Modeling Workflow Patterns

Bizagi Suite

Table of Contents

Modeling workflow patterns

Implementing the patterns

In this document we will explain how to diagram the modeling patterns proposed by Professor Van Der Aalst using the Bizagi Process Modeler. These patterns are very useful to model different situations in real business processes.

Below you will find some examples for each pattern in order to give you an idea about how they work.

Basic control flow patterns

This group of patterns shows the basic aspects for controlling process flows.

WCP 1- Sequence

Description

This pattern is used to model dependencies between tasks so that one task cannot start before another is finished (serial execution).

Example

When an insurance company receives a claim related to a policy, several activities must be performed in a logical sequence. First, the client submits the claim, then, the claim is evaluated in order to authorize its payment and finally the amount is disbursed.

Note that the claim cannot be evaluated before it is submitted nor can the amount of the claim be disbursed before the claim is evaluated.

Implementation

To model this pattern, it is necessary to connect the activities (in the performance order) by using sequence flow connectors as shown in the diagram 1.

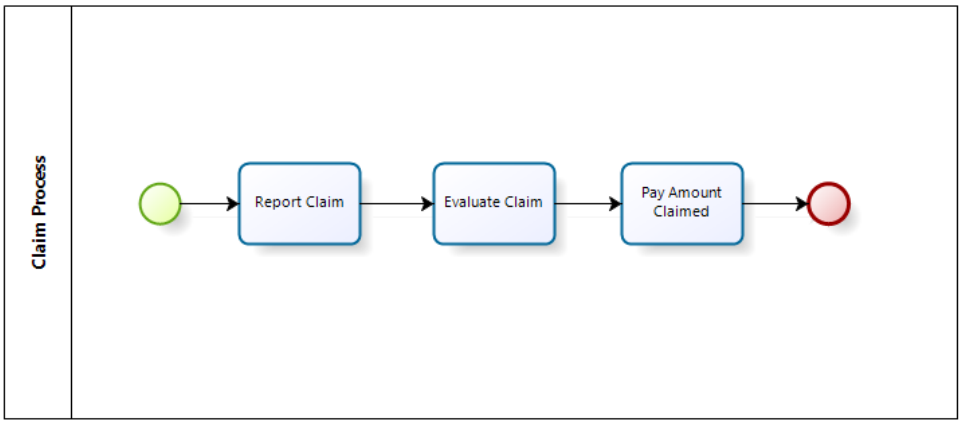


Diagram 1. Sequence pattern example

WCP 2- Parallel Split

Description

A parallel split is a point in the workflow process where a single branch is divided into two or more parallel branches which are executed concurrently [2]

Example

When a new employee arrives to a company, it is necessary to perform many activities such as, grant access to company information, sign some legal documents and set up his or her workstation.

Implementation

To implement the parallel split, it is necessary to use a Parallel gateway. This gateway creates all the alternative paths without checking any conditions.

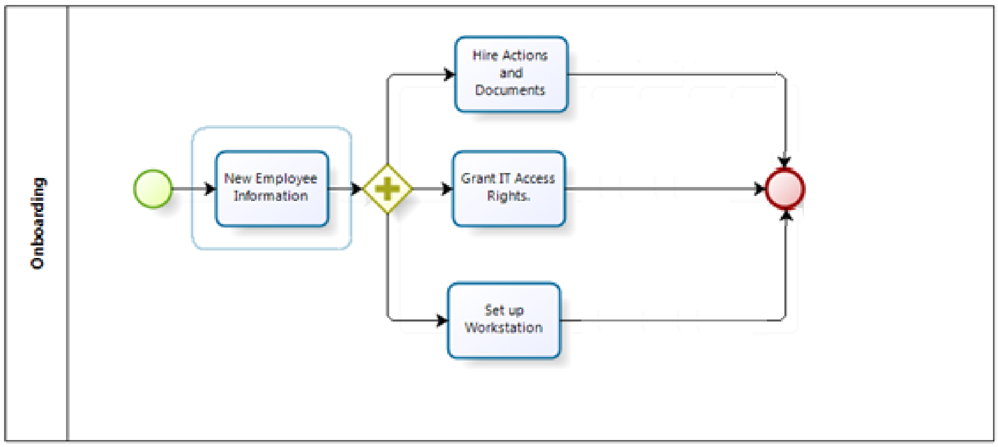


Diagram 2. Parallel split pattern example

WCP 3 - Synchronization

Description

Synchronization is a point in the process where two or more different branches of the process merge into one single branch. It is called synchronization because it expects all merged branches to be completed before it can continue with the next activity [2].

Example

When a new employee arrives to a company it is necessary to perform some activities such as grant access to company information, sign some legal documents and set up his or her workstation. The employee cannot begin to work until all the activities are completed.

Implementation

In the example, we use a Parallel gateway as a convergence means to synchronize the paths previously enabled.

The synchronization pattern can also be modeled with inclusive and exclusive gateways according to the required business conditions.

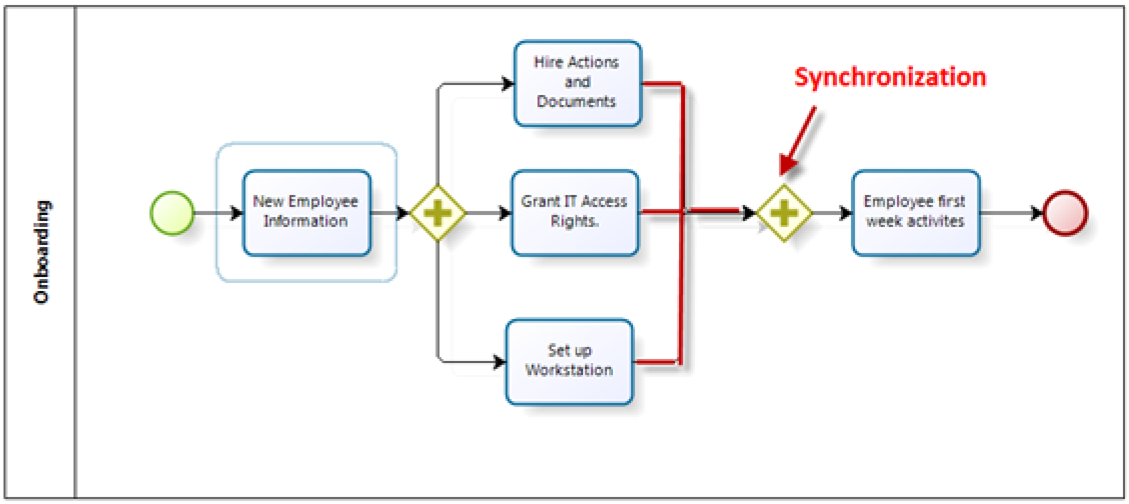


Diagram 3. Synchronization pattern example

WCP 4 - Exclusive Choice

Description

The exclusive choice is a point in the process where a path is chosen from several available paths based on a decision or process data [2].

Example

The Accounting Department receives and pays invoices. The payment of an invoice can be made by a bank transfer, check or credit card. Only one payment method is used.

Implementation

The exclusive choice pattern can be modeled using the Exclusive Gateway. The exclusive decision has three outgoing sequence flows, but only one of them can be taken based on data expression conditions.

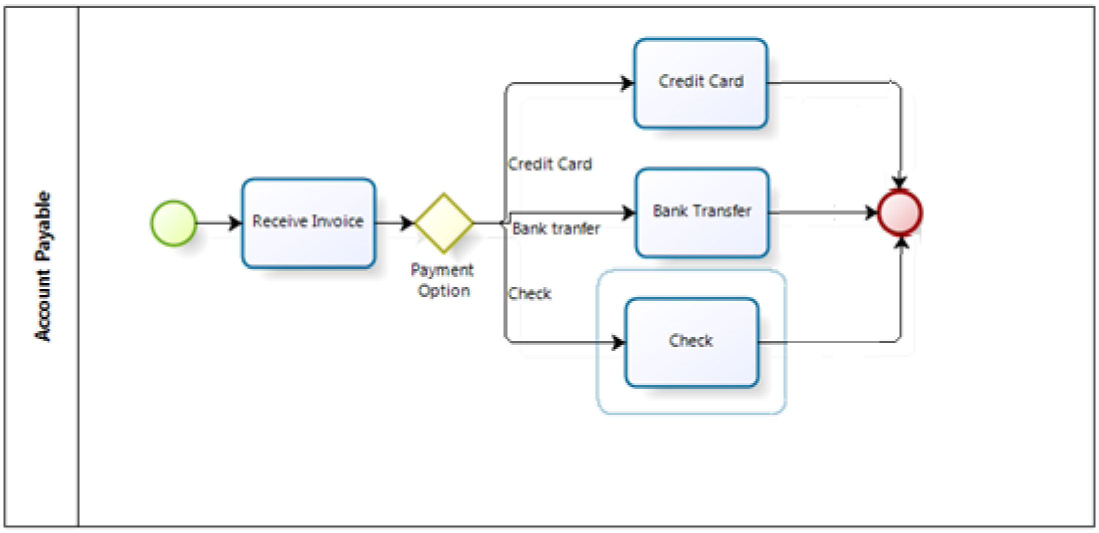


Diagram 4. Exclusive choise pattern example

WCP 5- Simple Merge

Description

The simple merge is a point in the process where two or more alternative branches come together without synchronization. It is an assumption of this pattern that none of the alternative branches are ever executed in parallel [2].

Example

Let´s take the previous example for the WCP 4. Now suppose that after making the payment by one of the possible means, the financial ERP is updated. Note in diagram 4 that the *Update financial ERP* activity will be enabled when only one of the incoming paths is taken.

Implementation

This implementation uses sequence flow connectors to connect the incoming activities with the merge activity.

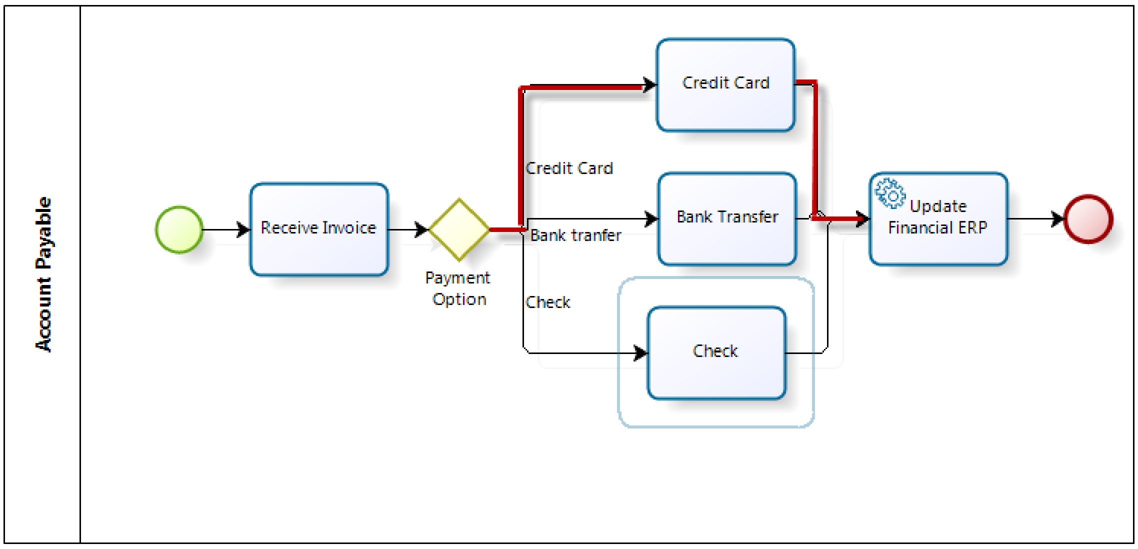


Diagram 5. Simple Merge pattern example

Advanced Branching and Synchronization Patterns

This group presents several patterns which characterize more complex branching and merging concepts.

WCP 6 - Multi-Choice

Description

The Multi-choice pattern is used to model a point in the workflow process where a number of branches are chosen based on a decision or a workflow control data [2].

Example

During an auditory process it is very common to find nonconformities. The nonconformities must be evaluated and corrected by the process owner.

The correction of the nonconformity can be done in different ways, for example, with a corrective action, a preventive action, an immediate action or with more than one of these options.

Implementation

To implement the multi-choice pattern it is necessary to use an Inclusive Gateway. This gateway allows enabling one or more paths according to process data.

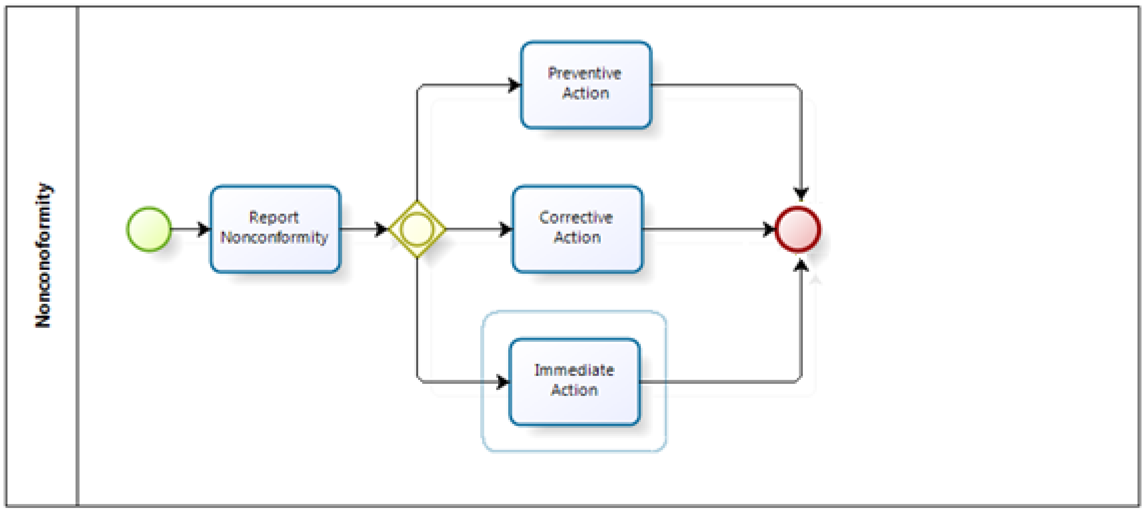


Diagram 6. Multi Choice pattern example

WCP 7 - Structured Synchronizing Merge

Description

It is a point in the process where multiple paths that were enabled earlier in the process converge into one single thread. [2].

Example

Let´s take the previous example for the WCP 4. Now suppose the nonconformity cannot be closed until all the necessary activities are completed.

Implementation

To implement this pattern it is necessary to use two inclusive gateways, one for divergence (activate some of the branches), and the other for the synchronization (synchronize the activated branches).

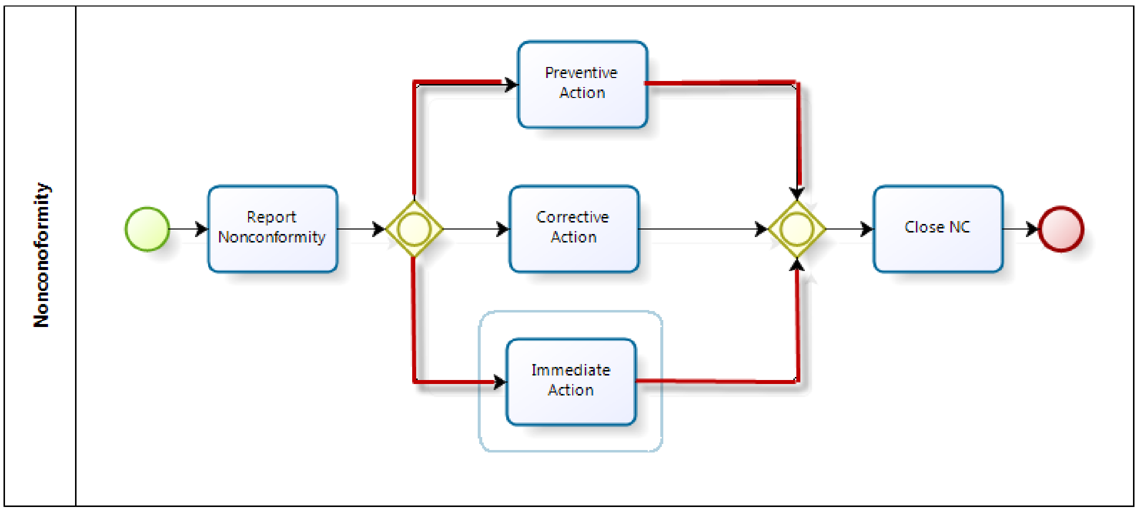


Diagram 7. Structured Synchronizing merge pattern example

WCP 8 - Multi Merge

Description

The multi merge pattern is used to model the convergence of two or more branches into a single path. Each time an incoming branch is enabled it results in the activation of the next activity within the process. [2]

Example

During a Recruitment process it is necessary to check the references of the future employee. It is very important to check the personal and professional references provided by the employee. Each time a reference is checked, the Human Resources Manager must be notified.

Implementation

The pattern uses a parallel gateway to enable two paths. Each time the activities of a path are finished, the next activity is performed (in this case Inform About Reference).

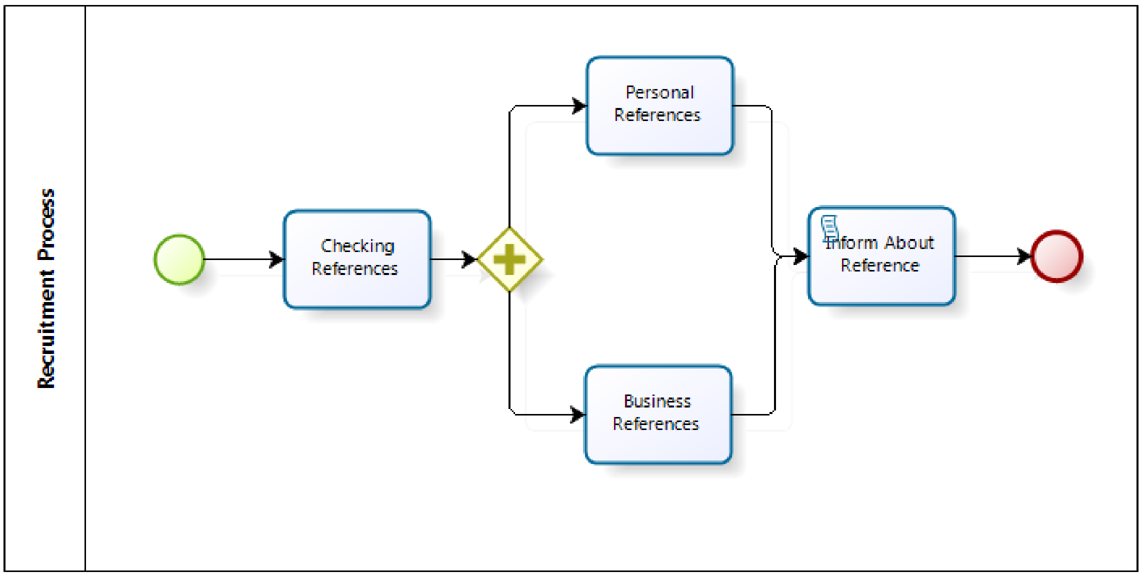


Diagram 8. Multi merge pattern example

WCP 9 - Structured Discriminator

Description

This pattern describes a point in the process that waits for one of the incoming branches to complete before activating the subsequent activity. The other incoming branches are omitted after been completed. Once all the incoming branches are completed the discriminator is reset [2].

Example

An employee requests a loan from the company. To grant the credit to the employee, it must be approved by the boss or by the financial area. When either of them approves the request, the money is disbursed to the employee.

Implementation

This pattern uses Parallel or Inclusive gateways to split a branch. To model a discriminator it is necessary to use a Complex gateway that waits for one of the incoming branches for the process to continue.

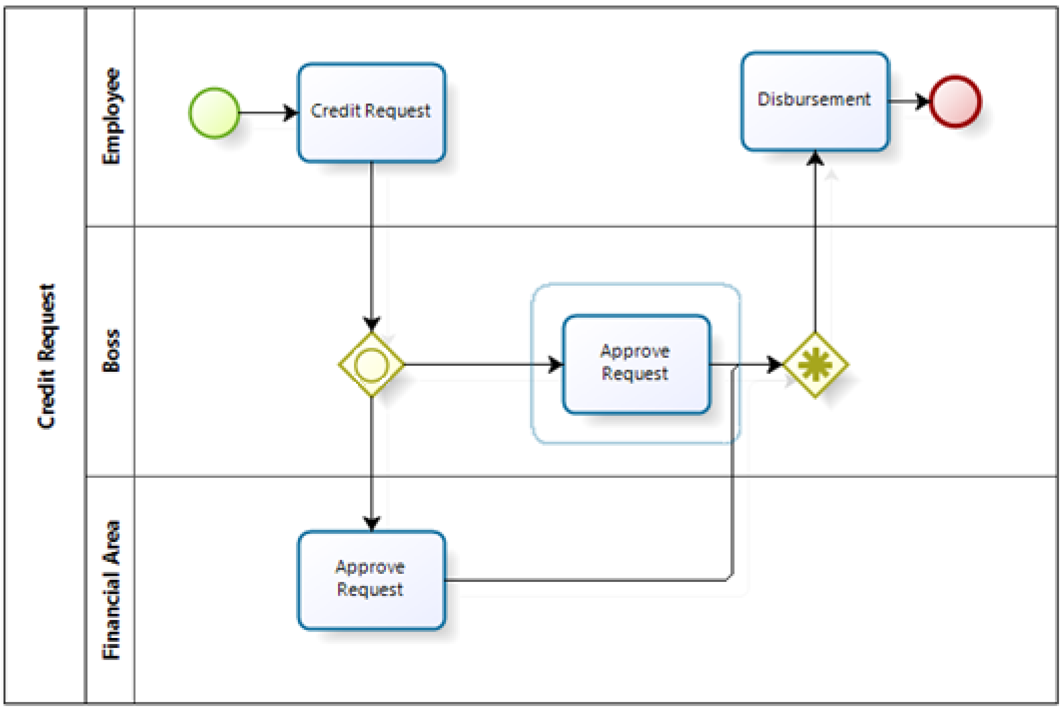


Diagram 9. Structured discriminator pattern example

WCP 30 - Structured Partial Join

Description

This pattern models the convergence of M branches into a single subsequent branch following a corresponding divergence earlier in the process model. The process continues when N of the incoming branches have been enabled.

Example

An employee requests a loan from the company. To grant the credit to the employee, it must be approved by the boss and by the financial area. When both of them approve the request, the money is disbursed to the employee.

Implementation

This pattern uses Parallel or Inclusive gateways to split a branch. A Complex Gateway is used as a convergence means because the number of branches (N) needed for the process to continue can be defined.

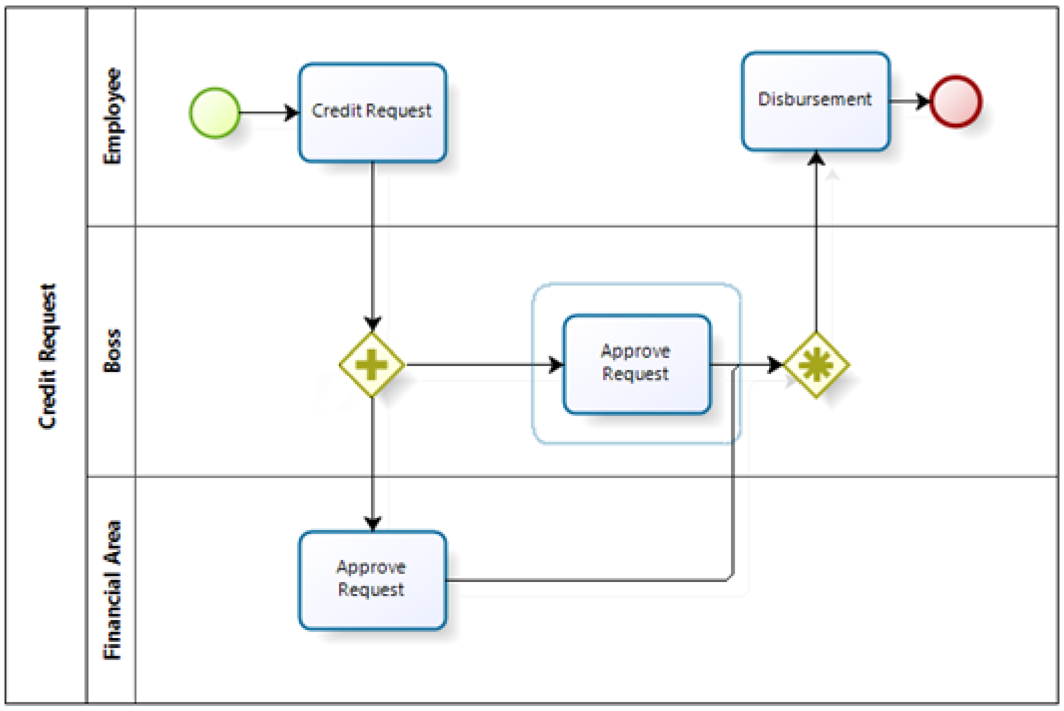


Diagram 10. Structured Partial Join pattern example

WCP 31 - Blocking Partial Join

Description

The convergence of two or more branches into a single subsequent branch following one or more corresponding divergences earlier in the process model. The thread of control is passed to the subsequent branch when N of the incoming branches have been enabled. The join construct resets when all active incoming branches have been enabled once for the same process instance. Subsequent incoming branches cannot be enabled until the join has reset.[1]

Example

A company decides to invest in business proposals. These proposals are submitted by students from many universities and each one is evaluated by 3 judges individually. If 2 of the 3 judges give their approval, the applicant will be notified about the decision. When the last judge gives his/her deliberation a new proposal can be evaluated.

Implementation

This pattern uses *Parallel* or *Inclusive gateways* to split a branch. A complex gateway is used as a convergence to define the number of branches (N) needed for the process to continue.

In order to block further incoming branches until the join is reset, a *Conditional Event* is used. This event evaluates if there are approvals in progress and it will stop the incoming flows until all the approvals have been issued.

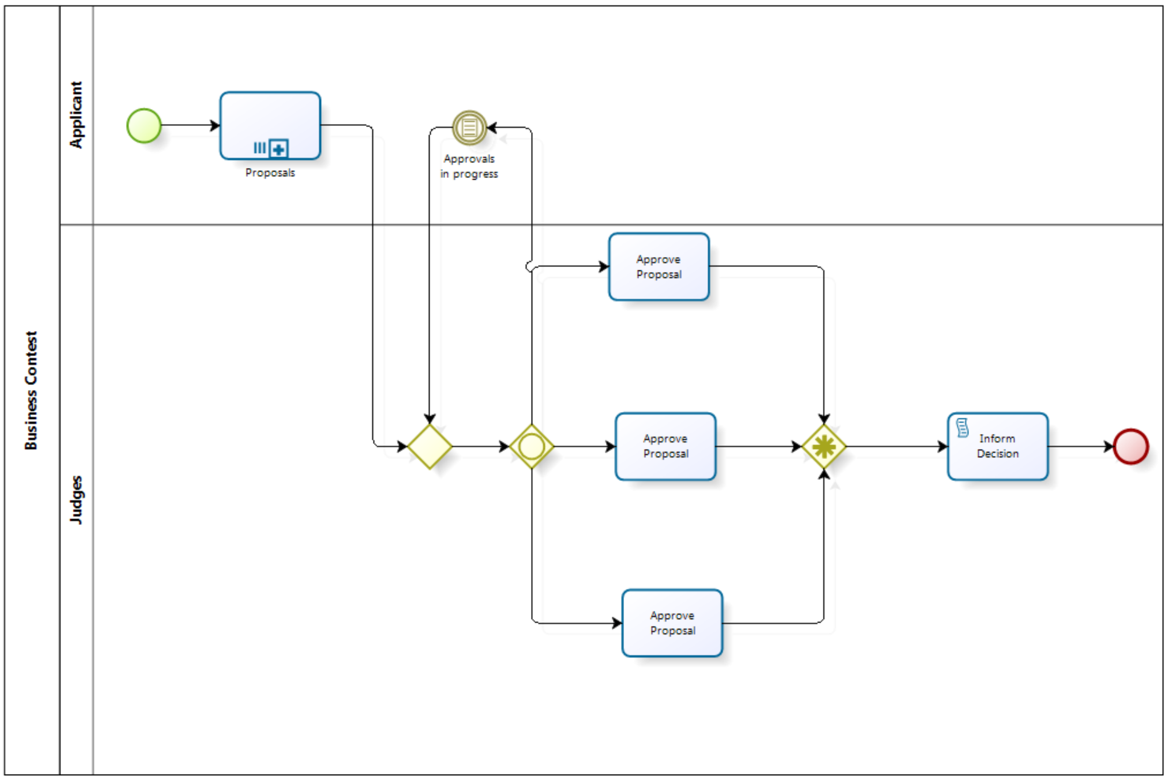


Diagram 11. Blocking Partial Join pattern example

WCP 32 - Cancelling Partial Join

Description

The convergence of two or more branches into a single subsequent branch following one or more corresponding divergences earlier in the process model. The thread of control is passed to the subsequent branch when N of the incoming branches have been enabled. Triggering the join also cancels the execution of all of the other incoming branches and resets the construct.

Example

A company decides to invest in business proposals. These proposals are submitted by students from many universities and each one is evaluated by 3 judges individually. If 2 of the 3 judges give their approval, the applicant will be notified about the decision, the remaining activity for the last approval will be disabled, and a new proposal can be evaluated.

Implementation

This pattern uses *Parallel* or *Inclusive gateways* to split a branch. A complex gateway is used as a convergence to define the number of branches (N) needed for the process to continue.

In order to block incoming branches until the join is reset, a *Conditional Event* is used. This event evaluates if there are approvals in progress and it will stop the incoming flows until all the necessary approvals have been issued.

Once N of the M incoming branches have been enabled, an escalation event is triggered to cancel the pending or uncompleted approvals.

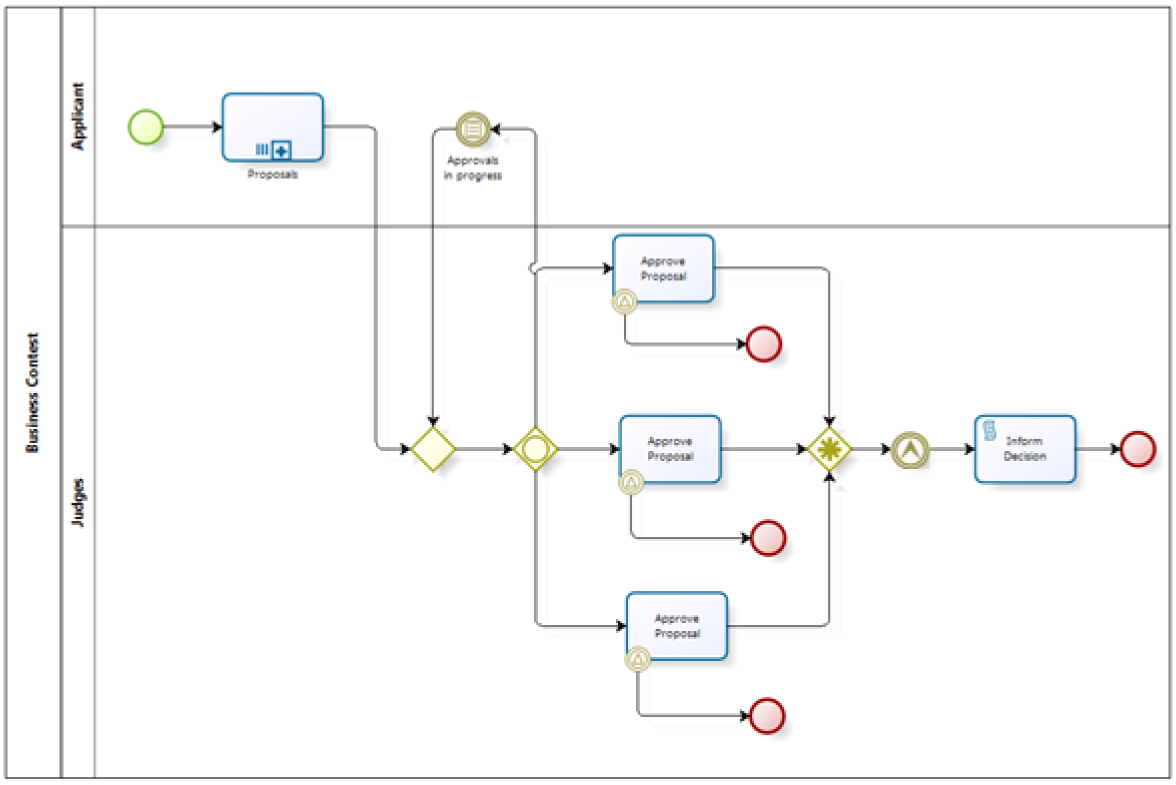


Diagram 12. Cancelling Partial Join pattern example

WCP 33 - Generalized AND Join

Description

The generalized AND join pattern describes how to merge two or more branches into one single outgoing branch once all the incoming branches have been enabled. Unlike the Synchronization pattern, it supports the situation where one or more incoming branches may receive multiple triggers for the same process instance [2]

Example

When a new employee arrives to a company it is necessary to perform some activities such as grant access to company information, sign some legal documents and set up his or her workstation. Some of these activities are multiple activities. The employee may not begin to work until all the activities are completed.

Implementation

In the example, we use a *Parallel gateway* as a convergence to synchronize the paths previously enabled.

The synchronization pattern can also be modeled with inclusive and exclusive gateways according to the required business conditions.

The sub process synchronization is done internally, it means the conditions to the sub process to be considered as completed are defined in its properties so when it happens, its related path will reach the *convergence parallel gateway* and will wait for the other active paths to allow the process to continue its normal flow.

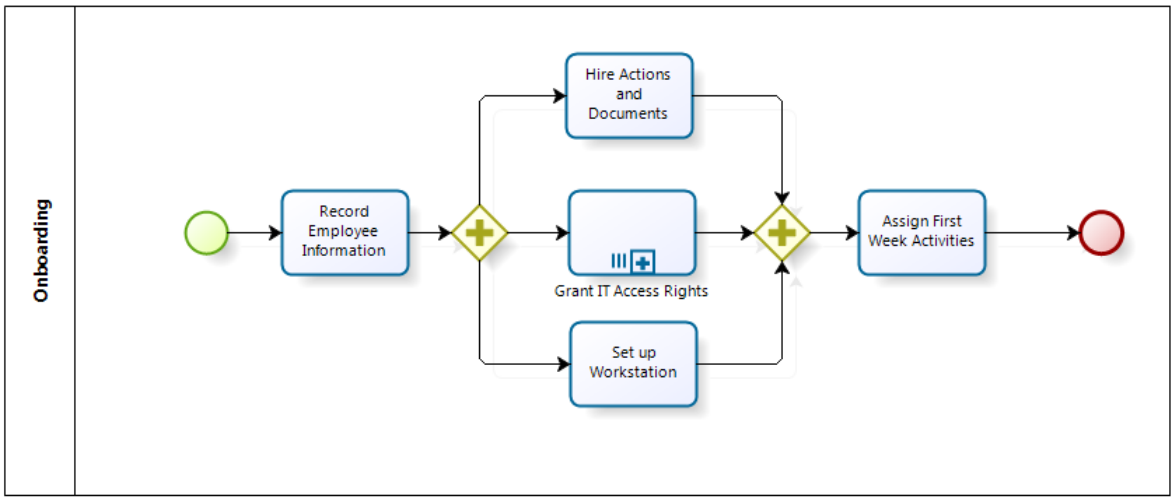

Onboarding
Record Employee Information
Hire Actions and Documents
Grant IT Access Rights
Set up Workstation
Assign First Week Activities

Diagram 13. Generalized AND join pattern example

WCP 37 – Acyclic Synchronizing Merge

Description

It describes the convergence of two or more branches which diverged earlier in the process into a single subsequent branch. Determination of how many branches require synchronization is made on the basis of information available locally to the merge construct.

Example

The marketing department of a company has three possible means to perform a marketing campaign. These means are radio, television and daily newspaper. If the marketing department decides to perform the campaign via radio or television these two will be launched at the same time. If marketing decides to use the daily newspaper they can decide later if the radio and television are launched at the same time or they first launch the daily newspaper and later launch the other means.

Implementation

This pattern can be modeled by using *Conditional Events.* These events are used to coordinate the synchronization between the necessary paths without forcing synchronization of all active paths. The conditional events will allow the flow to continue when the activities that must be coordinated have been accomplished.

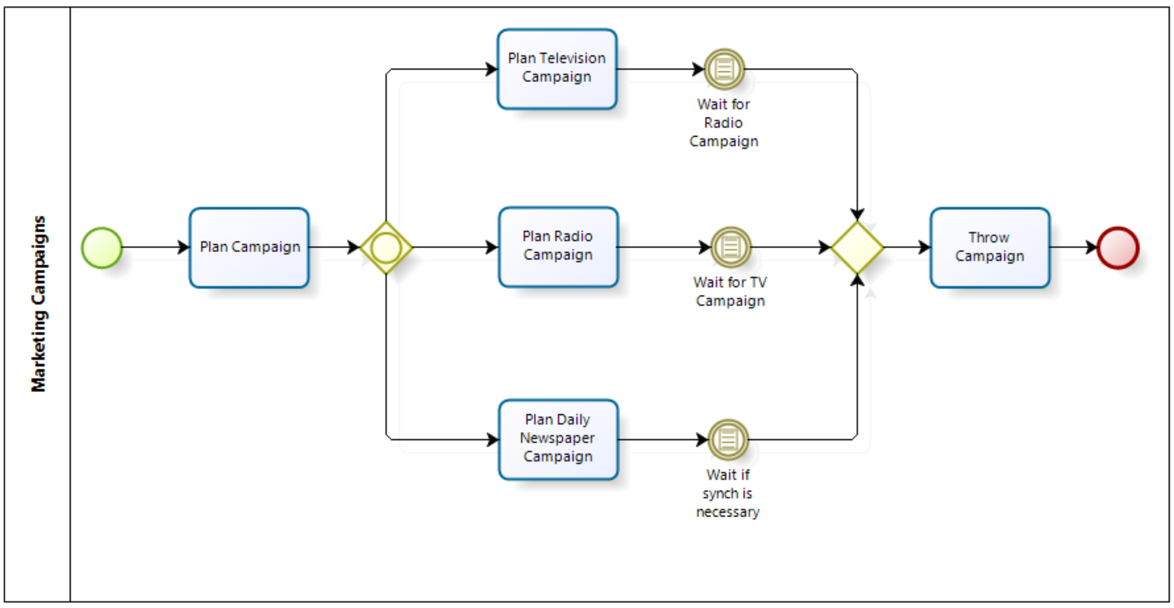


Diagram 14.Acyclic Synchronizing merge pattern example

WCP 38 - General Synchronizing Merge

Description

It describes the convergence of two or more branches which diverged earlier in a single branch. The process continues when all the active incoming branches have been enabled or it is not possible that a branch will be enabled at any future time [2].

Example

A couple is going to get married. A wedding agency coordinates the main activities during this special day. They plan the wedding and receive the bride and groom, their parents and siblings in order to organize the ceremony. However the siblings are abroad so they are not expected until the last moment. If they do not arrive on time they are not received and the wedding begins without them.

Implementation

This pattern uses *Interrupting events*. This kind of event is attached to the boundaries of an activity and it will activate an exceptional flow. Depending on the people who attend the ceremony, the *Inclusive gateway* enables their reception. One or more activities have interrupting events attached to allow the process to continue even if unexpected situations arise. In this case this situation is that the siblings do not arrive on time.

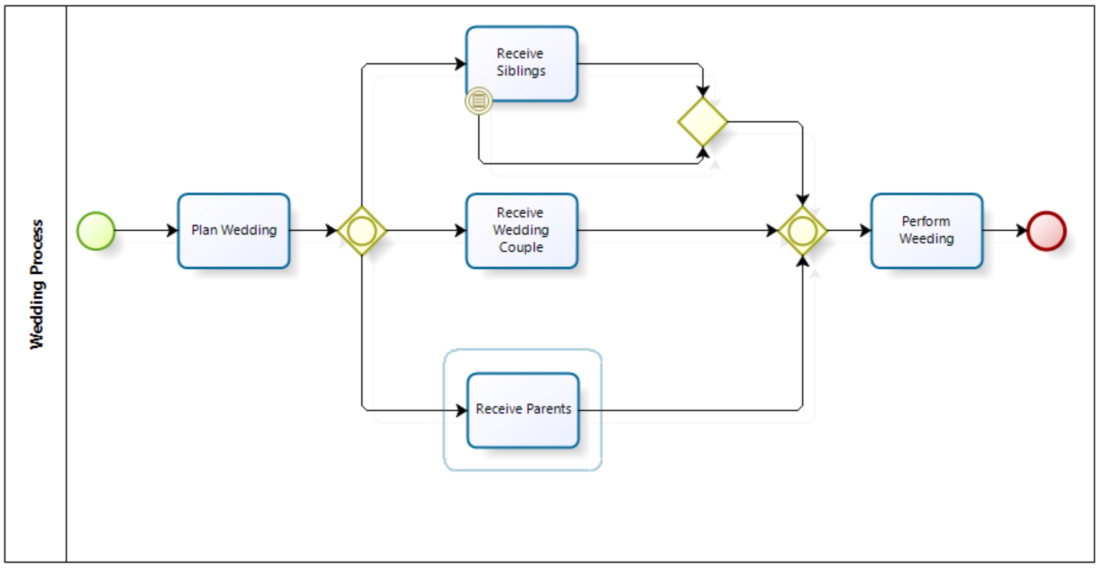


Diagram 15.General Synchronizing merge pattern example

WCP 41 - Thread Merge

Description

The pattern describes a point in a process where a distinct number of execution threads in a single branch of the same process instance should be merged together into a single thread of execution [2].

Example

A company decides to evaluate the perception of their employees about the job environment. An inquiry is sent to 100 employees. Once they have replied, the analyst evaluates them.

Implementation

To model this pattern it is necessary to use sub processes in order to create multiple instances of an activity. In this case, the Reply inquiry sub process is instantiated 100 times.

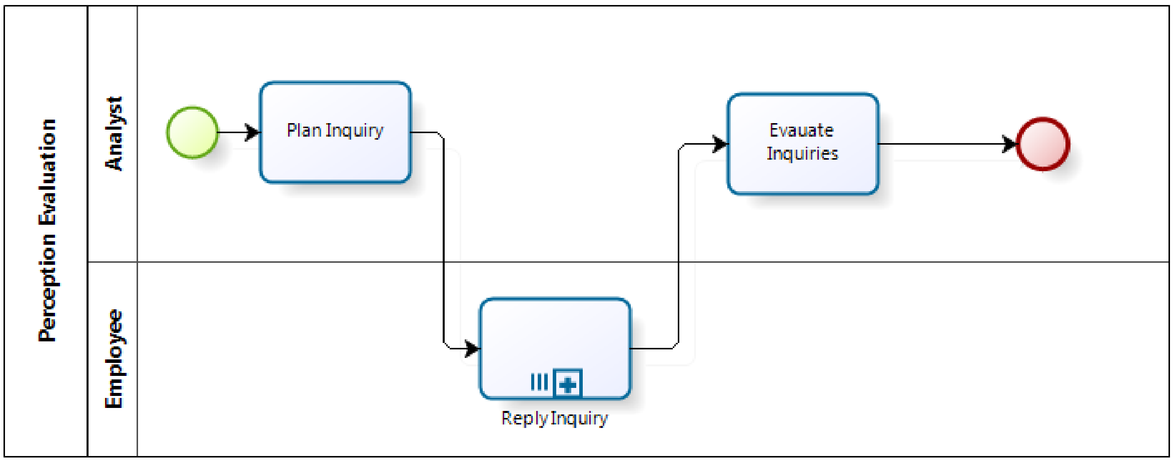
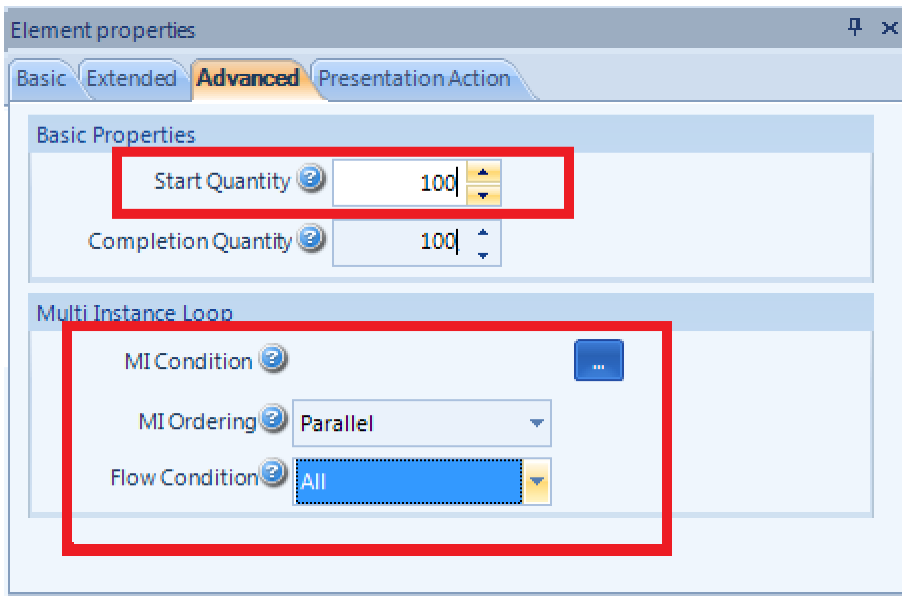


Diagram 16. Thread Merge pattern example

In order to configure the sub process behavior for following the required conditions, the following properties have to be defined:



Start Quantity: 100
Completion Quantity: 100
MI Condition: ...
MI Ordering: Parallel
Flow Condition: All

Start Quantity = 100: It means the sub process will be instantiated 100 times.

Completion Quantity= 100: The sub process is considered as complete when all instances have been accomplished.

MI Condition = None: It means there is no condition to the number of instances for the sub process

MI Ordering = Parallel: The sub process instances will be executed in parallel.

Flow Condition = All: The flow will continue to the next activity when they are all completed.

WCP 42 - Thread Split

Description

It describes that at a given point in a process, a nominated number of execution threads can be initiated in a single branch of the same process instance [2].

Example

A company decides to evaluate the perception of their employees about the job environment. An inquiry is sent to 100 employees. Inquiries are evaluated by the analyst once they are replied.

Implementation

To model this pattern it is necessary to use sub processes in order to create multiple instances of an activity. In this case, the Reply inquiry sub process is instantiated 100 times.

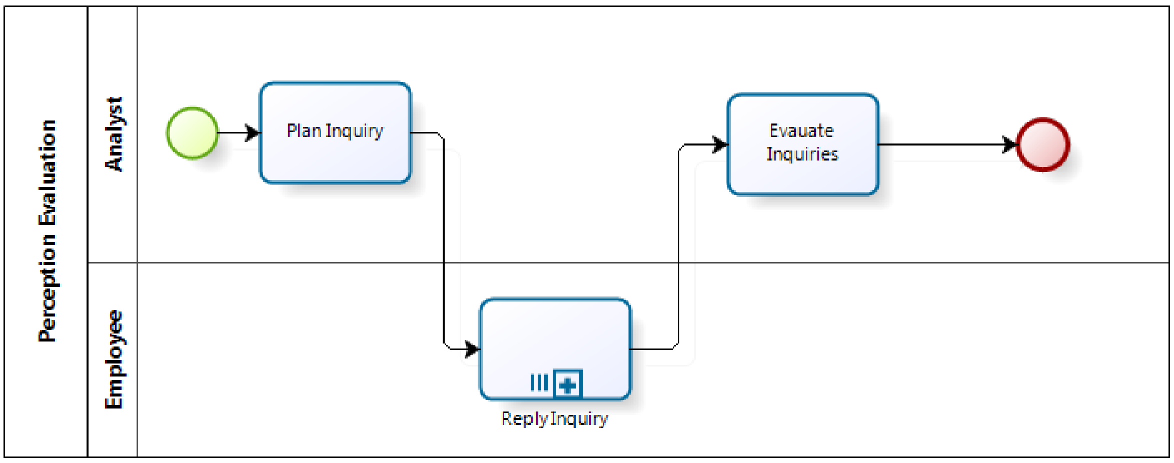
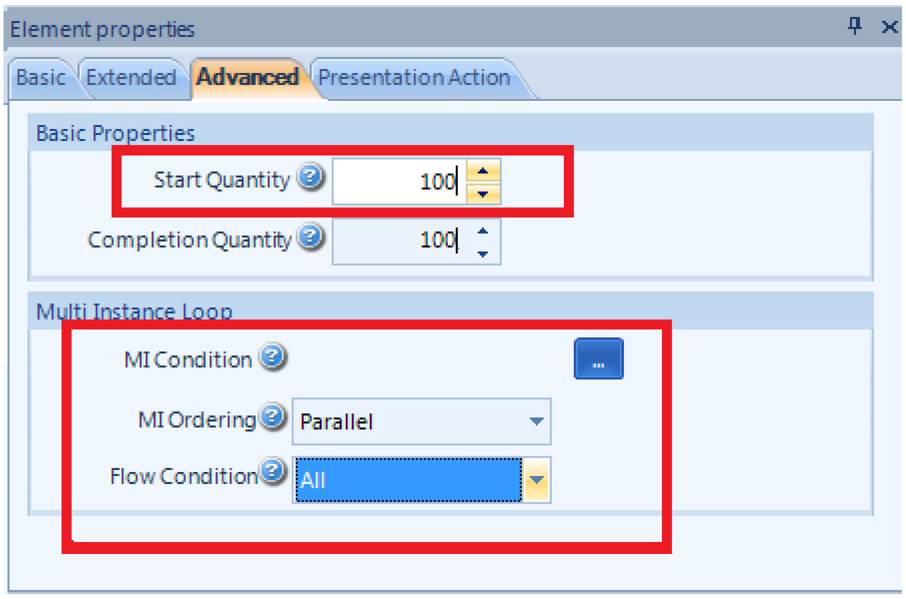


Diagram 17. Thread Split pattern example

In order to configure the sub process behavior for following the required conditions, the following properties have to be defined:



Start Quantity: 100

Completion Quantity: 100

MI Ordering: Parallel

Flow Condition: All

Start Quantity = 100: It means the sub process will be instantiated 100 times.

Completion Quantity=100: The sub process is considered as complete when all the instances have been accomplished.

MI Condition = None: It means there is no condition to the number of instances for the sub process

MI Ordering = Parallel: The sub process instances will be executed in parallel.

Flow Condition = All: The flow will continue to the next activity when all instances are completed.

Multiple Instances Patterns

Multiple instances patterns describe situations where there are multiple threads of execution active in a process model which relate to the same activity (and hence share the same implementation definition) [1]. Multiple instances can arise for three different reasons:

1. An activity is able to initiate multiple instances of itself when triggered.

2. An activity is initiated multiple times as a consequence of receiving several independent triggers, for example as a part of a loop.

3. Two or more activities in a process share the same implementation definition.

WCP 12 - Multiple Instances without Synchronization

Description

This pattern is used to model activities that have to be instantiated many times within a process and do not need to be synchronized for the flow to continue. This pattern is also known as Multi-threading without synchronization or Spawn off facility

Example

A company is planning to implement a new project. The stakeholders are notified about the project scope and they can send their comments or suggestions if they wish. The process flow continues even if no suggestions are received.

Implementation

To model this pattern it is necessary to use sub processes in order to create multiple instances of an activity. The "Suggestions" sub process is enabled for the stakeholders to make their comments or suggestions. The process continues its normal flow once the sub process instances are activated.

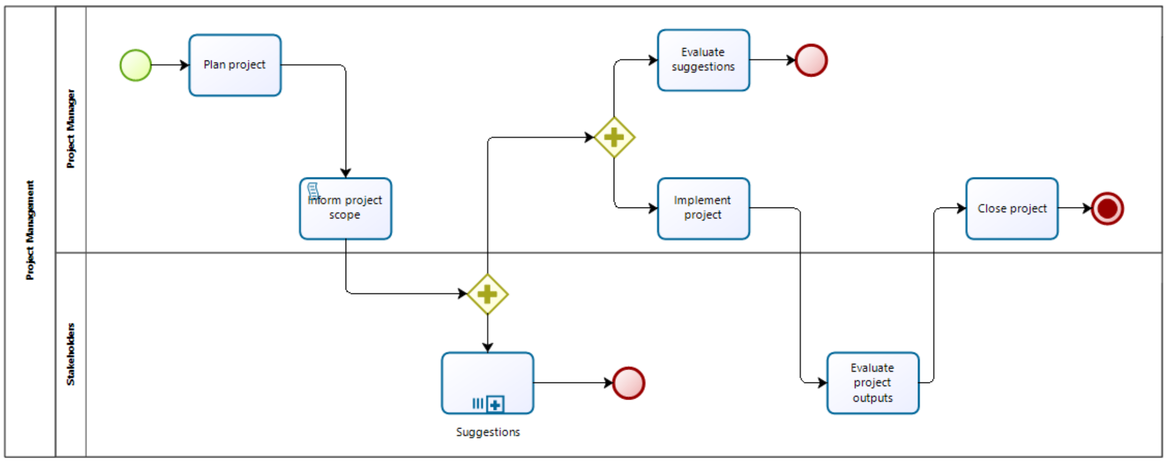
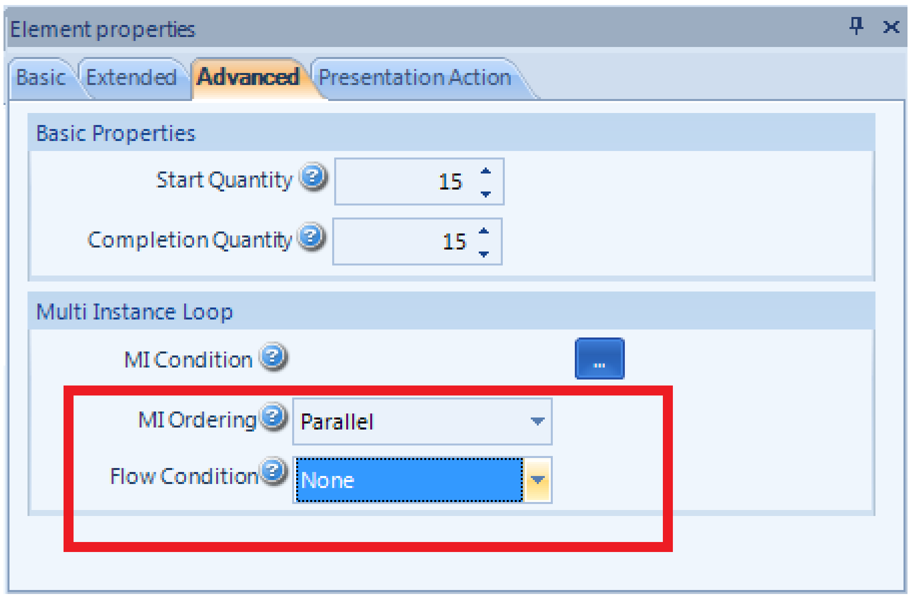


Diagram 18. Multiple Instances Without Synchronization pattern example

In order to configure the sub process behavior for following the required conditions, the following properties have to be defined:



Start Quantity: 15

Completion Quantity: 15

MI Ordering: Parallel

Flow Condition: None

Start Quantity =15 : It means the sub process will be instantiated 7 times. Suppose there are 15 stakeholders.

Completion Quantity= 15: The sub process is considered as complete when all the instances have been accomplished.

MI Condition = None: It means there is no condition to the number of instances for the sub process

MI Ordering = Parallel: The sub process instances will be executed in parallel.

Flow Condition = None: The flow will continue to the next activity every time an instance is completed.

WCP 13- Multiple Instances with a Priori Design-Time Knowledge

Description

This pattern allows an activity to be instantiated multiple times in a process. The amount of instances is known and they are executed concurrently. It is necessary to finish all the activity instances before subsequent activities can be enabled.

Example

A company decides to evaluate the perception of the employees about the job environment. An inquiry is sent to 100 employees. Once all of them have replied, the analyst starts its evaluation.

Implementation

To model this pattern it is necessary to use sub processes in order to create multiple instances of an activity. In this case, the Reply inquiry sub process is instantiated 100 times.

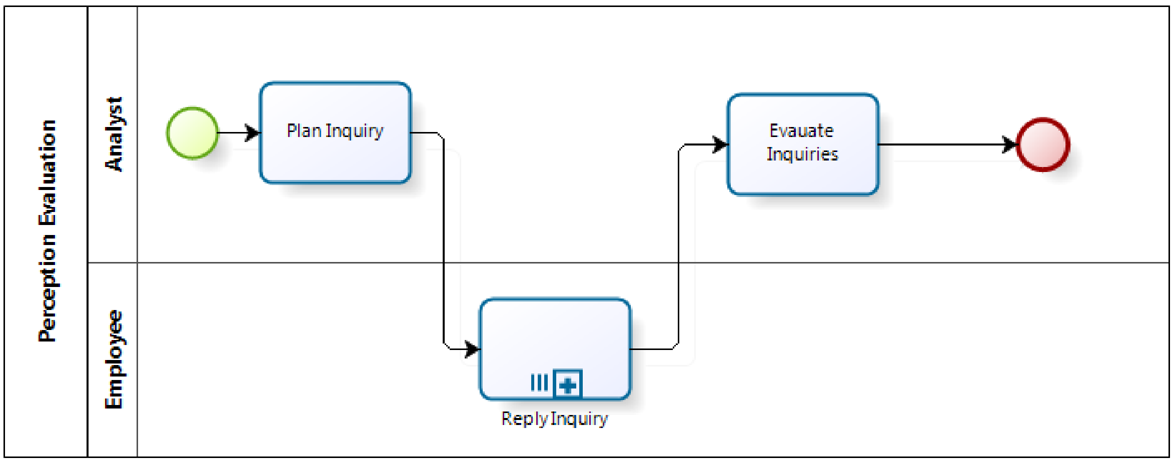
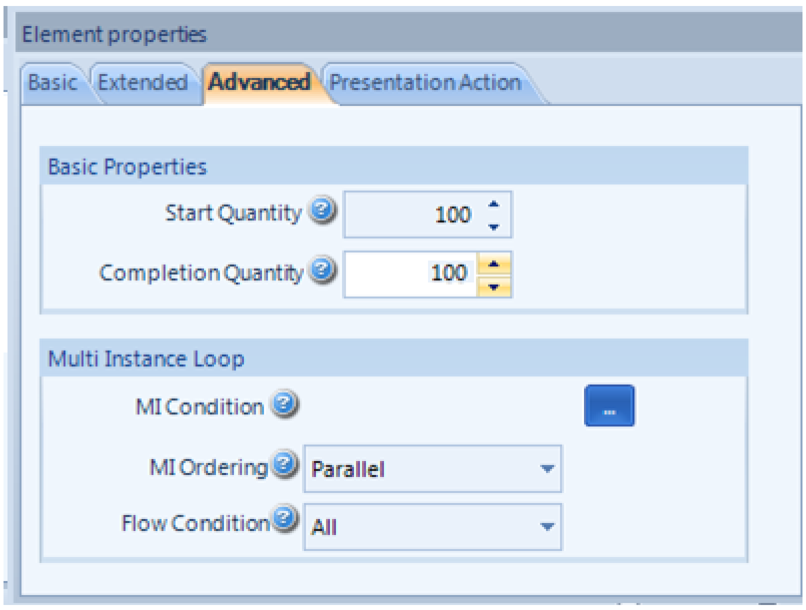


Diagram 19. Multiple Instances without a priori design-time knowledge example

In order to configure the sub process behavior for following the required conditions, the following properties have to be defined:



Basic Properties

Start Quantity 100

Completion Quantity 100

Multi Instance Loop

MI Condition

MI Ordering Parallel

Flow Condition All

Start Quantity =100 : It means the sub process will be instantiated 100 times.

Completion Quantity: 100: The sub process is considered as complete when all the instances have been accomplished.

MI Condition = None: It means there is no condition to the number of instances for the sub process

MI Ordering = Parallel: The sub process instances will be executed in parallel.

Flow Condition = All: The flow will continue to the next activity when all the instances are completed.

WCP 14 - Multiple Instances with a Priori Run-Time Knowledge

Description

This pattern allows an activity to be instantiated multiple times in a process. The number of instances is not known but they must be created during the run time, before the activity is executed. Activities are run concurrently and it is necessary to finish all the instances before subsequent tasks can be enabled.

Example

A company decides to evaluate the perception of their employees about the job environment. A set of employees is chosen at random and an inquiry is sent them. Once all the inquiries are replied to, an analyst starts their evaluation.

Implementation

To model this pattern it is necessary to use sub processes in order to create multiple instances of an activity. In this case, the *Reply inquiry* sub process is instantiated as many times as employees who have replied to the inquiry in the *Plan Inquiry activity*.

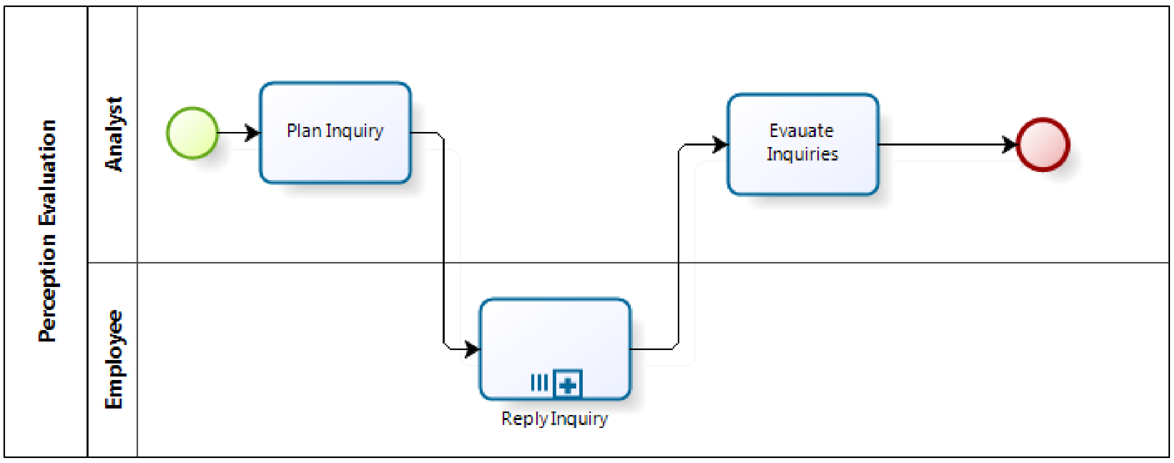
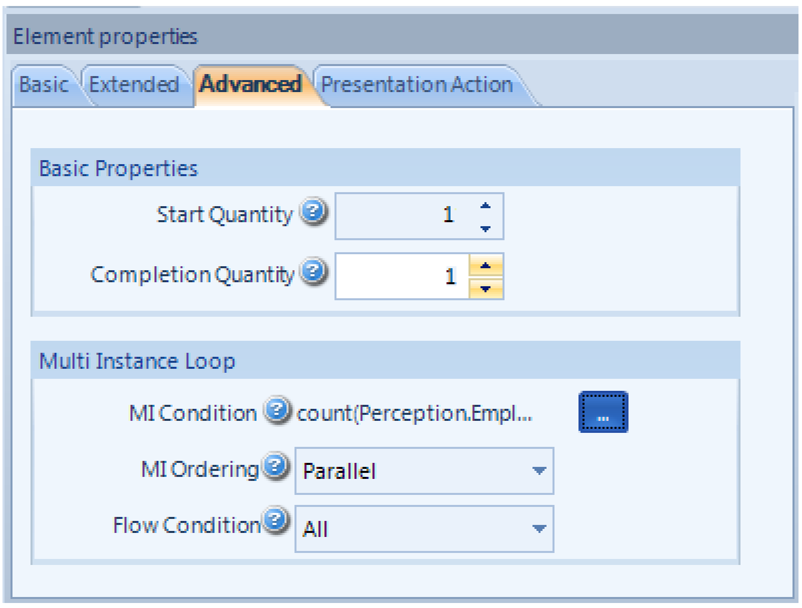


Diagram 20. Multiple Instances with a priori run-time knowledge example

To configure the sub process to follow the required conditions, the following properties have to be defined:



Element properties

Basic | Extended | Advanced | Presentation Action

Basic Properties

Start Quantity: 1

Completion Quantity: 1

Multi Instance Loop

MI Condition: count(Perception.Empl...

MI Ordering: Parallel

Flow Condition: All

MI Condition = count(Perception.Employees): It means there will be as many instances as employees recorded in a master entity to reply to the inquiries.

MI Ordering = Parallel: The sub process instances will be executed in parallel.

Flow Condition = All: The process will continue when all the instances are completed.

WCP 15 - Multiple Instances without a Priori Design-Time Knowledge

Description

This pattern allows an activity to be instantiated multiple times in a process. The number of instances is not known during the design or run time. Activities are run concurrently but, whilst they are running, new ones can be created. It is necessary to finish all the instances before subsequent tasks can be enabled.

Example

A project committee holds a meeting to start the execution of a project. They plan the activities that must be performed and then, they are executed. However, unexpected situations can arise and new tasks might be required. The committee can create as many tasks as necessary before all the active instances finish.

Implementation

Diagram 21 shows how to model this pattern. In the *Committee Meeting Activity* an initial number of sub process instances is set, then, an event is used in order to be able to create new ones. Once the active instances are completed, the event is disabled and no additional instances can be created.

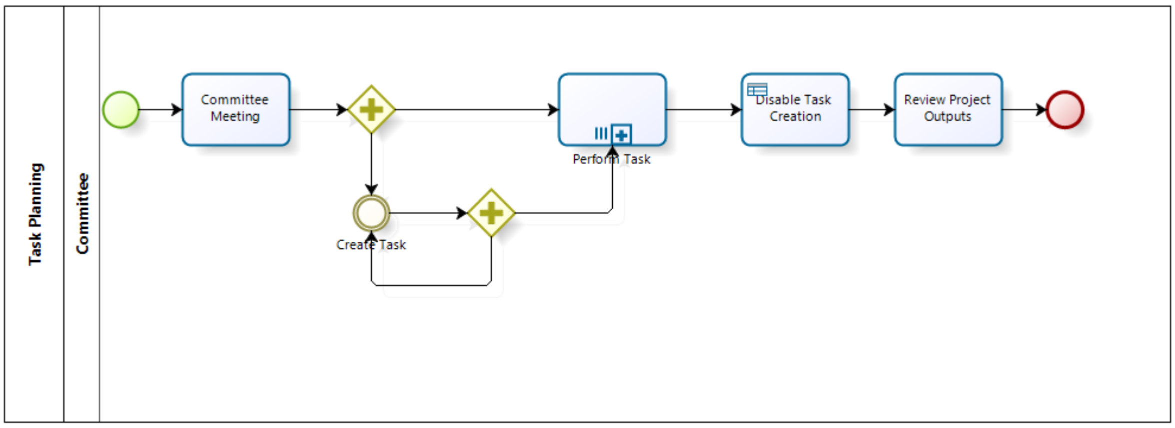
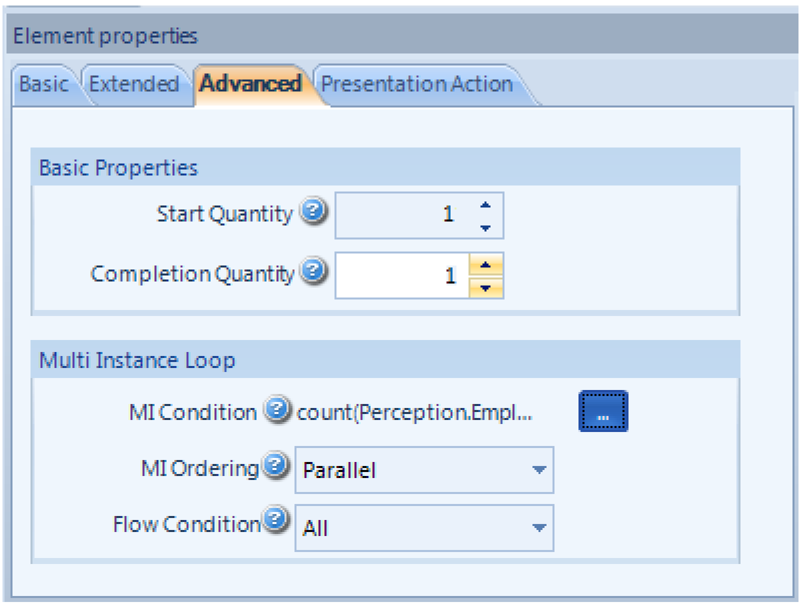


Diagram 21. Multiple Instances without a priori design-time knowledge example

To configure the sub process to follow the required conditions, the following properties have to be defined:



Basic Properties

Start Quantity: 1

Completion Quantity: 1

Multi Instance Loop

MI Condition: count(Perception.Empl...

MI Ordering: Parallel

Flow Condition: All

MI Condition = count(Project.Task): It means there will be as many instances tasks recorded in an established master entity.

MI Ordering = Parallel: The sub process instances will be executed in parallel.

Flow Condition = All: The process will continue when all of the instances are completed.

WCP 34 - Static Partial Join for Multiple Instances

Description

This pattern allows an activity to be instantiated multiple times in a process. The number of instances is known during the design or run time. Activities are run concurrently, and only *N* of the *M* instances created (where N<M) must be finished before subsequent tasks can be enabled. Subsequent completions of the remaining M-N instances are inconsequential. [1]

Example

A company decides to evaluate the perception of their employees about the job environment. An inquiry is sent to 100 employees but an analyst waits for at least 60 replies before starting the evaluation.

Implementation

To model this pattern it is necessary to use sub processes in order to create multiple instances of an activity. In this case, the *Reply inquiry* sub process is instantiated 100 times. A complex gateway is used to control that at least 60 inquiries have been replied in order to start their evaluation.

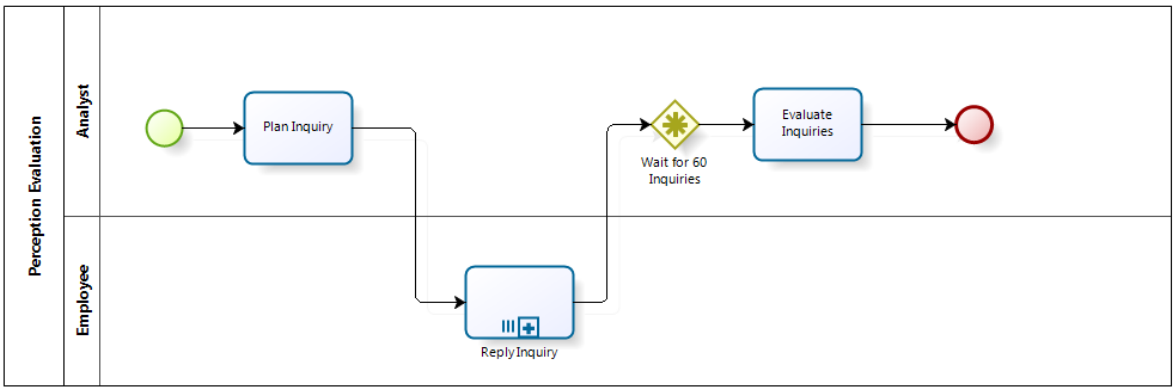
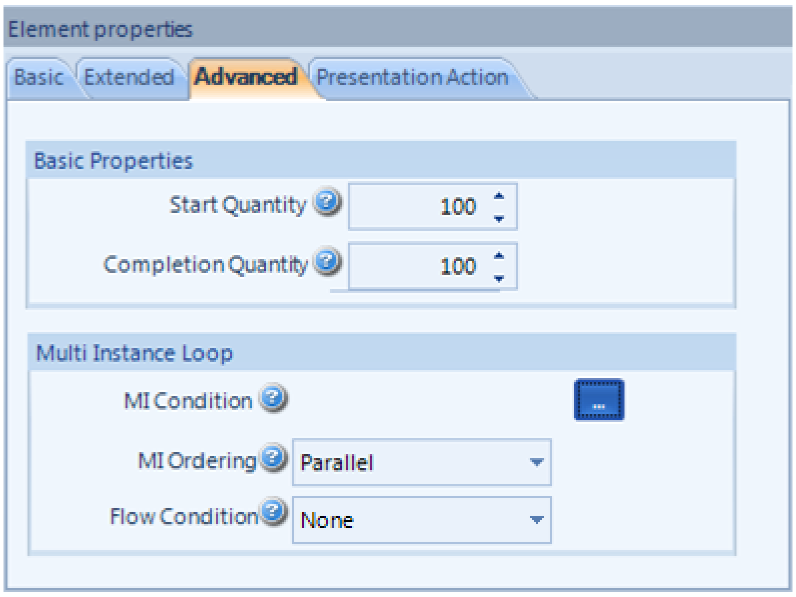


Diagram 22. Multiple Instances without a priori design-time knowledge example

In order to configure the sub process behavior for following the required conditions, the following properties have to be defined:


Element properties

Basic | Extended | Advanced | Presentation Action

Basic Properties

Start Quantity: 100

Completion Quantity: 100

Multi Instance Loop

MI Condition

MI Ordering: Parallel

Flow Condition: None

Start Quantity =100 : It means the sub process will be instantiated 100 times.

Completion Quantity=100: The sub process is considered as completed when all the instances have been accomplished.

MI Condition = None: It means there is no condition to the number of instances for the sub process

MI Ordering = Parallel: The sub process instances will be executed in parallel.

Flow Condition = None: The flow will continue to the next activity each time an instance is completed.

WCP 35 - Cancelling Partial Join for Multiple Instances

Description

This pattern allows an activity to be instantiated multiple times in a process. The number of instances is known during the design or run time. Activities are run concurrently, and only *N* of the *M* instances created (where N<M) must be finished before subsequent tasks can be enabled. Once the *N* instances have been completed, the remaining M-N instances are cancelled.

Example

A company decides to evaluate the perception of their employees about the job environment. An inquiry is sent to 100 employees but an analyst waits for at least 60 replies before starting the evaluation. The remaining 40 inquiries will be cancelled.

Implementation

To model this pattern it is necessary to use sub processes in order to create multiple instances of an activity. In this case, the *Reply inquiry* sub process is instantiated 100 times.

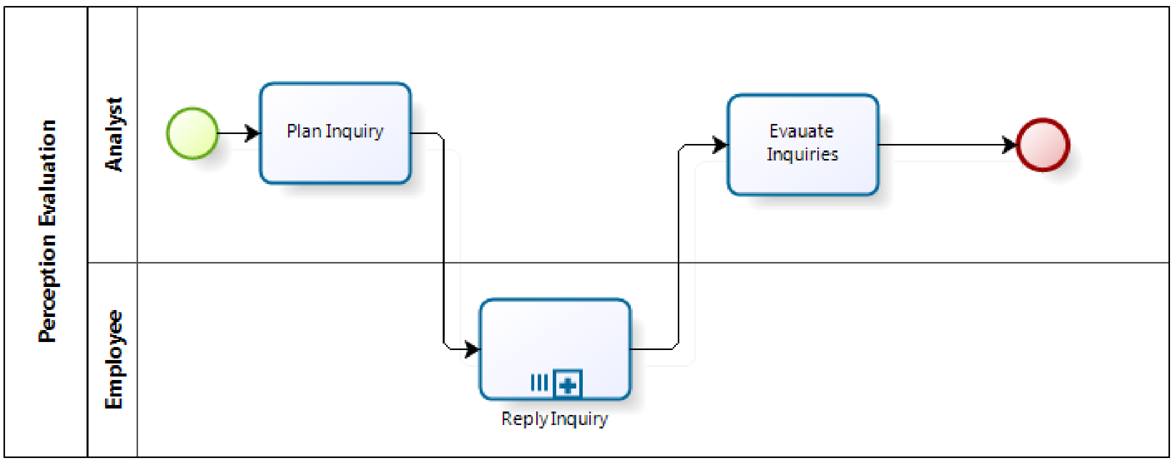
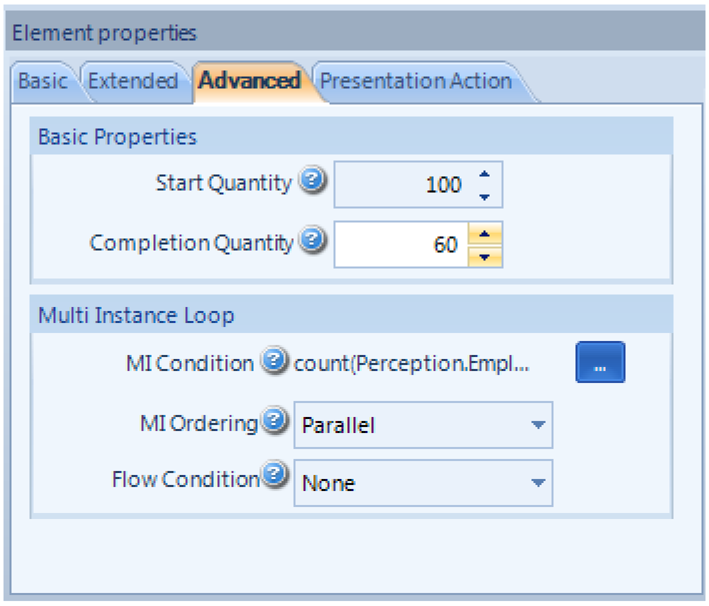


Diagram 23. Cancelling partial join for multiple instances example

In order to configure the sub process behavior for following the required conditions, the following properties have to be defined:



Start Quantity 100

Completion Quantity 60

MI Condition count(Perception.Empl...

MI Ordering Parallel

Flow Condition None

Start Quantity =100 : It means the sub process will be instantiated 100 times.

Completion Quantity= 60: The sub process is considered as completed when 60 instances have been accomplished.

MI Condition = count(Perception.Employees): It means there will be as many instances as employees recorded in an established master entity to reply to the inquiry.

MI Ordering = Parallel: The sub process instances will be executed in parallel.

Flow Condition = None: The flow will continue to the next activity each time an instance is completed.

WCP 36 - Dynamic Partial Join for Multiple Instances

Description

This pattern allows an activity to be instantiated multiple times in a process. The number of instances is not known during the design or run time. Activities are run concurrently but, whilst they are running, new ones can be created. A completion condition is specified and it is evaluated each time an instance of the task completes.

Once the completion condition evaluates to true, the preceding task in the process is triggered regardless of the state of the remaining task instances. Subsequent completions of the remaining activity instances are inconsequential and no new instances can be created. [1]

Example

A company decides to evaluate the perception of their employees about the job environment. An inquiry is sent to specific people. New inquiry requests can be sent before all the active instances finish. The analyst waits for at least 60 replies before starting the evaluation. The remaining 40 inquiries can be received.

Implementation

To model this pattern it is necessary to use sub processes in order to create multiple instances of an activity. In this case, the *Reply inquiry* sub process is instantiated as many times as necessary. A complex gateway is used to control that at least 60 inquiries have been received in order to start the evaluation.

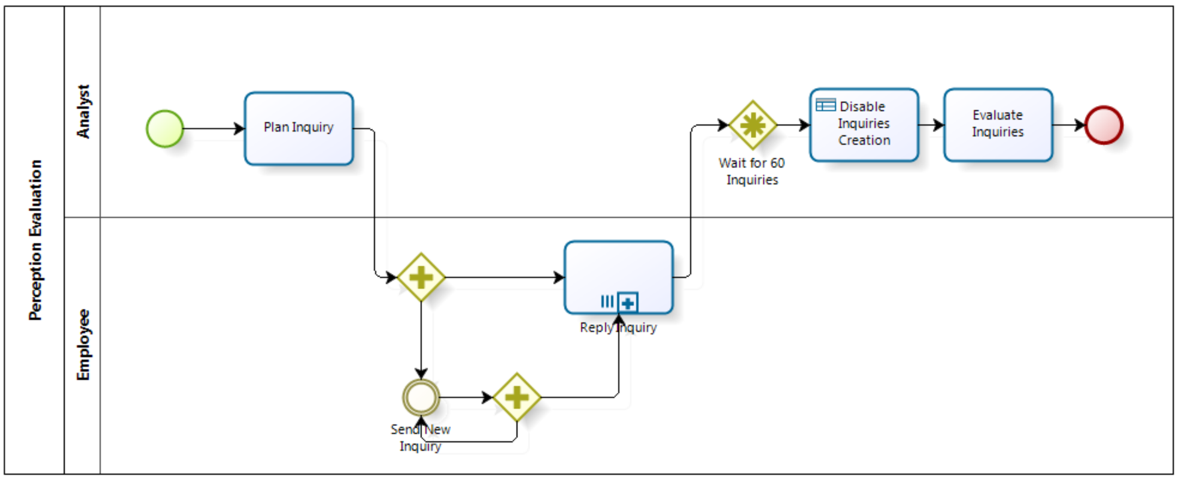
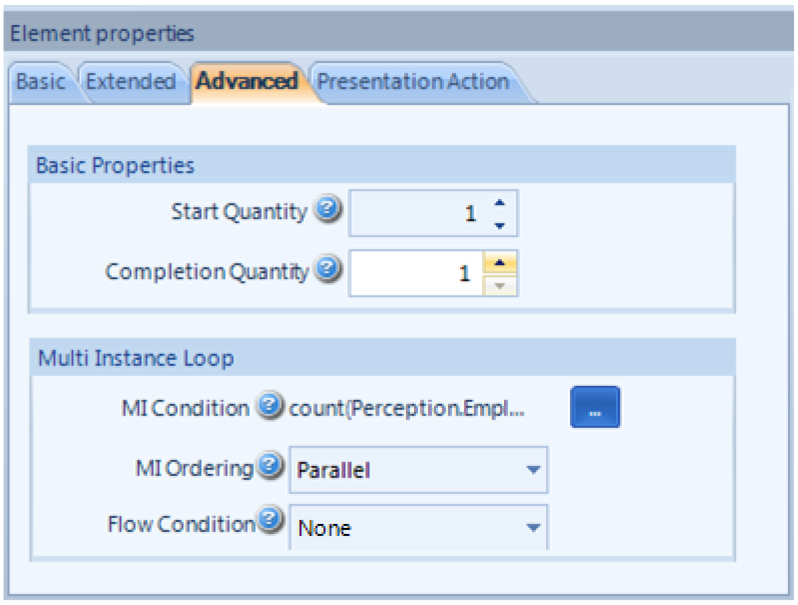


Diagram 24. Dynamic Partial Join for Multiple Instances pattern example

In order to configure the sub process behavior for following the required conditions, the following properties have to be defined:



Start Quantity: 1
Completion Quantity: 1
MI Condition: count(Perception.Empl...
MI Ordering: Parallel
Flow Condition: None

MI Condition = count(Perception.Employee): It means there will be as many instances as employees recorded in an established master entity to reply the inquiry.

MI Ordering = Parallel: The sub process instances will be executed in parallel.

Flow Condition = None: The flow will continue to the next activity each time an instance is completed.

State-based Patterns

The group of state-based patterns reflects situations for which solutions are most easily accomplished by the notion of states

WCP 16 - Deferred Choice

Description

The Deferred Choice pattern describes a point in the process where one of several branches is chosen. The decision is taken based on interaction with the process environment instead of process data. When one branch of the process is enabled, the others should be disabled.

Example

When a client makes a credit request, it is necessary to ask for different documents. If the client does not bring the documents after five days it is necessary to contact him and decide if he continues with the process. If the client does not continue with the process, it is not necessary to wait for the documents and the process must end. If the client brings the documents, it is not necessary to contact him.

Implementation

To implement this pattern it is necessary to use an event based gateway. This gateway represents a point in the process where only one of many paths of the process can be selected based on an event. Remaining paths will be disabled.

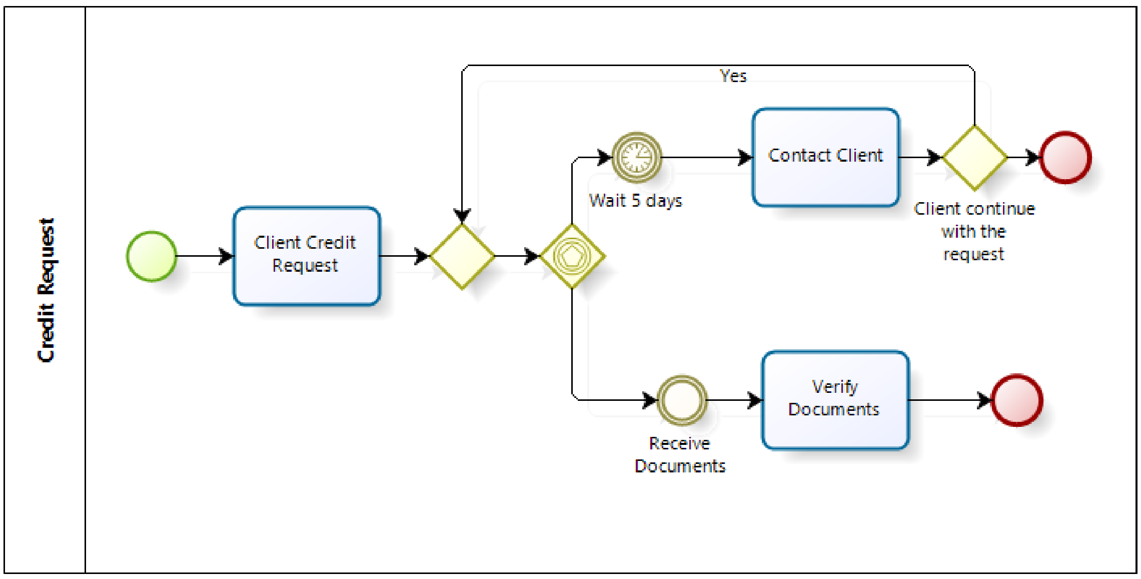


Diagram 25. Deferred Choice pattern example

WCP 17 - Interleaved Parallel Routing

Description

A set of activities has a partial ordering defining the requirements with respect to the order in which they must be executed. Each activity in the set must be executed once and they can be completed in any order according to the partial order.

However, as an additional requirement, no two activities can be executed at the same time. [1]

Example

The Human Resources Manager established that an applicant must face three different tests in order to be able to be contracted: a psychological test, an intelligence test and a round of introduction.

The order in which the three tests are performed is relevant. The intelligence test must be checked before performing the psychological test. The round of introduction can be accomplished at any moment. However two tests cannot be performed concurrently.

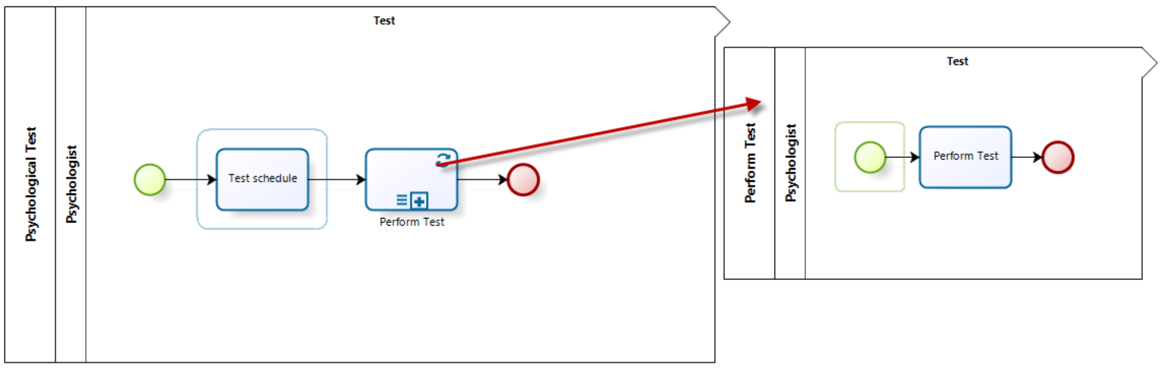
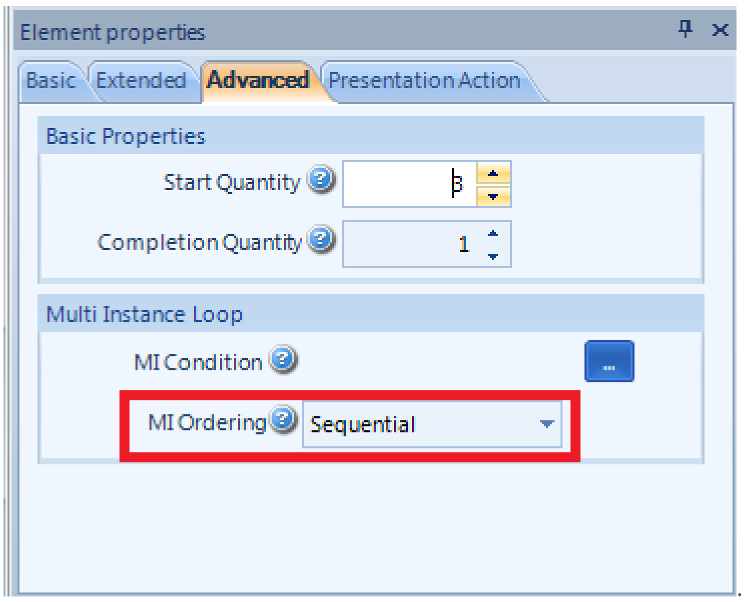


Diagram 26. Interleaved Parallel Routing pattern example

Implementation

This pattern is modeled by using a multiple sub process. In this case the sub process depicts the tests that must be performed so it must be configured as a sequential subprocess in the *Element Properties menu*


Element properties

Basic | Extended | Advanced | Presentation Action

Basic Properties

Start Quantity: 3

Completion Quantity: 1

Multi Instance Loop

MI Condition

MI Ordering: Sequential

The sequence in which the test will be performed is defined in the first activity of the process.

WCP 18 - Milestone

Description

It defines that an activity is only enabled when the process instance is in specific state (milestone). If the process instance has progressed beyond this state, then the task can no longer be enabled.

Example

In a travel agency, flights, car rental, and hotels may be booked as long as the invoice is not printed.

Implementation

This pattern uses an event based gateway to control that the change of the booking only is possible if the Print Invoice activity has not been performed.

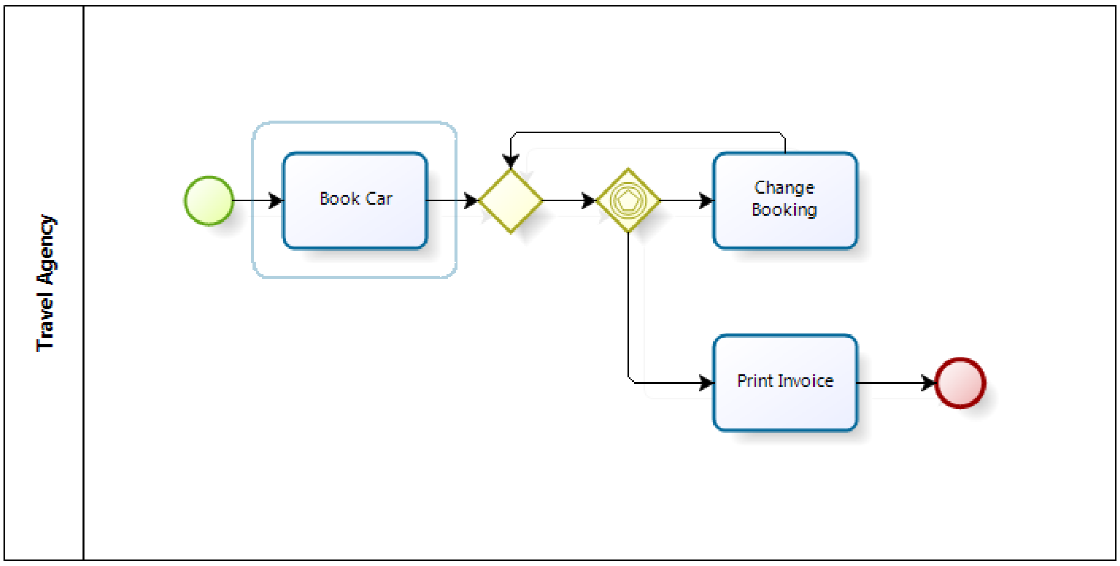


Diagram 27. Milestone pattern example

WCP 39 - Critical Section

Description

The Critical Section pattern describes that two or more connected sub-processes are identified as critical sections. When one of these critical sections is active - which means that an activity inside this section is enabled - no other critical section can be activated. The business process waits for completion or stopping one critical section to be able to activate another critical section.

Example

Two administrators have to access a server for changing some settings. As long as one admin is currently on the server there is no chance for the other administrator to access the server too. He or she has to wait until the other administrator has left the server.[3]

Implementation

This pattern is modeled by using an E*vent Based Gateway* to control the non-simultaneous execution of critical activities. In this case the server updating is carried out by one administrator, the other one is not able to do it. Once the server is updated an *Exclusive Gateway* evaluates if more changes are necessary to enable the server updating again or finish the process.

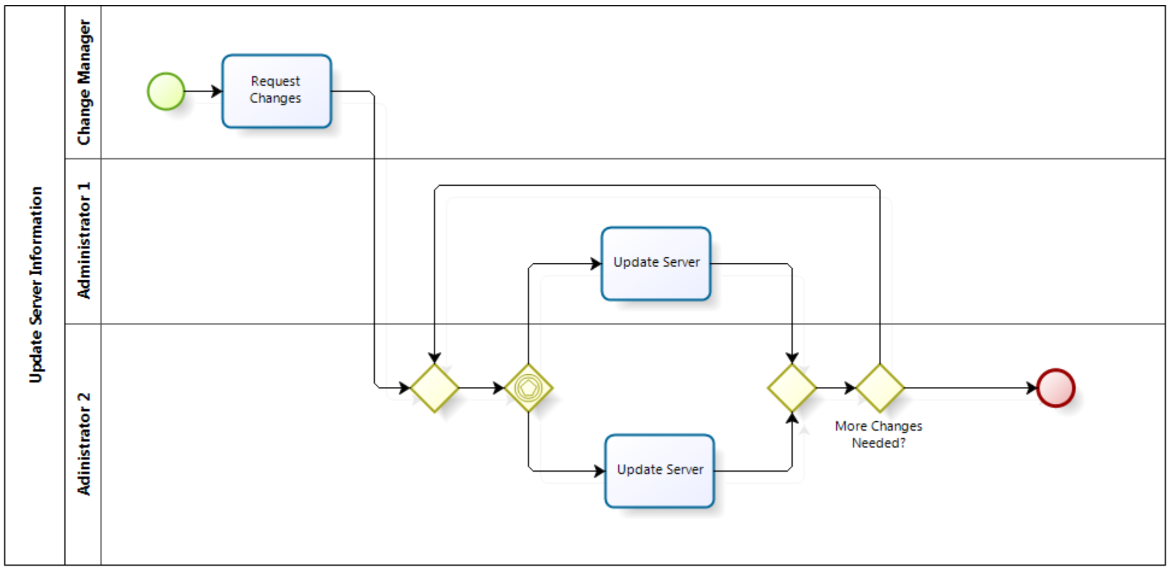


Diagram 28. Critical Section pattern example

WCP 40 Interleaved Routing

Description

The Interleaved Routing pattern describes that each member of a set of tasks must be executed once. The tasks can be performed in any order. However, it is not possible to perform two tasks simultaneously.

Example

A student must face three different tests in order to be admitted to a University: a psychological test, an admission test and a general knowledge test.

The order in which the three tests are performed is not relevant. However two tests cannot be performed concurrently.

Implementation

This pattern is modeled by using an *Inclusive Gateway* to control the activity that must be performed each time. In this case the first activity evaluates which activity to perform, the *inclusive gateway* enables that activity and the *exclusive gateway* controls that all activities have been performed before allow the process to continue its normal flow.

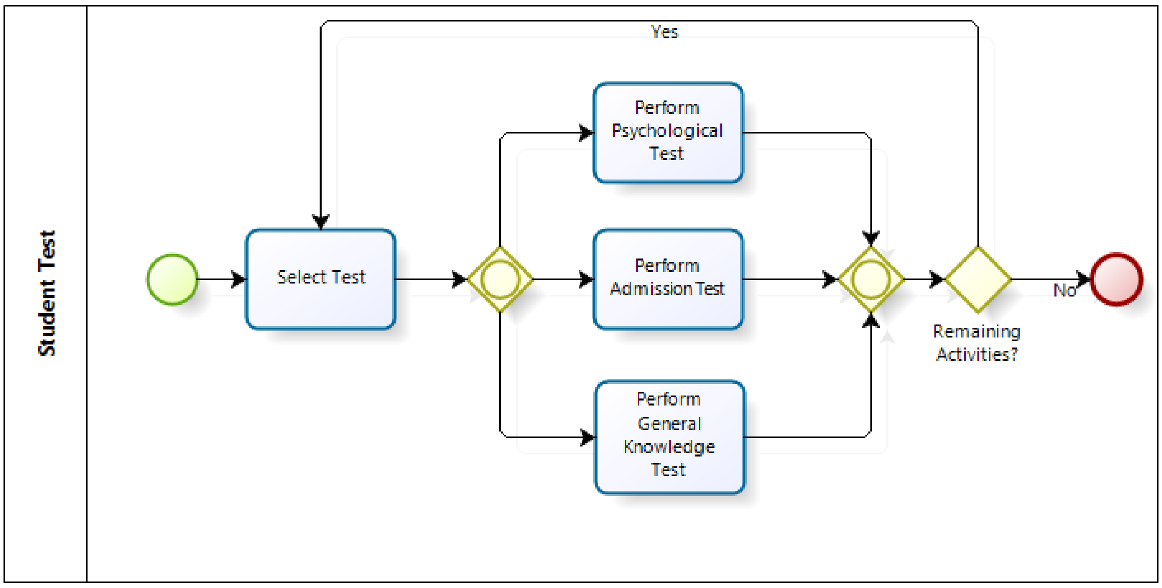


Diagram 29. Interleaved Routing pattern example

Cancellation and Force Completion Patterns

The pattern in this group applies the concept of cancelling or withdrawing activities.

WCP 19 - Cancel Task

Description

The Cancel Task pattern describes the possibility of cancelling or withdrawing an enabled task [2].

Example

In a travel request it is necessary to book several things such as a hotel, car and flight, as well as provide an advance to the employee. If necessary, the employee must be able to cancel the travel before the bookings are completed.

Implementation

These patterns can be modeled using an Event Based Gateway and an Intermediate Event. Event Based Gateways enable the activity (or thread) to manage bookings and advances, and an event in order to be able to cancel the request.

Only the activity or the Intermediate event can be performed (not both). If the event takes place, the activity is disabled and the process finishes, if not, the process continues its normal flow.

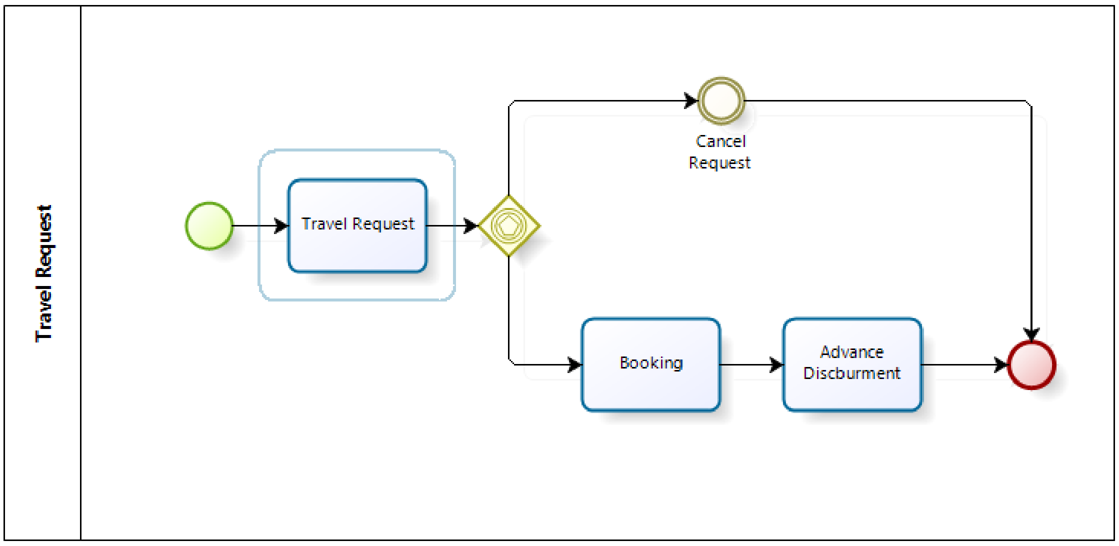


Diagram 30. Cancel Task pattern example

WCP 20 - Cancel Case

Description

The Cancel Case pattern describes the removal of a complete process instance. This includes currently running tasks as well as those which may be executed at some time in the future [2].

Example

When an incident occurs, it must be reported to the Help Desk of the company. After reporting the incident, the Help Desk should be able to resolve the incident or close the case at any moment.

Implementation

To implement this pattern you can use a parallel gateway and a Terminate End Event. Once the process reaches the terminate end event, any activity that is active is canceled and the case ends.

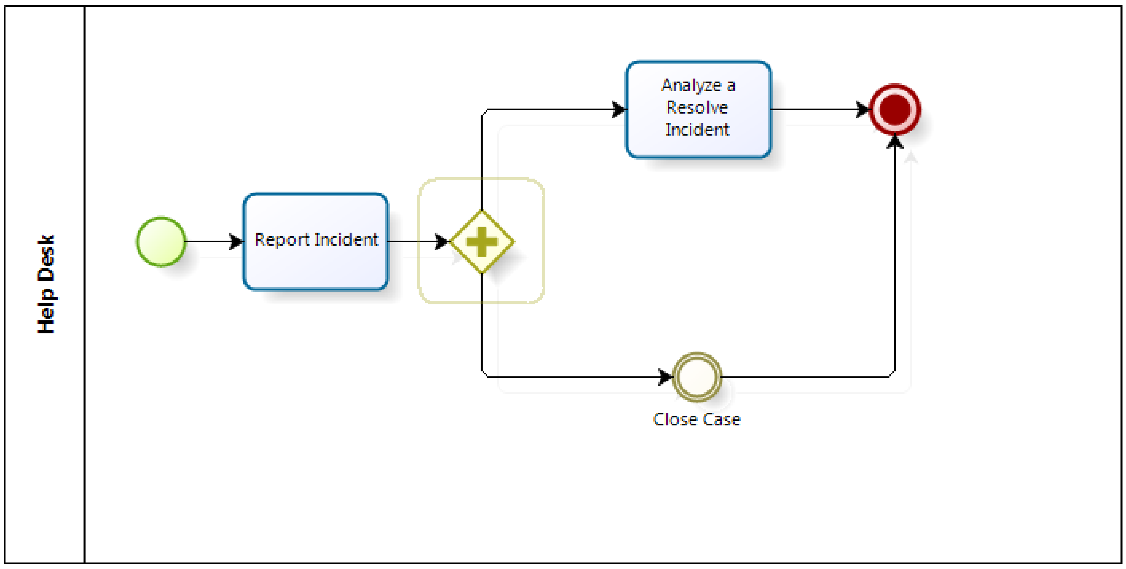


Diagram 31. Cancel Case pattern example

WCP 25 - Cancel Region

Description

The cancel region pattern describes the ability to disable a set of activities in a process instance. If any of the activities are in execution they will be cancelled.

Example

When a travel request is made, it is necessary to make reservations. If they are not successfully completed it is not possible to make the trip.

Implementation

Bizagi supports this pattern implementation through a transactional sub process. The transactional sub process allows you to interrupt the activities without saving information about the task. Then the process continues to the intermediate error event.

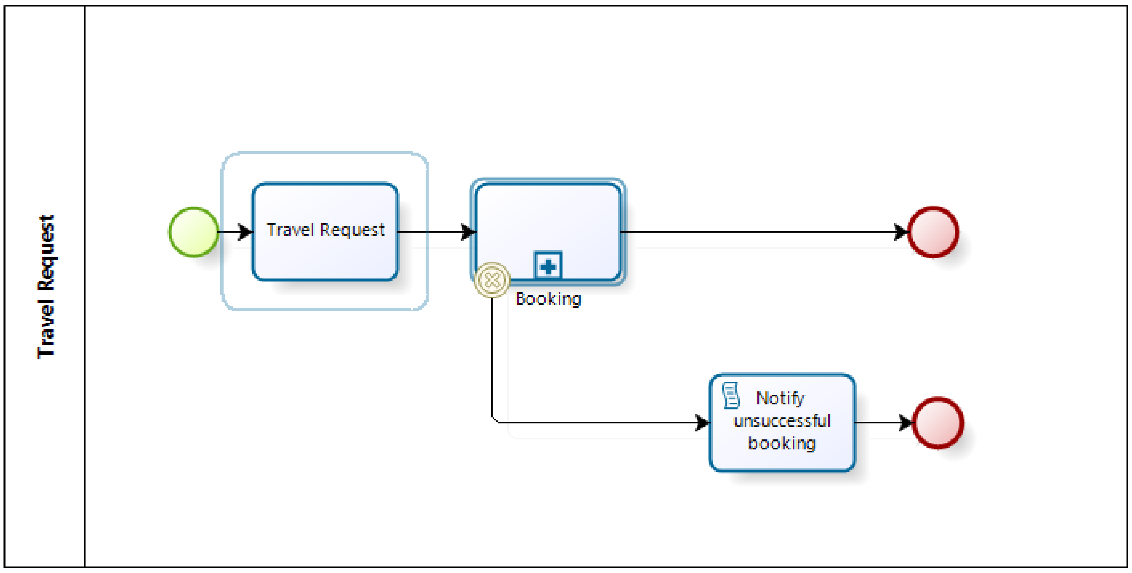


Diagram 32. Cancel Region pattern example

WCP 26 - Cancel Multiple Instance Activity

Description

The Cancel Multiple Instance Task pattern describes the ability of cancelling a whole Multiple Instance task by withdrawing all instances which have not yet been completed. All others - already completed - are not affected [2].

Example

A project manager creates and assigns deliverables for the execution of a project. Each deliverable needs to be performed sequentially. The project manager can cancel the execution of the unfinished deliverables at any time, if necessary.

Implementation

This pattern can be modeled by using interrupting events. This kind of event is attached to the boundaries of an activity or sub process and will interrupt it when triggered. In this case when a special condition is fulfilled (related to the cancelation of uncompleted deliverables) the pending instances for the *Deliverables Sub process* will be cancelled and the process will continue its normal flow.

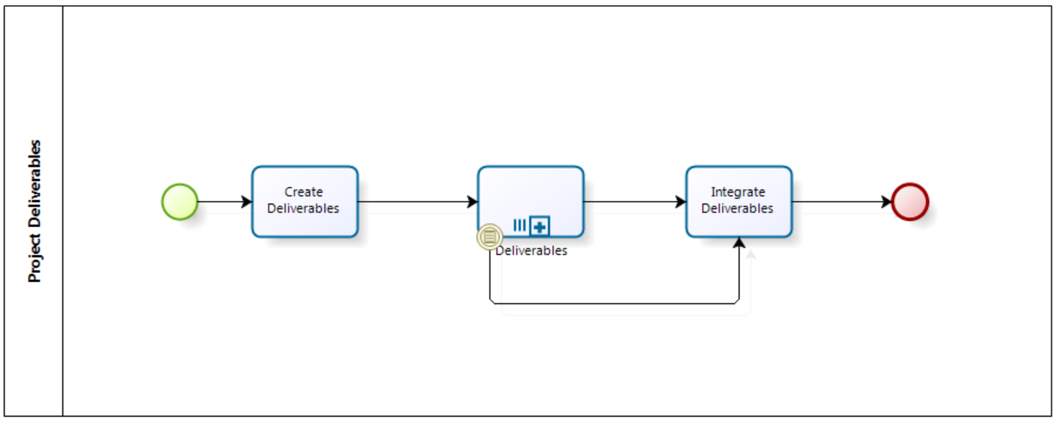


Diagram 33. Cancel Multiple Instance Activity pattern example

WCP 27 - Complete Multiple Instance Activity

Description

The Cancel Multiple Instance Task pattern describes the ability of completing a whole Multiple Instance task by withdrawing all instances which have not yet been completed.

Implementation

Bizagi does not support this pattern.

Iteration Patterns

WCP 26 - Arbitrary Cycles

Description

This pattern models a point in a workflow process where one or more activities can be done repeatedly.

Example

The deliverables of a project need approvals from several people. The number of approvals needed is a number defined by the project manager.

Implementation

This pattern is modeled using an exclusive gateway that is controlled by a counter, each time that the branch (in this case the activity) is executed the counter is increase by one. Once the number of executions needed is complete, the process continues.

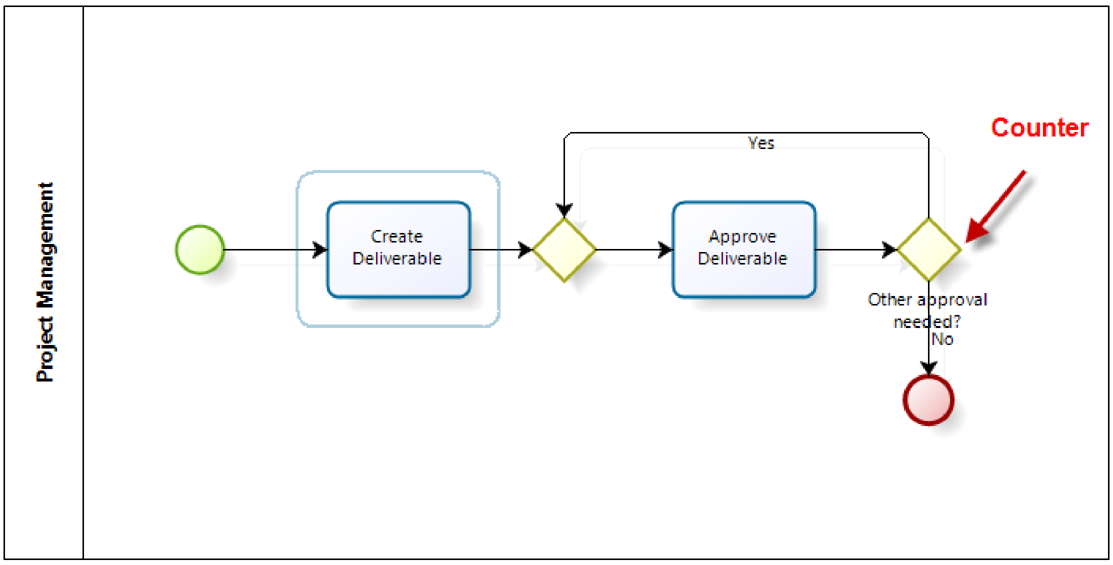


Diagram 35. Arbitrary Cycles pattern example

WCP 10 - Structured Loop

Description

This pattern describes the ability to execute an activity or sub-process repeatedly. The loop has either a pre-test or post-test condition associated with it. The condition is either evaluated at the beginning or end of the loop to determine whether it should continue. The looping structure has a single entry and exit point [2].

Example

In the Account Payable process it is necessary to receive and approve invoices. If the invoice is not approved , the supplier must correct it and send it again, until the invoice meets all requirements.

Implementation

To implement this pattern it is necessary to include an exclusive gateway. This gateway validates a condition; if the condition is not fulfilled the process returns to the activity where the condition is set, in other cases, the process continues its normal flow.

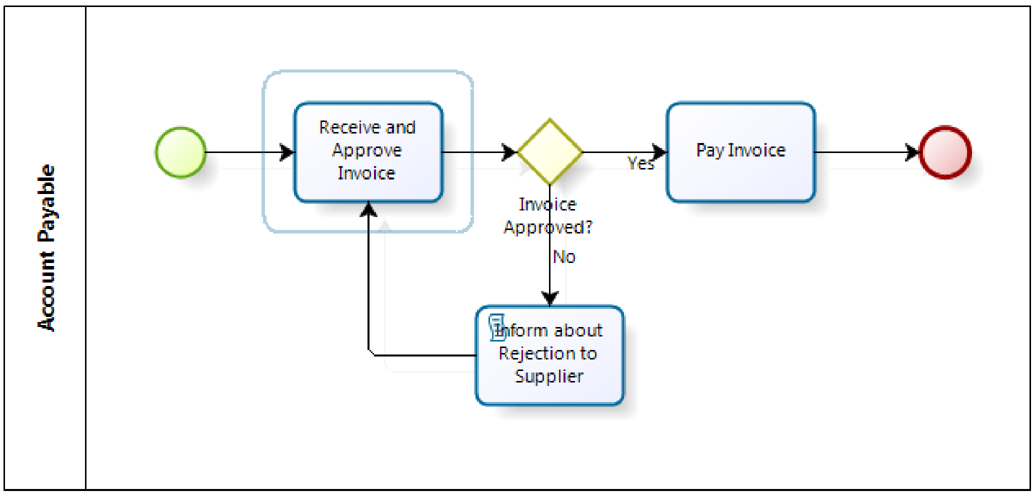

Account Payable
Receive and Approve Invoice
Invoice Approved?
Yes
No
Pay Invoice
Inform about Rejection to Supplier

Diagram 36. Structured Loop pattern example

WCP 22 - Recursion

Description

The Recursion pattern describes the ability of a task to invoke itself in terms of the overall decomposition structure with which it is associated.

Implementation

Bizagi does not support this pattern.

Termination Patterns

Termination patterns are used to determine when a process instance finishes according to desired business conditions.

WCP 11 Implicit Termination

Description

This pattern is used to determine when a process instance is considered as complete. A given process (or sub-process) instance should terminate when there are no remaining work items that are able to be done either now or at any time in the future. [1]

Example

Consider the Reverse Logistics in a manufacturing company. When a refund is received, two activities must be performed after which the whole process can be finished. The first activity is to generate a replacement order and the second is to document the causes for the refund.

Implementation

The implicit termination pattern for the exposed example is represented as follow:

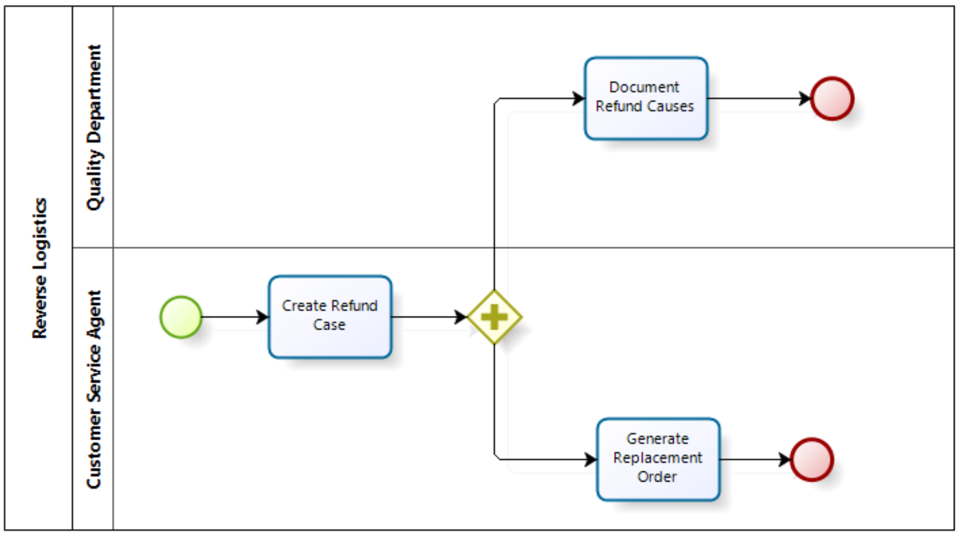


Diagram 37. Implicit Termination pattern example

Note the pattern is represented by adding *End Events* at the end of the paths that must be completed. The process will be considered as finished when each *End Event* is reached.

WCP 43 - Explicit Termination

Description

A given process (or sub-process) instance should terminate when it reaches a nominated state. Typically this is denoted by a specific end node. When this end node is reached, any remaining work in the process instance is cancelled and the overall process instance is recorded as having completed successfully. [1]

Example

Suppose a Credit Request process has a sub process called Documents Request. In this sub process the credit analyst establishes which documents must be summited by the customer in order to complete the credit request, then, an email is sent to him to request those documents. If after 3 days the customer has not sent the requested documents, the credit request is considered as cancelled and the sub process is finished.

Implementation

The explicit termination pattern for the exposed example is represented as follow:

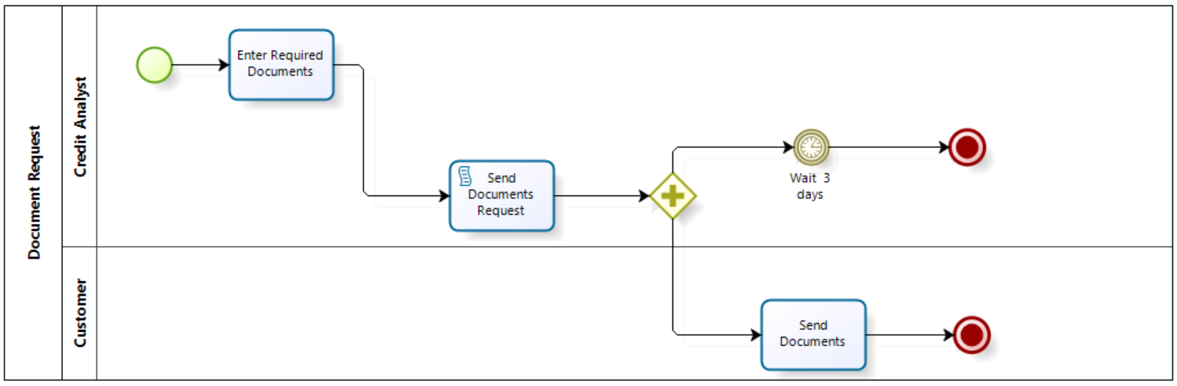


Diagram 38. Explicit termination pattern example

Note the pattern is represented by adding *Terminate Events* at the end of the paths that must be completed. The process will be considered as finished when one of the *Terminate Events* is reached.

Trigger Patterns

Triggers Patterns allow modeling processes that must interact between each other through message flows.

WCP 23 - Transient Trigger

Description

This pattern allows an activity to be triggered by a signal or message from another part of the process or from the external environment. These triggers are transient in nature and are lost if not acted upon immediately by the receiving activity. [1]

Example

In the purchase request process, the purchase department receives a requirement. The best supplier is selected and a purchase order is generated. The purchase process will not continue to the payment until all the ordered products have been received and inspected.

Implementation

This pattern can be represented by using Message Events or Signals in the process flow.

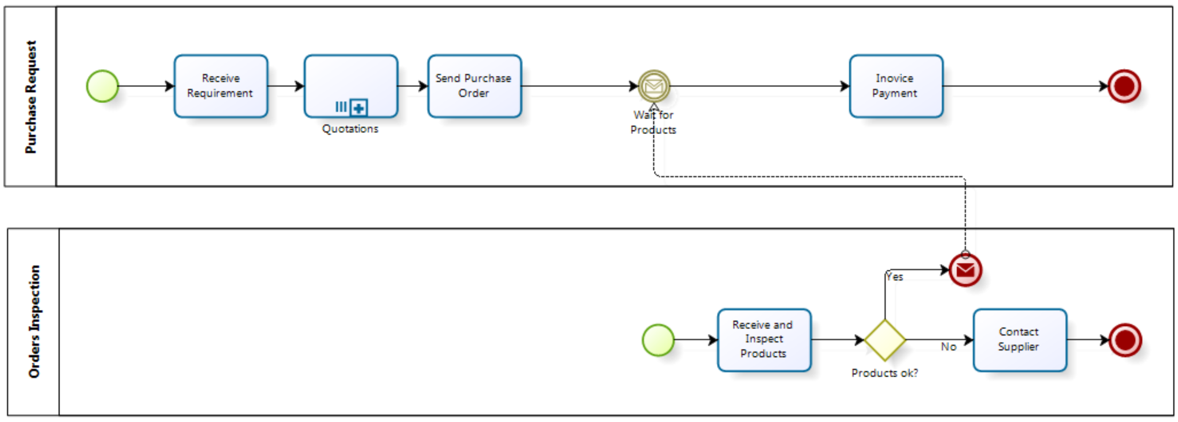


Diagram 39.Transient Trigger pattern example

In this case the business process is then detained until the message is received. For more information about how to configure message events please refer to http://wiki.bizagi.com/en/index.php?title=Collaboration

WCP 24 - Persistent Trigger

Description

This pattern allows an activity to be triggered by a signal from another part of the process or from the external environment. These triggers are persistent in form and are retained by the workflow until they can be acted upon by the receiving activity. [1]

Example

An upholstery company produces sport seats for luxury cars. Once a specific set of seats is produced according to the customer preferences, a signal is sent to the main car assembly line in order to allow the line to continue its normal flow.

Implementation

This pattern is used in collaborative models and retains a message or signal sent by a process until another process can catch it. In this case the upholstery company finishes the seats even when the car is just in the start of the assembly line, the message is sent once but it is not lost if the assembly line is not yet ready to install the seats.

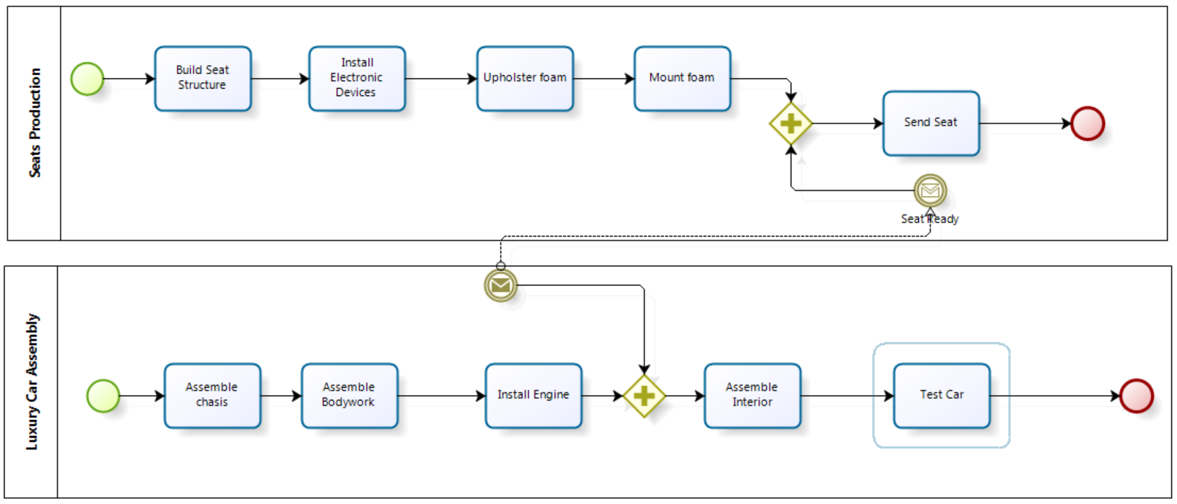


Diagram 39. Persistent Trigger pattern example

References

[1] Nick Russell, Arthur H.M. Ter Hofstede, "Workflow Control-Flow Patterns, A Revised View".BPM Group, Queensland University Of Technology, Australia. 2006.

[2] W.M.P. van der Aalst, A.H.M. ter Hofstede, B. Kiepuszewski and A.P. Barros, "Workflow Patterns", Department of Technology Management, Eindhoven University of Technology, Australia, 2002.

[3] Marcus Goetz, "Modeling Workflow Patterns through a Control-flow perspective using BPMN and the BPM Modeler Bizagi", Institute of Applied Informatics and Formal Description Methods, University Karlsruhe.