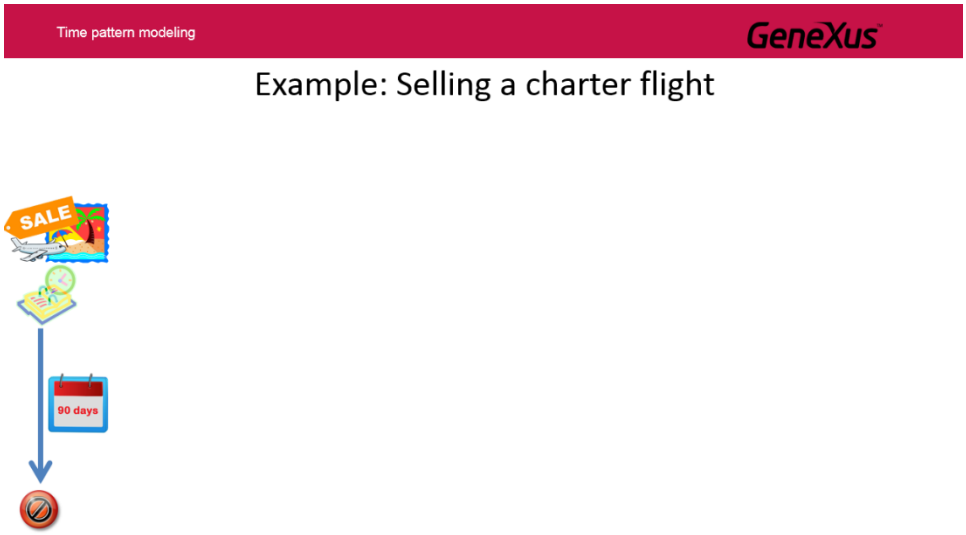
Modeling Time Patterns

When modeling business processes, we frequently need to represent situations related to time, such as waiting periods, delays, expiry dates or deadlines for the activities included in a process.

To model these situations, events of Timer type are used; also, the type of pattern that we will use to model a situation will depend on the type of situation to be represented. Let's see different cases with an example.

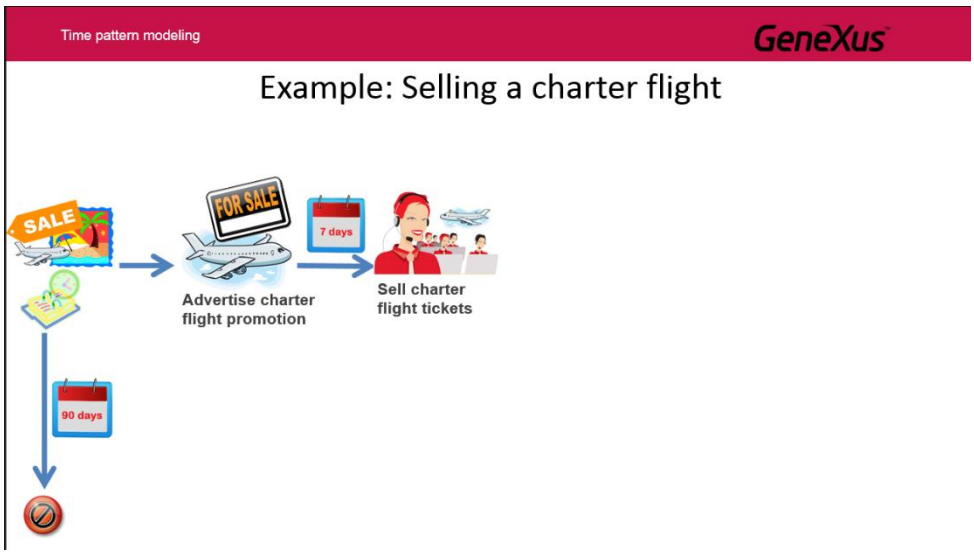
Using the Travel Agency process modeling, suppose that the Agency has requested us to represent the sales process of a charter flight as a way to promote a certain tourist attraction.

The entire process must have a maximum duration of 3 months and, when this period has elapsed, the process must end permanently.

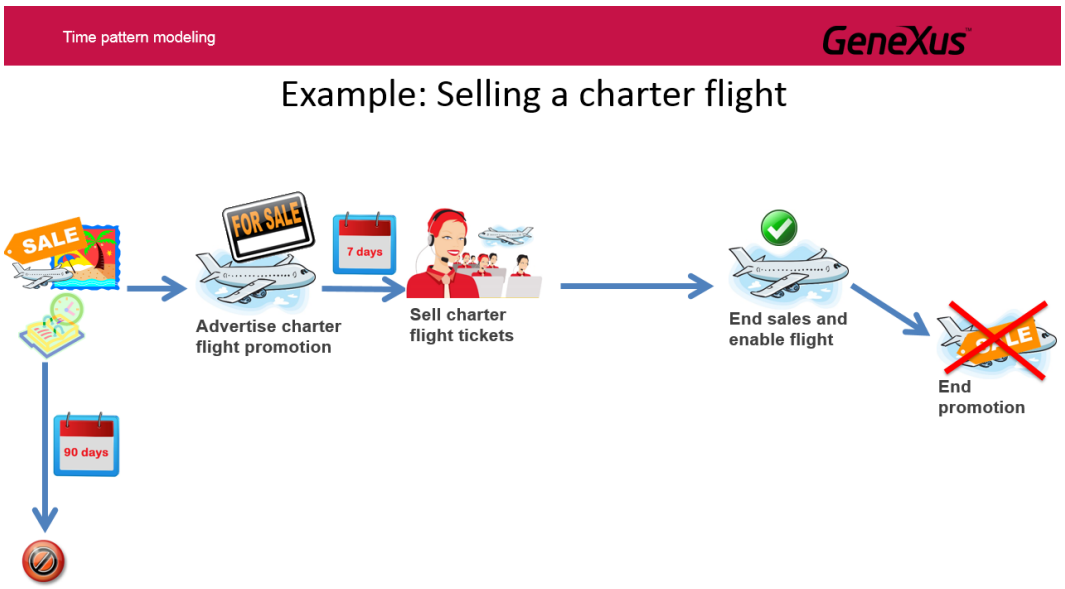


The sales process of a charter flight begins when the promotion is published in several media outlets, social networks and web pages. The objective is to sell all the seats available on the flight, so as to maximize the profits.

One week after launching the promotion, tickets are made available for sale.

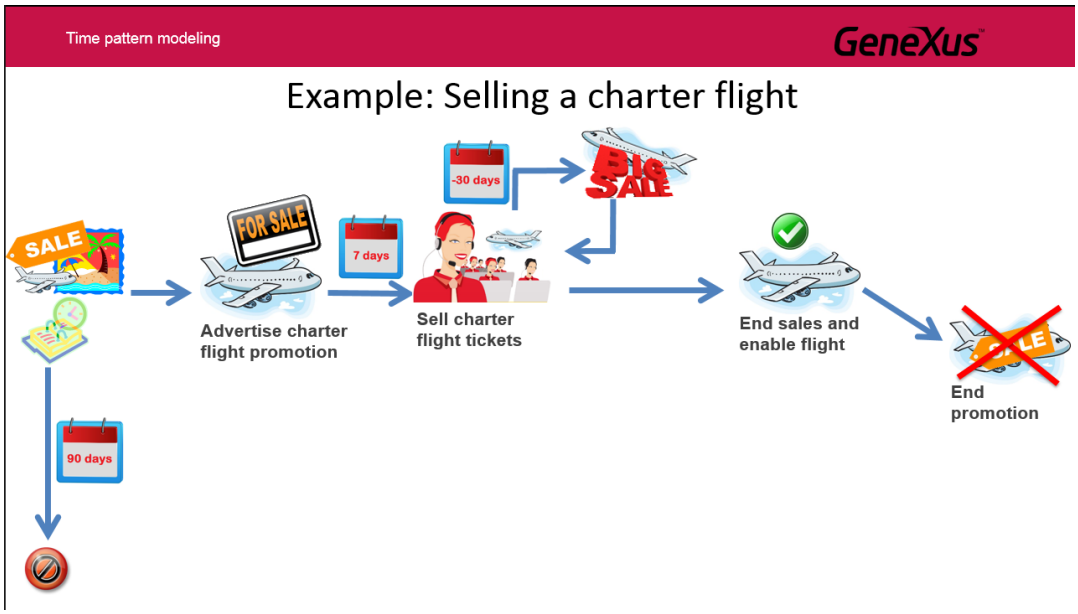


When all the tickets have been sold, the flight is enabled (which implies that this flight will actually operate) and the promotion is removed from the various media where it was published.

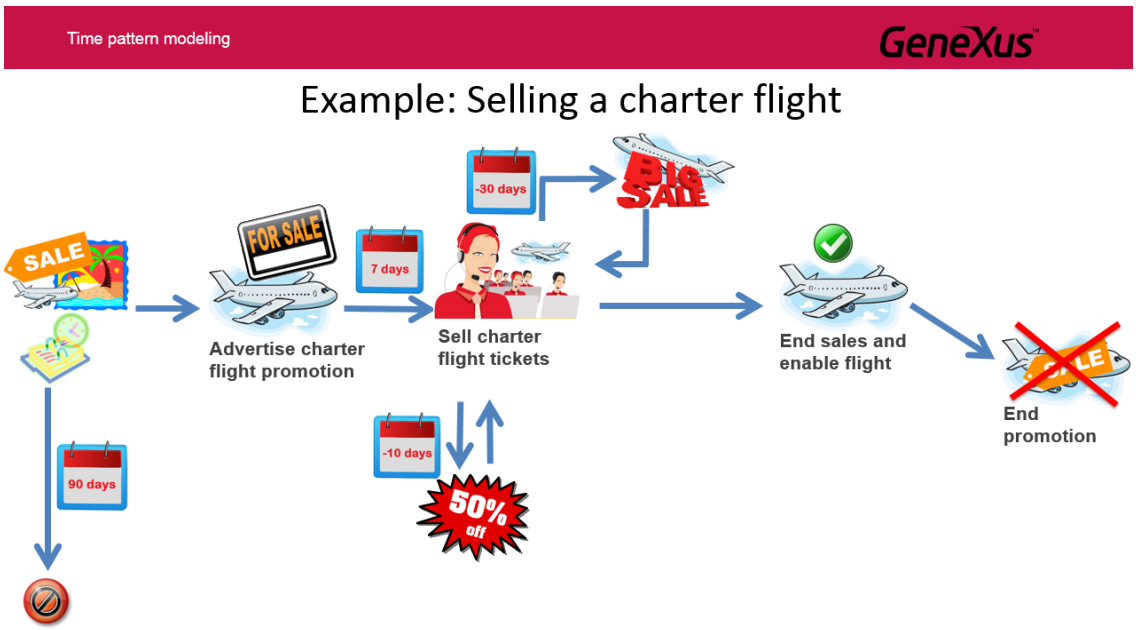


As the flight tickets are sold, certain actions are taken as time goes by.

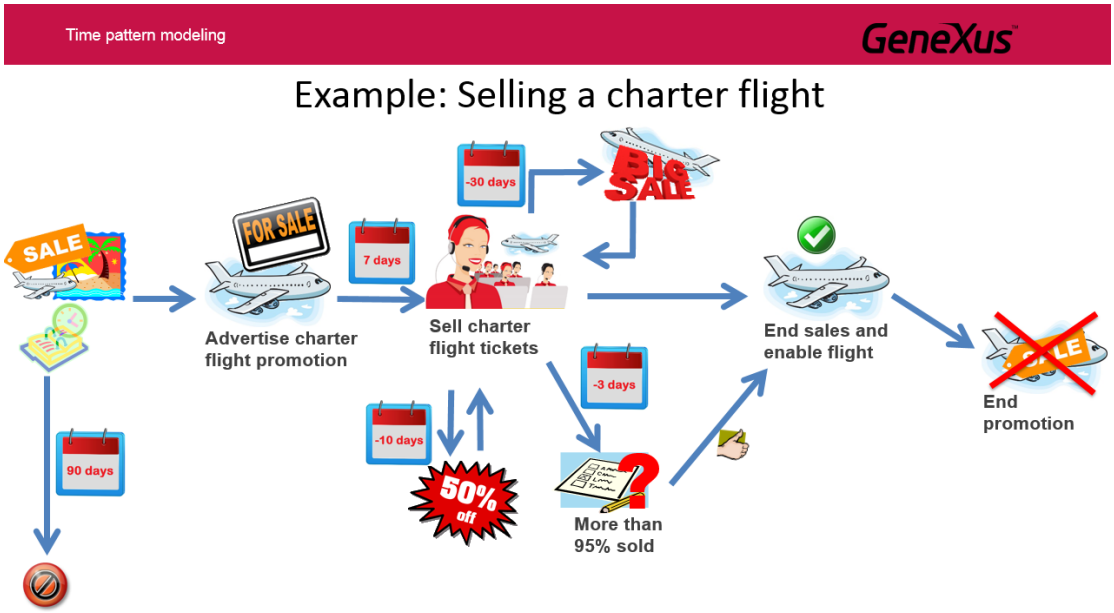
For example, if the sales process ends in 30 days and not all the seats have been sold yet, a warning is triggered that starts a sub process to increase the promotion advertising in the various sales channels.



When 10 days are left before the end of the process and seats are still available on the flight, an event is triggered. Even though this event doesn't interrupt the sales activity, it starts an activity to lower the prices advertised in order to attract more passengers, since the objective is to sell as many tickets as possible.

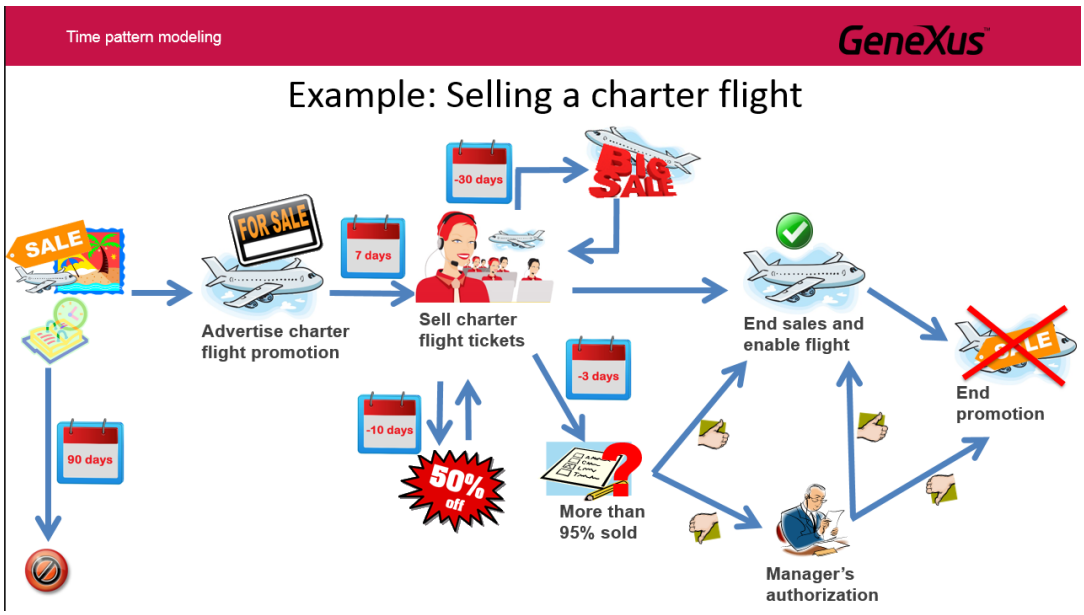


When 3 days are left to complete the 3-month period of the promotion, the sales task is interrupted and it is evaluated whether more than 95% of the seats available have been sold. If this percentage has been reached, ticket sales are stopped and the flight is enabled; also, the promotions advertised are removed and the process is ended.



If less than 95% of the flight tickets have been sold, an authorization from the Travel Agency's management is requested to check whether the flight will depart or be canceled due to not meeting the objective of selling most of the seats available.

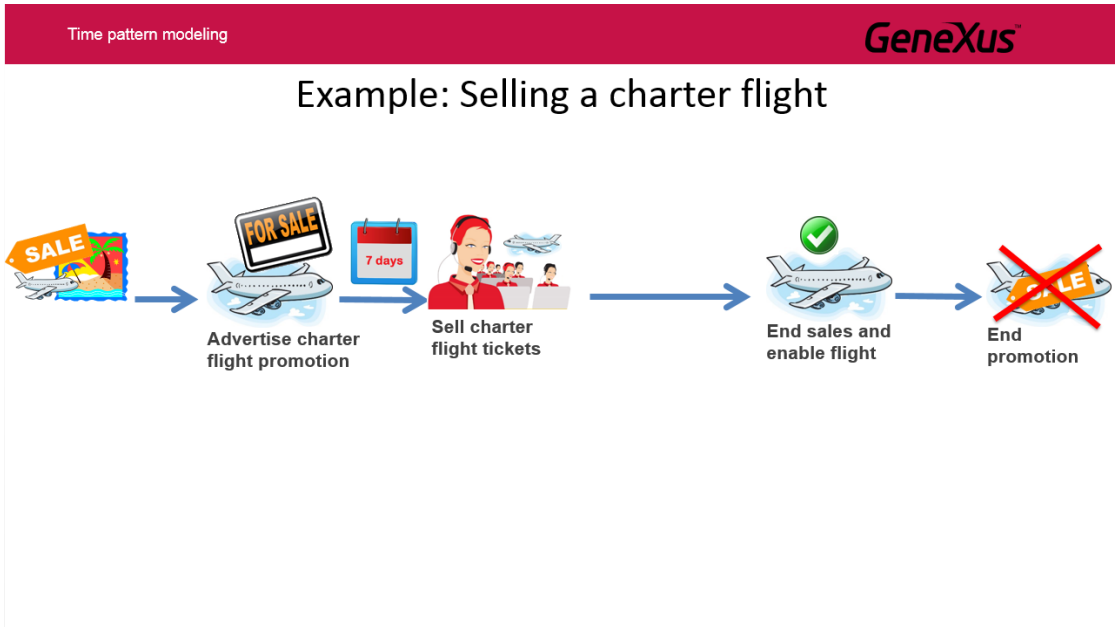
If the flight is authorized, it is enabled; in addition, the promotion is removed and the process is ended. If the Travel Agency's management doesn't authorize the flight, the promotion is removed and the process is ended.



Let's see how we can model each of these time-related situations.

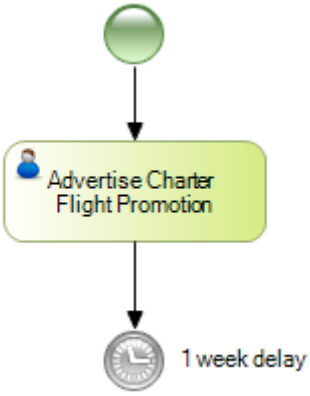
First, we run the GeneXus Business Process Modeler and create a BPDiagram object called CharterFlightSale.

We will focus on modeling the most common flow first; that is to say, the flow that takes place if there are no inconveniences. In this case, it involves publishing the charter flight promotion, selling all the tickets, enabling the flight, removing the promotion, and ending the process.

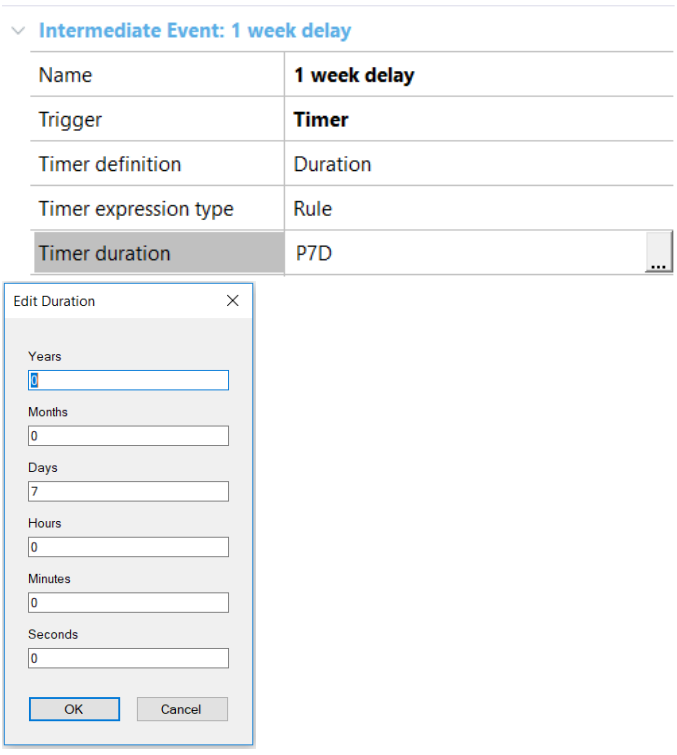


To start the process, from the toolbar we drag a None Start Event and an interactive task joined from the None Start Event. We call this task **Advertise Charter Flight Promotion**.

According to the Travel Agency's instructions, once the promotion has been published, we must wait 1 week to start selling the flight tickets. To implement a delay, we use an event of timer type, so we drag an intermediate timer event to the diagram and join it from the task **Advertise Charter Flight Promotion**. Next, we press F2 and add the description “1 week delay.”



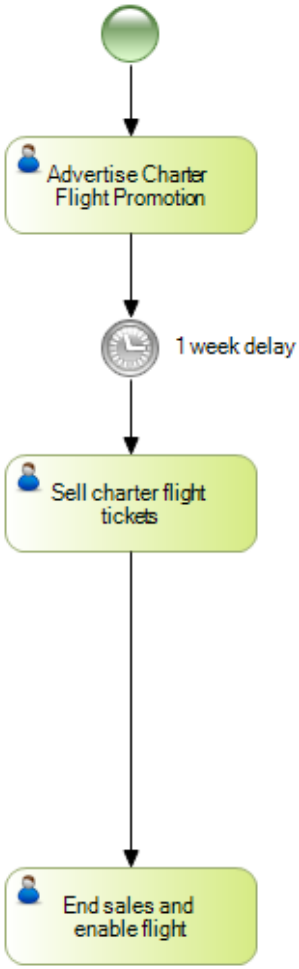
In the properties of the intermediate timer event, in the value Days we type 7.



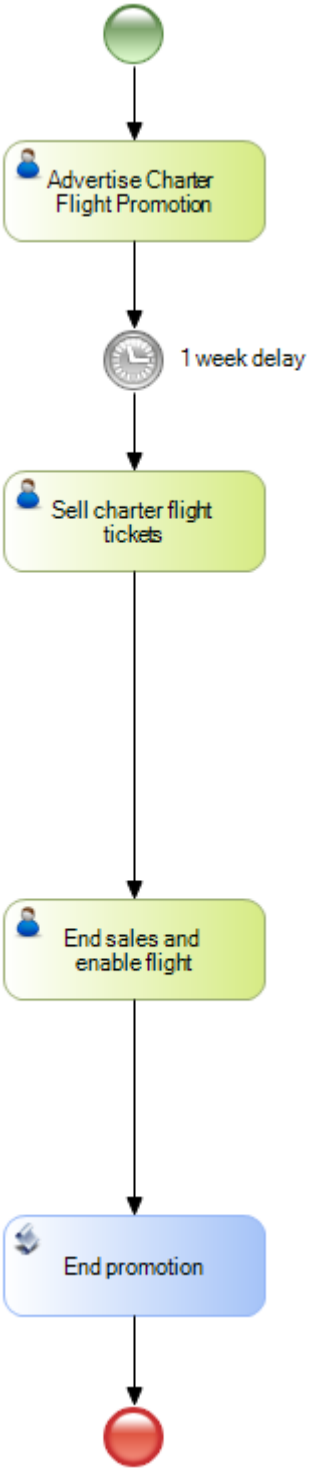
Now we drag another task of User type, call it **Sell charter flight tickets**, and join it from the timer event.

Once the task of publishing the charter flight promotion has been completed, the timer used in this way will stop the flow for a week before starting the task of selling the flight tickets.

Once all the tickets have been sold, the flight must be enabled; to do so, we drag another interactive task called **End sales and enable flight** and connect it from the ticket sales task.



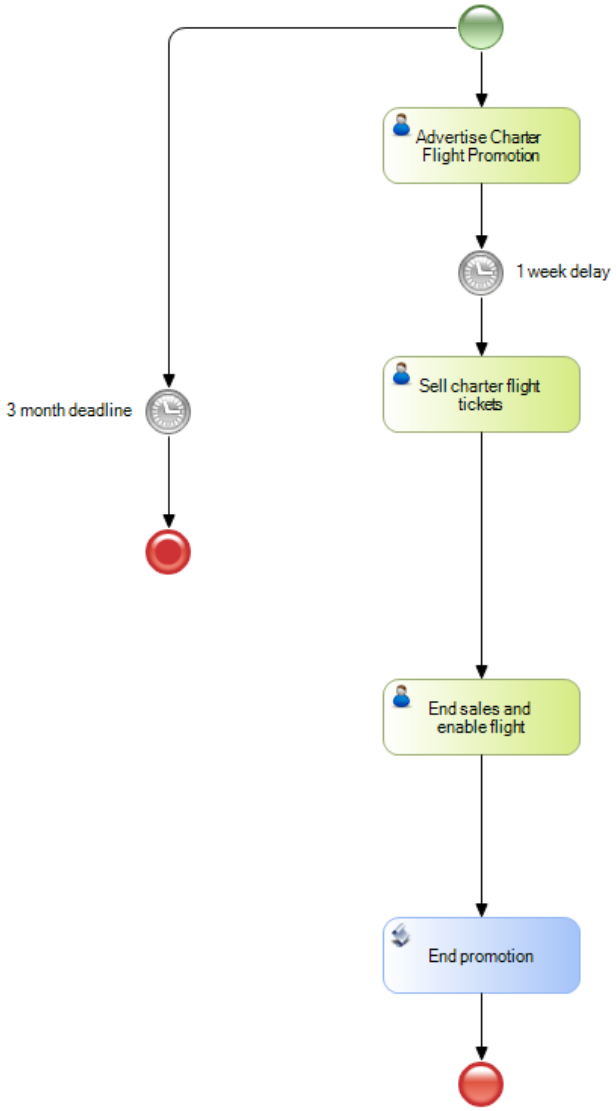
When all the charter flight tickets have been sold and the flight is enabled to operate, we must remove the promotion that we had published and end the process. To do so, we drag a task of script type called **End promotion** and connect it from the previous task. Lastly, we drag a None End Event to end the process.



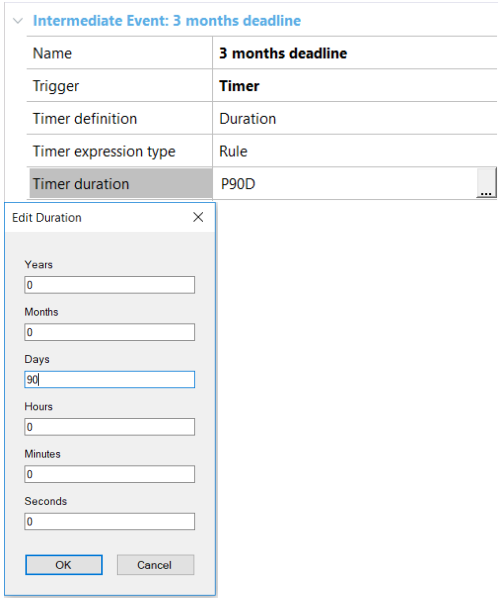
What we have modeled so far assumes that all of the charter flight tickets are always sold in optimum time, but we still have to add several controls and requirements that the agency has asked for.

First, we must make sure that the entire process doesn't last more than 3 months. To this end, we drag an intermediate timer event, join it from the beginning of the process –that is to say, from the None Start Event– and give it the description “3-month deadline.”

To make sure that the process actually ends in this period, meaning that all the process flows are ended, we drag a Terminate End Event from the toolbar.

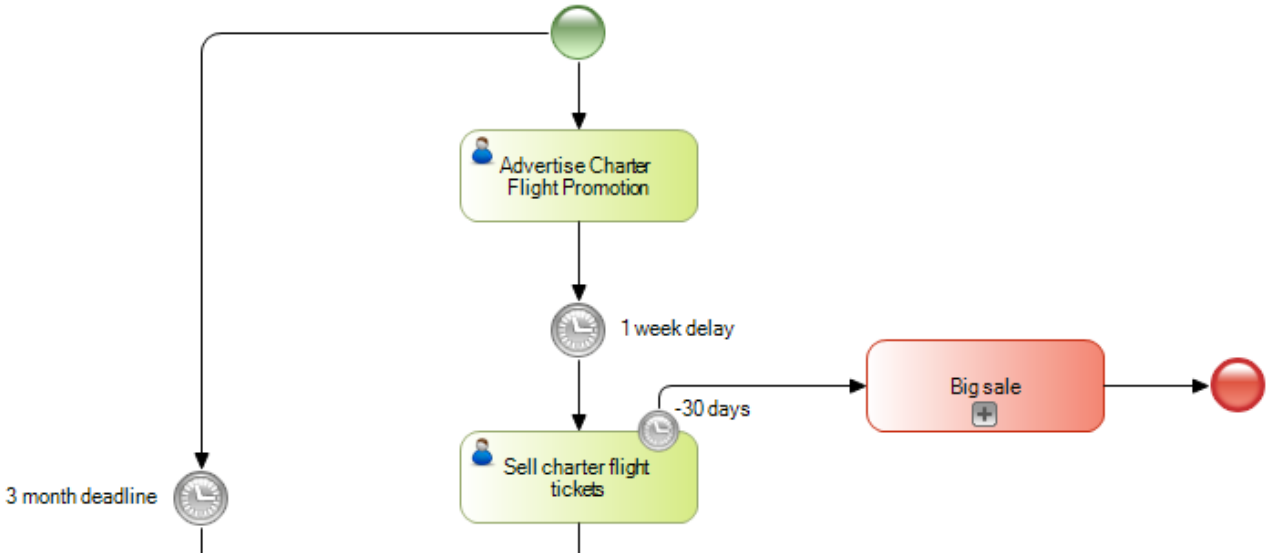


In the timer properties, we assign the Days time unit and set the value 90. It will last 90 days, which is the equivalent to the 3 month-period requested.



Now we will model the various notifications or deadlines involved in the sales process of a charter flight tickets.

The first action takes place when 30 days are left to reach the maximum period allowed for the sales process. At this point, the advertising and sales efforts must be increased.
To model this, we drag an intermediate event of timer type from the toolbar and drop it on the task **Sell charter flight tickets**. We add the description "**-30 days.**"



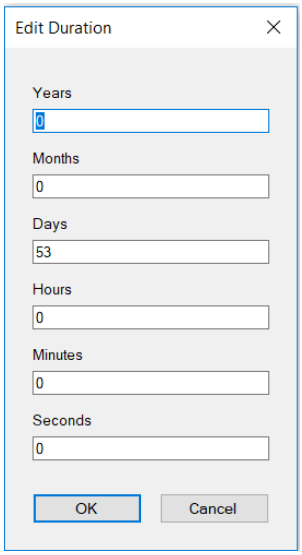
Next, we insert an embedded sub process called Big Sale and connect it from the timer; also, we insert a None End Event and connect it from the sub process.

In the timer properties we placed on the task, we set **Timer usage** to **Warning**, because we want a warning to be displayed when the sub process starts to be run. Note that after selecting the Warning value, the property **Interrupts activity** disappears. The reason is that it doesn't make sense because we only want to show a warning.

Remember that the time set in the timer event starts to count when the task containing the inserted event is executed. Since we want the sub process that increases advertising to begin 30 days before reaching the maximum period of 90 days allowed for the entire process, we have to make some calculations.

We know that the 90 days start to count when the process is run. Assuming that the task **Advertise charter flight promotion** doesn't consume time, we know that the process **Sell charter flight tickets** starts 7 days after that; that is to say, 83 days before the process deadline. Therefore, the sub process **Big sale** should start 83 – 30 = 53 days after running the task **Sell charter flight tickets.**

In the properties of the intermediate timer event, we set the value 53.

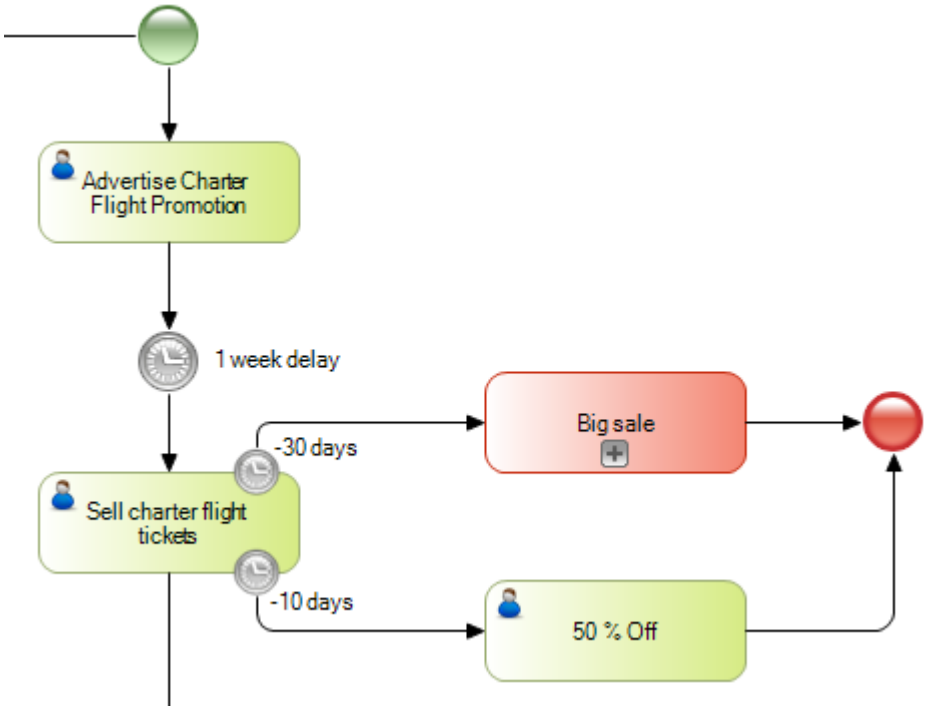


Intermediate Event: -30 days

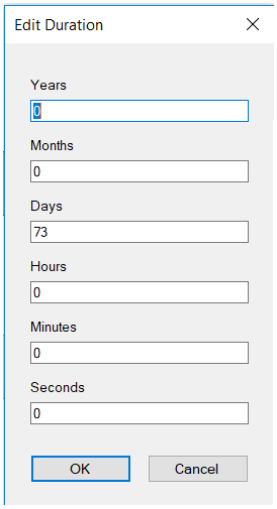
Name	**-30 days**
Trigger	**Timer**
Timer definition	Duration
Timer expression type	Rule
Timer duration	P53D
Timer usage	**Warning**

What we have modeled ensures that the timer event will issue a warning when 30 days are left to complete the process. This warning will be displayed when running the task in the Workflow client inbox.

Now, when 10 days are left to end the charter flight sales process, we want to lower prices so as to ensure that all tickets are sold. To do so, we drag an Intermediate timer event again to the task **Sell charter flight** tickets and assign it the description "**-10 days**." Next, we insert an interactive task and assign it the name **50 % Off** to represent the price reduction process of the flight tickets to half of the initial price. We connect the output of this task to the none end event of the sub process.



We go to the properties of the timer event and set the **Interrupts activity** property to **False**. In the **Timer usage** property, we select the value **None** because we want the price reduction process to start without issuing a warning and without interrupting the ticket sales process. This event must take place 10 days before the end the process; that is to say, 20 days after the warning we modeled before. So, we select Days and enter 73.



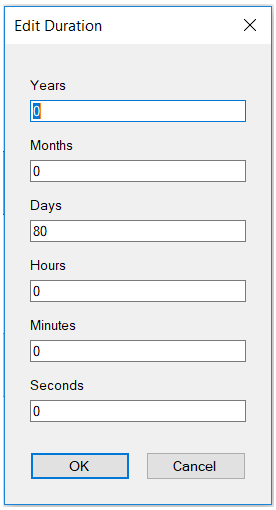
Intermediate Event: -10 days

Name	**-10 days**
Trigger	**Timer**
Interrupts activity	**False**
Timer definition	Duration
Timer expression type	Rule
Timer duration	P73D
Timer usage	**None**

In this way, when 10 days are left to end the process, the price reduction task will be automatically triggered, and the ticket sales task will continue.

Now we will take the last action before reaching the deadline to sell the charter flight tickets: 3 days before ending the process, if 95% or more of the tickets have been sold, the sales process will be closed and the charter flight will be automatically enabled. We want this control to be a deadline that interrupts the sales task. If the desired percentage is not reached, we leave this decision to the manager.

For this last control, we drag another timer event to the task **Sell charter flight tickets** and call it "**-3 days**." In the **Timer usage** property we leave the default value –Deadline– and in the **Interrupts activity** property we also leave the default value –True. We must have it triggered 7 days after the last deadline; that is to say, we assign the value 80.

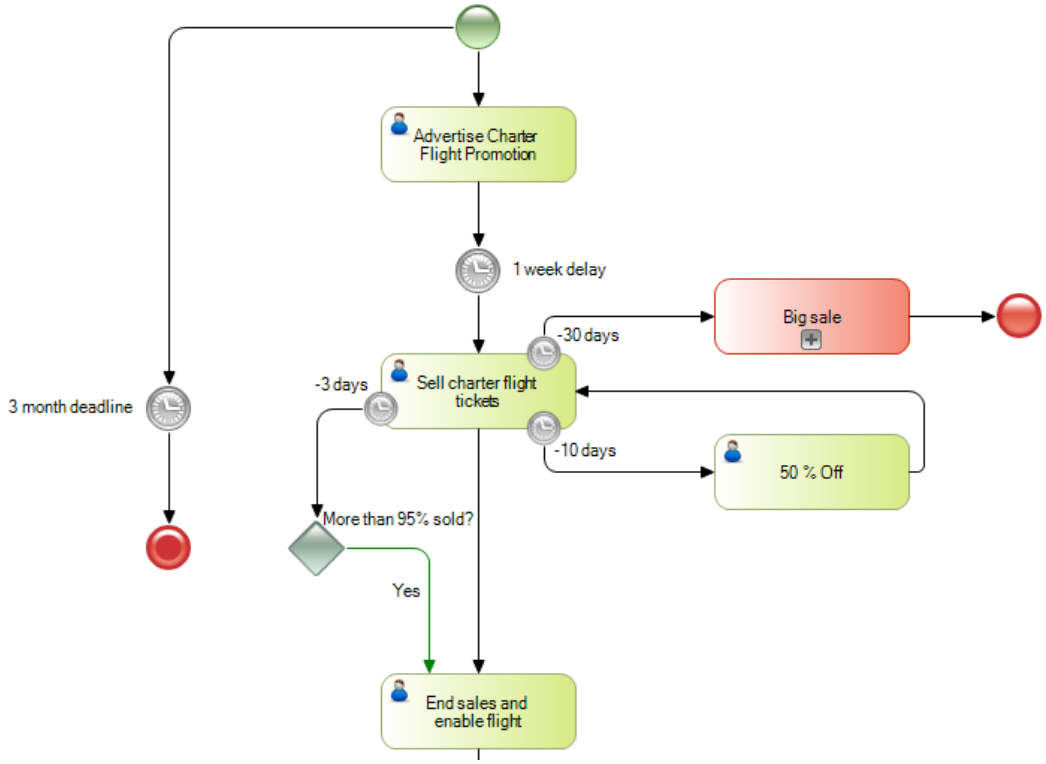


Intermediate Event: -3 days

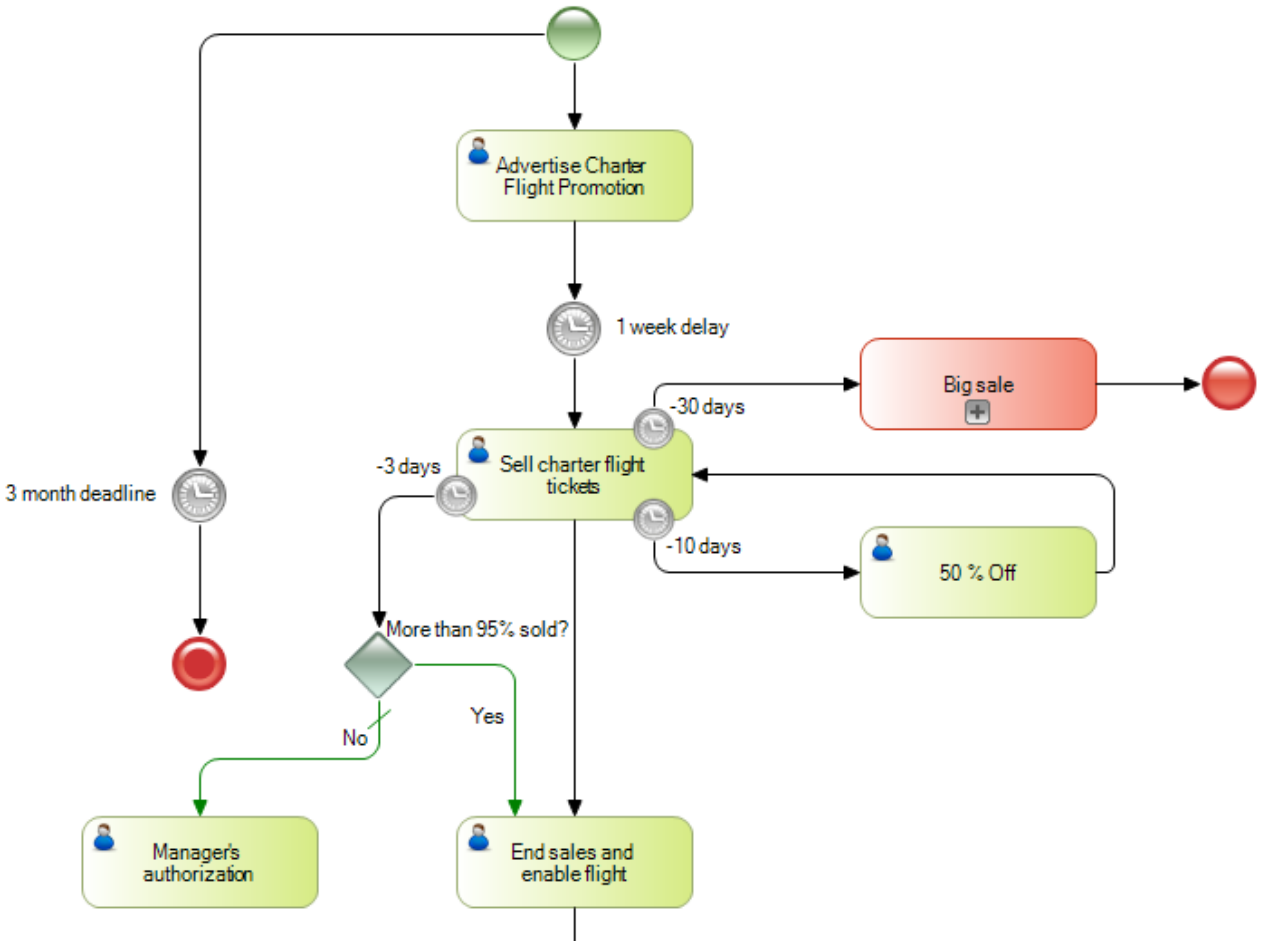
Name	**-3 days**
Trigger	**Timer**
Interrupts activity	True
Timer definition	Duration
Timer expression type	Rule
Timer duration	P80D
Timer usage	Deadline

These 80 days plus the 7 days we waited to start selling the tickets total 87 days, which means that this deadline will be triggered 3 days before the 90-day limit set for the entire process.

As for the percentage of sales reached, we assume that there is relevant data that stores this information, and that it is continuously updated from the ticket sales task.
To check whether 95% of the sales have been reached, we insert an exclusive Gateway joined from the timer event "**-3 days"** and add the description "**More than 95% sold?**"
To the right we connect it with the task **End sales and enable flight** and add the "Yes" tag to the flow.



We insert a user task called **Manager's authorization** and join it from the exclusive Gateway. We add a "No" tag to the join and set the Condition Type property to the Default value, because it will be the default path.

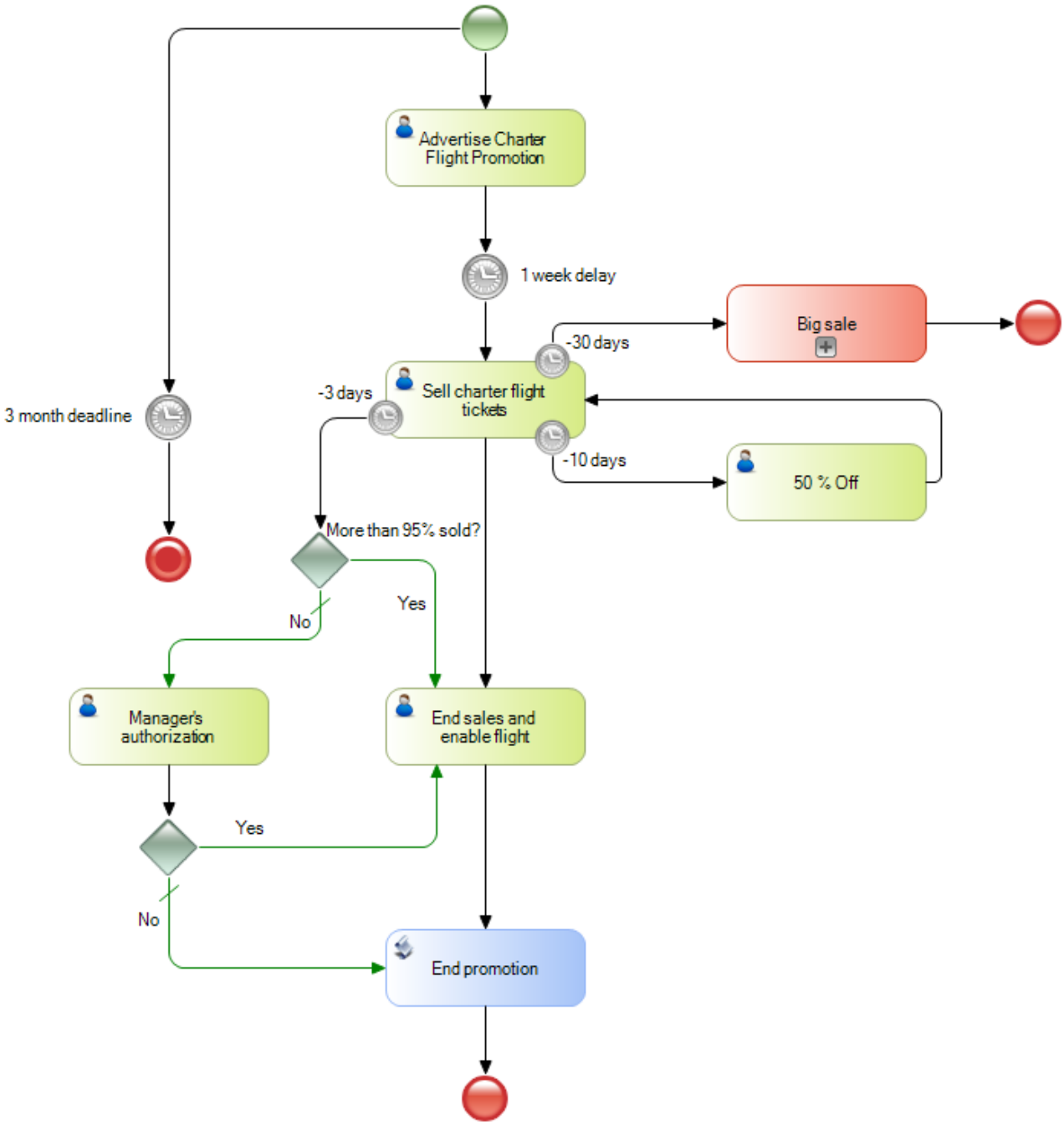


This task represents the manager's action to authorize or cancel the charter flight. To query the decision

made, we insert another exclusive Gateway and connect it from the **Manager's authorization** task.

To the right of the Gateway we connect it with the task **End sales and enable flight**, and add a "Yes" tag to this flow to represent the manager's authorization for the flight to depart. Next, the task that ends the sale and enables the flight will be executed. Lastly, the promotion is removed and the process is ended.

We join the Gateway downwards and connect it with the task **End Promotion**; we add a "No" tag to this flow and set it as the default path. Thus, if the manager doesn't authorize the charter flight, the promotion will be removed and the flight ticket sales process will end.



In this process we had the chance to see how to model delays, control the maximum duration of a process, issue warnings, and manage deadlines, with or without interruption of the task to be monitored.
This encompasses most of the cases in which timer events are used to model time patterns.