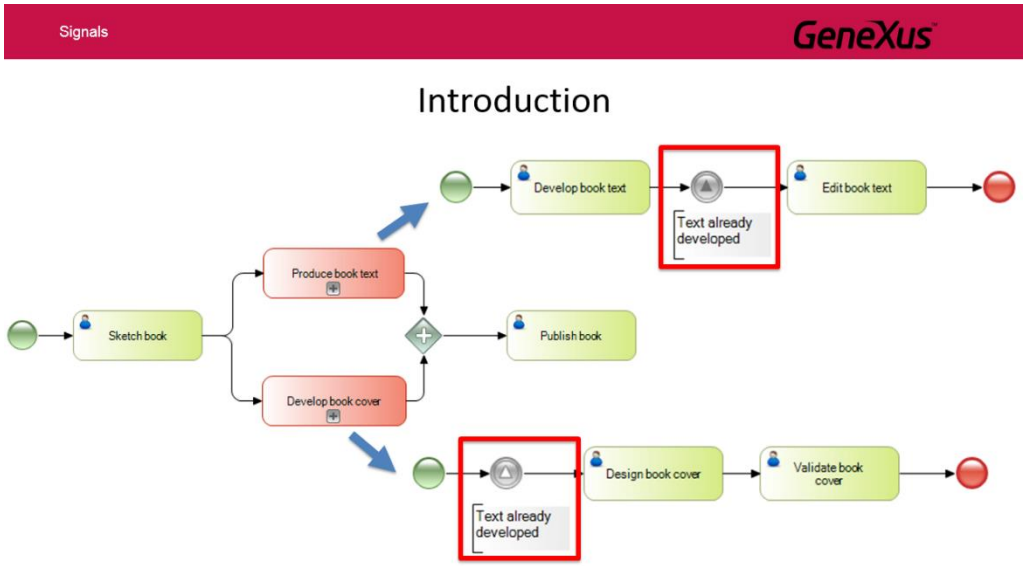
Eventos del tipo Señal

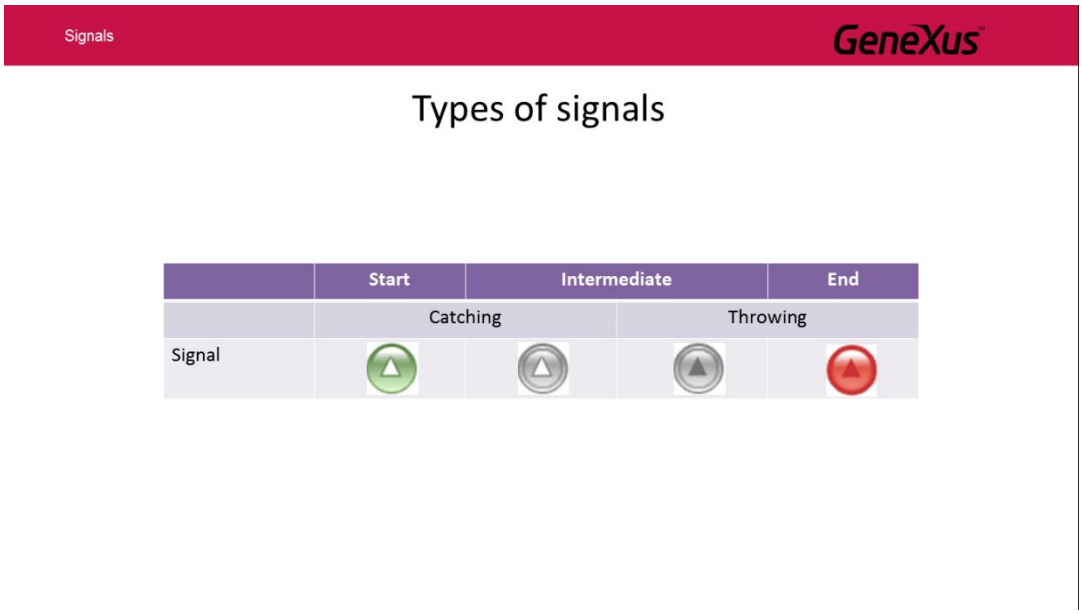
En este video vamos a ver algunos usos del evento del tipo Señal (o Signal en inglés).

Los eventos Signal, son utilizados para enviar o recibir señales dentro o fuera del proceso, por lo que son útiles tanto para la comunicación entre partes de un mismo proceso como para la comunicación entre procesos que estén unidos por una relación jerárquica.



Esto implica que es posible señalizar la ocurrencia de un evento que podrá ser detectado en cualquier parte del proceso, subprocesos e incluso procesos ancestros.

El comportamiento de las señales varía dependiendo de si son eventos de inicio, eventos intermedios o eventos de fin.
A su vez estos eventos pueden disparar una señal (throwing) o capturar una señal (catching).



En función de esto, la notación cambia. Por ejemplo los eventos intermedios tienen un borde doble, mientras que los de inicio y fin tienen borde simple. Cuando son eventos de disparo el triángulo es oscuro y cuando se captura una señal el triángulo es claro.

Un evento de señal de inicio, permite que un proceso comience cuando se recibe una señal desde otra parte

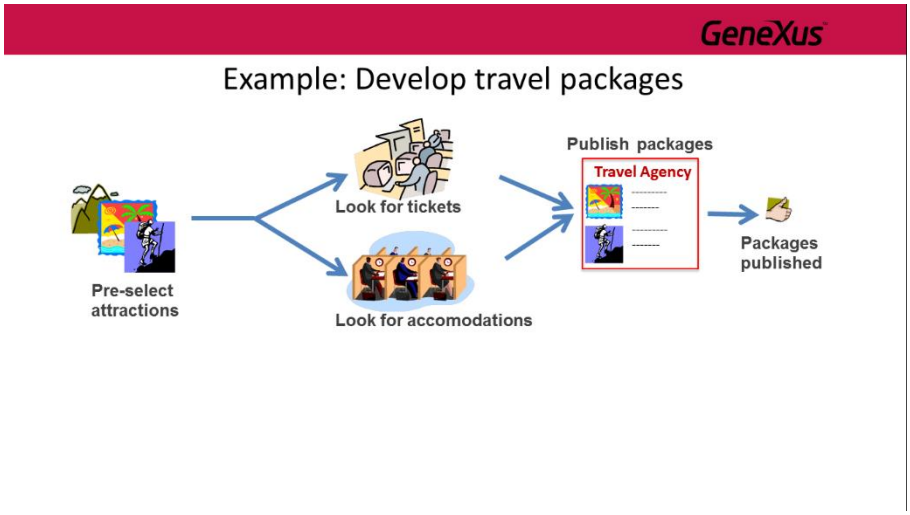
del proceso o procesos relacionados jerárquicamente con el proceso donde es definido. Por este motivo son siempre del tipo de captura (o “catch”).

Un evento de señal del tipo de fin, permite que al finalizar un flujo del proceso se envíe una señal a otra parte del proceso o procesos relacionados jerárquicamente. Son siempre de disparo (o “throw”).

Un evento de señal del tipo intermedio, puede ser de disparo o de captura, dependiendo del valor de su propiedad “Is throw” (True o False respectivamente) y pueden ser colocados en cualquier parte del proceso, normalmente entre actividades.

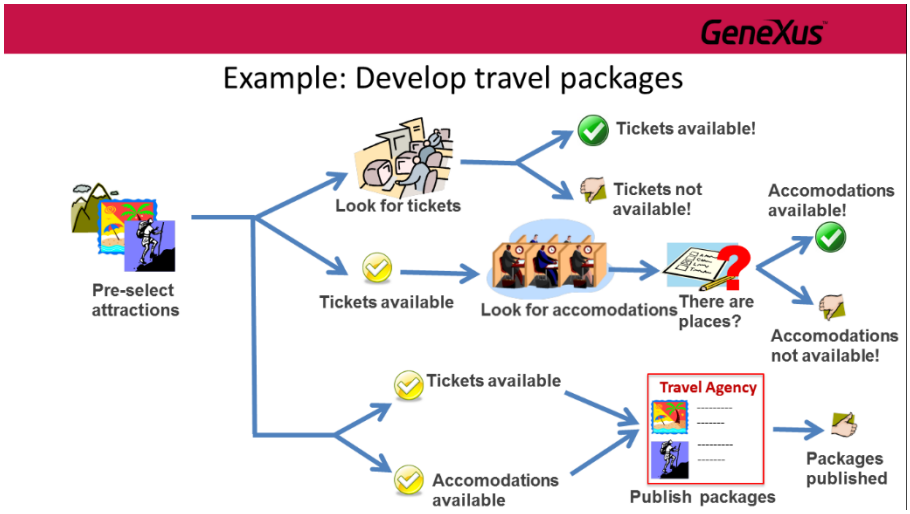
Veamos algunos de estos símbolos en acción.

La agencia de viajes nos solicitó que modeláramos el proceso de publicación de paquetes turísticos para sus clientes. Cada paquete involucra varias reservas de pasajes y varias reservas de hotel y un paquete puede ser publicado solamente si hay disponibilidad de estas reservas.



Esto implica que el proceso deberá permitir asegurar un cierto cupo de pasajes y de habitaciones, como para poder publicar paquetes de una cierta atracción turística.

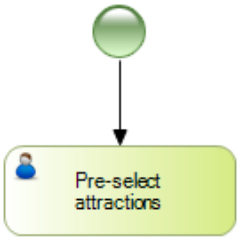
A su vez, el proceso debe ser eficiente, es decir las averiguaciones en las aerolíneas y en los hoteles deben comenzarse al mismo tiempo, pero si ya no se logra obtener los pasajes necesarios, debe cancelarse la búsqueda de los cupos hoteleros.



El proceso debe asegurar que solamente se publiquen los paquetes, si tanto los pasajes como los alojamientos están asegurados.

Para implementar este proceso en GeneXus, abrimos el GeneXus Modeler, creamos un objeto del tipo Business Process Diagram y lo llamamos **DevelopTravelPackages**.

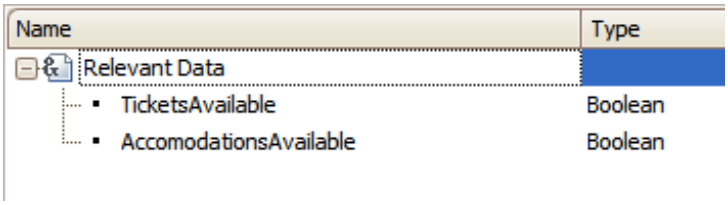
Para comenzar arrastramos un símbolo de None Start Event. Luego insertamos una tarea interactiva, le ponemos de nombre "**Pre-select attractions**" y la conectamos desde el None Start Event.



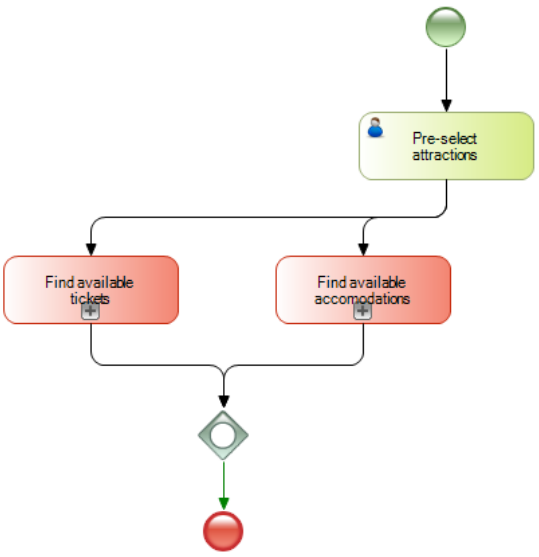
Esta tarea resume la actividad de analizar qué atracciones son más aptas para ser incluidas en un paquete. Luego de esto, debemos iniciar el proceso de búsqueda de pasajes y otro de búsqueda de alojamientos.

Para esto arrastramos un símbolo de un subproceso embebido, le llamamos **Find available tickets** y lo unimos desde la tarea antes definida. Agregamos otro subproceso embebido **Find available accomodations** y también lo unimos desde la tarea Pre-select attractions.

Para que podamos terminar el proceso de desarrollo de paquetes turísticos, debemos asegurar que ambos subprocesos hayan terminado exitosamente. Esto lo podríamos hacer definiendo un dato relevante del tipo booleano que fuera seteado en cada proceso, dependiendo de si dicho proceso terminó exitosamente o no.



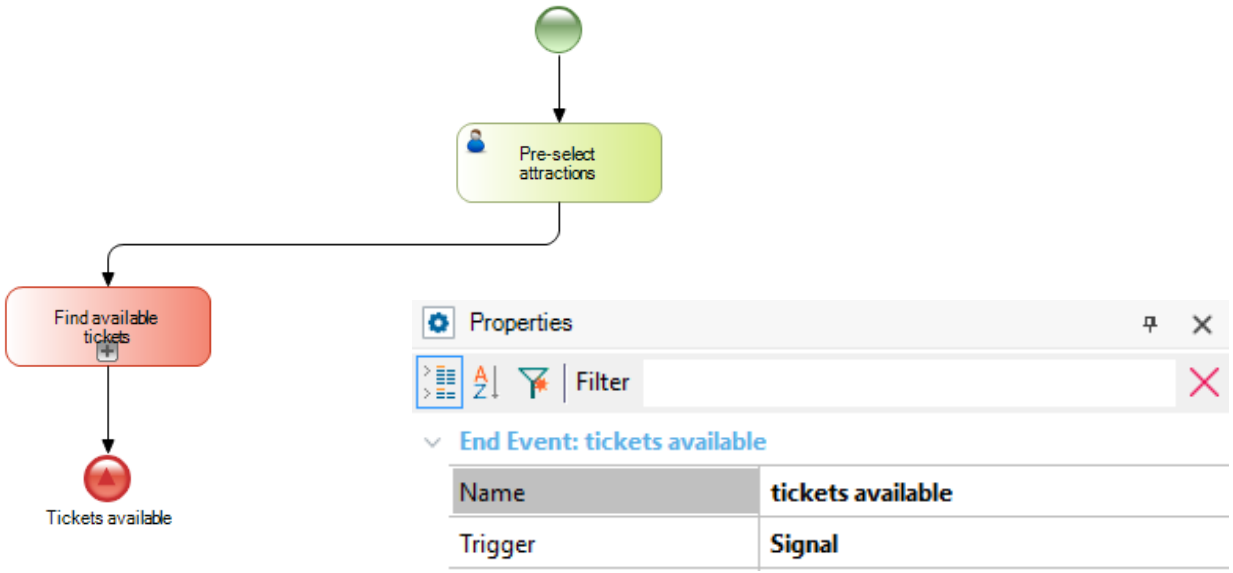
Y luego utilizar un Inclusive Gateway para sincronizar los caminos provenientes de cada subproceso, de forma de que solamente se termine el proceso de desarrollo de paquetes turísticos, si ambos subprocesos fueron exitosos.



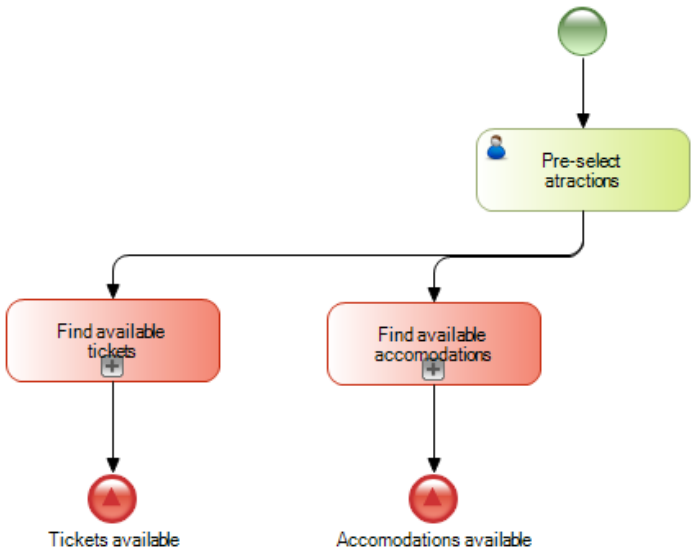
Si bien esta solución es posible, implica definir y setear datos relevantes. Podemos obtener un resultado

similar si usamos Señales, sin necesidad de estos datos relevantes.

En lugar del inclusive Gateway, insertamos un End Event del tipo Signal y lo conectamos a la salida del subproceso de buscar pasajes disponibles. Asignamos a la señal el nombre **Tickets available** y vemos que un **evento de fin del tipo señal** ya es del tipo "throw" porque el triángulo es oscuro. En un signal end event no podemos cambiar el valor de su propiedad **Is Throw** ya que la misma no está disponible y su valor es siempre True.

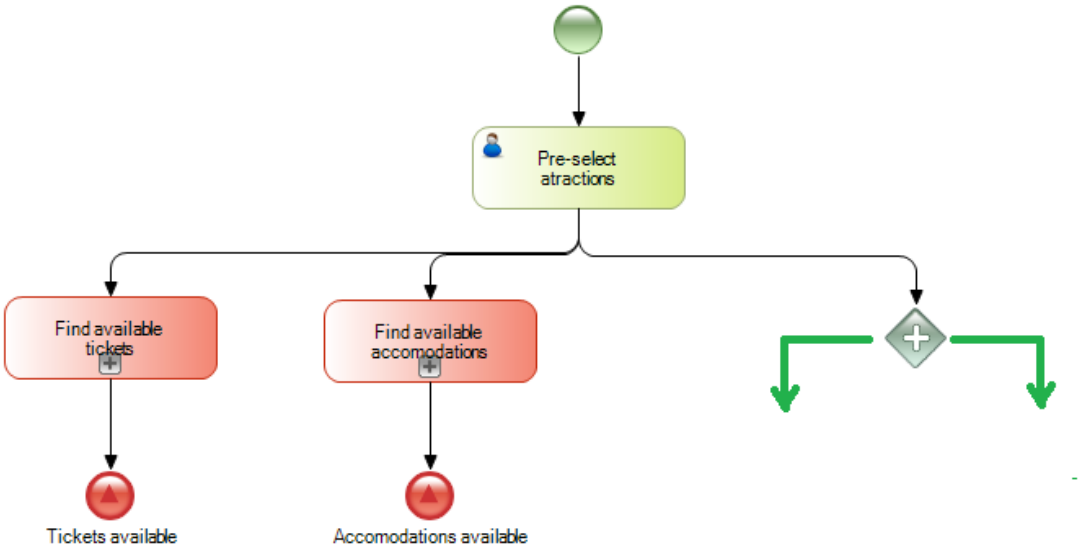


Análogamente, insertamos un Signal End Event a la salida del subproceso de buscar alojamientos y le damos el nombre **Accomodations available**.



Ahora debemos capturar estas señales para poder continuar con el proceso de desarrollo de paquetes turísticos, es decir con la publicación de los mismos y finalmente terminar el proceso.

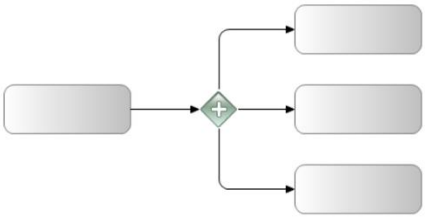
Para esto, insertamos un Parallel Gateway al cual conectamos desde la tarea Pre-select attractions. Esto nos permitirá bifurcar el flujo en dos caminos.



A diferencia del Exclusive Gateway que se utiliza cuando el camino por donde debe seguir el flujo del proceso depende de la evaluación de una condición del tipo verdadero / falso, un Parallel Gateway nos permite dividir el flujo en dos o más caminos, sin evaluar ninguna condición.

GeneXus

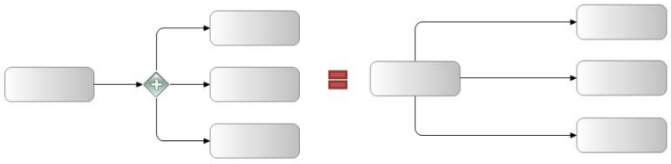
Paralell Gateway



Debido a esto, también es posible modelar este mismo comportamiento sin utilizar un **Paralell** gateway sino simplemente uniendo dos conectores. Sin embargo, el uso del **Parallel** gateway puede llegar a clarificar el diagrama en determinadas circunstancias.

GeneXus

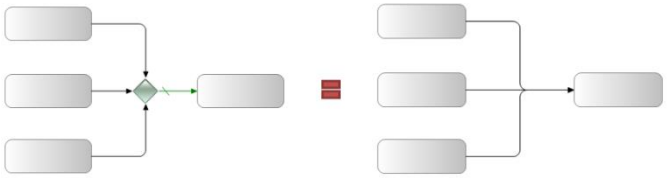
Paralell Gateway



En el caso de queramos unir varios caminos para seguir por uno solo (a lo que llamamos **sincronización**), los **Exclusive** gateway también pueden ser utilizados para sincronización, aunque su utilización rara vez es necesaria para el modelado ya que generalmente puede ser modelado sin el mismo, obteniéndose el mismo comportamiento.

GeneXus

Exclusive Gateway

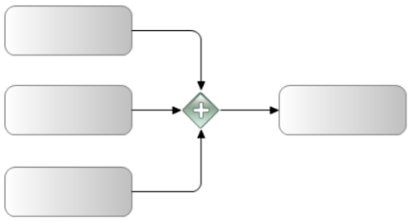


Esta forma de modelar tiene su riesgo, ya que al no haber sincronización, la tarea siguiente a la unión de los caminos podría ejecutarse varias veces, una por cada camino que fue unido, a medida que van llegando las secuencia de flujo de cada camino.

Para evitar esto, debemos utilizar un **Parallel** Gateway para unir caminos. Un parallel Gateway espera por todas las secuencias de flujo entrantes y no continúa hasta que todas están presentes.

GeneXus

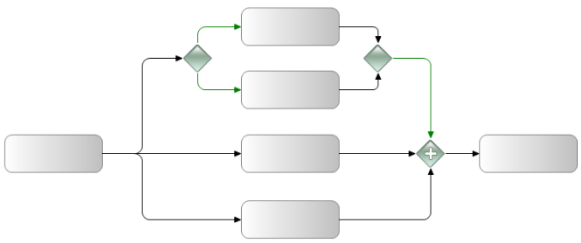
Paralell Gateway



Sin embargo, hay ciertas situaciones en las que un **Exclusive** gateway si es requerido para sincronizar.

GeneXus

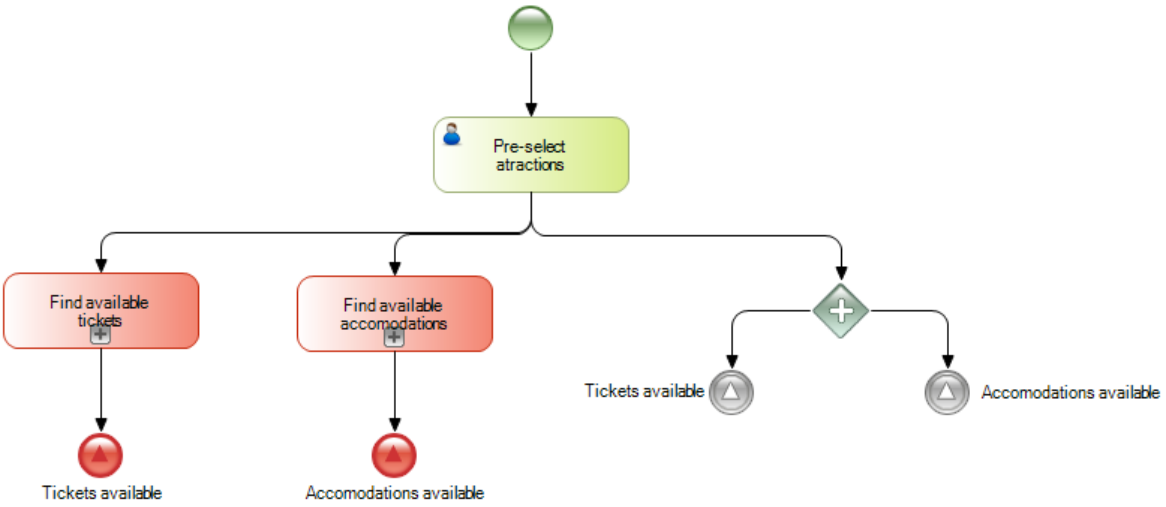
Exclusive Gateway



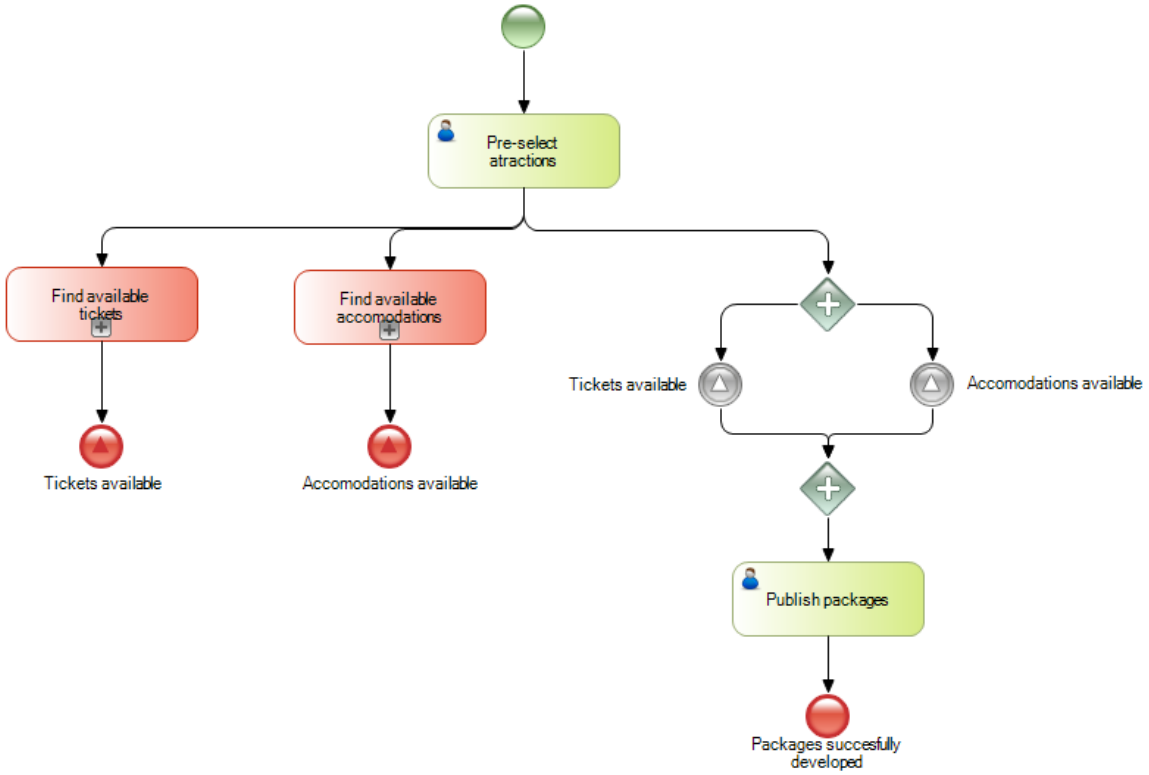
En el ejemplo que vemos, si no se utilizara el **Exclusive** para sincronizar el resultado de un gateway anterior, el **Parallel** gateway tendría cuatro conectores de secuencia de entrada. Sin embargo, sólo tres de las cuatro secuencias de flujo podrían pasar en cada vez (dado el Exclusive Gateway de la bifurcación), por lo tanto, el proceso quedaría atascado en ese **Parallel** gateway.
Continuando con nuestro modelo, agregamos un Intermediate event del tipo signal para cada camino y los conectamos desde el parallel Gateway.

A la señal del camino izquierdo la llamamos **Tickets available** y la dejamos con la propiedas **Is throw** en False. Observemos que esta señal del tipo “catch”, capturará la señal proveniente de la del tipo “throw” del mismo nombre, que se dispara al finalizar el proceso de búsqueda de pasajes.

Hacemos lo mismo con la otra señal, la llamamos **Accomodation available** y la dejamos también con **Is throw** en False. Esta a su vez capturará la señal que se envíe cuando se termine el proceso de búsqueda de alojamientos.



Luego insertamos otro parallel Gateway para unir ambos caminos, lo que asegurará que no pueda continuarse si no se reciben ambas señales.
Finalmente insertamos una tarea **Publish packages**, la conectamos desde el parallel Gateway y agregamos un none end event para finalizar el proceso, al cual le agregamos la descripción **Packages succesfully developed**.



Con esto lograríamos una implementación que cumple con el requisito de que no pueda darse por terminado el proceso, si no se obtienen los tickets y los alojamientos necesarios.

No obstante, esto no cumple que el proceso sea eficiente. Recordemos que la Agencia de viajes nos pidió que no pueda empezar el proceso de búsqueda de alojamientos si no se han conseguido los tickets previamente. Para poder hacer esto, usamos también señales.

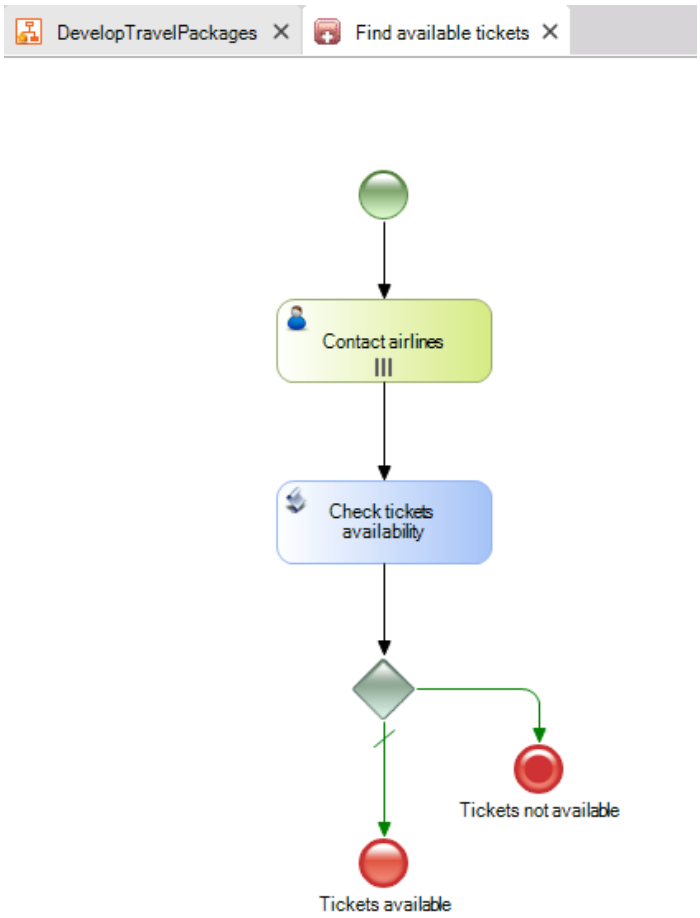
Vamos a dar doble clic sobre el proceso **Find available tickets** para definirlo. Vemos que se abre una ventana en blanco, donde vamos a agregar los símbolos que representen el proceso de obtener tickets.

Para empezar insertamos un None Start Event y una tarea **Contact Airlines**. Como esta tarea va a ser ejecutada varias veces, una por cada aerolínea contactada, vamos a su propiedad Loop type y asignamos el valor Multi-Instance.

Para analizar la información obtenida, una vez contactadas todas las aerolíneas, insertamos una tarea script con el nombre **Check availability** y la conectamos desde la tarea **Contact Airlines**.

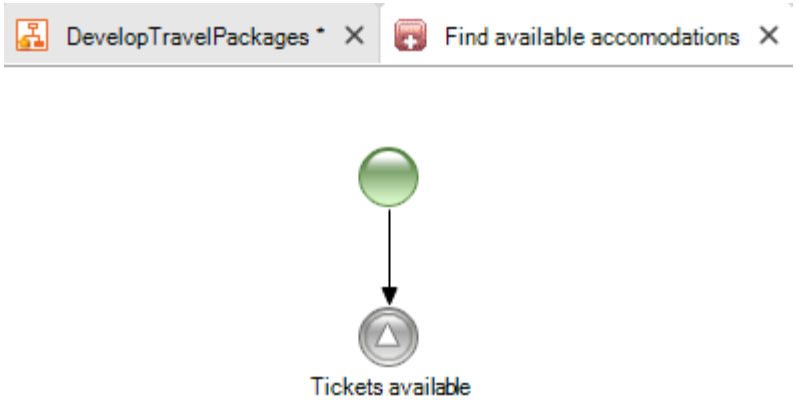
Ahora arrastramos un exclusive Gateway.

Si hay disponibilidad de tickets terminamos el proceso con un end event (el camino por defecto) y si no hay tickets como para armar un paquete turístico terminamos el proceso con un terminate end event, que asegura que no solo se termine el subproceso, sino también el proceso principal de diseño de paquetes.

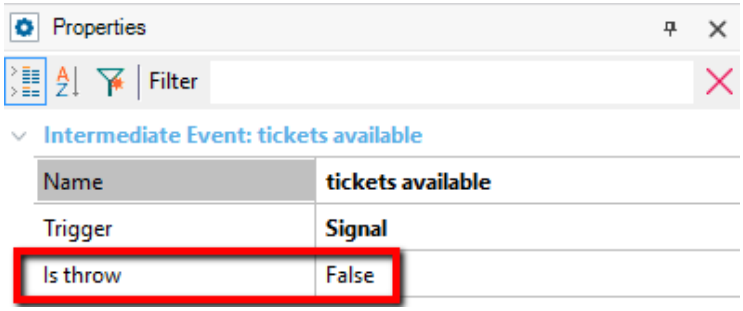


A continuación, vamos a definir el proceso de búsqueda de lugares en los hoteles para ver si podemos ofrecer un paquete para la atracción pre-seleccionada.

Hacemos doble clic sobre el subproceso **Find available Accomodations**. Arrastramos un None Start Event y a continuación insertamos un símbolo de un evento intermedio del tipo signal.



Le ponemos de nombre **Tickets available**, tal como llamamos al Signal End Event al final del proceso de búsqueda de tickets para el paquete. Vamos a sus propiedades y verificamos que la propiedad IsThrow está en False, ya que queremos que este evento sea de "catch" y capture la señal enviada al finalizarse el proceso de búsqueda de pasajes.

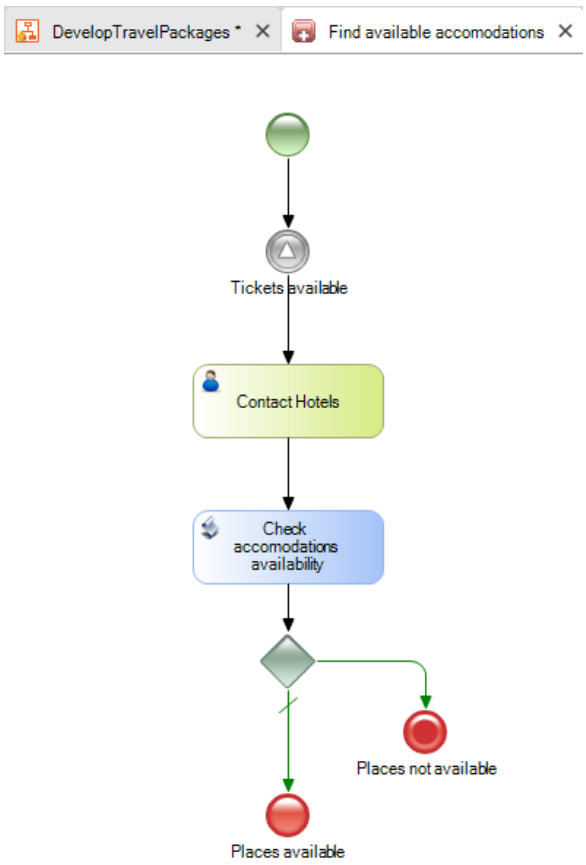


De esta manera, solamente si hay tickets disponibles para el paquete, se continuará con el proceso de búsqueda de plazas en los hoteles.

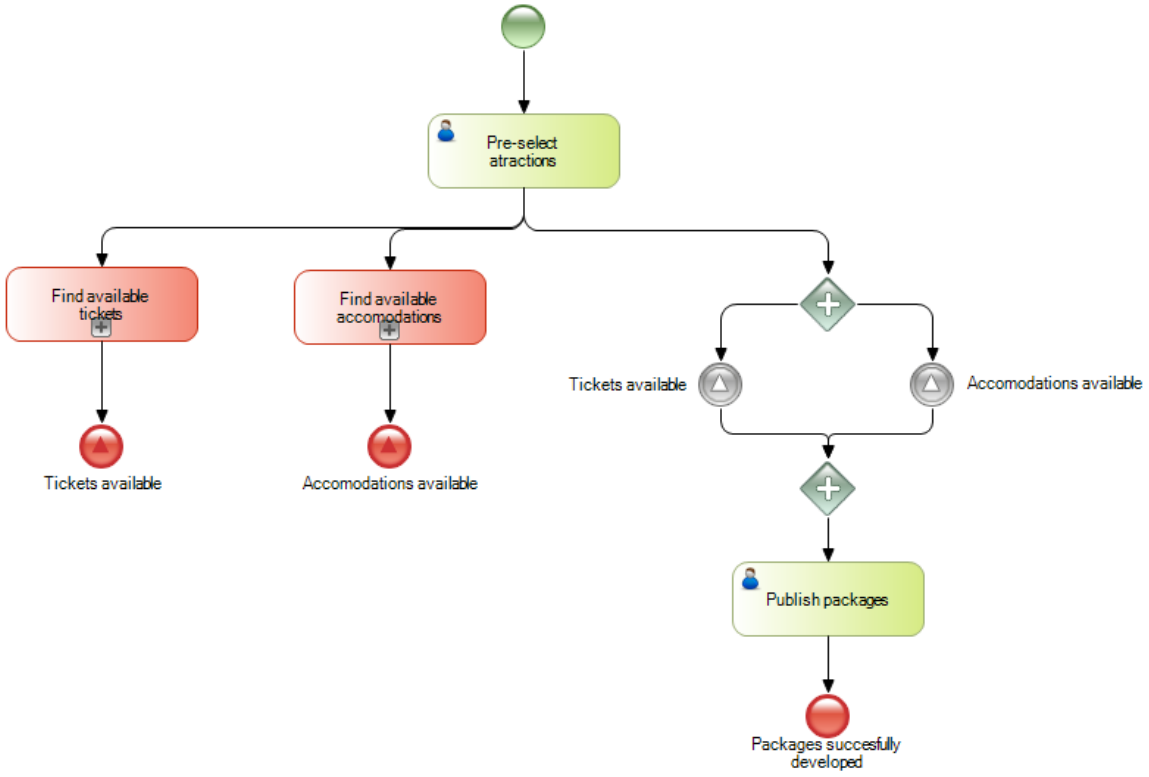
Continuando con la definición del proceso, insertamos una tarea interactiva de nombre Contact Hotels, a la cual le definimos la propiedad Loop type en el valor Multi-instance, ya que esta tarea debería repetirse varias veces, una por cada hotel contactado.

A continuación y en forma similar al proceso de verificación de pasajes, insertamos una tarea del tipo batch a la que llamamos **Check Accomodations availability** que se encargará de verificar la existencia de suficientes plazas en los hoteles para ofrecer el paquete que estamos diseñando.

Para terminar insertamos un exclusive Gateway y si hay disponibilidad de lugares en los hoteles (que es el resultado esperado) terminamos el proceso con un None End Event con la etiqueta "Places available" y para el caso contrario insertamos un Terminate end event con la etiqueta "Places not available".



Cerramos la ventana del subproceso, volvemos al proceso principal y salvamos.



De esta manera hemos definido un proceso que cumple con el objetivo de diseñar un paquete turístico de

forma eficiente, tal como nos habían solicitado de la Agencia de Viajes.

Las señales nos otorgaron un mecanismo de comunicación entre las distintas partes del proceso principal y éste y los subprocesos, para asegurar los objetivos que nos planteamos.

Si desea saber más de este tema, visite el link que aparece en pantalla.

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