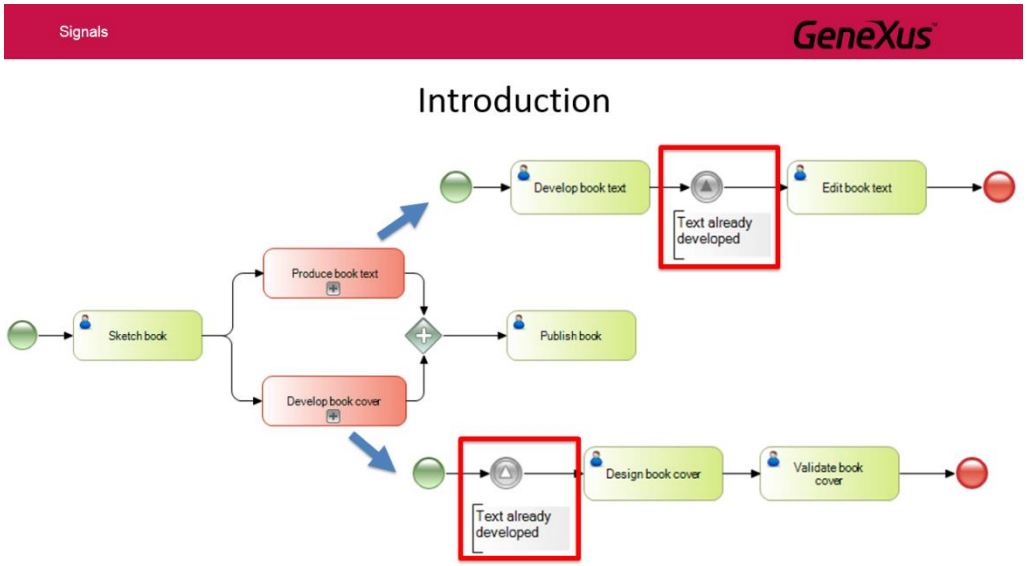
Eventos do tipo Sinal

Neste vídeo, veremos alguns usos do evento do tipo Sinal (ou *Signal* em inglês).

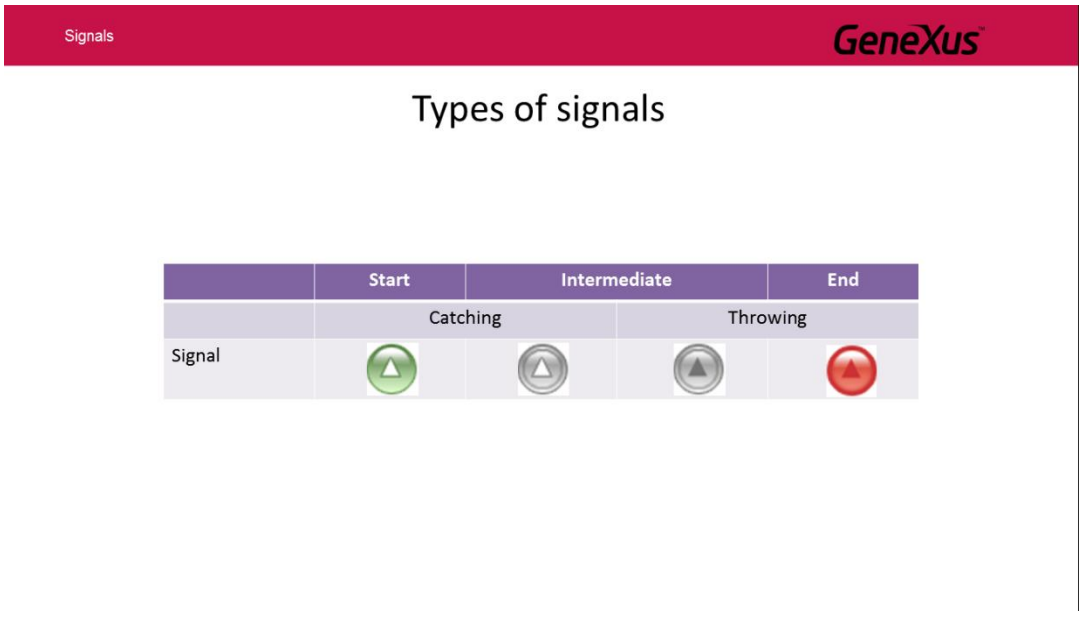
Os eventos Sinal são usados para enviar ou receber sinais dentro ou fora do processo, por isso são úteis tanto para comunicação entre partes de um mesmo processo quanto para comunicação entre processos que estão vinculados por uma relação hierárquica.



Isso implica que é possível sinalizar a ocorrência de um evento que pode ser detectada em qualquer parte do processo, subprocessos e até processos ancestrais.

O comportamento dos sinais varia dependendo se são eventos de início, eventos intermediários ou eventos finais.

Por sua vez, esses eventos podem desencadear um sinal (throwing) ou capturar um sinal (catching).



Com base nisso, a notação muda. Por exemplo, os eventos intermediários têm uma borda dupla, enquanto os eventos de início e fim têm uma borda simples. Quando são eventos de disparo, o triângulo é escuro e quando um sinal é capturado, o triângulo é claro.
Um evento de sinal de início permite que um processo comece quando um sinal é recebido de outra parte

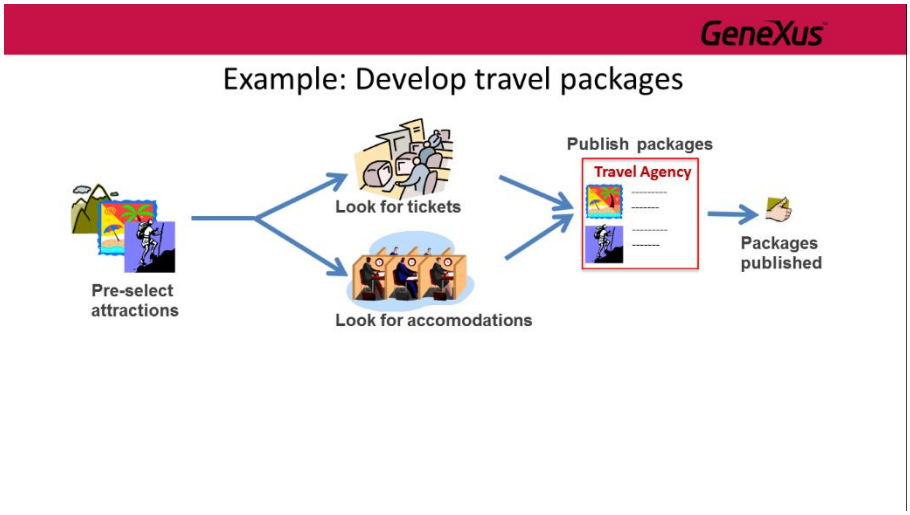
do processo ou processos hierarquicamente relacionados ao processo onde está definido. Por esta razão, eles são sempre o tipo de captura (ou "*catch*").

Um evento de sinal do tipo fim permite que ao finalizar um fluxo de processo se envie um sinal à outra parte do processo ou a processos hierarquicamente relacionados. Eles são sempre de disparo (ou "*throw*").

Um evento de sinal do tipo intermediário pode ser de disparo ou de captura, dependendo do valor de sua propriedade "Is throw" (True ou False, respectivamente) e podem ser colocados em qualquer parte do processo, geralmente entre atividades.

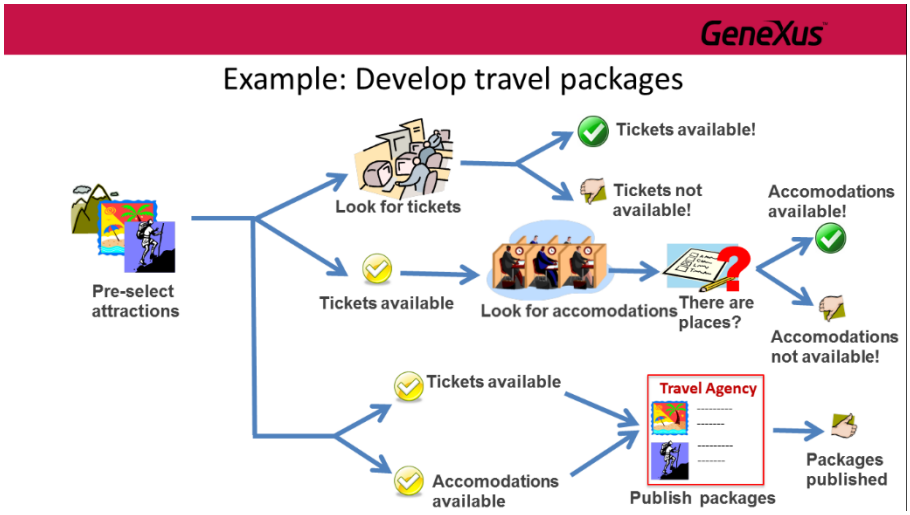
Vamos ver alguns desses símbolos em ação.

A agência de viagens nos pediu para modelar o processo de publicação de pacotes turísticos para seus clientes. Cada pacote envolve várias reservas de bilhetes e várias reservas de hotéis e um pacote só pode ser publicado se houver disponibilidade dessas reservas.



Isso implica que o processo deve garantir um certo número de passagens e quartos, de modo a poder publicar pacotes de uma certa atração turística.

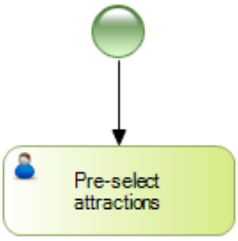
Ao mesmo tempo, o processo deve ser eficiente, ou seja, as consultas sobre as companhias aéreas e os hotéis devem começar ao mesmo tempo, mas se os bilhetes necessários não puderem ser obtidos, a busca de quartos em hotéis deve ser cancelada.



O processo deve garantir que apenas os pacotes sejam publicados, se ambas as passagens e as acomodações estiverem asseguradas.

Para implementar este processo no GeneXus, abrimos o GeneXus Modeler, criamos um objeto do tipo Business Process Diagram e chamamos **DevelopTravelPackages.**

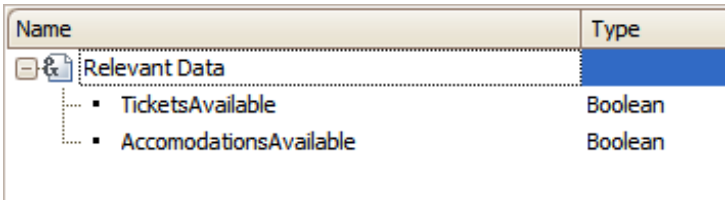
Para começar, arrastamos um símbolo de None Start Event. Em seguida, inserimos uma tarefa interativa, nós a chamamos de **"Pre-select attractions"** e conectamos ao None Start Event.



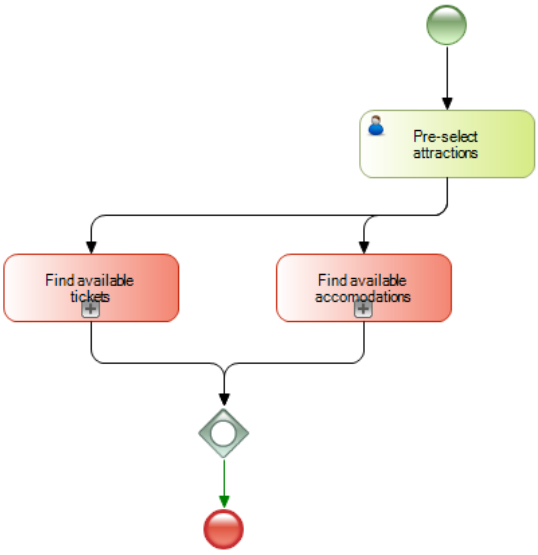
Esta tarefa resume a atividade de analisar quais atrações são mais adequadas para serem incluídas em um pacote. Depois disso, devemos iniciar o processo de busca de passagens e outra busca de acomodação.

Para isso, arrastamos um símbolo de um subprocesso incorporado, nós o chamamos de **Find available tickets** e nós o juntamos a tarefa previamente definida. Nós adicionamos outro subprocesso incorporado **Find available accomodations** e também unimos a ele a tarefa **Pre-select attractions.**

Para que possamos completar o processo de desenvolvimento de pacotes turísticos devemos garantir que ambos os subprocessos tenham sido concluídos com sucesso. Poderíamos fazer isso definindo um dado relevante do tipo booleano que foi configurado em cada processo, dependendo se o processo foi concluído com sucesso ou não.



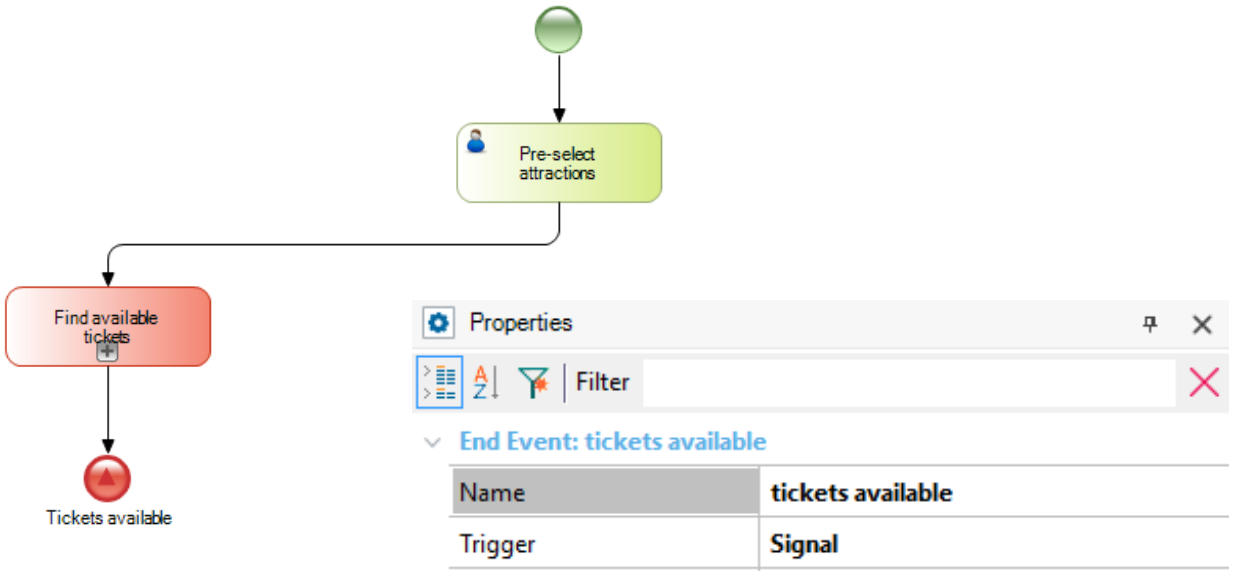
E, em seguida, utilizar um Gateway Inclusivo para sincronizar os caminhos provenientes de cada subprocesso, de modo que apenas o processo de desenvolvimento de pacotes turísticos termina, se ambos



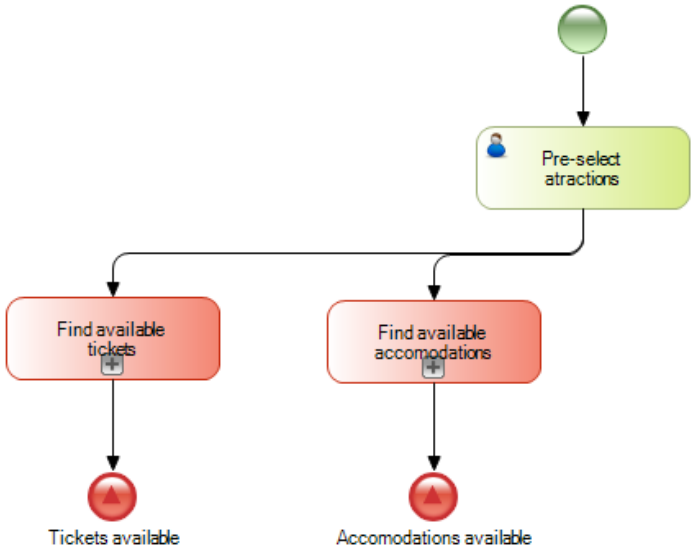
os subprocessos tiverem sido bem sucedidos

Embora esta solução seja possível, envolve a definição e configuração de dados relevantes. Podemos obter um resultado semelhante se usar Sinais, sem a necessidade desses dados relevantes.

Em vez do Gateway Inclusivo, inserimos um End Event do tipo Sinal e o conectamos à saída do subprocesso de buscar passagens disponíveis. Nós atribuímos o nome de **Tickets available** para o sinal e vemos que um **evento final do tipo sinal** já é do tipo "throw" porque o triângulo está escuro. Em um signal end event não podemos alterar o valor de sua propriedade **Is Throw**, uma vez que não está disponível e seu valor é sempre True.

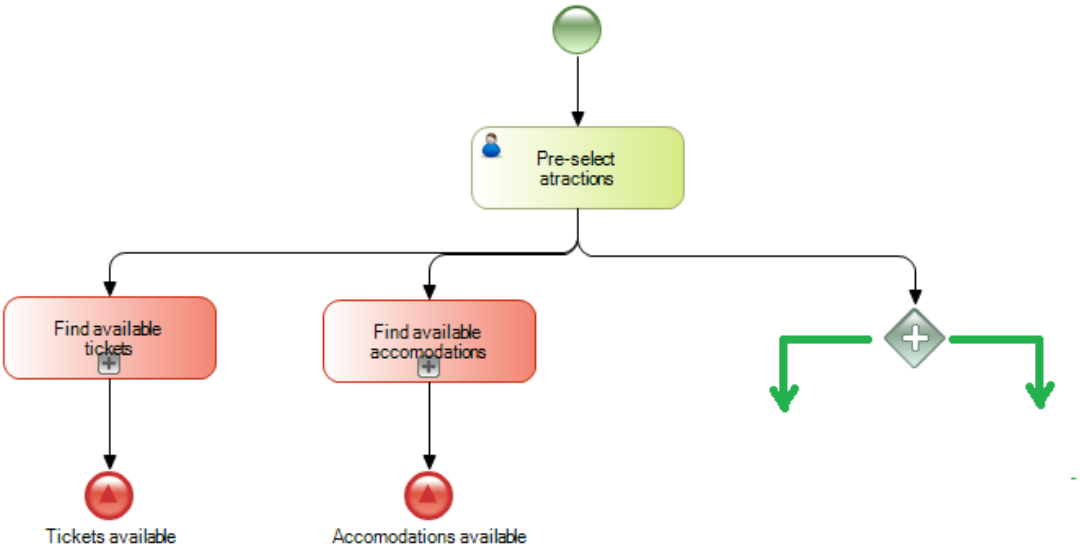


Da mesma forma, inserimos um Signal End Event na saída do subprocesso de encontrar acomodações e nós damos o nome de **Accomodations available**.



Agora devemos capturar esses sinais para poder continuar com o processo de desenvolvimento de pacotes turísticos, ou seja, com a publicação do mesmo e finalmente terminar o processo.

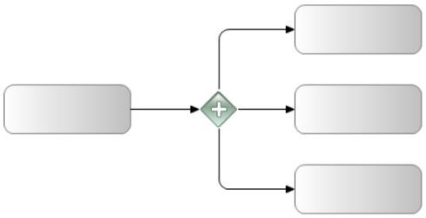
Para isso, inserimos um Parallel Gateway ao qual nos o conectamos a partir da tarefa Pre-select attractions. Isso nos permitirá bifurcar o fluxo em dois caminhos.



Ao contrário do Exclusive Gateway usado quando o caminho que o fluxo do processo deve seguir depende da avaliação de uma condição de tipo verdadeiro / falso, um Parallel Gateway nos permite dividir o fluxo em dois ou mais caminhos, sem avaliar nenhuma condição.

GeneXus

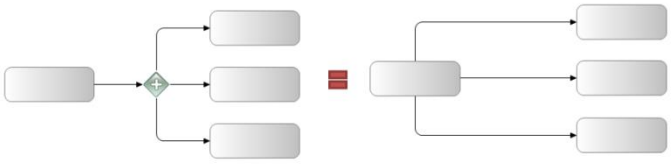
Paralell Gateway



Devido a isso, também é possível modelar esse mesmo comportamento sem usar um **Paralell Gateway**, mas simplesmente juntando dois conectores. No entanto, o uso do **Parallel** gateway pode esclarecer o diagrama em determinadas circunstâncias.

GeneXus

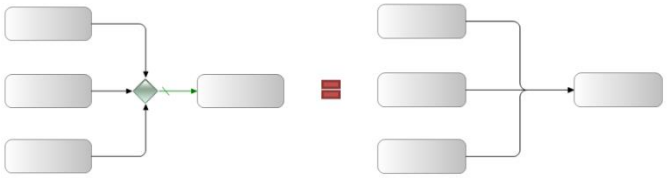
Paralell Gateway



Caso queiramos unir vários caminhos para seguir por apenas um (o que chamamos de **sincronização**), o **Exclusive** gateway também pode ser usado para sincronização, embora seu uso seja raramente necessário para modelagem, pois geralmente pode ser modelado sem o mesmo, obtendo o mesmo comportamento.

GeneXus

Exclusive Gateway

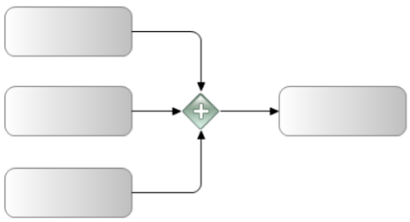


Esta forma de modelar tem seu risco, uma vez que na ausência de sincronização, a tarefa após a união dos caminhos poderia ser executada várias vezes, uma para cada estrada que se juntou, à medida que a sequência de fluxo de cada caminho está chegando.

Para evitar isso, devemos usar um **Parallel** Gateway para juntar caminhos. Um parallel gateway aguarda todas as sequências dos fluxos recebidos e não continua até que todas estejam presentes.

GeneXus

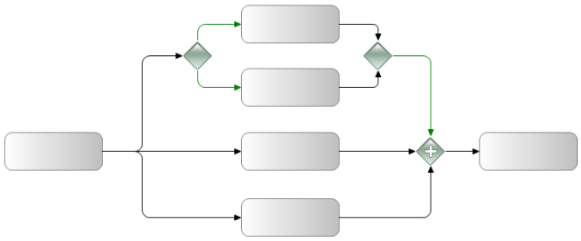
Paralell Gateway



No entanto, existem certas situações nas quais um **Exclusive** gateway é necessário para sincronizar.

GeneXus

Exclusive Gateway

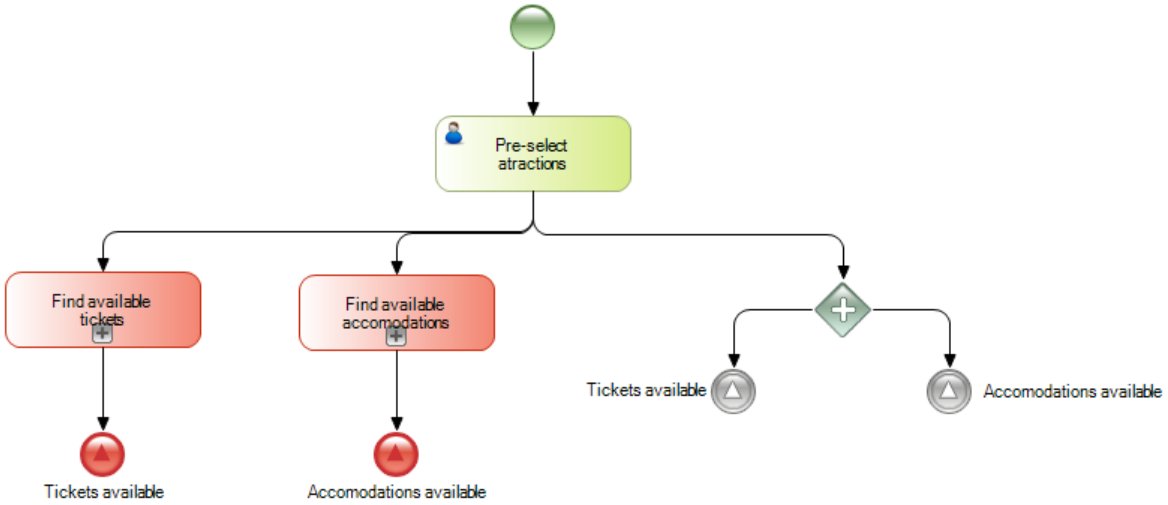


No exemplo que vemos, se o **Exclusive** não fosse usado para sincronizar o resultado de um gateway anterior, o **Parallel** gateway teria quatro conectores de seqüência de entrada. No entanto, apenas três das quatro sequências de fluxos podem ser passados a cada vez (segundo o Exclusive Gateway da bifurcação), portanto, o processo ficaria preso nesse **Parallel** gateway.

Continuando com nosso modelo, adicionamos um evento intermediário do tipo sinal para cada caminho e os conectamos do Parallel Gateway.

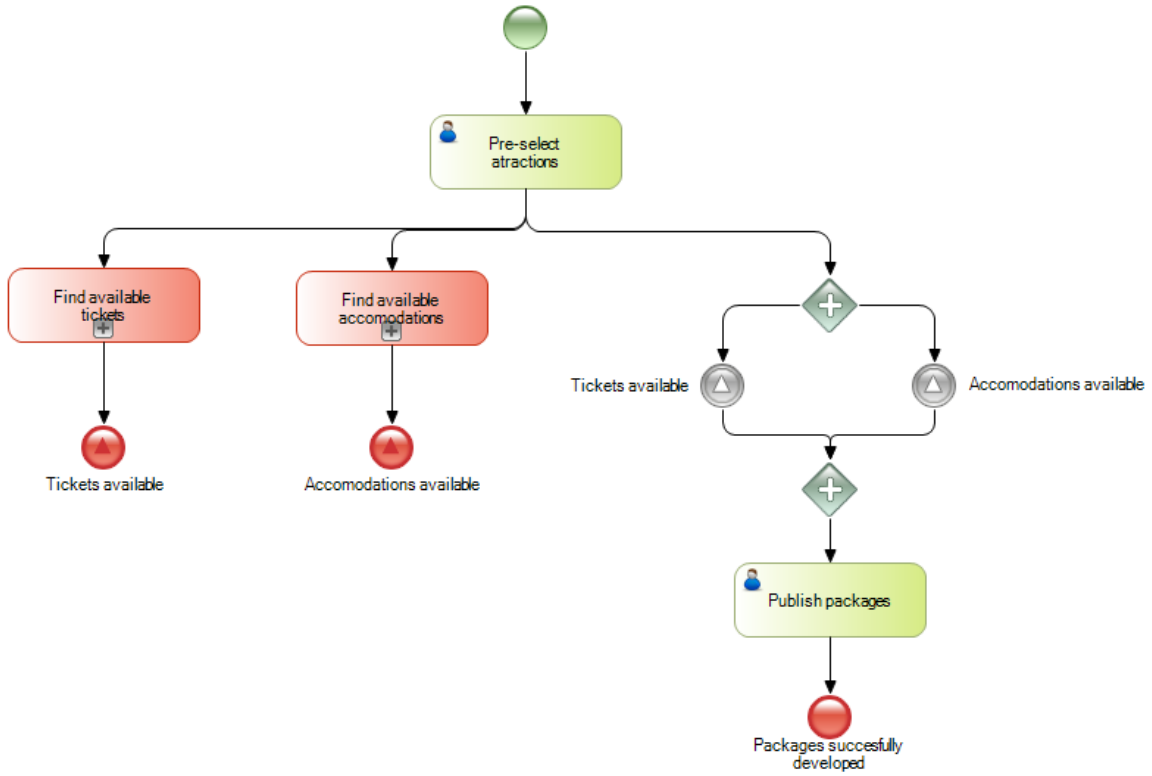
No sinal do caminho à esquerda, chamamos de **Tickets available** e deixá-lo com a propriedade **Is throw** em Falso. Observe que esse sinal de tipo "catch" capturará o sinal do tipo "throw" do mesmo nome, que é acionado no final do processo de busca de passagens.

Fazemos o mesmo com o outro sinal, o chamamos **Accomodation available** e também deixamos **com Is throw** em Falso. Isto, por sua vez, capturará o sinal que é enviado quando o processo de busca de alojamento estiver concluído.



Em seguida, inserimos outro parallel Gateway para unir os dois caminhos, o que garantirá que não se pode continuar se ambos os sinais não forem recebidos.

Finalmente, inserimos uma tarefa de **Publish packages**, a conectamos ao parallel Gateway e adicionamos um none end event para concluir o processo, ao qual adicionamos a descrição **Packages succesfully developed**.



Com isso, conseguimos uma implementação que atenda ao requisito de que o processo não pode ser considerado finalizado, se as passagens e acomodações necessárias não forem obtidas.

No entanto, isso não torna o processo eficiente. Lembre-se de que a agência de viagens nos pediu para não iniciar o processo de encontrar acomodações se você não obteve os ingressos anteriormente. Para fazer isso, também usamos sinais.

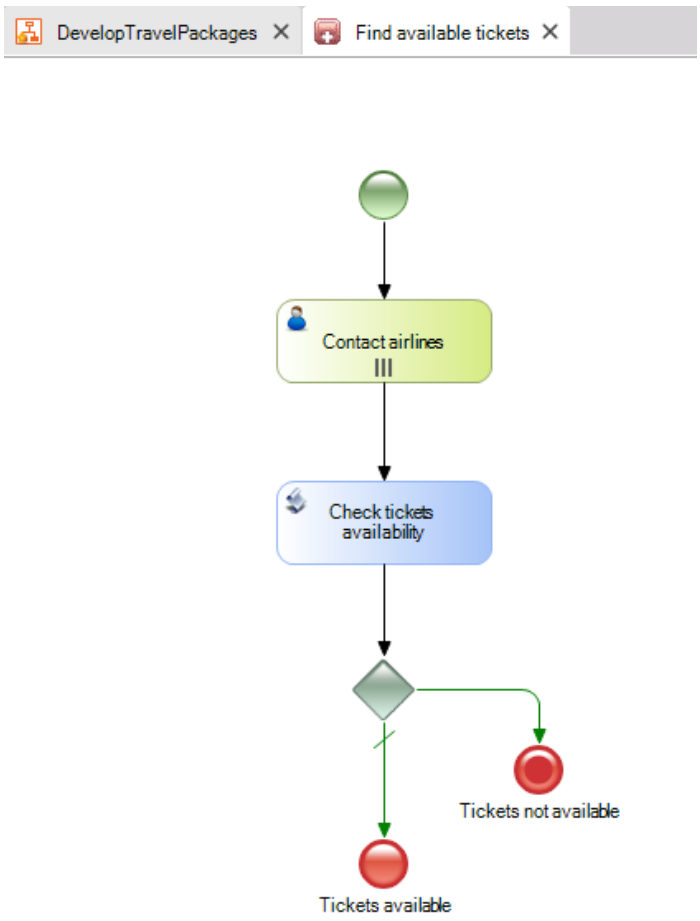
Vamos clicar duas vezes no processo **Find available tickets** para defini-lo. Vemos que uma janela em branco se abre, onde iremos adicionar os símbolos que representam o processo de obtenção de passagens.

Para começar, inserimos um None Start Event e uma tarefa de **Contact Airlines**. Como esta tarefa será executada várias vezes, uma para cada companhia aérea contatada, nós vamos para sua propriedade de tipo Loop e atribuímos o valor Multi-Instance.

Para analisar as informações obtidas, uma vez que todas as companhias aéreas foram contatadas, inserimos uma tarefa de script com o nome **Check availability** e a conectamos da tarefa da **Contact Airlines**.

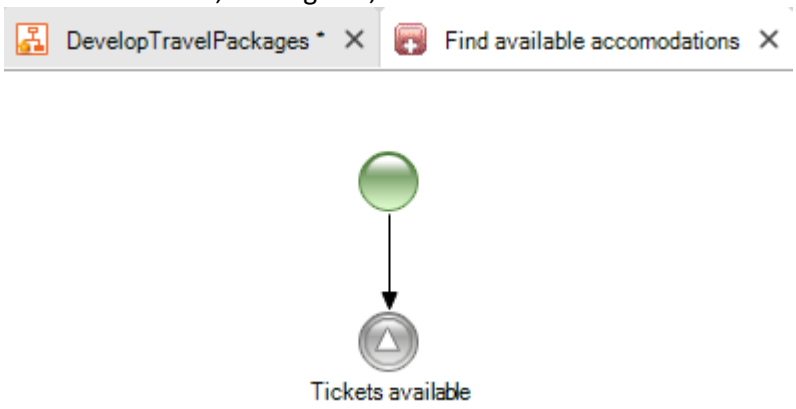
Agora arrastamos um exclusive Gateway.

Se houver passagens disponíveis, terminamos o processo com um end event (o caminho padrão) e, se não houver passagens para construir um pacote turístico, terminamos o processo com um end event, o que garante que não apenas o subprocesso esteja concluído, mas também o processo principal de desenho do pacote.

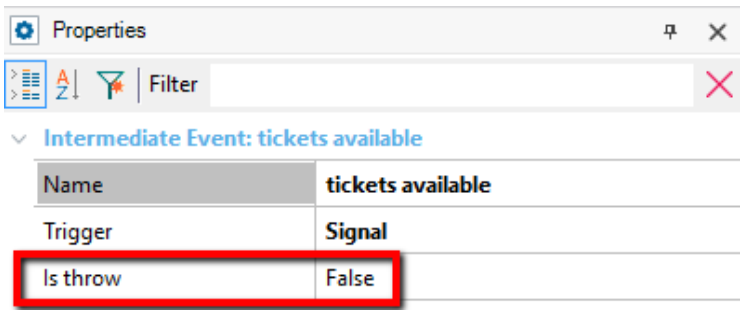


Em seguida, definiremos o processo de encontrar acomodações nos hotéis para ver se podemos oferecer um pacote para a atração pré-selecionada.

Clicamos duas vezes sobre o subprocesso **Find available Accomodations**. Arrastamos um None Start Event e Evento de Início e, em seguida, inserimos um símbolo de um evento intermediário de tipo de sinal.



Colocamos o nome de **Tickets available**, como chamamos o Signal Event End no final do processo de pesquisa de passagens para o pacote. Nós vamos às suas propriedades e verificamos que a propriedade IsThrow está em Falsa, pois queremos que este evento seja "*catch*" e capture o sinal enviado quando o processo de busca de passagen estiver concluído.

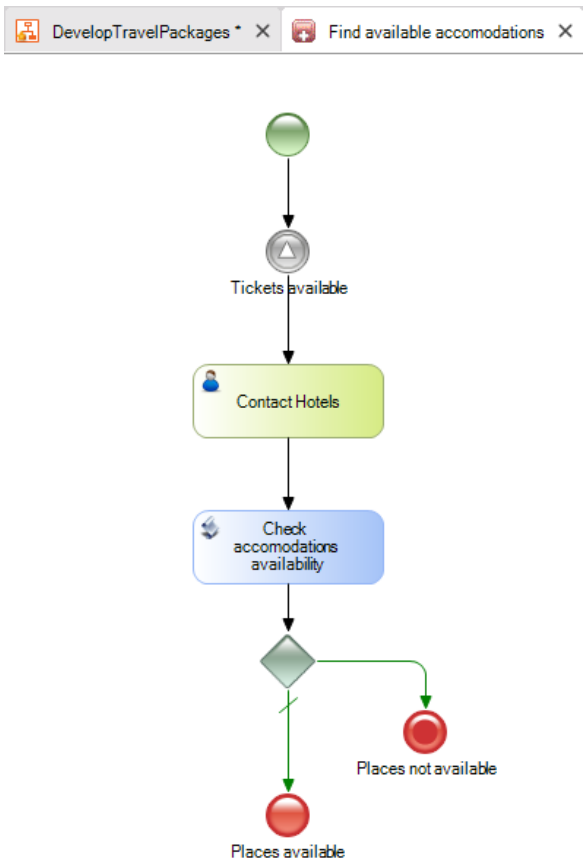


Desta forma, apenas se houver ingressos disponíveis para o pacote, o processo de encontrar lugares nos hotéis continuará.

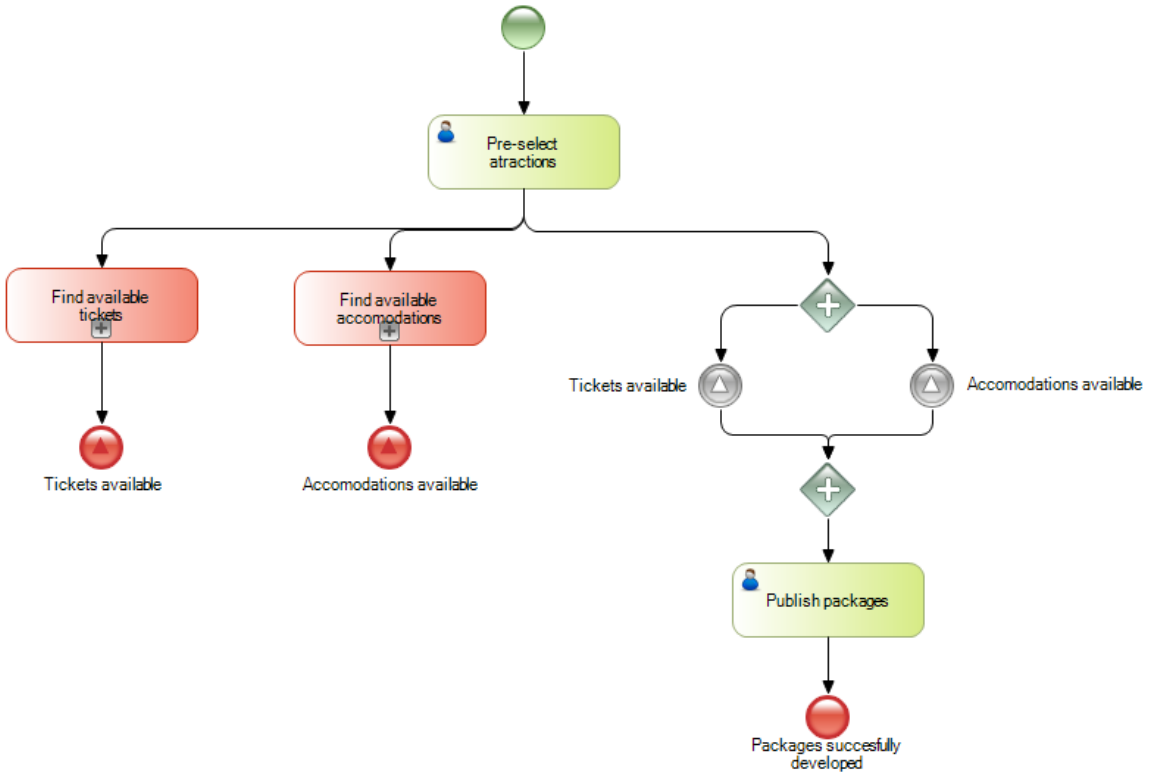
Continuando com a definição do processo, inserimos uma tarefa interativa denominada Contact Hotels, na qual definimos a propriedade Loop tipo no valor Multi-instance, uma vez que esta tarefa deve ser repetida várias vezes, uma para cada hotel contatado.

Em seguida e de forma semelhante ao processo de verificação de passagens, inserimos uma tarefa do tipo de batch chamado **Check Accomodations availability**, que verificará a existência de lugares suficientes nos hotéis para oferecer o pacote que estamos desenhando.

Para terminar, inserimos um exclusive Gateway e, se houver disponibilidade de lugares nos hotéis (que é o resultado esperado), encerramos o processo com um None Event End com a etiqueta "Places available" e, caso contrário, inserimos um Terminate end event com a etiqueta "Places not available".



Fechamos a janela de subprocesso, retornamos ao processo principal e salvamos.



Desta forma, definimos um processo que atende ao objetivo de projetar um pacote de turístico de forma eficiente, como pedimos na Agência de Viagens.

Os sinais nos deram um mecanismo de comunicação entre as diferentes partes do processo principal e este e os subprocessos para garantir os objetivos que estabelecemos.

Se quiser saber mais sobre este tema, visite o link que aparece na tela.

http://wiki.genexus.com/commwiki/servlet/hwikibypageid?24913