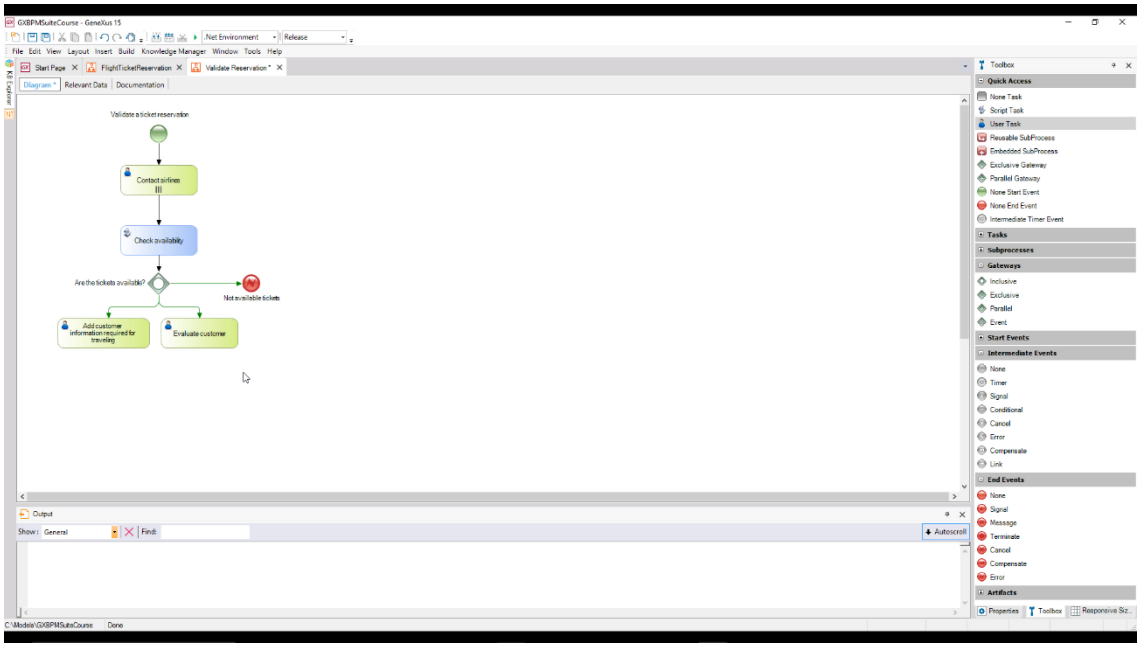
Forking and merging of paths, generation of regular notifications and signal management.

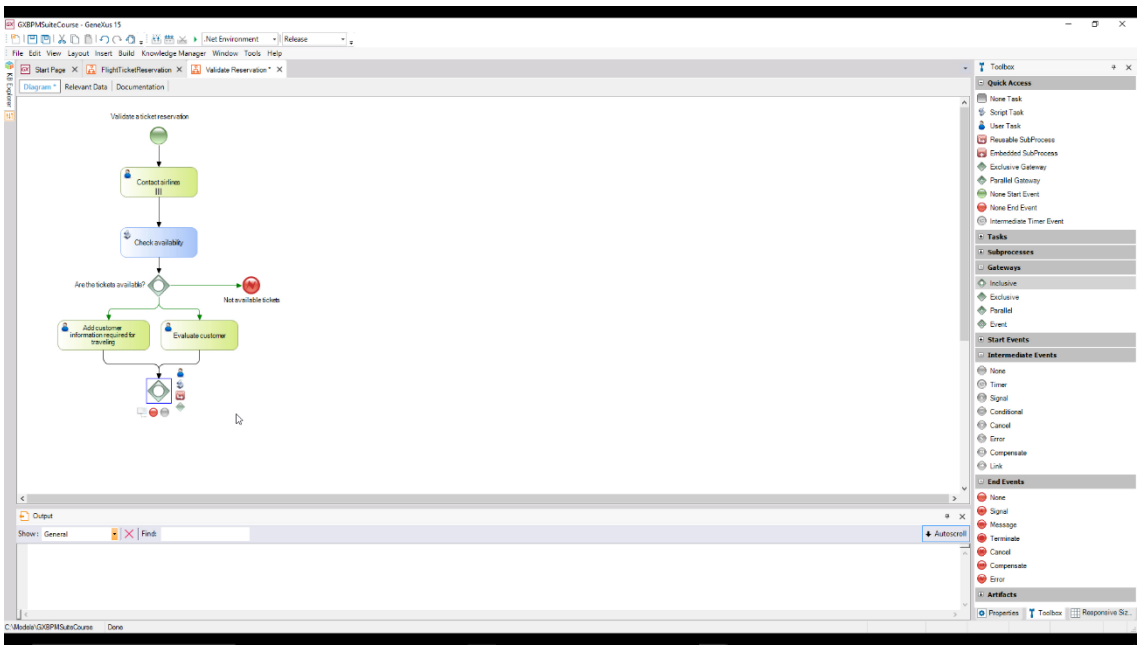
Going back to the model of the reservation validation process, if tickets were available we should continue through 2 paths at the same time. We add the interactive tasks “Add customer information required for traveling” and “Evaluate Customer”, connecting both of them from the inclusive gateway.



In this case, we’re using the inclusive gateway to create branching paths. Each path has a condition defined and the process will follow one or more paths at the same time, depending on what conditions are met.

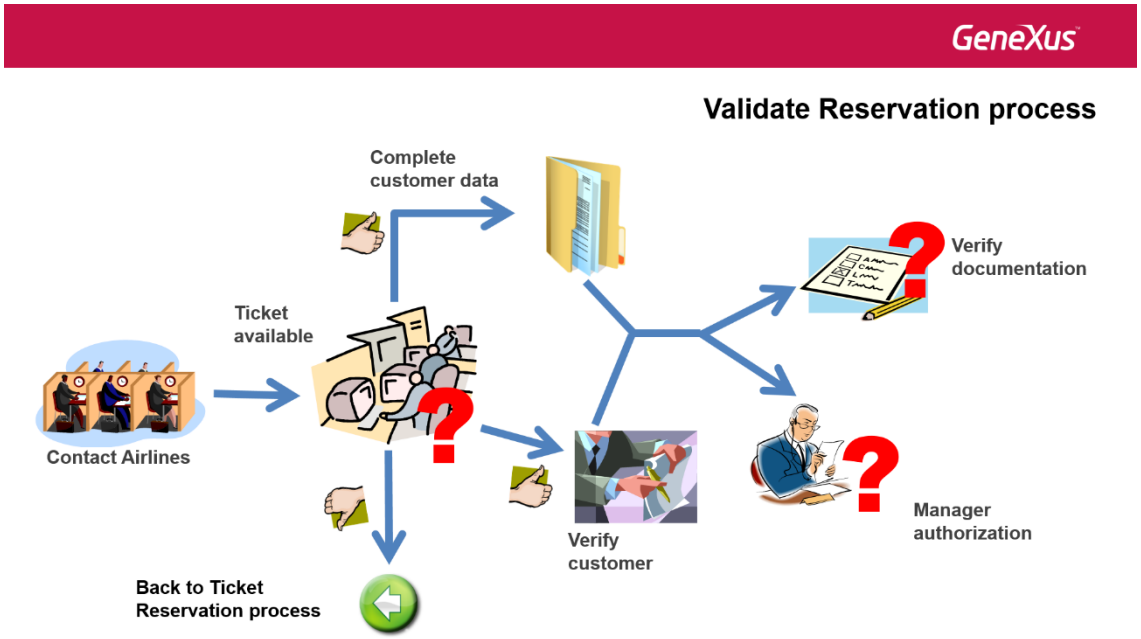
Once the tasks of obtaining information for the trip and checking the customer's financial situation have been completed, we should continue the process. In order to move on, both tasks must be completed.

That is to say, if one task ends before the other, the process will have to wait for the second task to be completed in order to continue. To achieve this, we also use an inclusive gateway, so we drag it from the toolbar and connect it from both tasks.

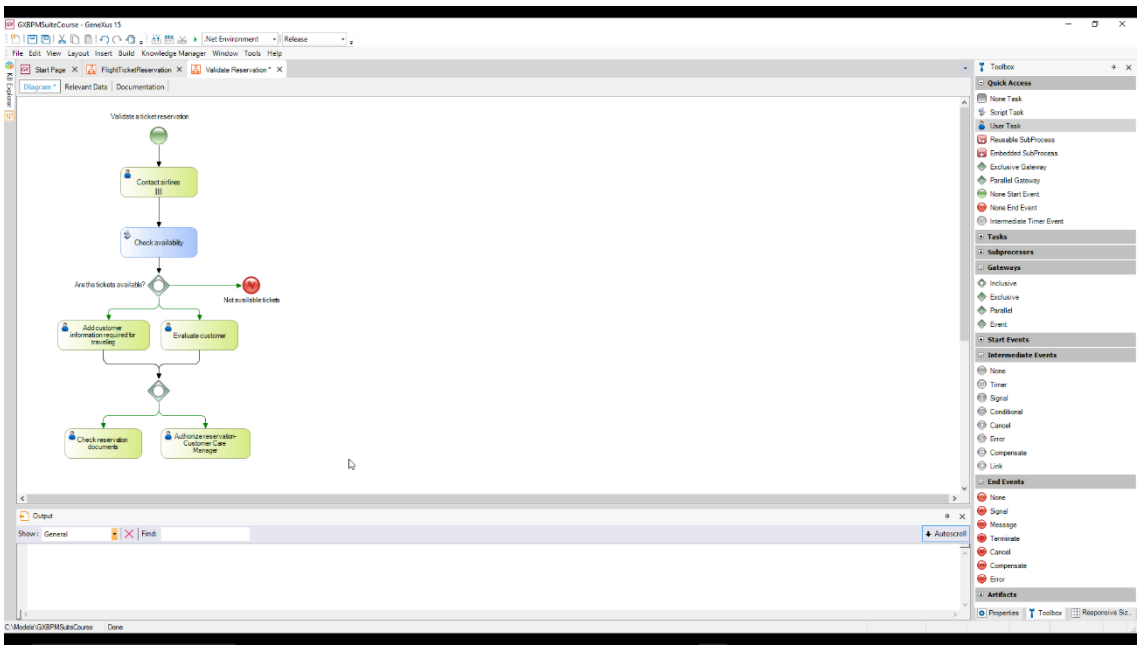


When paths are joined, an inclusive gateway synchronizes the paths through which the process actually moved. These are not necessarily all the paths to the gateway.

In our case, the process can move on from any of both tasks, which are the only incoming paths to the inclusive gateway. Therefore, once they are both completed, our process will continue. It will also involve the execution of 2 activities: checking the necessary details for the trip and obtaining the manager's authorization.



To model the flow divergence into 2 paths we will use an inclusive gateway as in the previous case. So, we make use of the same inclusive gateway we had used to join the paths and create the tasks Check reservation documents and Authorize Reservation -Customer Care Manager, connecting them from this node.

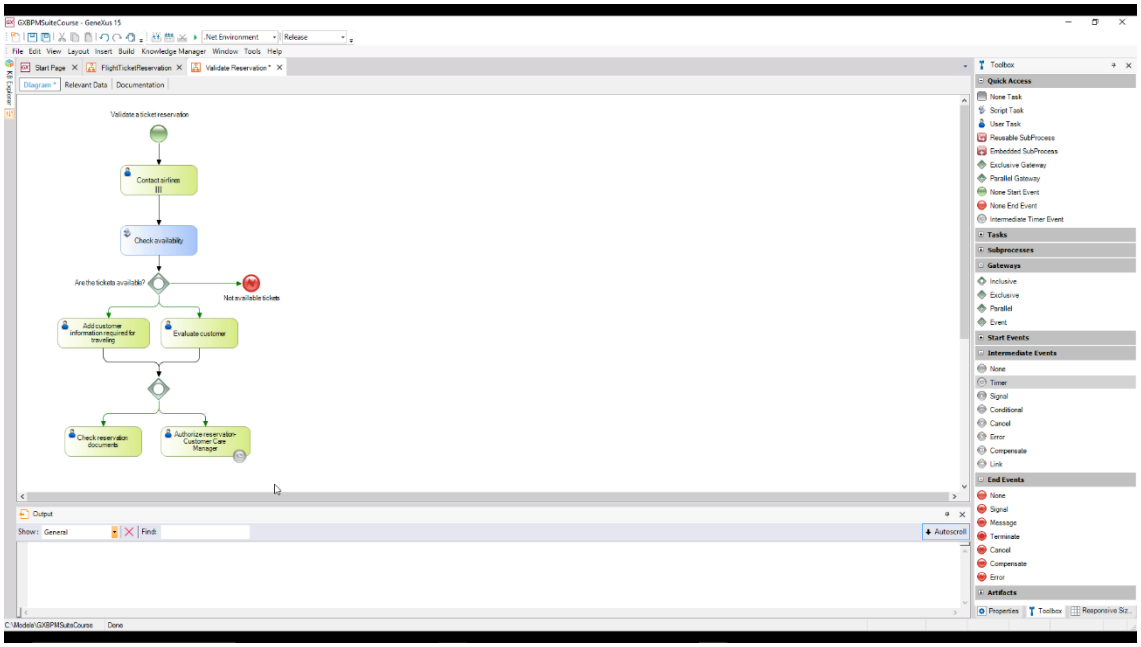


The task executed by the customer care manager must have a certain time frame to be completed, and we want the system to notify him, in a regular and repetitive manner, that a reservation is pending authorization.

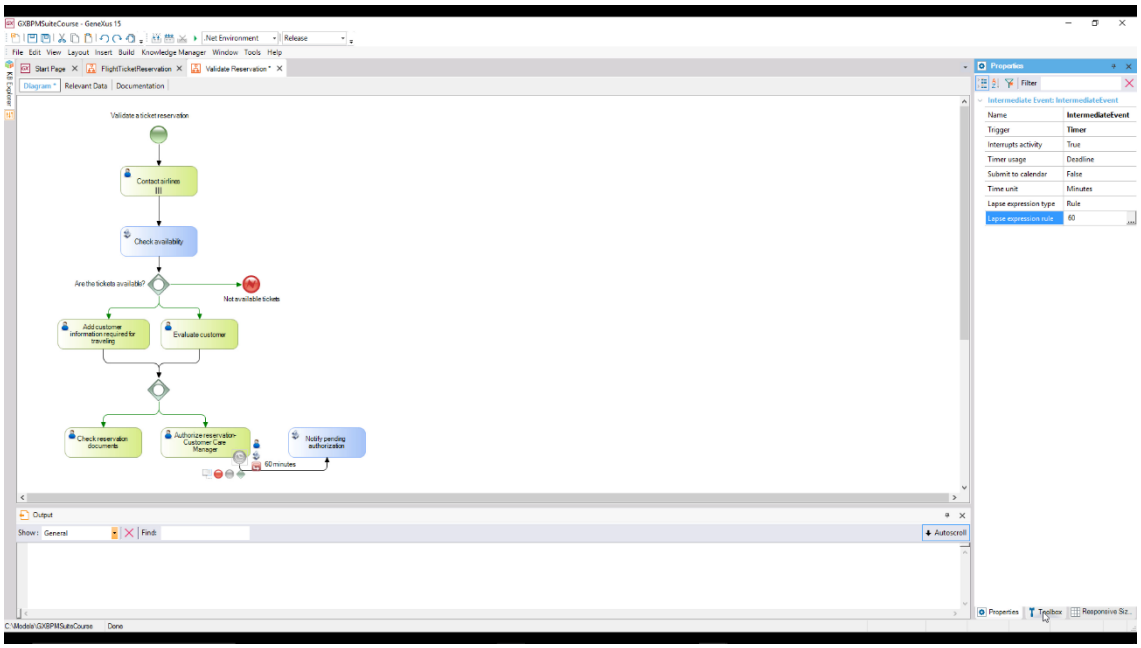
To mark the time elapsed, we add an intermediate event of timer type to the task.

An event is something that happens during the execution of a process; it can start, pause, interrupt or end said process. Depending on the nature of the event, we can classify it as a start, intermediate or end event. We have already used the easiest examples of start and end events, such as none start event and none end event, respectively.

Now we will use an intermediate event of **timer** type. The timer event associated with a task will interrupt its execution after a certain period of time has elapsed, thus limiting the duration of the task.

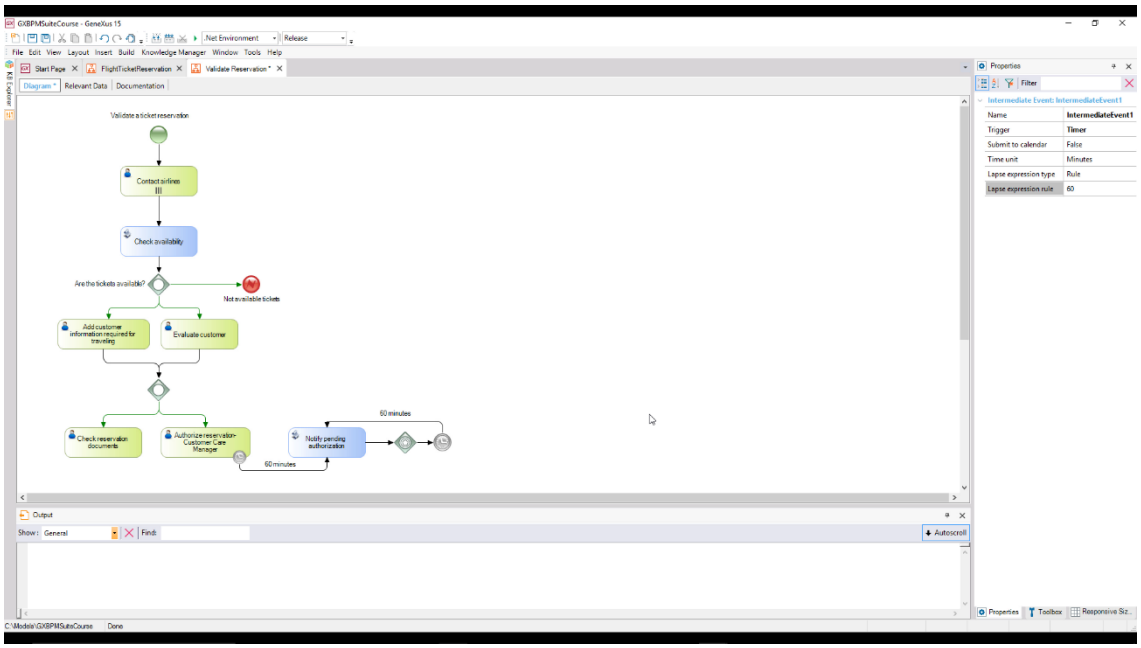


The notification is automatically displayed, so we add a script type task with the description “Notify pending authorization” and connect it from the timer event. We tag this connection with the period of time between regular notifications, which in this case is 60 minutes.



We also open the properties of the intermediate timer event and in the Lapse expression rule property we type 60.

Once the warning is displayed, we want it to be repeated every 60 minutes. To achieve this, we use an Event Gateway and a new timer whose output is connected to the batch task. We also set this connection to 60 minutes and adjust the timer property to do so.

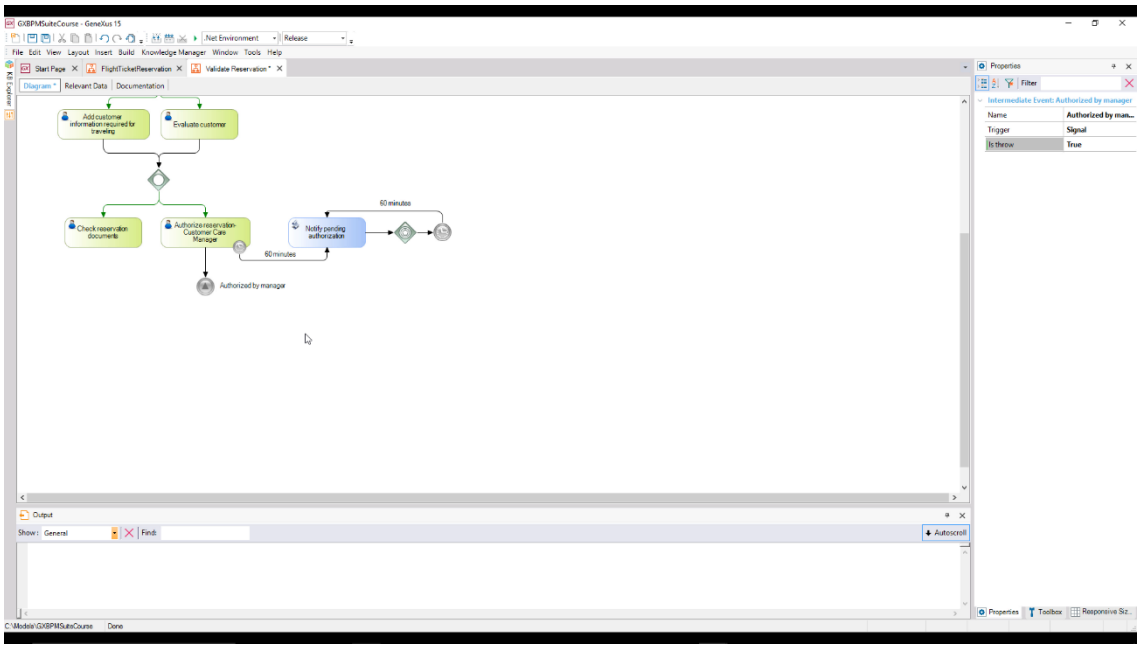


The gateway event is used so that the process continues through a certain path only if a specific event occurs. It’s a particular case of the exclusive gateway because only one path can be followed; in this example it doesn't depend on a condition, it depends on the occurrence of an event.

Continuing with our model, if the manager authorizes the reservation, this notification cycle has to be interrupted. To generate a notification, we need a signal to be triggered and detected to interrupt the notifications.

To model an event that **throws** a signal, we add a symbol of “signal intermediate event”, we type the description “Authorized by manager” and connect it from the task “Authorize reservation - Customer Care Manager”.

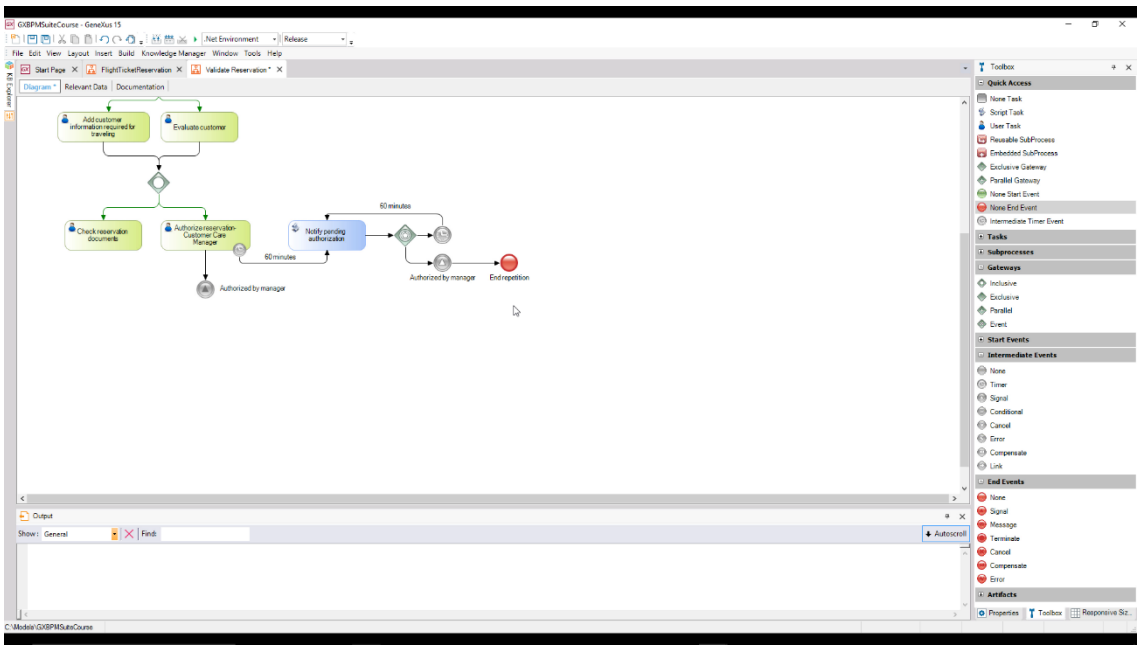
Since we want to configure the signal to be generated, we change the **Is Throw** property and set it to True.



Now we must **catch** this signal in the notification generation process.

To do so, we add another “signal intermediate event” with the same description as that generated by the signal “Authorized by manager” and connect it from the Gateway event. Next, we add a none end event and connect it from the signal event.

Since this signal intermediate event has the function of capturing the signal, we set the Is Throw property to False.

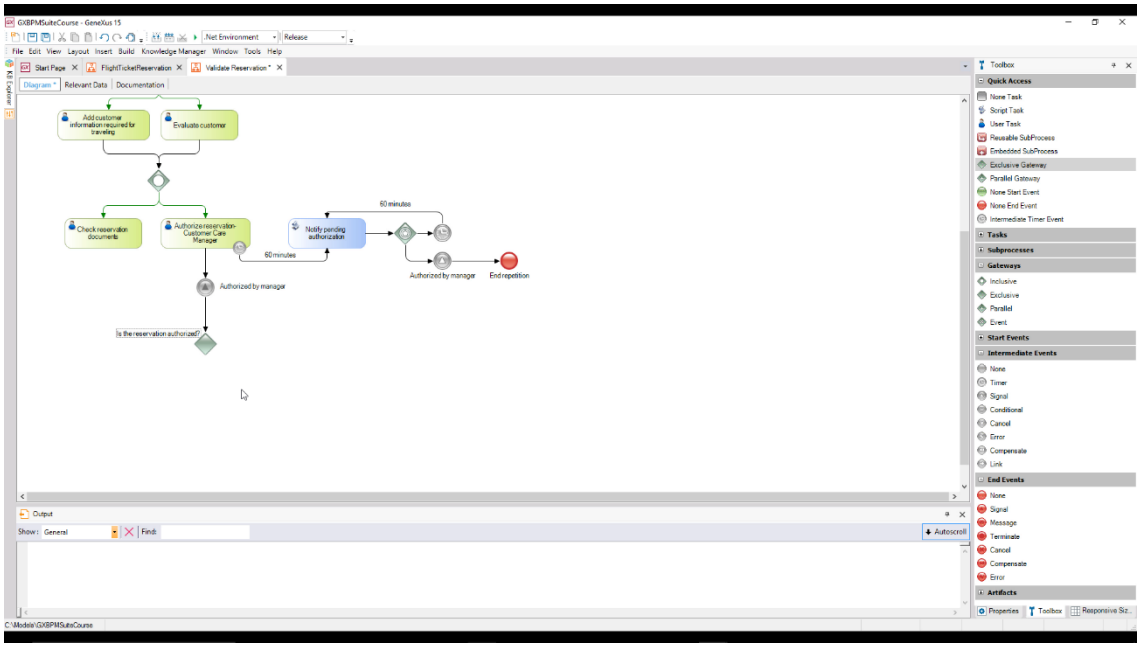


All the types of intermediate events can **catch** events, and some of them can also **throw** them. When a process flow reaches a *catch* type event, the process stops until the expected event occurs. On the other hand, when the flow reaches a *throw* type event, it is thrown right away and the flow moves on.

The types of events that catch events have outline icons, and the events that throw events have filled icons. This can be seen in the two signal events that we added.

Continuing with the process, once the customer care manager has examined the reservation, it may happen that it has been authorized or rejected.

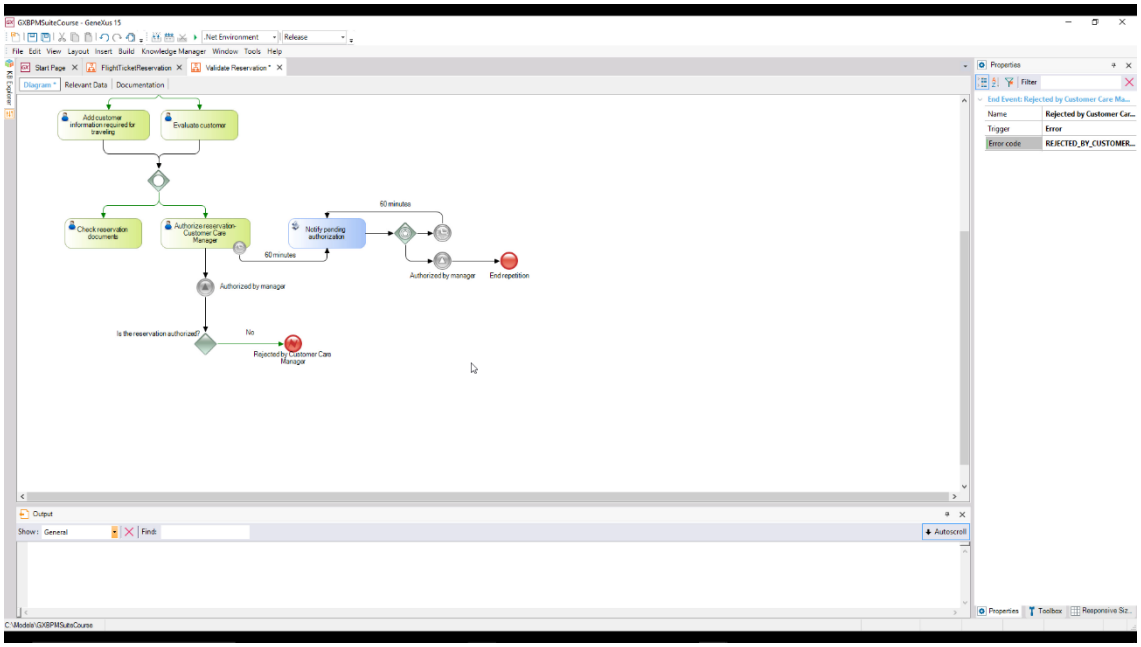
To depict this, we add an exclusive gateway with the description “Is the reservation authorized?” and connect it from the signal intermediate event.



If the reservation is not authorized, we must send an error signal to the FlightTicketReservation process, so that it notifies the customer about the rejection. To achieve this, we add an error end event connected from

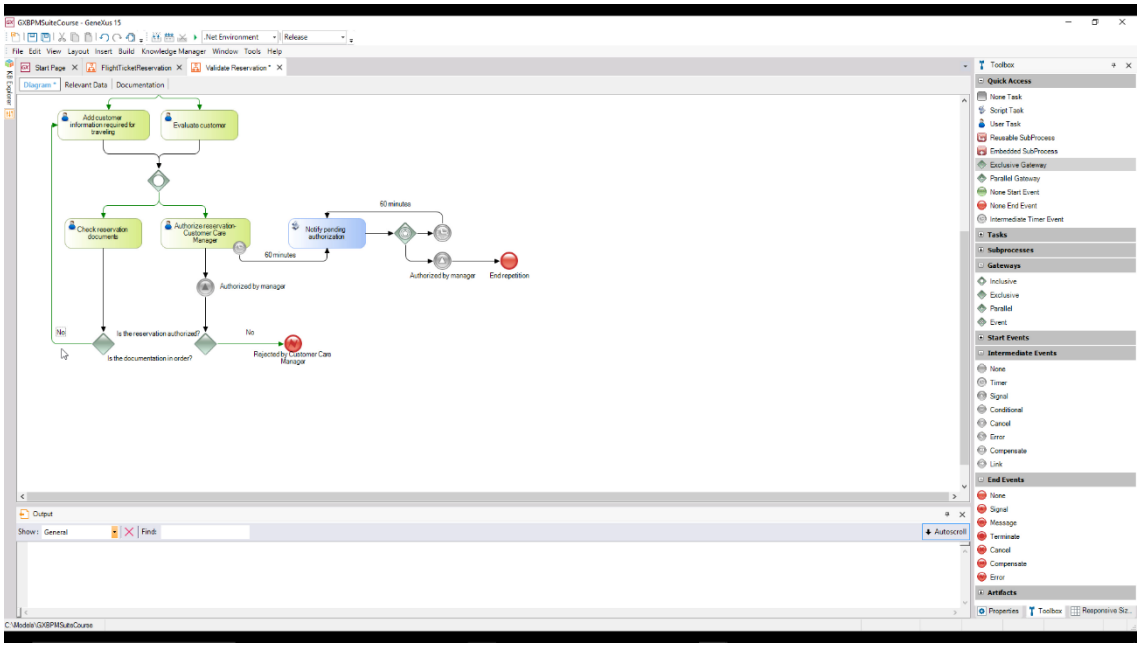
the exclusive gateway, we add a “No” tag to the connection and the description “Rejected by Customer Care Manager” to the end event.

In the Error Code property we type **REJECTED_BY_CUSTOMER_CARE_MANAGER**.

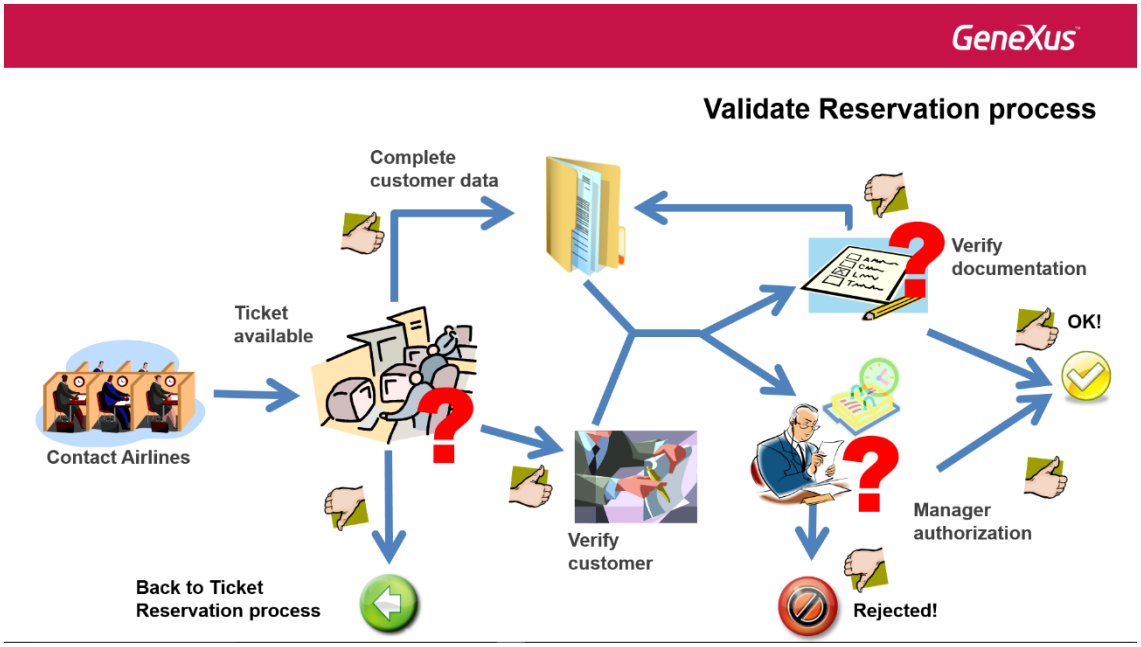


After completing the task of reviewing the documentation provided, it may happen that it isn’t in order. In this case, the system must go back to the data collection task. To perform this checking, we add an exclusive gateway with the description “Is the documentation in order?” and connect it from the task “Check reservation documents”.

Lastly, we connect the gateway to the task “Add customer information required for traveling” and to the connector we add the “No” tag.

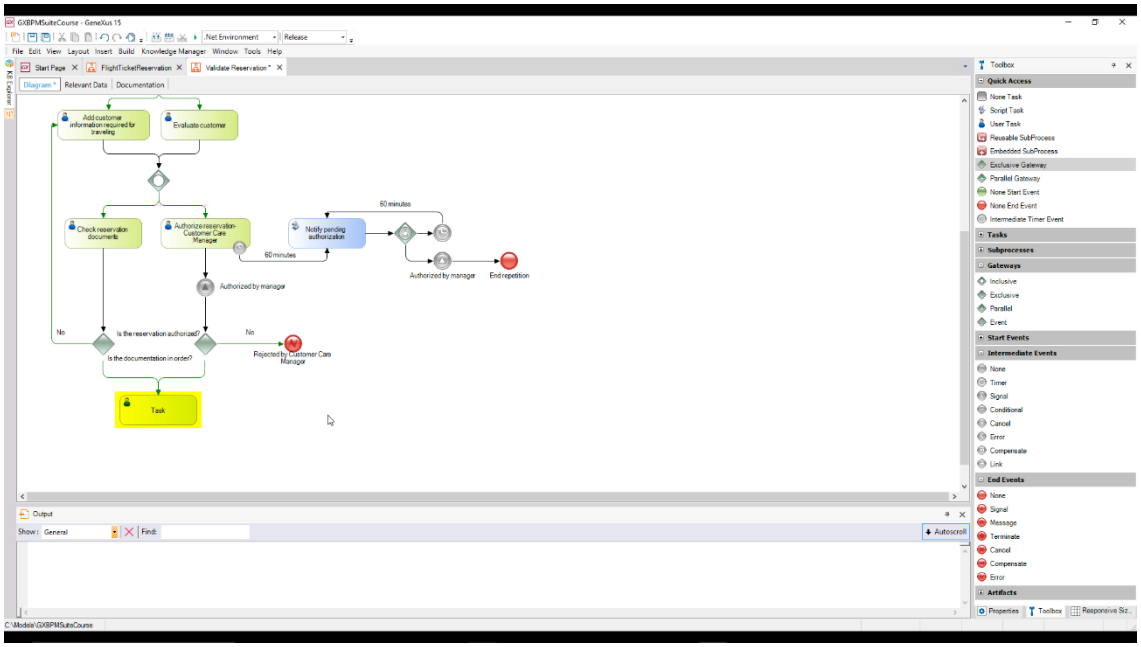


Once the documentation has been successfully verified, and the customer care manager has issued an authorization, the reservation must be considered as valid.



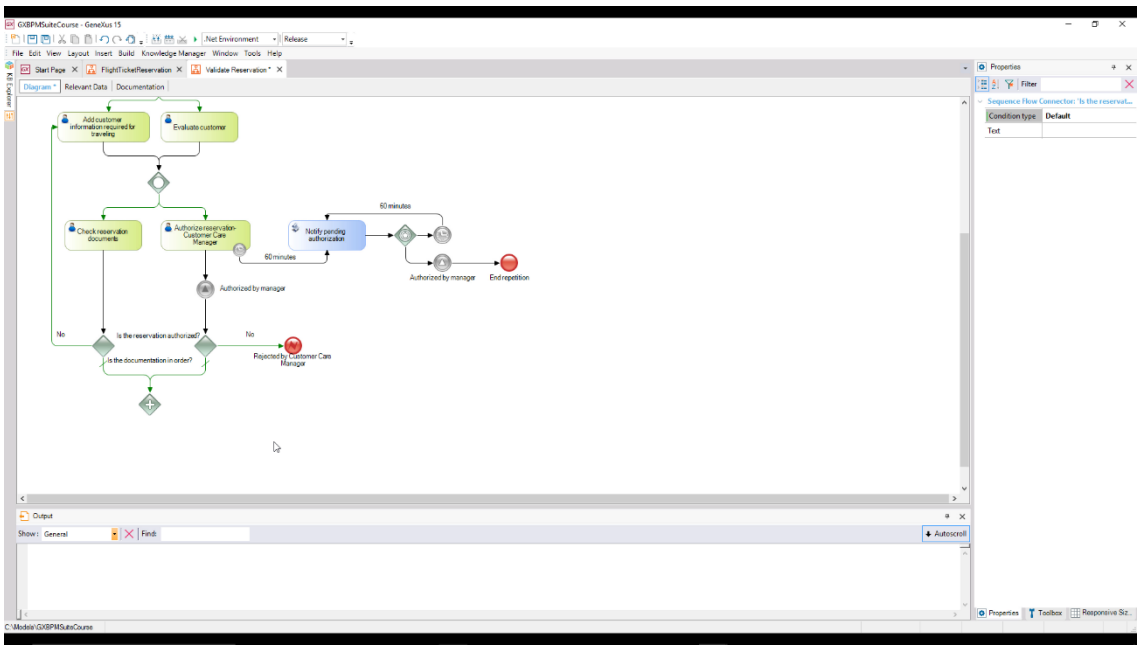
In order to continue, both tasks must be completed. If necessary, the system must wait for a task to end.

Synchronization is necessary, because if we simply joined both paths…



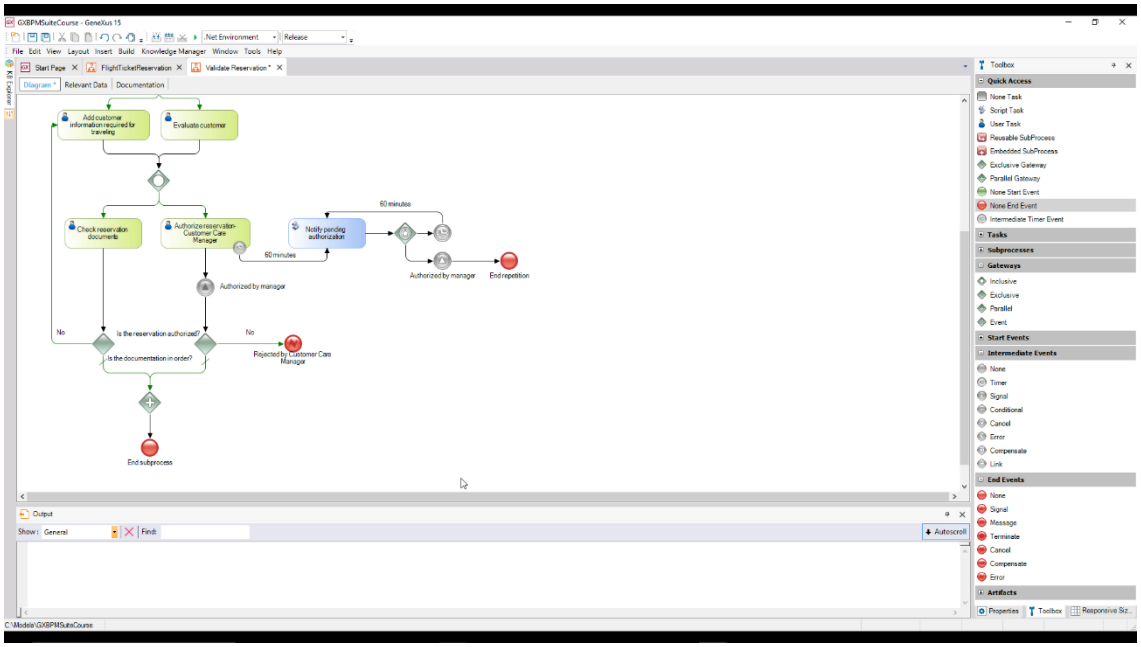
… it may happen that one of the previous stages is executed first, so the following task would be executed. Next, when the pending stage is executed, the task following the join would be executed for the second time!

To perform this synchronization, in which we need to wait for both paths, we add a **parallel exclusive gateway** and join it from the exclusive gateways.

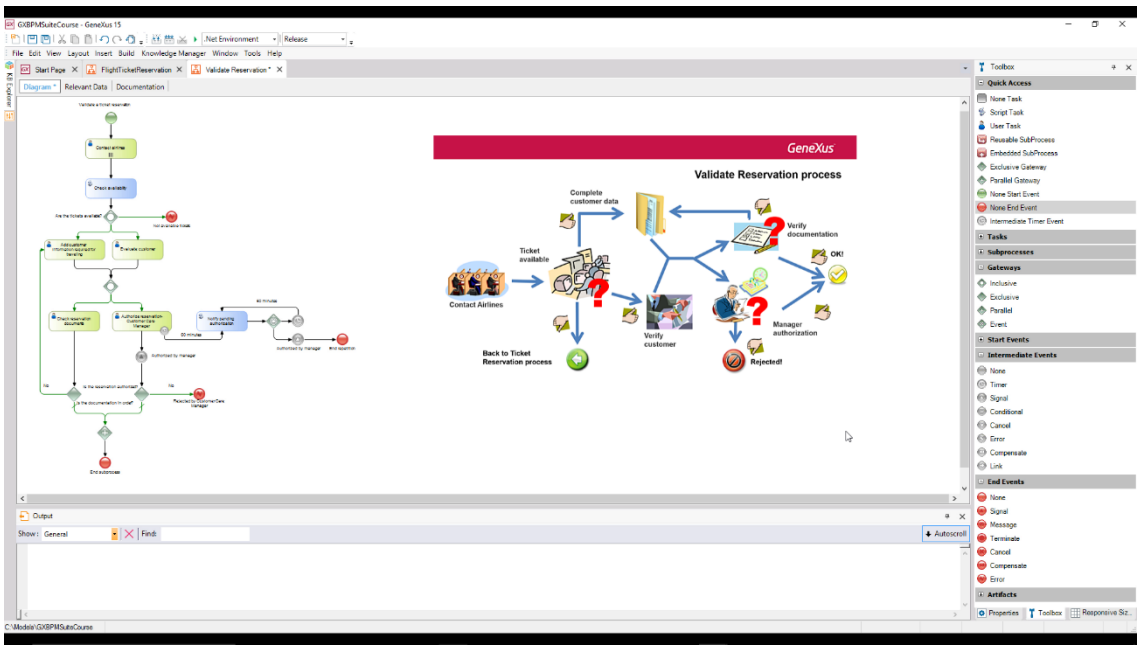


To indicate that both paths coming out of the exclusive gateways are default paths, we set their corresponding Condition type properties to Default.

Once both paths have ended, the ticket reservation validation process must end, so we enter a none end event and connect it from the parallel gateway.

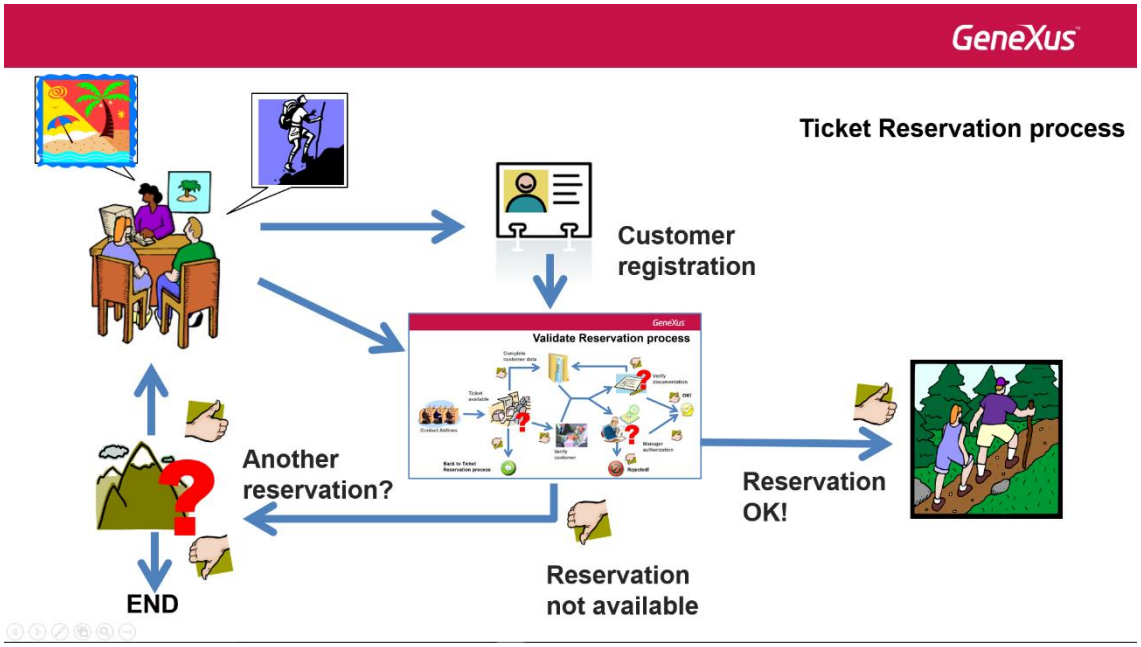


We right-click on the diagram and adjust the zoom by Zooming Out until we can see the entire diagram.



In this way we've completed the diagram of the ticket reservation validation sub-process…

which is part of the main process of airline ticket reservation.



This video, together with Modeling – Part One, has given us a process modeling overview, a short introduction to the BMPN standard and the advantages provided by GeneXus to build diagrams that follow this standard.

In upcoming videos we will see other aspects of business process modeling, such as their automation, monitoring, optimization, deployment and maintenance.