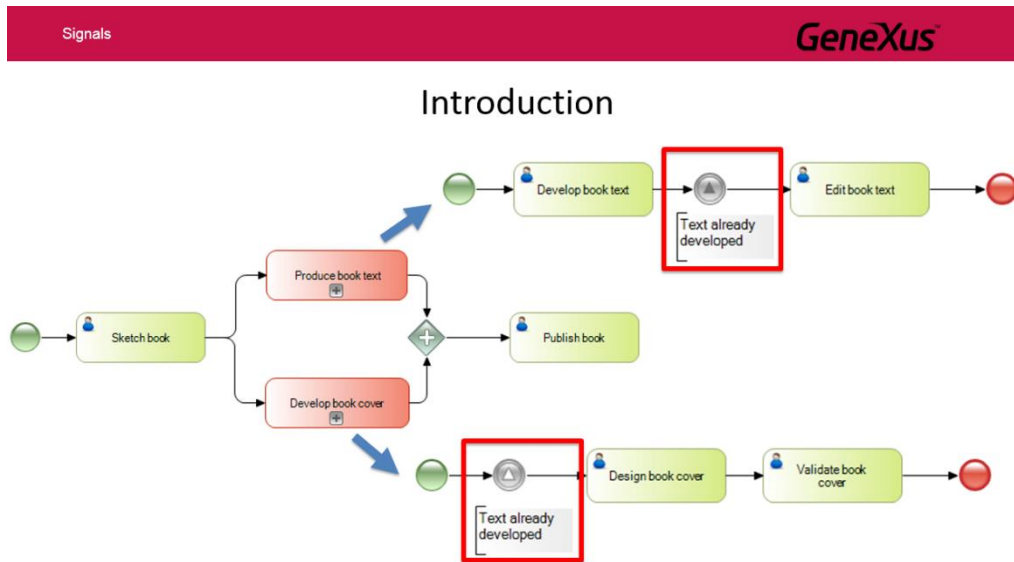


シグナルタイプのイベント

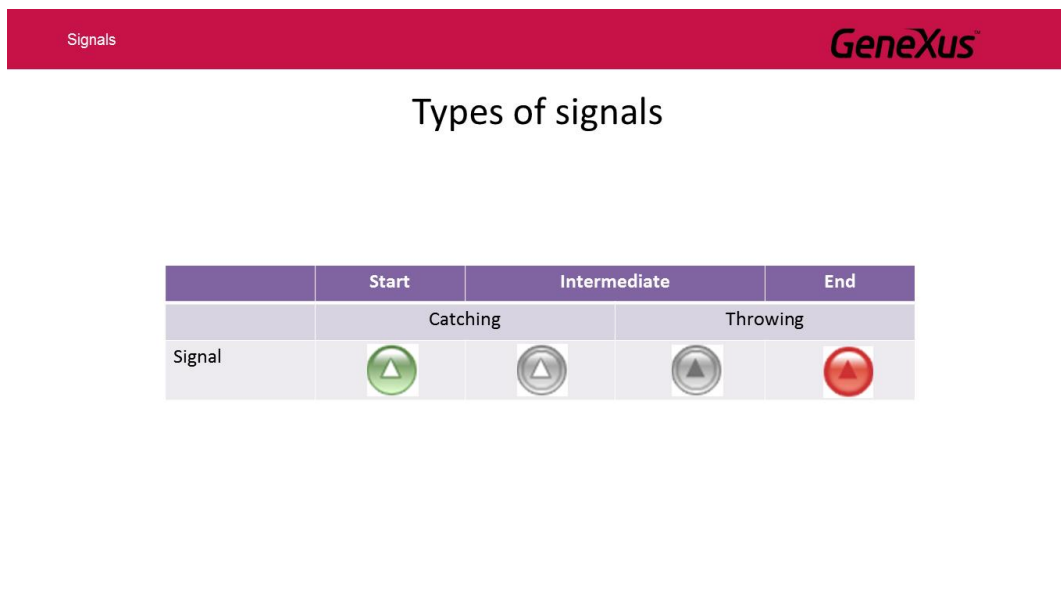
この章では、シグナルタイプイベントの使用方法をいくつか見ていきます。

シグナルイベントは、プロセス内またはプロセス外でシグナルを送受信するために使用されます。 そのため、これらは同じプロセスの部分間の通信と、階層関係によってリンクされたプロセス間の通信の両方に役立ちます。



これによって、プロセス、サブプロセス、さらには親プロセスの任意の部分で検出できるイベントの発生を通知できます。

シグナルの動作は、開始イベント、中間イベント、終了イベントのいずれであるかによって異なります。さらに、これらのイベントはシグナルをトリガーしたり（スロー）、シグナルを受信したり（キャッチ）することができます。



それに応じて使用される表記も変わります。たとえば、中間イベントには二重の境界線があり、開始イベントと終了イベントには単純な境界線が付いています。イベントをトリガーしている場合、塗りつぶされた三角形です。シグナルを受信している場合、白抜きの三角形です。

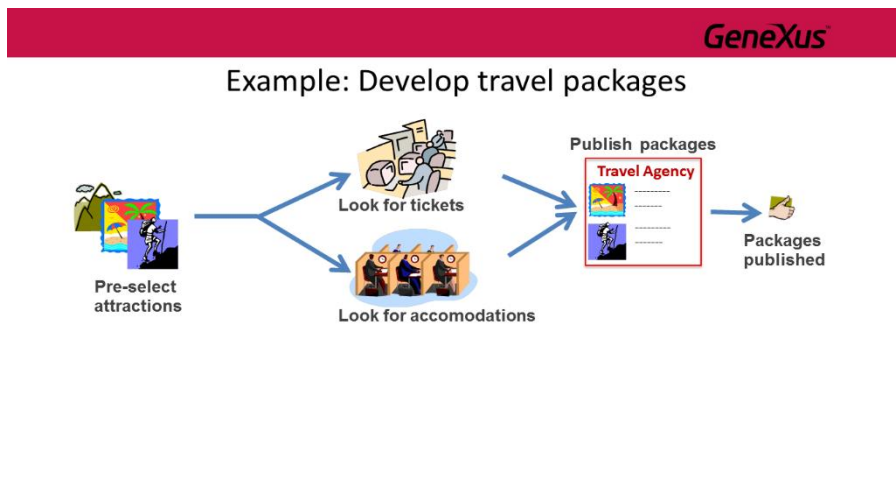
開始シグナル イベントを使用すると、シグナルが定義されているプロセスに階層的に関連するプロセスの別の部分またはプロセスからシグナルを受信したときに、プロセスを開始できます。このため、常に「catch」タイプになります。プロセスフローが終了すると、終了タイプのシグナルイベントにより、プロセスの別の部分または階層的に関連するプロセスにシグナルを送信できます。それらは常に「throw」タイプです。

中間タイプのシグナルイベントは、「Is throw」プロパティの値（それぞれ True または False）に応じて「throw」または「catch」タイプにすることができ、プロセスの任意の部分（通常はアクティビティの間に）に配置できます。

これらのシンボルのいくつかが実際に動作する様子を見てみましょう。

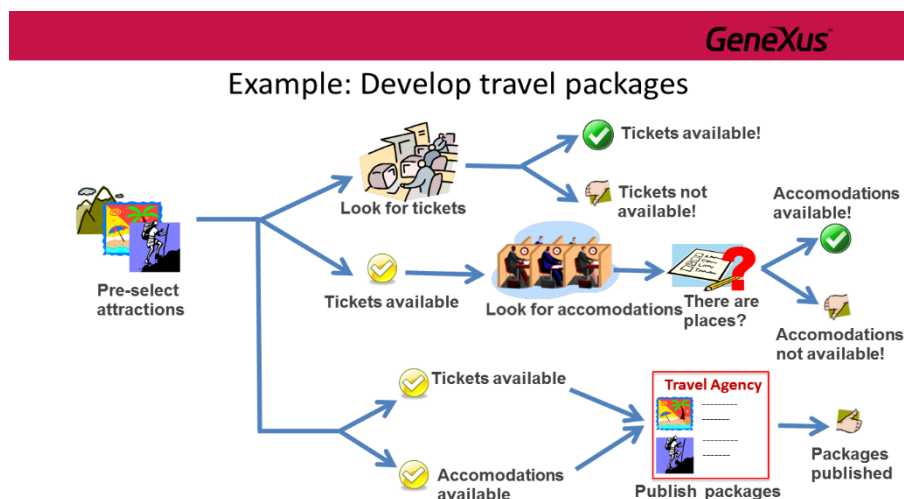
旅行代理店は、顧客向けに旅行パッケージを発行するために使用されるプロセスをモデル化するように私たちに依頼しました。

各パッケージには複数のチケットとホテルの予約が含まれており、チケットとホテルが利用可能な場合にのみ 1 つのパッケージを公開できます。



これは、特定の観光名所のパッケージを宣伝できるように、プロセスで一定数のチケットと部屋を確保できる必要があることを意味します。

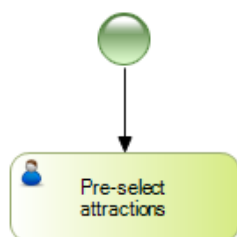
さらに、プロセスは効率的でなければなりません。つまり、航空会社とホテルへの問い合わせを同時に開始し、最低枚数の航空券が入手できなかった場合はホテルの部屋探しを中止しなければなりません。



このプロセスでは、チケットと宿泊施設の両方が予約された場合に、パッケージのみが公開されることを保証する必要があります。

GeneXus でこのプロセスを実装するには、GeneXus を開き、ビジネスプロセスダイアグラムオブジェクトを作成し、「DevelopTravelPackages」というオブジェクト名にします。

まず、「開始イベント/トリガーなし」シンボルをドラッグします。次に、インタラクティブなタスクを挿入し、「Pre-select attractions」という名前を付け、「開始イベント/トリガーなし」から接続します。



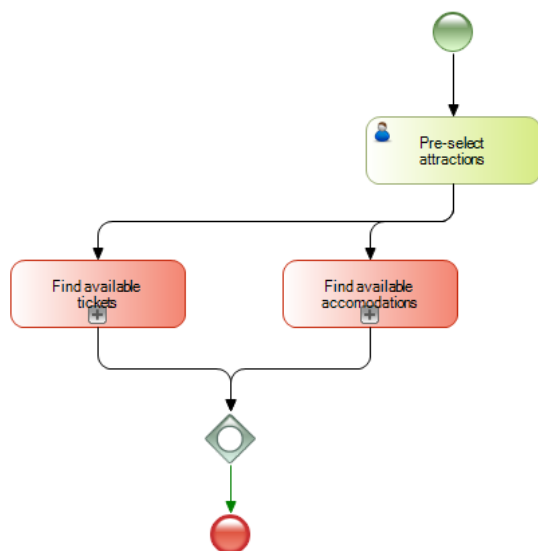
このタスクには、パッケージに含めるのにどのアトラクションがより適しているかを検討するアクティビティが含まれます。

次に、チケットを検索するプロセスとホテルを検索する別のプロセスを開始する必要があります。

旅行パッケージの開発プロセスを完了するには、両方のサブプロセスが正常に完了していることを確認する必要があります。これは、このプロセスが正常に完了したかどうかに応じて、各プロセスに設定された「Boolean型」のデータ項目を定義することによって実行できます。

Name	Type
Relevant Data	
▪ TicketsAvailable	Boolean
▪ AccomodationsAvailable	Boolean

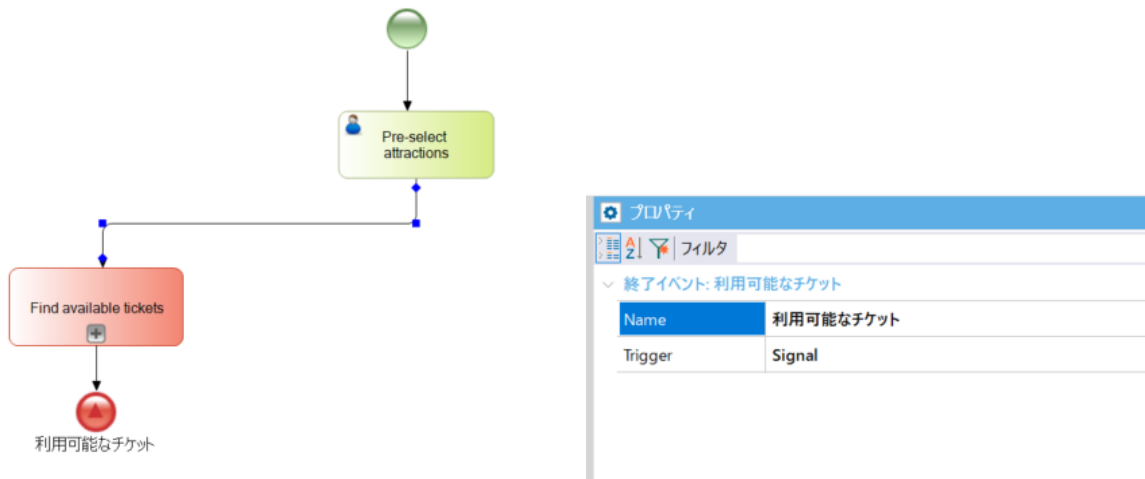
次に、包括的ゲートウェイを使用して各サブプロセスからのパスを同期し、両方のサブプロセスが正常に完了した場合にのみパッケージ開発プロセスが終了するようにする必要があります。



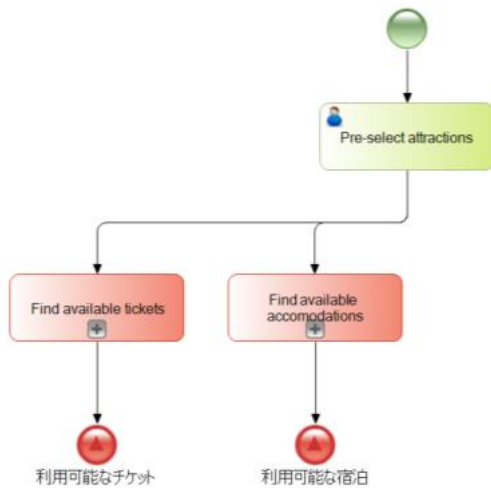
このソリューションは可能ですが、関連するデータの定義と設定が必要になります。

この関連データを必要とせずに、シグナルを使用しても同様の結果を得ることができます。

包括的ゲートウェイの代わりに、シグナルタイプの終了イベントを挿入し、利用可能なチケットを探すサブプロセスの出力に接続します。シグナルの名前は「利用可能なチケット」とします。三角形が暗いため、シグナルタイプの終了イベントがすでにスロータイプであることがわかります。シグナル終了イベントでは、「Is throw」プロパティの値は使用できず、その値は常に True であるため、値を変更することはできません。

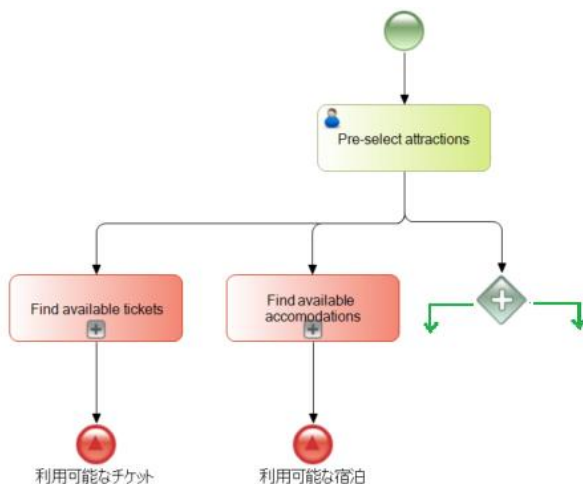


同様に、ホテルを検索するサブプロセスの出力にシグナルタイプの終了イベントを挿入し、それを「利用可能な宿泊」とします。



次に、旅行パッケージを開発するプロセスを続行するには、パッケージを公開して最終的にプロセスを終了するためにこれらのシグナルをキャッチする必要があります。

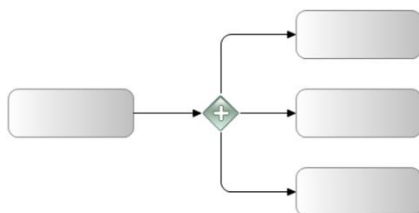
これを行うには、アトラクションの事前選択タスクから接続する平行ゲートウェイを挿入します。 これにより、フローを 2 つのパスに分岐させることができます。



プロセスフローがたどるパスが true/false 条件の評価に依存する場合に使用される排他的ゲートウェイとは異なり、並列ゲートウェイを使用すると、条件を評価せずにフローを 2 つ以上のパスに分割できます。

GeneXus®

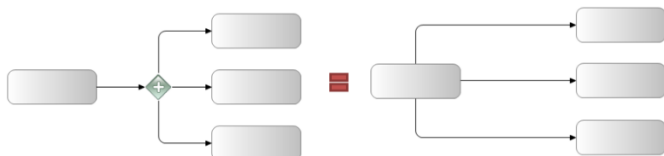
Parallel Gateway



このため、平行ゲートウェイを使用せずに、2 つのコネクタを結合するだけで、これと同じ動作をモデル化することもできます。 ただし、平行ゲートウェイを使用すると、特定の状況ではダイアグラムを明確にすることができます。

GeneXus®

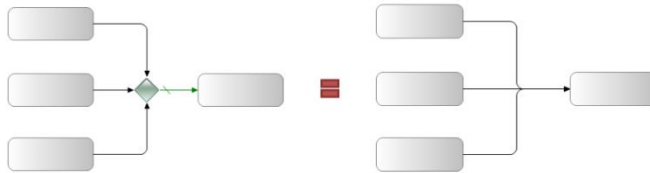
Parallel Gateway



複数のパスを結合してそのうちの 1 つだけをたどることを同期と呼びます。このコンテキストでは、排他的ゲートウェイは、通常は排他的ゲートウェイなしでモデル化でき、同じ動作が得られるため、モデリングに必要なことはほとんどありませんが、同期にも使用できます。

GeneXus®

Exclusive Gateway

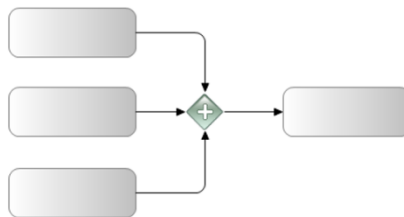


このモデリング方法は同期がないため、パスを結合した後のタスクが各パスのフローシーケンスが到着するたびに、結合されたパスごとに 1 回ずつ、複数回実行される可能性があるため、かなり危険です。

これを回避するには、平行ゲートウェイを使用してパスを結合する必要があります。平行ゲートウェイは、すべての受信フローシーケンスを待機し、すべてが到着するまで続行しません。

GeneXus®

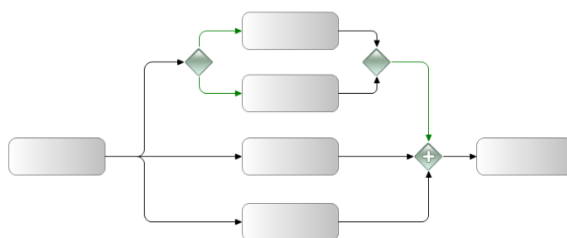
Parallel Gateway



ただし、場合によっては、同期のために専用ゲートウェイが必要になります。

GeneXus®

Exclusive Gateway



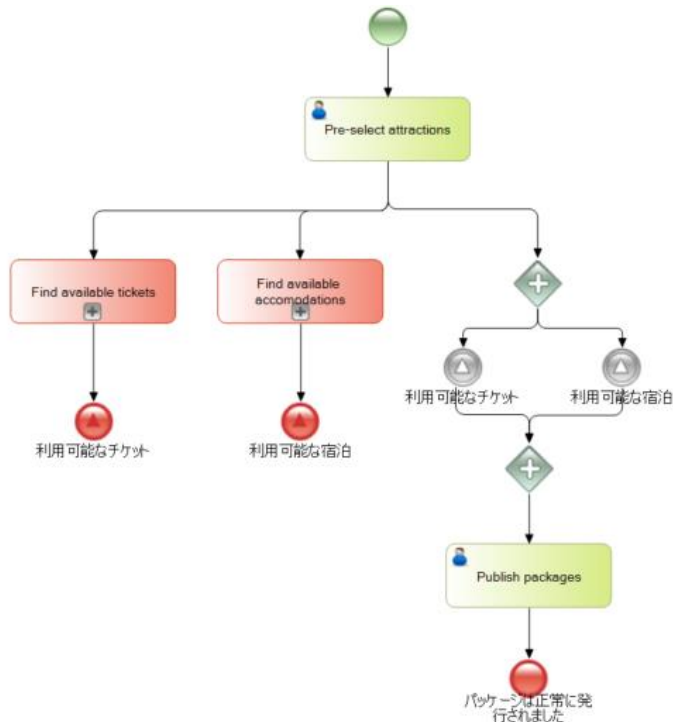
この例では、前のゲートウェイの結果を同期するために排他的ゲートウェイが使用されなかった場合、平行ゲートウェイには 4 つの入力シーケンスコネクタが存在します。ただし、4 つのフローシーケンスのうち一度に通過できるのは 3 つだけです(分岐点に排他的ゲートウェイがあるため)。したがって、プロセスはその平行ゲートウェイで停止します。

モデルに戻り、各パスにシグナルタイプの中間イベントを追加し、それらを平行ゲートウェイから接続します。

左側のパスのシグナルを「利用可能なチケット」と呼び、「Is throw」プロパティを「False」に設定したままにします。このシグナルは「catch」タイプであることに注意してください。チケット検索プロセスの終了時にトリガーされる同じ名前の「throw」シグナルをキャッチします。他のシグナルでも同じことを行います。これを「利用可能な宿泊」とし、こちらも「Is throw」プロパティを「False」のままにします。次に、これは、宿泊施設を検索するプロセスが終了したときに送信されるシグナルをキャッチします。

次に、別の平行ゲートウェイを挿入して両方のパスを結合します。 これにより、両方のシグナルが受信されない限りプロセスは続行できなくなります。

最後に、「Publish packages」というタスクを挿入し、平行ゲートウェイから接続し、「終了イベント/トリガーなし」を追加してプロセスを終了します。これに「パッケージは正常に発行されました」という説明を追加します。



このように実装することで、必要なチケットと宿泊施設を取得しない限りプロセスを完了しないという要件を満たします。

ただし、このプロセスは効率的ではありません。 旅行代理店は、チケットを事前に取得していない場合は宿泊施設を検索するプロセスを開始しないように要求していることに注意してください。 これを行うために、シグナルも使用します。

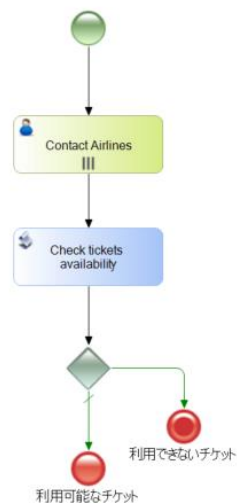
「Find available tickets」プロセスをダブルクリックして定義します。 空のウィンドウが開き、チケットを取得するプロセスを表すシンボルが追加されます。

まず、「開始イベント/トリガーなし」と「Contact Airlines」というユーザータスクを挿入します。 このタスクは連絡先の航空会社ごとに 1 回ずつ複数回実行されるため、タスクの「Loop type」プロパティを「Multi-Instance」に設定します。

取得した情報を調べるために、すべての航空会社にコンタクトを取った後、「Check tickets availability」というスク립トタスクを挿入し、タスク「Contact Airlines」から接続します。

次に、排他的なゲートウェイをドラッグします。

利用可能なチケットがある場合は、終了イベント「Condition type=default」でプロセスを終了します。 旅行パッケージを作成するのに十分なチケットがない場合は、サブプロセスだけでなくメインのパッケージ設計プロセスも終了する終了(終結)イベントでプロセスを終了します。



次に、宿泊施設の検索プロセスを定義して、事前選択したアトラクションのパッケージが提供できるかどうかを確認します。

サブプロセス「Find available accomodations」をダブルクリックします。「開始イベント/トリガーなし」をドラッグし、次にシグナルタイプの中間イベントのシンボルを挿入します。



パッケージのチケット検索プロセスの最後にあるシグナル終了イベントのように、これを「利用可能なチケット」と呼びます。プロパティを開いて、「Is throw」プロパティが False に設定されていることを確認します。これは、チケット検索プロセスの完了時に送信されたシグナルをこのイベントにキャッチするためです。

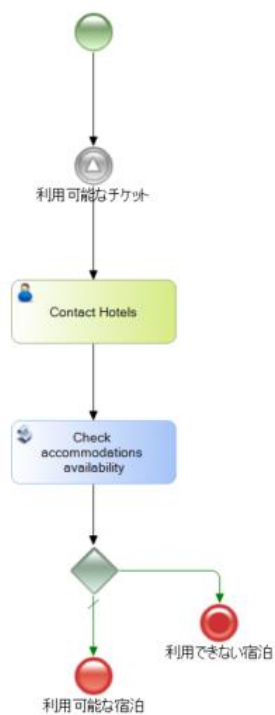
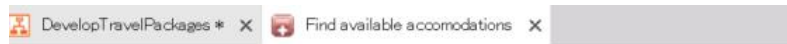


このようにして、宿泊施設を検索するプロセスは、パッケージに利用可能なチケットがある場合にのみ続行されます。

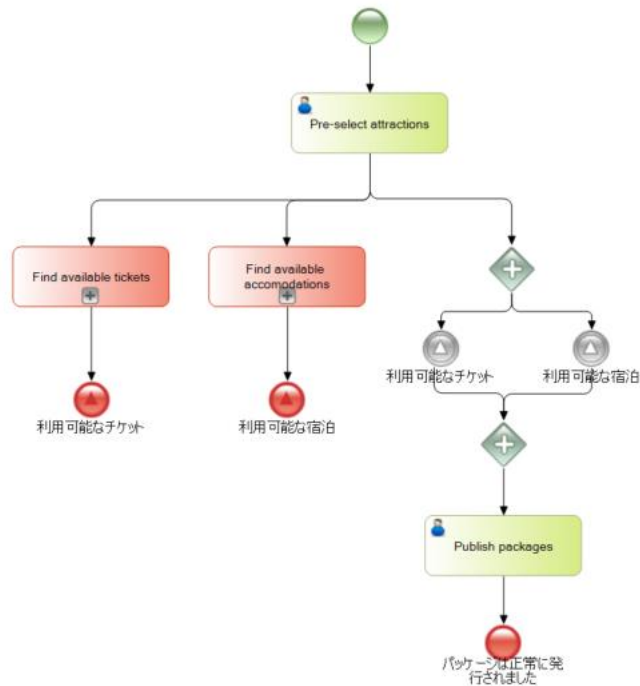
プロセスの定義を続けるために、「Contact Hotels」というユーザータスクを挿入します。このタスクは、連絡するホテルごとに1回ずつ、複数回実行する必要があるため、「Loop type」プロパティを「Multi-Instance」に設定します。

次に、チケット検索プロセスと同様に、設計中のパッケージを提供できる十分なホテルの部屋があるかどうかを確認する、「Check accommodations availability」というスクリプトタスクを挿入します。

最後に、排他的なゲートウェイを挿入し、利用可能なホテルの部屋がある場合（予想される結果）、「利用可能な宿泊」タグを持つ終了イベント(トリガーなし)でプロセスを終了します。それ以外の場合は、タグ「利用できない宿泊」を含む終了(終結)イベントを挿入します。



サブプロセスのウィンドウを閉じ、メインプロセスに戻って保存します。



このようにして、旅行代理店の要求どおりに旅行パッケージを効率的に設計するという目的を満たすプロセスを定義しました。

シグナルは、設定した目的を達成するためにメインプロセスのさまざまな部分間、およびメイン プロセスとそのサブプロセス間の通信方法を提供します。

このトピックの詳細については、以下のリンクを参照してください。

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