



FUEGO

The background features a light gray BPM diagram with various nodes (circles, some with internal symbols) and arrows indicating process flow. A thick horizontal line is positioned below the 'FUEGO' text.

# **BPM Process Patterns**

**Repeatable Designs  
for BPM Process  
Models**

FUEGOBPM

# BPM Process Patterns

---

PART NO. BPMPProcessPatternsWhitePaper.doc

Date January 17, 2006

This document is subject to change without notice. This document and the software described in this document contains proprietary trade secrets and confidential information of Fuego, Inc. and is also protected by U.S. and other copyright laws and applicable international treaties. Use of this document and the software is subject to the license agreement between you and Fuego, Inc. If no such license agreement exists, you may not use this document and software in any manner whatsoever. Unauthorized use of the document or software, or any portion of it, will result in civil liability and/or criminal penalties.

Fuego, FuegoBPM Studio, FuegoBPM Designer are trademarks or registered trademarks of Fuego, Inc.

All other trademarks, trade names, and service marks are owned by their respective companies.

## Table of Contents

Table of Figures .....	2
Introduction .....	3
A Practical Approach to Process Design .....	3
Process Design Patterns .....	4
Basic Control Patterns .....	5
Sequence Pattern .....	5
Exclusive Choice Pattern .....	5
Simple Merge Pattern .....	7
Parallel Split and Synchronization Patterns .....	8
Advanced Branching and Synchronization Patterns .....	10
Multiple Choice and Synchronizing Merge Patterns .....	10
Discriminator and N-out-of-M Join patterns .....	12
Multiple Merge Pattern .....	12
Structural Patterns .....	13
Arbitrary Cycles Pattern .....	13
Collaboration Pattern .....	14
Implicit Termination Pattern .....	14
Multiple Instance Patterns .....	15
Multiple Instances without Synchronization Pattern .....	15
Multiple Instances with Design and/or Runtime Knowledge Patterns .....	16
State Based Patterns .....	17
Deferred Choice Pattern .....	17
Milestone Pattern .....	18
Cancellation Patterns .....	18
Cancel Activity Pattern .....	18
Cancel Case Pattern .....	19
Summary .....	20
References .....	21
Appendix A – BPM Process Patterns .....	22

## Table of Figures

Figure 1: Process Pattern Learning Curve .....	4
Figure 2: Highway Patterns .....	4
Figure 3: Sequence Pattern .....	5
Figure 4: Exclusive Choice Pattern.....	6
Figure 5: Exclusive Choice Pattern - Handling the Unexpected .....	7
Figure 6: Simple Merge Pattern.....	8
Figure 7: Sequence Pattern Elapsed Time .....	9
Figure 8: Should-Be Parallel Split and Synchronization Pattern Elapsed Time.....	9
Figure 9: Sequence Pattern inside a Parallel Split Pattern.....	10
Figure 10: Conditional Transition in a Parallel Split Pattern.....	11
Figure 11: Synchronizing Merge Pattern.....	11
Figure 12: Discriminator and N-out-of-M Join Pattern .....	12
Figure 13: Merge Activity before Synchronization .....	13
Figure 14: Upstream Transition .....	13
Figure 15: Collaboration Pattern.....	14
Figure 16: End Activity Used as the Merge Point .....	15
Figure 17: Implicit Termination Pattern .....	15
Figure 18: Asynchronously Handling Batch Processing .....	16
Figure 19: Synchronously Handling Batch Processing .....	17
Figure 20: Notification Wait.....	17
Figure 21: Milestone Pattern.....	18
Figure 22: Cancel Activity Pattern .....	19
Figure 23: Cancel Case Pattern.....	20

# BPM Process Patterns: Repeatable Designs for BPM Process Models

## Introduction

Rigorous and skillful business process modeling is a commonly overlooked but key aspect to successfully developing BPM solutions. Before BPM, processes sometimes served as specifications that IT sometimes used to develop software solutions.

Business processes serve a much more critical role in BPM. Today they provide:

- **Solutions** - BPM Processes now serve as *both* the specification and the source code. The processes modeled become the actual solutions deployed. As changes are made to the processes, the business changes are automatically kept in synch.
- **Common Understanding** – Processes provide a simple communication tool between the end-users, business analysts, developers and management. The different constituencies now take an active role and develop buy-in and ownership of the solution.

Business analysts and developers new to BPM sometimes struggle with the art of discovering, modeling, understanding and explaining business processes. BPM Process patterns provide a time proven and simple technique to shorten the learning curve and improve productivity and quality of the processes designed.

## A Practical Approach to Process Design

One of the problems process patterns have is that they are confused with other software patterns. Object-Oriented (OO) software development benefited greatly in 1995 from the 23 patterns defined in the seminal book by Gamma, et al. *Design Patterns: Elements of Reusable Object-Oriented Software*. Each of the OO patterns they defined came with a clear explanation of the pattern's intent, motivation and known uses. OO software developers now could stand on the shoulders of giants.

Since then, software developers and architects have benefited from other types of patterns including software messaging, graphical user interface (GUI) design and application architecture. This rich history is both good and bad news for the adoption of process design patterns. As with most good ideas, the bad news is that over the last 10 years, patterns have at times been over hyped as a cure-all. Fear crept in because OO, software messaging and application architecture patterns are esoteric and intimidating to those of us not already well versed in these fields.

The good news is that 10 years experience has proven that there can be context specific practical pattern solutions to recurring real world design problems. Unlike other technologies - process design patterns are relatively simple to understand, learn and apply immediately. Most business analysts experience the learning curve shown in Figure 1 when using patterns to model business processes.

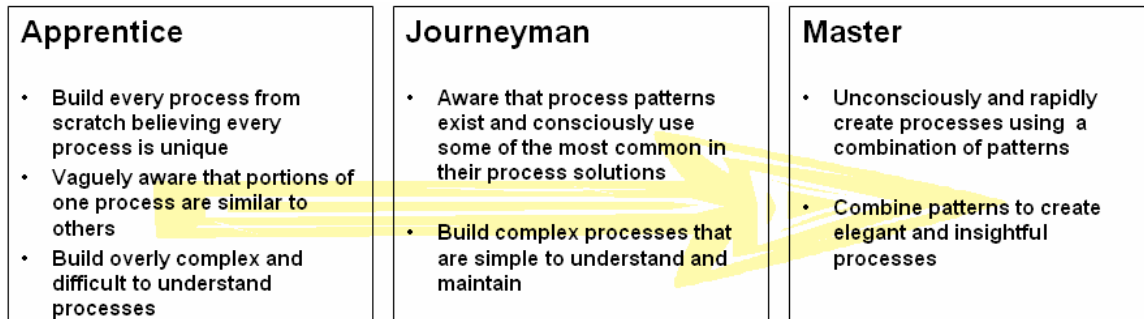


Figure 1: Process Pattern Learning Curve

## Process Design Patterns

Process patterns are examples that show how to connect activities together to solve a common problem. Processes are like highways. As we learned to drive we eventually became accustomed to similar and time proven highway on/off-ramps, signs, bridges and curves. Every country, state and city ensures that their engineers follow known and proven specifications. Because highway construction is fairly consistent from city to city, highways are constructed quicker for less cost. Traffic accidents are reduced because drivers are rarely surprised by non-standard lane widths, traffic dividers, turning radiuses and on-ramps. Process design patterns are the specifications for the on/off ramps, bridges and highways of processes.



Figure 2: Highway Patterns

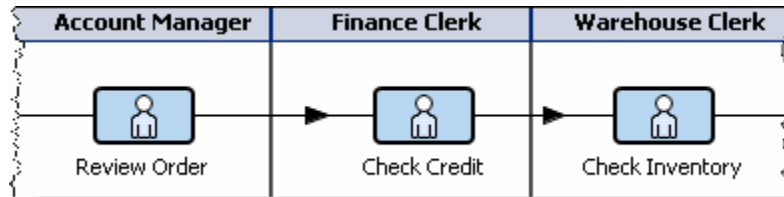
Similar to the approach taken by the OO world, process design pattern identification and documentation was well served by the article *Workflow Patterns* by van der Aalst, et al. (<http://www.workflowpatterns.com>). Gradually building in complexity, process patterns were broken down into the six categories:

- Basic Control,
- Advanced Branching,
- Structural,
- Multiple Instances,
- State Based and
- Cancellation.

## Basic Control Patterns

### Sequence Pattern

This is the most common and obvious of all the patterns. When business analysts begin to model the way things work today (“As-Is” process) usually much of the process looks like activities strung together in a series. Instances (individual items of work flowing through the process) step through the activities one by one.

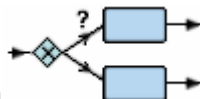


**Figure 3: Sequence Pattern**

In the example in Figure 3, individuals in the Account Manager role review the orders going through the process. Once reviewed, Finance Clerks then check the credit and then Warehouse Clerks check the inventory to ensure the items are in stock.

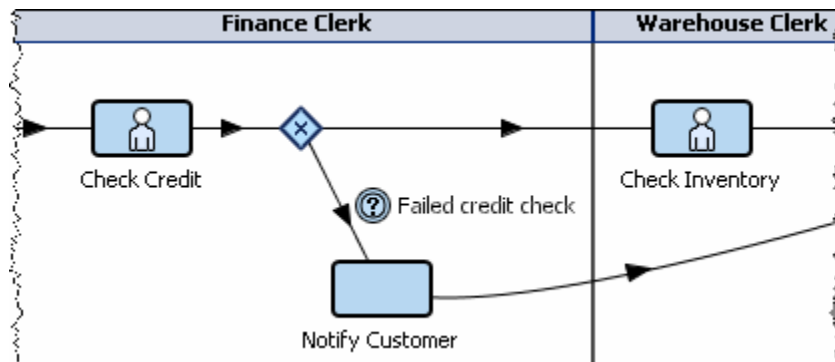
Although most processes contain at least one sequence pattern, a recommended practice during process walk-through sessions is to question the use of this pattern before it makes it into the BPM production (“Should Be”) process for three reasons.

1. If activities are done one after the other, the amount of time it takes to accomplish the three activities can be quite long. The activities may be able to be done simultaneously in parallel (discussed later in the Parallel Split pattern).
2. Each activity might have many other transitions (lines) coming out of them to handle different business rules. For example, during the process walk-through ask, “What happens if the customer’s credit check fails?”
3. The third question that should be asked when you see this pattern is “Do all account managers really need to see all the orders to be reviewed?” The reason this is an important topic of discussion during a process walk-through is because most BPM implementations use a shared queue as the default. All account managers would indeed see all the orders unless the specific account manager assigned to the order was designated upstream of the Review Order activity.




### Exclusive Choice Pattern


Every activity in the process should be examined to see if there might be more than one transition leaving it. As an instance leaves an activity, it normally exits through one and only one transition leaving the activity.



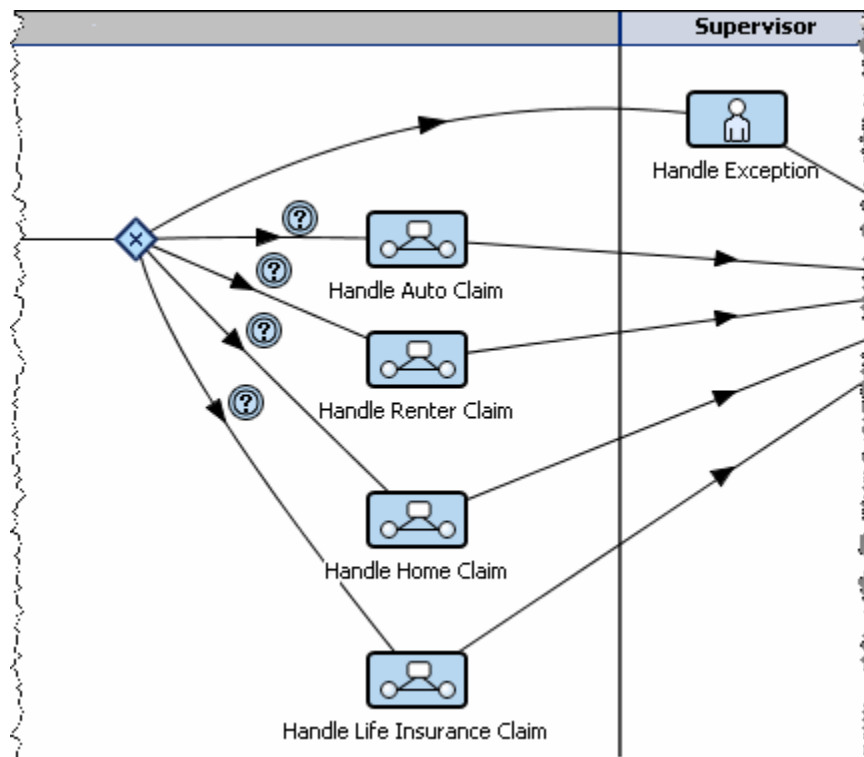
**Figure 4: Exclusive Choice Pattern**

As shown in Figure 4, it might be determined that there are two paths that could occur after the Check Credit activity (pass or fail). Depending on the credit score found in the Check Credit activity, the instance either flows to the Check Inventory (passed the credit check) *or* Notify Customer activity (failed the credit check).

The Business Process Modeling Notation (BPMN) standard shown here uses the  Conditional activity to represent the beginning of the Exclusive Choice pattern. As shown in Figure 5, there can be many transitions representing many different business rules leaving a single Conditional activity. In this insurance claim process, the Exclusive Choice pattern indicates a single insurance claim cannot be both an “Auto” *and* a “Renter” insurance claim. Instead it means that the claim can be either an “Auto” *or* “Renter” *or* “Home” *or* “Life” insurance claim. The instance leaving the Conditional activity goes through only one of the transitions.

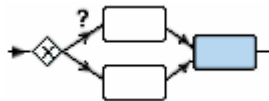
Note the “Handle Exception” activity in the Supervisor role. A recommended practice is to have a default “unconditional” transition leaving one of these activities. This is the transition taken if none of the business rules inside the activity’s  conditional transitions are true.






**Figure 5: Exclusive Choice Pattern - Handling the Unexpected**

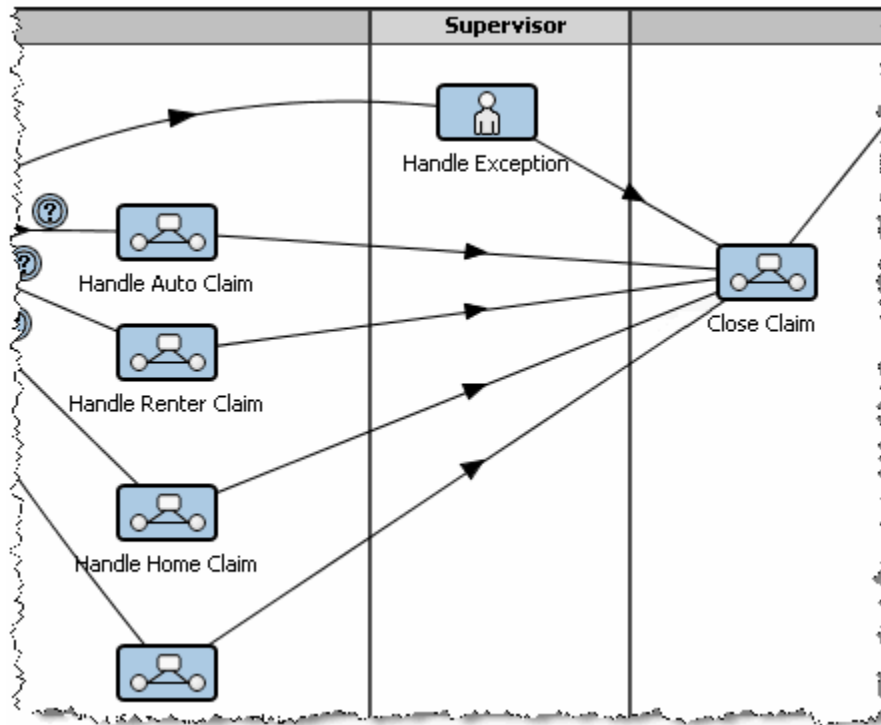
In the Exclusive Choice pattern shown in Figure 5, the transition to the “Handle Exception” activity is important. Even the most experienced business analysts cannot predict when new data might cause none of the conditional transitions to fire - resulting in a process bottleneck. The one unconditional transition leaving an activity keeps this from occurring. If none of the conditional transitions are taken, the instance will always flow through the one unconditional transition. If there are many conditional transitions leaving an activity, always use the one unconditional transition as an error handling activity.



### Simple Merge Pattern

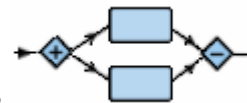
The power of patterns in a process becomes clearer when one pattern combines with others to form processes. Simple Merge is one of the many examples of this.

Anytime an Exclusive Choice pattern  occurs, somewhere downstream in the process there will almost always be a Simple Merge pattern. Simple Merge combines several transitions back into a single activity.




**Figure 6: Simple Merge Pattern**

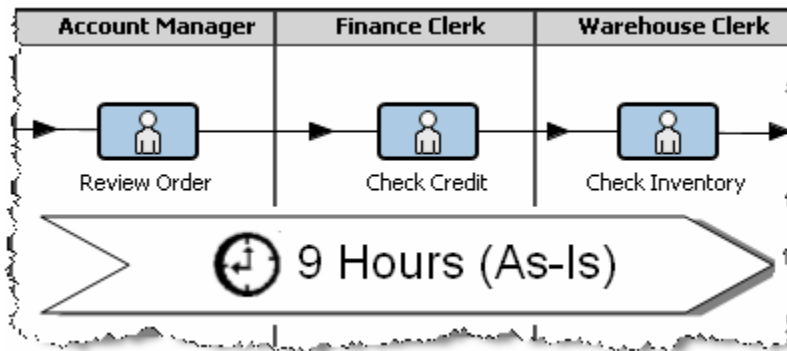
As shown in Figure 6, the “Close Claim” activity merges the transitions coming into it. The insurance claim instance coming into it came from only one of the activities upstream in the process.



## Parallel Split and Synchronization Patterns

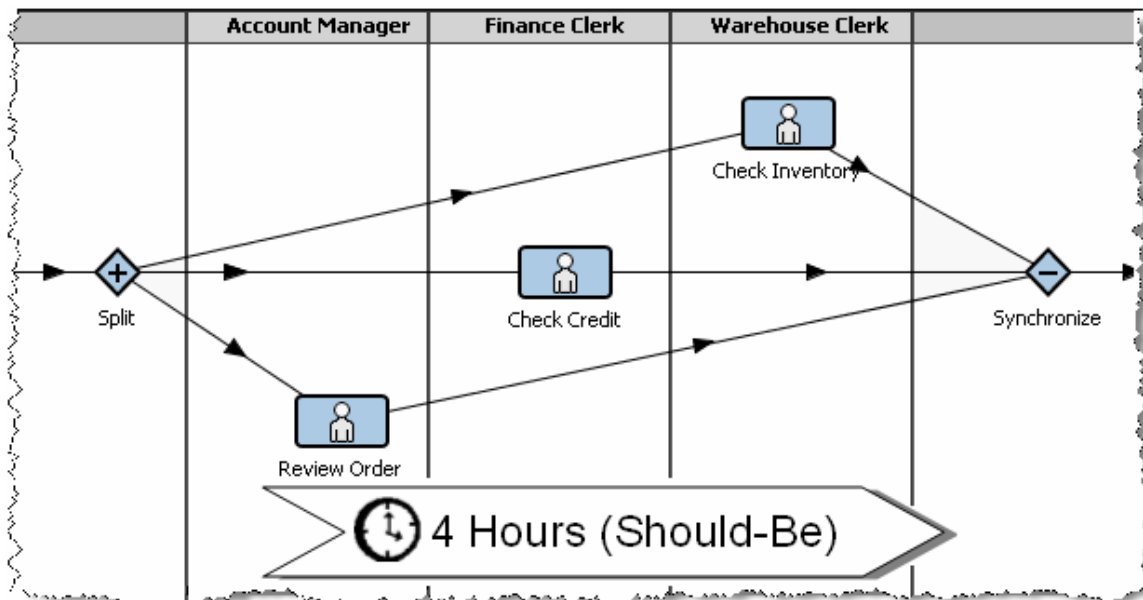
If parts of the current “As-Is” process look like a Sequence pattern , evaluate the process to see if it can be improved using a Parallel Split and Synchronization pattern.

For example, if the first, second and third activities in a sequence of activities shown in Figure 7 take 2, 3 and 4 hours respectively, the total time elapsed time to execute all three activities is 9 hours (the sum of the individual activity times).



### Figure 7: Sequence Pattern Elapsed Time

An improved “Should-Be” process might be able to use the Parallel Split and Synchronization pattern to speed up the process.



**Figure 8: Should-Be Parallel Split and Synchronization Pattern Elapsed Time**

In Figure 8 it was determined that simultaneously:

- an order can be reviewed (taking 2 hours), *while*
- the customer's credit is being checked (taking 3 hours), *while*
- the inventory for the items ordered is being checked (taking 4 hours).

Instead of taking 9 hours, the order now takes the 4 hours to complete the three activities (the longest of the three activities in the parallel paths).

The Parallel Split and Synchronization pattern speeds up the process by having the instance travel all the parallel paths through it simultaneously.

The order the activities execute is not important in this pattern. Using the example in Figure 8, an instance might execute the activities in the process in any one of these sequences:

- |                     |                     |                     |    |
|---------------------|---------------------|---------------------|----|
| (a) Check Inventory | (b) Check Credit    | (c) Review Order    | or |
| (a) Check Inventory | (b) Review Order    | (c) Check Credit    | or |
| (a) Check Credit    | (b) Review Order    | (c) Check Inventory | or |
| (a) Check Credit    | (b) Check Inventory | (c) Review Order    | or |
| (a) Review Order    | (b) Check Credit    | (c) Check Inventory | or |
| (a) Review Order    | (b) Check Inventory | (c) Check Credit.   |    |

van der Aalst, et al. refer to the execution of activities in any order as Interleaved Parallel Routing.

If one activity *must* be done before another, the two activities can still be placed sequentially in one of the parallel branches of this pattern.

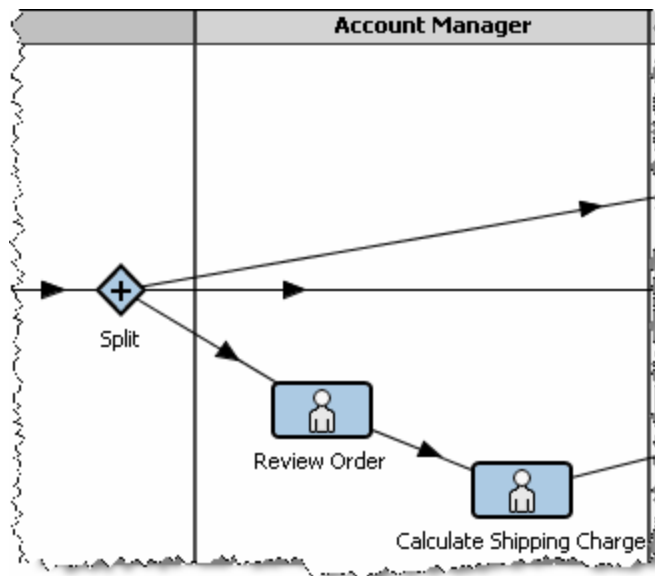
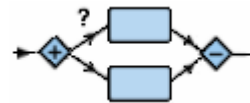


Figure 9: Sequence Pattern inside a Parallel Split Pattern

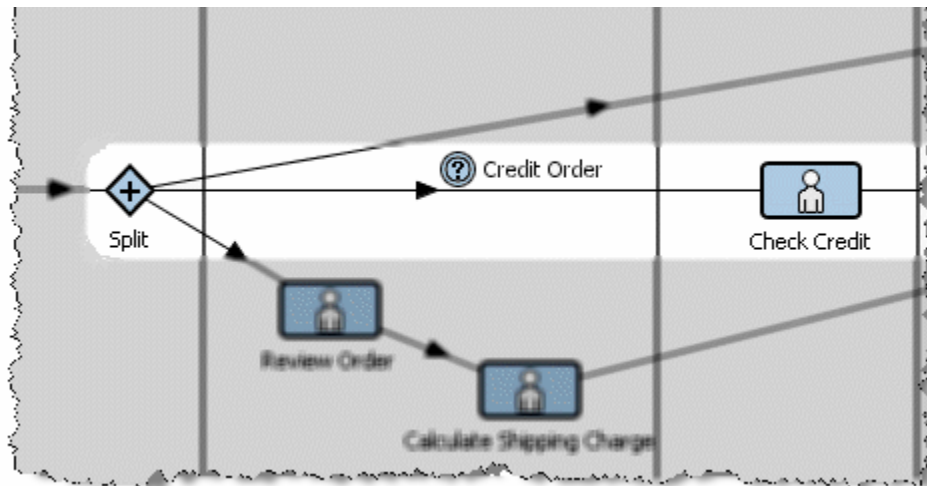
As shown in Figure 9, an order's shipping charge now can only be calculated after it has been reviewed.

## Advanced Branching and Synchronization Patterns

### Multiple Choice and Synchronizing Merge Patterns

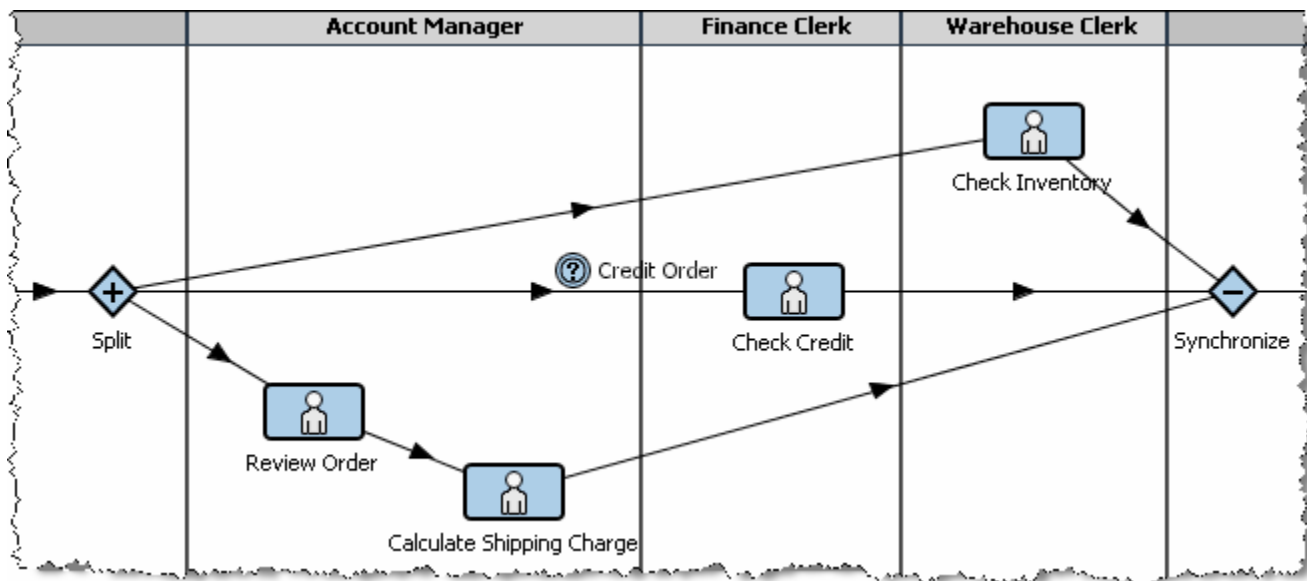


② Conditional transitions play an interesting and important role in a Parallel Split pattern.





**Figure 10: Conditional Transition in a Parallel Split Pattern**

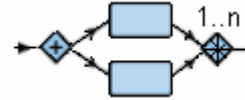
In Figure 10, the middle transition going to the Check Credit activity was changed to a conditional transition. Now the instance will flow to the Check Credit activity during runtime *only* if the order is a credit order. Since the other two transitions going to Review Order and Check Inventory are still unconditional, these two activities continue to always be executed.





**Figure 11: Synchronizing Merge Pattern**

In Figure 11, there are 3 transitions coming into the Synchronize activity on the right. The default behavior is to have the instance not continue beyond the Synchronize activity until all the valid transitions leaving the Split activity reach it. This  marshalling point is called a Join or Junction activity. Once again, suppose that the order is not being paid using credit. During runtime, the BPM server automatically detects this and knows to wait for only 2 out of the 3

transitions coming into the Synchronize activity (the transitions coming from the Check Inventory and Calculate Shipping Charge activities) before continuing. If the order is being paid by credit, the  Synchronize activity will automatically wait for all three transitions during runtime.



## Discriminator and N-out-of-M Join patterns

The Parallel Split and Synchronization pattern  can be extended to offer some flexibility as the transitions marshal in a  Join activity.

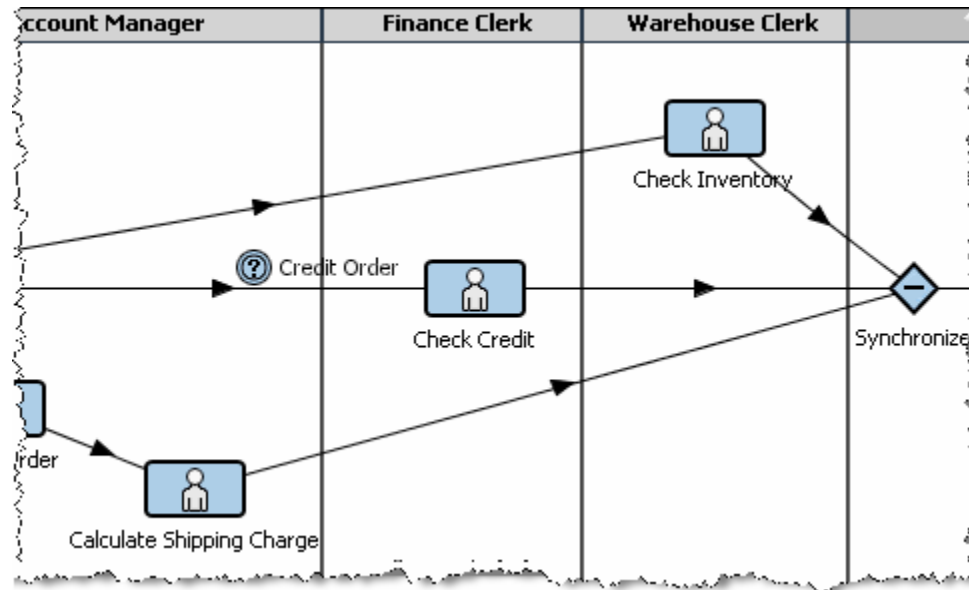

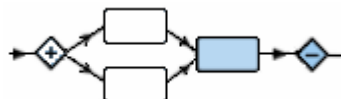




Figure 12: Discriminator and N-out-of-M Join Pattern

In the example in Figure 12, if an order's credit check fails, there is probably no need to wait for the transitions coming from either the Check Inventory or Calculate Shipping Charge activities. If the credit check fails, an indicator can be set in the Check Credit activity so that once it reaches the  Synchronize activity in the process the instance is immediately released. If this occurs, the BPM Server automatically removes the instances left stuck in the Check Inventory and Calculate Shipping, and the instance immediately continues on through the rest of the process.



## Multiple Merge Pattern

Any branch leaving the Split  activity in a Parallel Split and Synchronization pattern can be merged back into a single transition before it reaches the  Join

activity. In Figure 13, the Calculate Shipping Charge activity serves as the single merge activity and all transitions go through this activity before continuing on to the Synchronize activity.

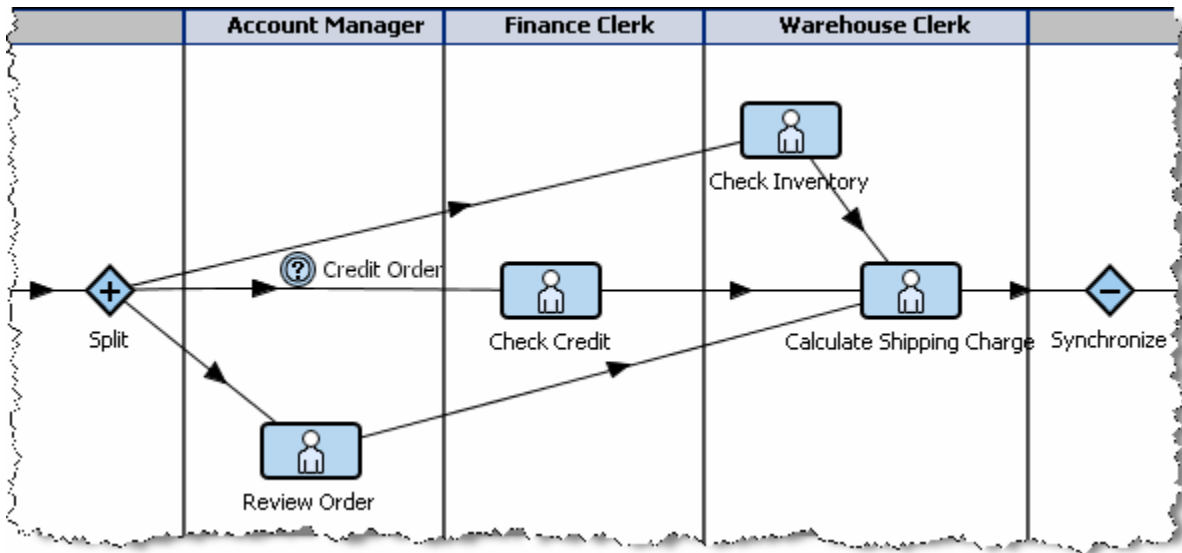
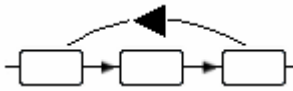


Figure 13: Merge Activity before Synchronization

## Structural Patterns

### Arbitrary Cycles Pattern



When discovering an “As-Is” process, you sometimes hear “We currently are able to go from any activity to any other activity in the process.” It is tempting to carry this forward into the “Should-Be” design. End-users sometimes appreciate that they will continue to work ad hoc. Work done upstream in the process does not have to be done particularly well or checked for completeness because it can always be “fixed” later by returning the instance back upstream.

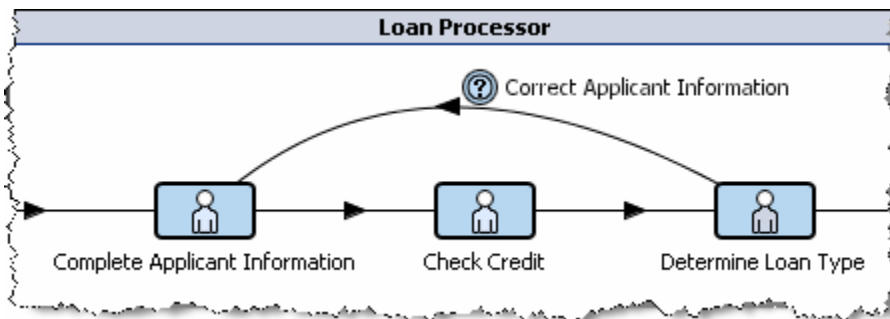


Figure 14: Upstream Transition

Transitions headed back upstream should be closely examined because:

- Instances take longer to complete the process – chances of streamlining and improving the process are greatly reduced.
- This might indicate a root problem upstream that should be resolved.
- This discourages the consistent and organized best practice execution of processes that BPM is meant to enable. This process pattern is more of a democracy than a process.

## Collaboration Pattern

Sometimes there are valid business reasons to send the instance back upstream. Collaboration patterns (Figure 15) are quite common in BPM processes.

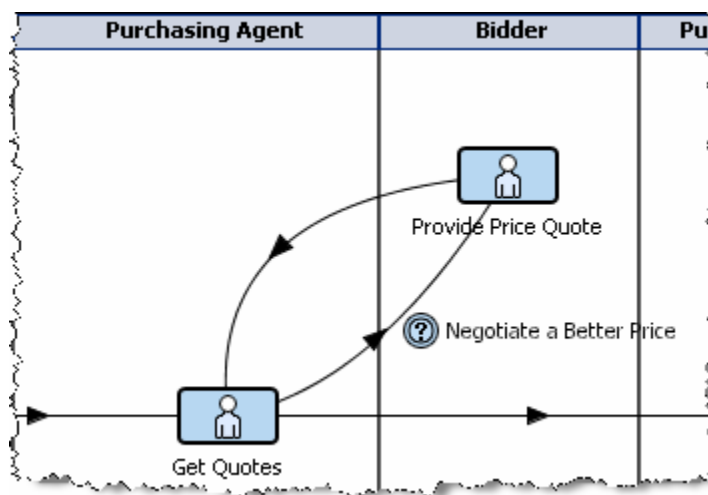
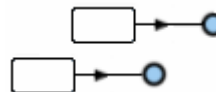


Figure 15: Collaboration Pattern

Here the instance continues in a loop until the purchasing agent decides the negotiations are over either because they have the lowest price possible or they decided to purchase from another bidder.

## Implicit Termination Pattern



This End activity is often seen as a single funnel out of the process. The End activity can then be used as the point of the merge (Figure 16) for the many transition branches in a process.



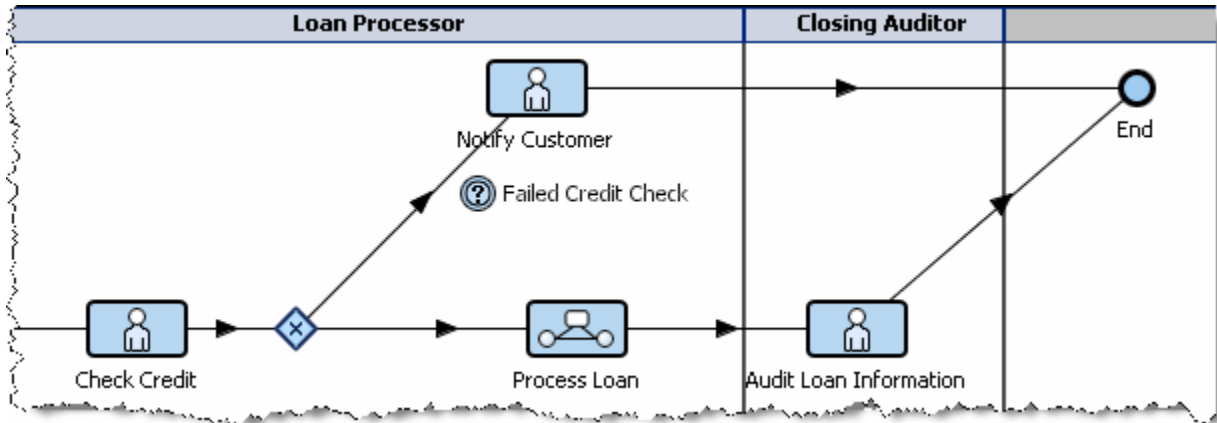


Figure 16: End Activity Used as the Merge Point

The Implicit Termination pattern provides an alternative to forcing transitions directly to the End activity. This is sometimes necessary because a process might need to reach the End activity from many activities in a process.

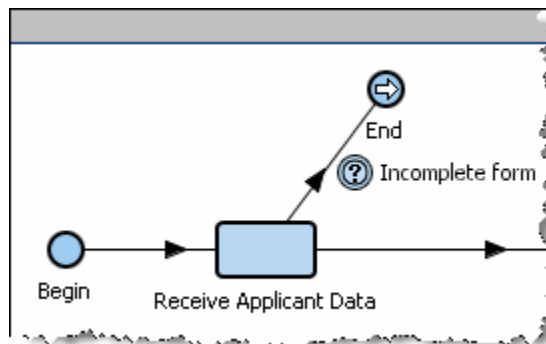



Figure 17: Implicit Termination Pattern

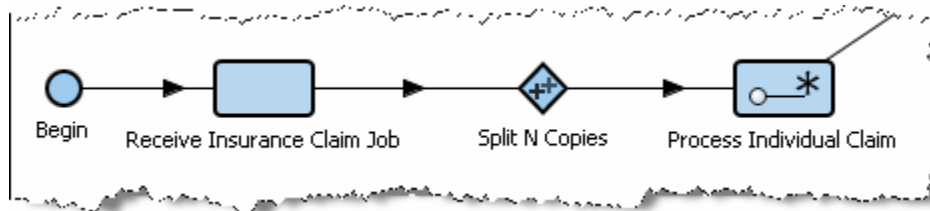
The  Connector icon shown in Figure 17 represents the End activity and prevents the unsightly clutter caused by drawing a transition across the entire width of the process.

## Multiple Instance Patterns



The patterns discussed to this point all deal with a single instance throughout the life of a process. Suppose now that each instance of a process handles a batch of insurance claims. Sometimes instance batches contain 40 claims and sometimes they contain 4000.


**Multiple Instances without Synchronization Pattern** 

This pattern takes a batch job and processes each individual item in it without worrying about synchronizing them back into the process. It is usually modeled as shown in Figure 18.





**Figure 18: Asynchronously Handling Batch Processing**

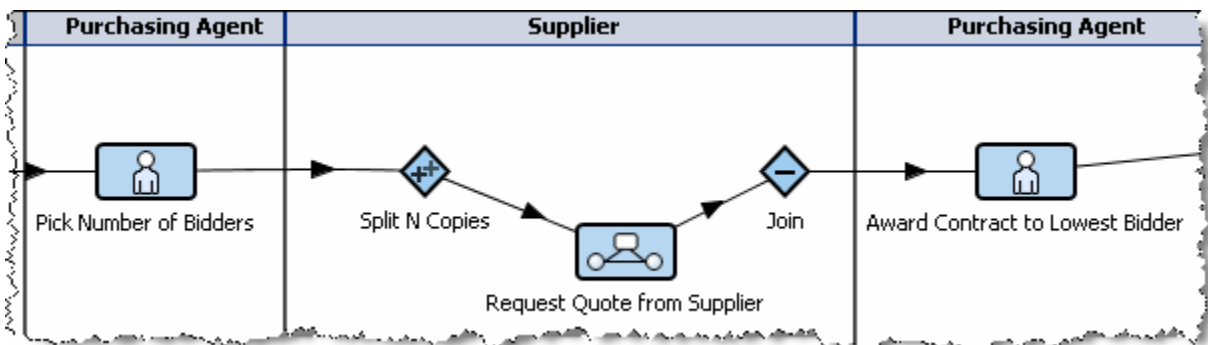
For example, depending on the number of claims in a job, the  Split-N activity parses each claim and each claim is then individually sent without synchronization (asynchronously) to a  subprocess activity. While the claim is sent to the subprocess, the context in the parent process shown in Figure 18 remains the original batch job. For every claim in the batch job, a new instance is created in the Process Individual Claim subprocess.

Although there is just the one transition drawn leaving the  Split-N activity, the power of this pattern rests in its ability to dynamically spawn new instances based on something known only at runtime (number of insurance claims in the batch job in this example).



### Multiple Instances with Design and/or Runtime Knowledge Patterns



Processes often need feedback from the instances spawned by the  Split-N activity. In the example in Figure 19, a purchasing agent decides at runtime how many suppliers will be allowed to bid. This number is used in the Split-N activity to spawn that number of instances in the  synchronous Request Quote from Supplier subprocess activity. As each supplier bids in the child subprocess, the bid information is carried back into this parent process.



**Figure 19: Synchronously Handling Batch Processing**

When the same number of bids reach the  Join activity that were spawned in the upstream  Split-N activity, the instance continues on. Use this pattern when you need information back from the subprocess. In this example the bids of the individual suppliers are compared and the contract is awarded to the lowest bidder in the Award Contract to Lowest Bidder activity.


## State Based Patterns

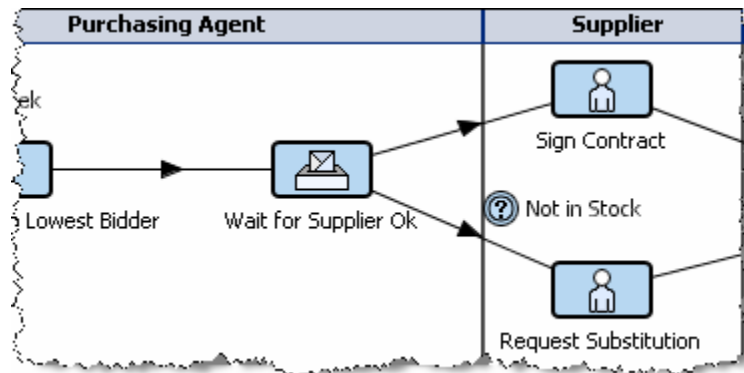
State based patterns show how to receive notifications from events outside the process and how to set Service Level Agreements (SLA) for activities in a process.



Processes sometimes need to wait for an event outside the process to occur before continuing. These events are fired from:


- Notifications from other processes currently running and
- Notifications from outside applications or web pages.

This friction built into the process is handled using an activity called a  Notification Wait activity.



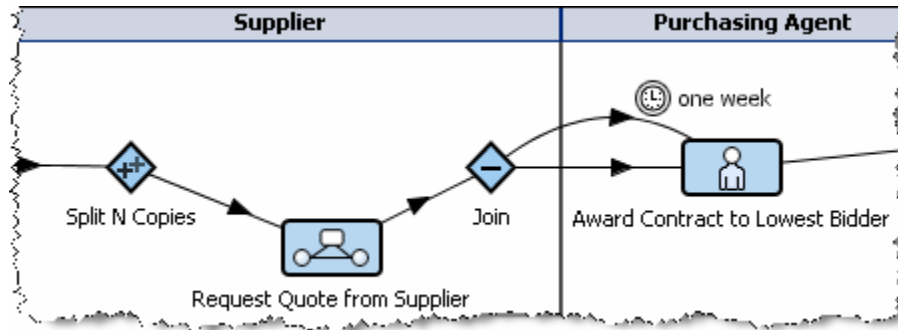
**Figure 20: Notification Wait**

In the example shown in Figure 20, the instance in the process will remain in the Wait for Supplier Ok activity until it has been notified from outside the process. The event causing the notification sends in information in this example to let the process know if the product was in stock. Based on this, the instance will travel

through either the  conditional transition (product not in stock) or the unconditional transition.



Sometimes, a Due transition can help keep a process instance moving along. Think of the Milestone pattern as a way to set an SLA for an individual activity. If an instance sits too long in the activity, it is automatically transitioned through the due transition to the next activity.



**Figure 21: Milestone Pattern**

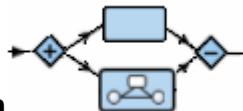
In Figure 21, the purchasing agent will award the contract to the lowest bidder if either one of two things occurs:

1. If not all the suppliers have placed their bids, the due transition times out after one week *or*
2. If the same number of bids reach the Join activity that were spawned in the upstream Split-N activity.

Due transitions are sometimes used when Join, synchronous subprocess invocations and Notification Wait activities might cause bottlenecks in processes.

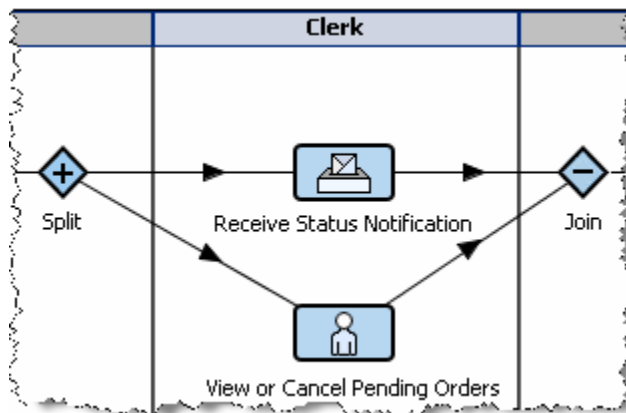
## Cancellation Patterns

One of the things quickly discovered when delivering a BPM solution is the need to cancel instances in individual activities and the process as a whole.



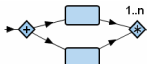
### Cancel Activity Pattern

Individual activities can create bottlenecks in the process.




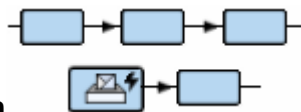
**Figure 22: Cancel Activity Pattern**

This can cause a problem in a process if the activity causing the bottleneck cannot be accessed by participants of the process. In the example in Figure 22, the Receive Status Notification has no human interaction and is waiting for an external notification that might never come.

Adding a Discriminator and N-out-of-M Join pattern  helps solve this problem for individual activities. The upper leg inside the Split/Join in this example has the transition with the long running activity with no human interaction. The lower transition was added here to:

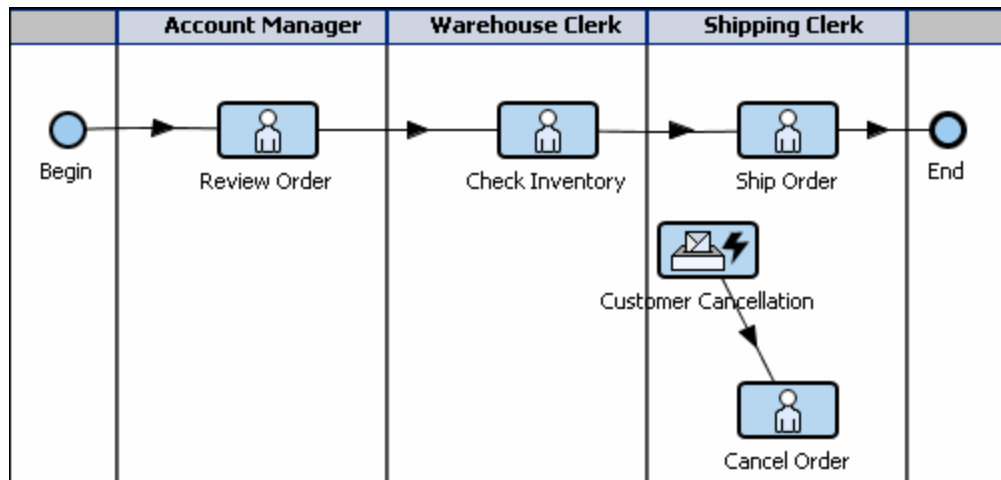
1. Give the clerk the chance to view the orders that are pending in the upper leg and
2. Allow the clerk the chance to cancel any orders that are pending in the upper leg.

Once the clerk cancels an instance, it reaches the  Join activity. The instance is removed from the Receive Status Notification activity and it continues on through the rest of the process.






### Cancel Case Pattern

Customers can call at any time and cancel their entire order no matter where the instance is located in the process. A cumbersome way to handle this is to have cancellation transitions in each activity of the process leading to the End.



**Figure 23: Cancel Case Pattern**

Instead consider the pattern shown in Figure 23. Here the  Customer Cancellation activity appears to not be part of the process flow. The  icon inside this activity indicates that it can interrupt the instance in any activity of the process and direct it immediately to this activity. Notifications cause instances sitting anywhere in the process ( Review Order, Check Inventory or Ship Order) to immediately and automatically move to the Customer Cancellation activity. Transitions do not need to be drawn to this type of activity.

## Summary

Like words that are combined to form sentences, process design patterns are combined to form complete processes. These patterns illustrate some of the best thoughts on modeling business processes today. No matter what BPM tool or diagramming standard in use, they are universally applicable solutions to the complex process problems that BPM projects encounter daily. New BPM business analysts and developers are brought up to speed quicker, and once the patterns become comfortable to the team, they become part of the shared vocabulary and experience. End users and managers find it easier to understand processes when the consistent and elegant process modeling patterns processes are applied.

Business process patterns are a practical, time proven approach to translating business specifications into outstanding BPM solutions.


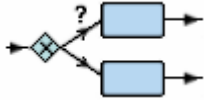
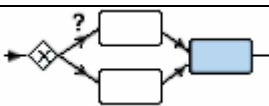
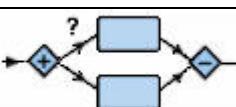
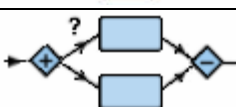
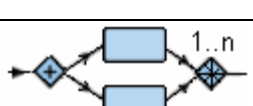


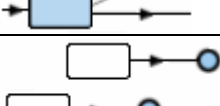
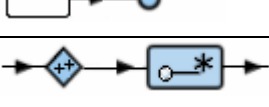
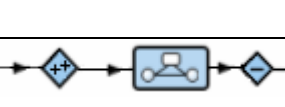
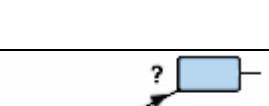
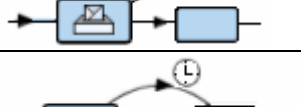
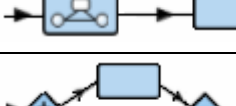
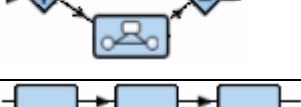
## References

1. Gamma, E., et al., Design Patterns: Elements of Reusable Object-Oriented Software, Addison Wesley, Reading, MA, 1995
2. van der Aalst, W.M.P., ter Hofstede, A.H.M., Kiepuszewski, B., and Barrios, A.P., “Workflow Patterns”, Technical Report, Eindhoven University of Technology, Eindhoven, Distributed and Parallel Databases, 14(3), pages 5-51, July 2003.
3. Havey, Michael, Essential Business Process Modeling, O'Reilly Media, Inc., 2005.

## **Appendix A – BPM Process Patterns**



## BPM Process Patterns \*

Pattern		Description
	<i>Sequence</i>	Activities executed one after the other in a series.
	<i>Exclusive Choice</i>	Two or more transitions representing different business rules leaving a single conditional activity. The instance exits through only one of the transitions
	<i>Simple Merge</i>	Combines several transitions back into a single activity.
	<i>Parallel Split and Synchronization</i>	Speeds the process by having the instance travel all the parallel paths through the branches in the pattern simultaneously.
	<i>Multiple Choice and Synchronizing Merge</i>	The instance flows from all the Split's unconditional transitions. The instance also flows through any conditional transitions that evaluate "true" during runtime.
	<i>Discriminator and N-out-of-M Join</i>	The instance flows from the Split's transitions. The instance continues when a certain number reach the complex gateway.
	<i>Multiple Merge</i>	Any branch leaving the Split activity can be merged back into a single transition before it reaches the Join activity.
	<i>Collaboration</i>	An instance continues in a loop until the conditional transition evaluates "false".
	<i>Implicit Termination</i>	Instances can reach the End activity from various locations in a process.
	<i>Multiple Instances without Synchronization</i>	Dynamically spawns new instances in an asynchronously invoked child subprocess.
	<i>Multiple Instances with Design and/or Runtime Knowledge</i>	Dynamically spawns new instances in a synchronously invoked child subprocess. The parent process receives feedback from all the child processes.
	<i>Deferred Choice</i>	Waits for an outside event before continuing.
	<i>Milestone</i>	An instance travels through a due transition after a time specified. This can help prevent bottlenecks in a process.
	<i>Cancel Activity</i>	An instance stuck in a long running activity is cancelled from the activity so it can continue to the next activity in the process.
	<i>Cancel Case</i>	An instance is cancelled from the entire process no matter where it is in the process.

\* Based on the process design patterns described by van der Aalst, W.M.P., ter Hofstede, A.H.M., Kiepuszewski, B., and Barrios, A.P., "Workflow Patterns", Technical Report, Eindhoven University of Technology, Eindhoven, *Distributed and Parallel Databases*, 14(3), pages 5-51, July 2003.