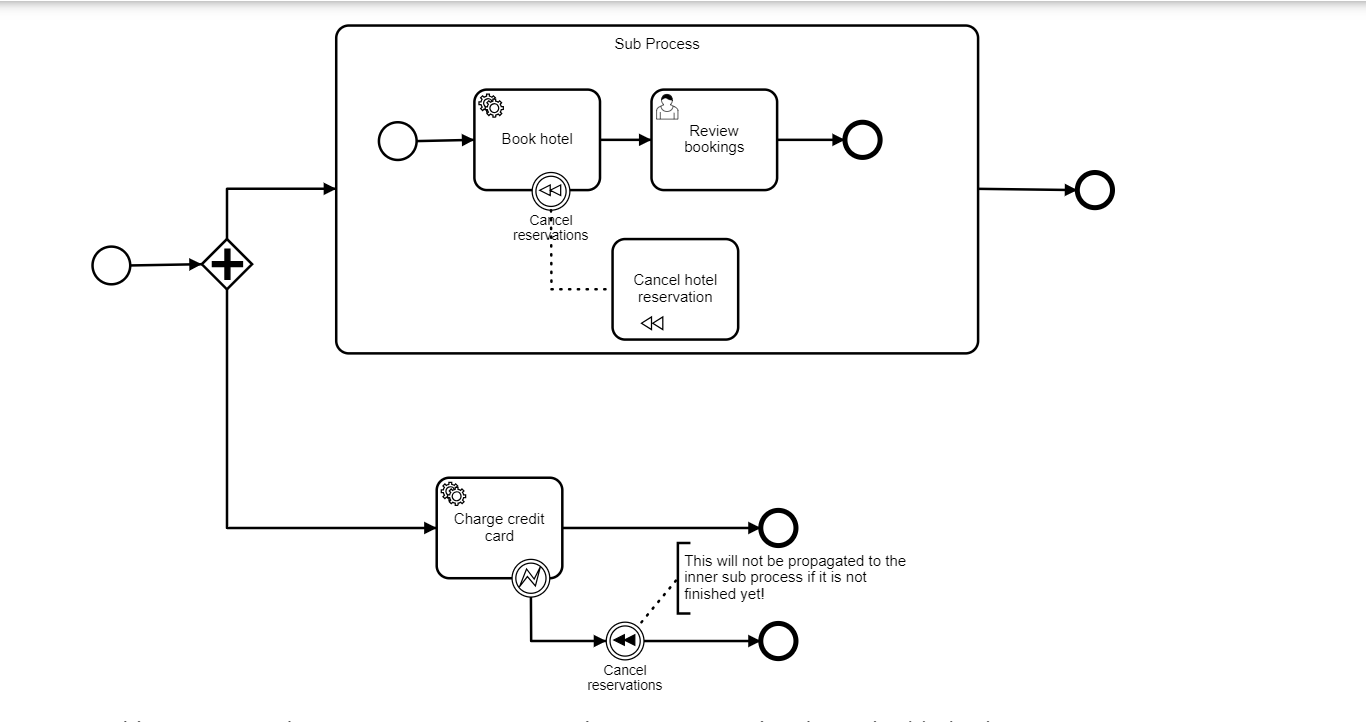
**How Camunda Compensation Event would help us to improve the Business processes in different domains?**

****

Compensation activities are useful for modeling business processes that involve transactions, contracts, or agreements that may need to be rolled back or modified in certain situations. The intermediate throwing compensation event can be used to compensate transaction subprocess which completed successfully

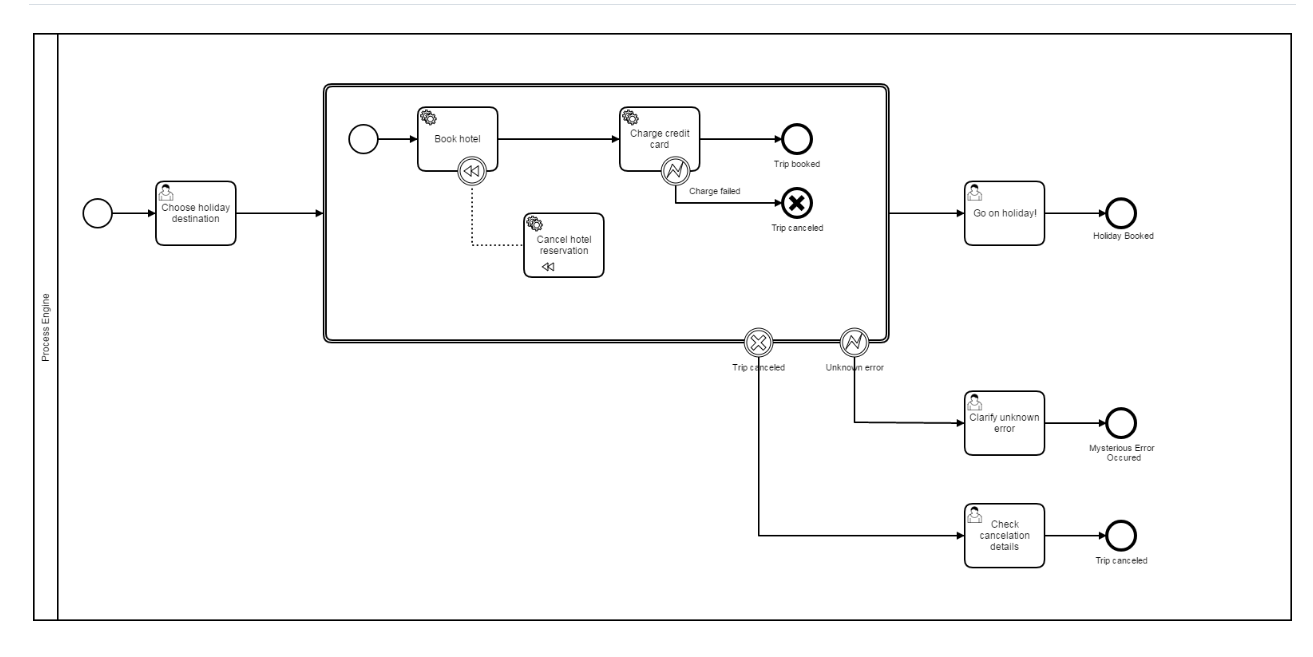
In this process we have two concurrent executions, one executing the embedded subprocess and one executing the “charge credit card” activity. Lets assume both executions are started and the first concurrent execution is waiting for a user to complete the “review bookings” task. The second execution performs the “charge credit card” activity and an error is thrown, which causes the “cancel reservations” event to trigger compensation. At this point the parallel subprocess is not yet completed which means that the compensation event is not propagated to the subprocess and thus the “cancel hotel reservation” compensation handler is not executed. If the user task (and thus the embedded subprocess) completes before the “cancel reservations” is performed, compensation is propagated to the embedded subprocess.

Note: When compensation is thrown for a multi instance activity, the associated compensation handler is only executed when all instances of this activity have ended. That means, the multi instance activity must be ended before it can be compensated.

Compensation event – Use Case

In the below workflow when the start event is triggered “Choose holiday destination” user task is activated then once the user claims and completes the task the flow goes to embedded subprocess and then “Book hotel” service task is executed and say for any reason booking is not done then Compensation boundary event is triggered and “Cancel hotel reservation” service task is executed in this case the information filled by the user is no longer needed in our DataBase so for this Compensation event itself rolls the data back or revert the data.

In any such scenario, we can go for Compensation Event.



## **Compensation Boundary Event**

In the above workflow we have used compensation boundary event can be used to attach a compensation handler to an activity or an embedded subprocess.

A compensation boundary event has a different activation policy than other boundary events. Other boundary events, such as the signal boundary event are activated when the activity they are attached to is started. When the activity is left, they are deactivated and the corresponding event subscription is canceled. The compensation boundary event is different. The compensation boundary is activated when the activity it is attached to completes successfully. At this point, the corresponding subscription to compensation events is created. The subscription is removed either when a compensation event is triggered or when the corresponding process instance ends.

This leads to the following points:

* When compensation is triggered, the compensation handler associated with the compensation boundary event is invoked the same amount of times that the activity it is attached to completed successfully.
* If a compensation boundary event is attached to an activity with multiple instance characteristics, a compensation event subscription is created for each instance.
* If a compensation boundary event is attached to an activity which is contained inside a loop, a compensation event subscription is created for each time the activity is executed.
* If the process instance ends, the subscriptions to compensation events are canceled.