[http://www.workflowpatterns.com](http://www.workflowpatterns.com/)

[**Oracle BPM Suite 12c Modeling Patterns**](https://learning.oreilly.com/library/view/oracle-bpm-suite/9781849689021/)

<https://learning.oreilly.com/library/view/oracle-bpm-suite/9781849689021/>

#### [****Design**** ****Principles**** for ****Process****-****driven**** ****Architectures**** ****Using**** ****Oracle**** ****BPM**** and ****SOA**** ****Suite**** ****12c****](https://learning.oreilly.com/library/view/design-principles-for/9781849689441/)

<https://learning.oreilly.com/library/view/design-principles-for/9781849689441/>

Business process modeling practice is not yet mature and common modeling mistakes can make models difficult to understand, and even incomprehensible. Business Process stakeholders may not support modeling initiatives if diagrams produced are to complex and inconsistent. Some individuals consider process modeling more an art than a science. In addition, design methodology are often the field of process though leaders, solution vendors, and consulting firms. These methodologies mainly focus on project management and organizational issues of process modeling or design initiatives. They often fail to address the ground work of process modeling itself. In response to such omission, it is important to consider modeling best practices.

A best practice can be seen as a successful way to treat a particular problem that may need to be adapted in skilful ways in response to prevailing conditions.

#### Following are commonly proposed process modeling best practices, regrouped by topics.

##### Process Scope

You should clearly define the scope of the Process by identifying the “Who”, “What”, “When”, “Where”, and “Why” of your process. The Process captures the “How”. It should be clear what each instance of your process represent. The process instances are discreet and identifiable, therefore you can refer to each one of them and can count them if desired. The potential alternative ways to trigger the Process should be identified, using Start Events. The potential alternative end states of the instances of the Process should also be identified using End Events. In BPMN, Start and End Events are optional. However, processes with implicit start and end events are undesirable and could lead to misinterpretations. Use Start and End Events in each Process and Sub-process to represent its beginning and completion.

##### Diagram Layouts

Aim for BPMN Diagrams that fit one page. The top-level diagram shows the whole Process on a single page, and Sub-processes can be used to expand process detail at nested diagram levels, so you can zoom in and out of your model to describe any level of detail. You should create alternative visualizations of the same Process for different communication purposes and stakeholders. For example:

* A summary Diagram with all Sub-processes and Call Activities collapsed and not showing any Data Objects.
* A verbose Diagram with all Sub-processes and Call Activities, showing Data Objects and Annotations.

Layout your BPMN Diagrams neatly to ease readability. Diagrams can become unreadable and very confusing when the process logic is not explicit and clear. Avoid crossed lines and keep a consistent direction of flow. The diagram reading will be easier and its communication efficient. Use consistent layout with horizontal Sequence Flows, and vertical Message Flows and Associations when your draw horizontally. BPMN Diagrams are not strict temporal orderings, as it is possible to loop back, but most readers expect a left-to-right ordering. It should be clear what the primary scenario, the “Happy path” of the Process, is. Whenever possible, externalize the business rules from the Process using Business Rule Tasks to create more concise and more agile Process models. Remember that a BPMN diagram is obscured by:

* Long, meandering, crossing lines.
* Mixed flow of the primary and alternative scenarios.

Process Partitioning and Structure A business process modeller should create a hierarchical Process model with multi layers of details for the Process. BPMN Sub-processes are used to split the Process into “phases” or “layers”. Use BPMN Call Activities to re-use other Processes or fragment of these Processes into your current Process.

##### Start and End Events

Explicit instantiation and termination of the Process should be enforced by always using Start and End Events. Alternative instantiations of the process should be depicted with separate Start Events. Success and failure end states in a Process or a Sub-process should be distinguished with separate End Events. Flows that end in the same end state should be merged to the same End Event.

##### Gateways

To make it explicit, always use Gateways to depict split or merge of flow. Do not mix Gateway types when both diverging and converging flows. For example, when a flow is divided with a Parallel Gateway, the resulting parallel flows should be consolidated via another Parallel Gateway if or when required. Always place an Activity that determines the diverging condition(s) just before a diverging Gateway of type Exclusive, Inclusive and Complex. A benefit of this best practice is that this decision Activity can now be interrupted if need be. Abstract multiple daisy-chained diverging Gateways into a Business Rule Task, simplifying potential overloaded diagrams.

##### Collaborations

Do not model the internal process under focus into a Pool. Without a pool to label, one will not have the opportunity to fall into the bad practice of naming the Pool with the Process Name. Message flows can add valuable business context to your diagram, but it is important to use them consistently. For example, if you are going to show any message flows from and to a requester of your process, you really should show all of them, and show them consistently in each level of your model.

##### Other considerations

Make the process logic visible in the diagram by adequately labeling diagram elements and by showing the exception handling logic explicitly in the diagram. BPMN provides a business-friendly notation for describing exception-handling behavior. Even though the BPMN specification gives the modeler great freedom, best practice is to learn specific diagram patterns to distinguish each type of exception, and use them consistently. Make sure your model is valid. If you want others to understand your model, you need to start by making it valid, so you should validate with appropriate tools, peer reviews, etc. Keep a unique format along your diagrams and focus on a clean and friendly look and feel. Using different font sizes, colors, boxes sizes or overlapping labels might make the diagrams reading a challenge.

# BPM Modeling Patterns

# Flow Control Patterns

* Sequence flow pattern: enforces a transitive temporal ordering to process activities.

There are different types of sequence flows which are as follows:

* Default sequence flow/unconditional sequence flow
* Conditional sequence flow
* Exclusive choice (XOR or Event Based Gateway) and simple merge pattern ( XOR-Join)

The decision mechanisms are categorized as follows:

* **Data**: An example of data is conditional expression. The conditional expressions are evaluated at the gateway when the process token reaches the gateway. That path whose evaluation result is true is followed, and it can route to only one flow
* **Events** (for example, the receipt of alternative messages): An event-based XOR gateway represents a divergence point where the alternatives paths are picked based on the event that occurs at that instance in the process flow. The event could be a receipt of message or a timer event. In an event-based gateway, it's the events that determine the path to be taken and not the conditional evaluations.

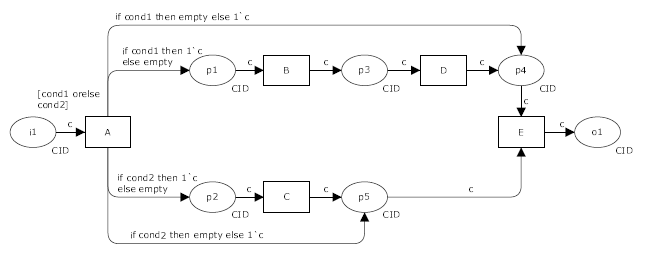
Events receive communication, and hence, correlation needs to be defined to correlate them with the main process instance.

* Multichoice (Inclusive gateway) and synchronizing merge(OR-Join) pattern

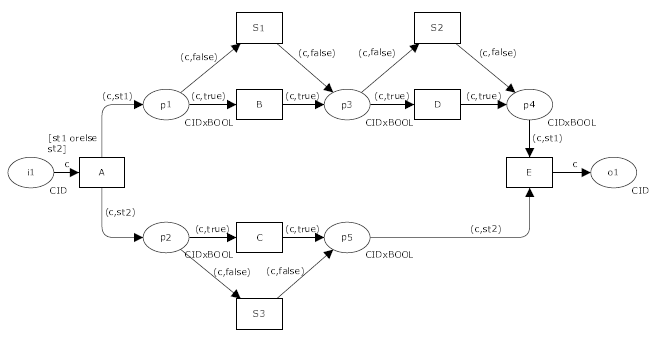
The process tokens can advance to multiple outgoing flows/paths. The sequence flow is picked based on the conditional evaluation where a token is generated for each flow for which the condition is evaluated as true; otherwise a default sequence flow is picked. The tokens are merged at the convergence, which can be an inclusive gateway.

* Structured synchronizing merge pattern

Synchronizing merge, also known as structured synchronizing merge, is implemented using the inclusive gateway. All of the tokens associated with a multichoice divergence point must reach the structured synchronizing merge before they can fire. In the case of structured synchronizing merge, there will be a single multichoice divergence point, and the structured synchronizing point will merge all the paths from that particular multichoice divergence point.



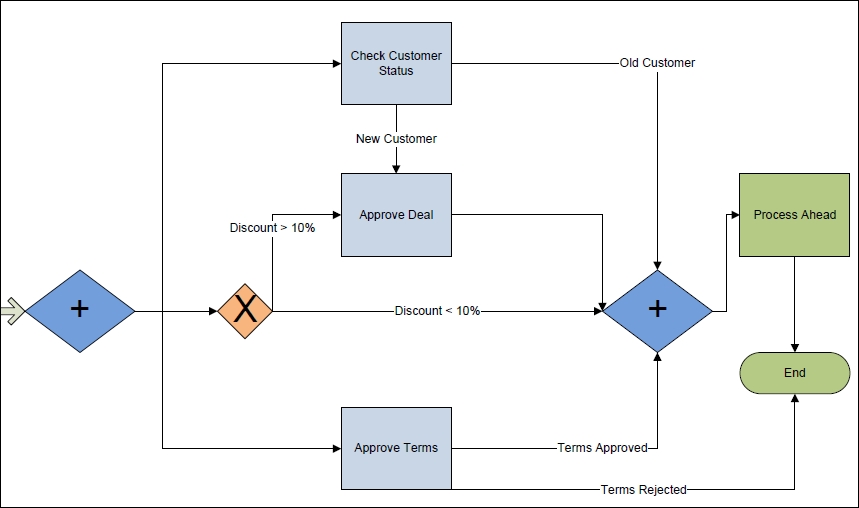
* + Local synchronizing merge pattern



* *Parallel split and synchronization pattern*

Parallel gateway split and join

* Conditional parallel split and parallel merge pattern

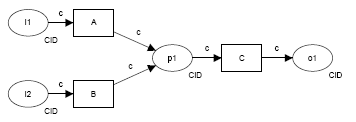


Anti-pattern: one cannot use conditional parallel split and merge by just merging some of the gateways.

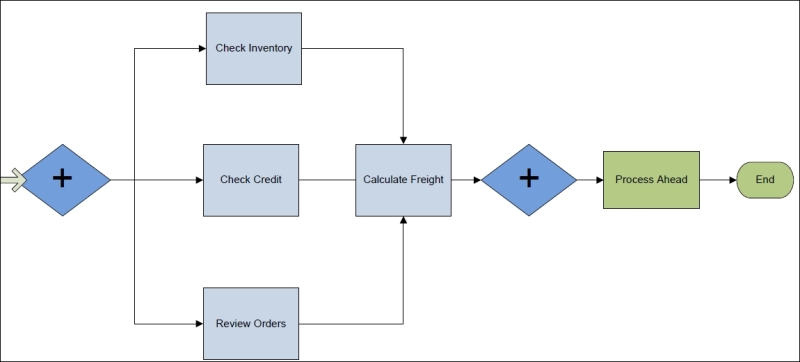
* Multimerge pattern: Convergence point without synchronization.

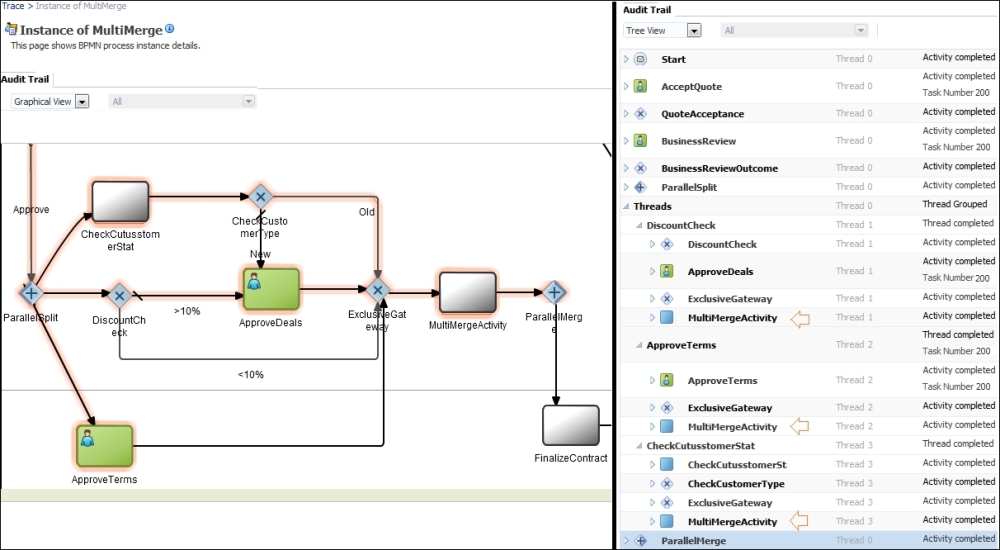
Simple merge: XOR merge. Multi-in, single out. The Simple Merge pattern provides a means of merging two or more distinct branches without synchronizing them.

Multi-merge: The convergence of two or more branches into a single subsequent branch such that each enablement of an incoming branch results in the thread of control being passed to the subsequent branch. XOR-join gateway. Accepts multiple incoming parallel sequence flow and passes the tokens as they arrive to the subsequent activity.



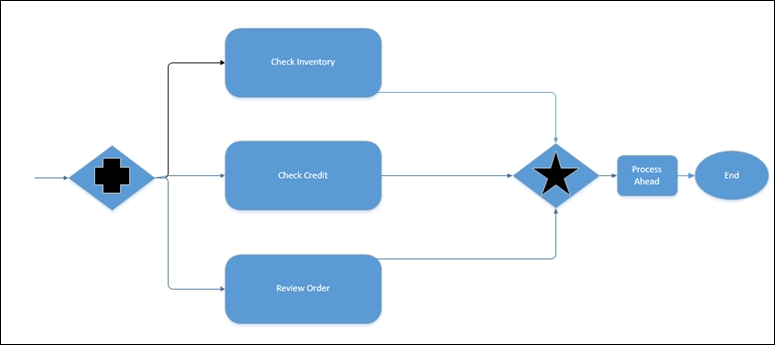
The distinction between this pattern and the Simple Merge is that it is possible for more than one incoming branch to be active simultaneously and there is no necessity for place p1 be safe.





with the usage of parallel for divergence, it would always need either a parallel gateway for convergence or a complex gateway. This means that it would always lead to synchronization of the token, either all of the tokens (with parallel gateway as convergent point) or some of the tokens (with complex gateway as the convergent point).

* Discriminator and partial join pattern



need a mechanism to set a trigger or an indicator in the converging point. When conditions related to the indicator meet, the synchronize activity in the process instance will be immediately released, and the BPM engine will automatically remove the instances struck in **Check Inventory** and **Review Order**. Then, the process instance converges at the convergence point and continues on through the rest of the process.

# Blocking Discriminator: (COMPLEX-join)

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 the first active incoming branch has been enabled. The Blocking Discriminator construct resets when all active incoming branches have been enabled once for the same process instance. Subsequent enablements of incoming branches are blocked until the Blocking Discriminator has reset.

Example: The check credentials task can commence once the confirm delegation arrival or the security check task has been completed.

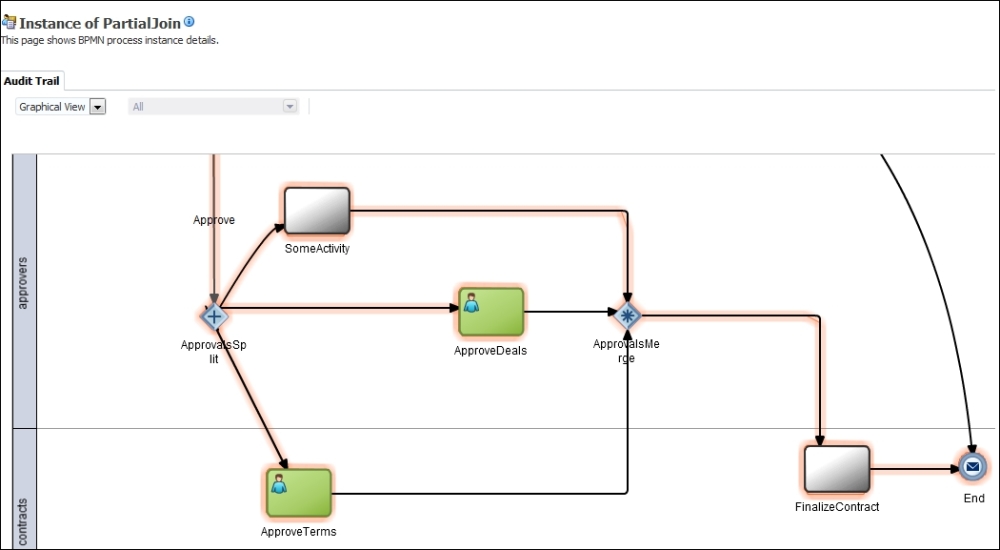
* + Structured discriminator pattern (1-out-of-M join,  COMPLEX-join)

One out of M joins. It's a special case of M out of N Join, that is, structured partial join.

Structured discriminator occurs in a structured context, that is, there must be a single parallel split construct earlier in the process model with which the structured discriminator is associated, and it must merge all of the branches that emanate from the structured discriminator.

* + Structured partial join pattern (such as 2 out of 3, complex-join)

For "N" out of "M" joins, the convergence point will trigger synchronization when the defined threshold "N" is reached.



* Complex synchronization pattern

Complex gateway gets activated when the conditional expression is evaluated as true. Once the complex gateway gets activated, it would create a token on the output sequence flow. At this time, if **Abort pending flows** is set on the complex gateway properties, then complex gateway will abort all the pending flows and the remaining tokens will be suppressed. They will not be able to trigger any subsequent branch

* + Canceling discriminator pattern (COMPLEX-join)
    1. out-of-M joins with a flag being set, is to set to abort the remaining flow pattern.
  + Canceling partial join pattern (COMPLEX-join)

N-out-of-M joins and a flag being set to **Abort Remaining Flows**.Others:

# Thread Merge: Example, Instances of the register-vehicle task run independently of each other and of other tasks in the Process Enquiry process. They are created as needed. When ten of them have completed, the process-registration-batch task should execute once to finalise the vehicle registration system records update.

Directly supported by setting the StartQuantity attribute on activities immediately following the MI activity.

# Thread Split: example, At the completion of the confirm paper receival task, initiate three instances of the subsequent independent peer review task.

Directly supported by setting the Quantity attribute on the outgoing sequence flow from an activity.

# 2. Multi-instance and State-based Patterns

* Multiple instances with prior design-time knowledge pattern

Supported via multiple instance task with MI Flow\_Condition attribute set to all.

This pattern is applicable in a multi-instance subprocess, and split and join.

This pattern has a context associated with it, which will determine the number of tasks/activities/subprocess instances; the context will be supplied at design time and will be a static value. Instances can be executed in parallel/sequence and must be synchronized before completion. This pattern behaves as a parallel split of the instances and as parallel merges at the downstream of those instances.

* Multiple instances with prior runtime knowledge pattern

This refers to the multi-instance subprocess and a variable to determine loop cardinality.

* Multiple instances without prior runtime knowledge pattern

The number of concurrent threads is not known in advance at the design time and is calculated at runtime. However, the number of instances is not known until the last instance is executed. Concurrent thread synchronization must be performed.

The BPMN service engine runs a subprocess with a MultiInstance loop marker and will create a set of instances, one for each element in the collection. The number of instances is determined by the collection size, that is, the size of the array. Once each instance gets completed, the instance tokens get synchronized and the process moves to subsequent activities.

* Static partial join for multiple instances pattern

Notionally supported via multiple instance task with MI Flow\_Condition attribute set to complex where ComplexMI \_FlowCondition is an expression that evaluates to true when exactly M instances have completed and passes on a single token to the following activity. However, it is unclear exactly how the ComplexMI\_FlowCondition should be specified.

The determination of the number of instances is performed before the first instance gets executed. The process instance moves to subsequent activities when a given number of task instances have been completed, rather than requiring all of them to finish.

This refers to the multi-instance subprocess with loop completion condition and loop cardinality.

* Canceling partial join pattern

The BPMN service engine runs a subprocess with a multi-instance loop marker and will create a set of instances. The number of to-be instances, m, is computed by the loop cardinality before the first instance of the subprocess gets started. The given number of instances n, that can allow the execution of subsequent tasks is determined by **Completion Condition**. Once n instances are completed, the subsequent completion of m-n instances is trivial and m-n instances get cancelled.

The MultiInstance subprocess with the loop completion condition and loop cardinality.

* Dynamic partial join for multiple instances pattern

Multi-instance subprocess with completion condition.

The determination of the number of instances is not performed until the final instance. A completion condition can be specified, which is evaluated each time an instance of the task completes. Once the completion condition evaluates to true, the next task in the process is triggered. Subsequent completions of the remaining task instances are cancelled and no new instances can be created.

* Structured loop pattern

Both while and repeat loops are directly supported by activity looping.

Iteration patterns are the foundation of many complex patterns. Structured loop patterns are an implementation of the while-do or repeat-until (do-while) loop.

Multi-instance subprocesses with the loop characteristics property set as loop. Loop characteristics (while-do or do-while) are defined by setting the evaluation order of loop characteristics.

* Arbitrary cycle pattern

The arbitrary cycle pattern offers a looping construct that allows multiple entry and exit points in and out of the loop.

This unstructured loop (iteration/cycle) pattern offers the flexibility to have multiple entry and exit points in the process. The arbitrary cycle pattern provides a mechanism to repeat the process parts in an unstructured way.

When a modeler is working on defining an "As-Is" process, there are requirements to shuffle from one activity to another. There are cases in which a task or an activity performed initially in the process needs to be changed/altered after reaching a certain stage in the process. This translates to the fact that one can work on the process in an ad hoc manner. The process should allow you to arbitrarily visit tasks/activities that need to be changed or altered after reaching a certain stage in the process. When a process does this, you can use exclusive gateways to realize arbitrary cycles.

Exclusive gateways can be used in nonblock structured process models.

* Trigger patterns

Trigger patterns are a set of patterns that deal with external signals.

We can use event subprocesses and boundary catch events. A boundary catch event can be attached to the activity or subprocess, and it can be configured to accept a certain event (signal). When this signal arrives or this event is raised and the activity on which the boundary catch event is configured for this event is active, then the activity/subprocess can act on this event.

* + Transient trigger pattern

The explicit initiation/termination of a task (activity/subprocess) by a signal from the same process or from an external environment.

For example, an activity can be cancelled by a cancel event. However, to cancel the activity, this activity needs to be active. When that activity is active, only the cancellation event meant for the activity will have an effect on the activity. What if cancellation event is raised when the process token has moved out of the activity for which the cancellation event is meant? In this case, the activity will not be cancelled.

We can use event subprocesses and boundary catch events. A boundary catch event can be attached to the activity or subprocess, and it can be configured to accept a certain event (signal). When this signal arrives or this event is raised and the activity on which the boundary catch event is configured for this event is active, then the activity/subprocess can act on this event.

BPMN: Not supported. Triggers are supported through durable messages.

* + Persistent trigger pattern

The explicit initiation of a task (activity/subprocess) by a signal from the same process or from an external environment.

There are cases when an external signal arrives at the process or the process itself raises signal. However, this signal will not be lost and is dealt with by the process.

For example, a process instance can be cancelled by a cancel instance event. The cancel instance event can arrive any time in the life cycle of the process.

We can use the event subprocess.

BPMN: Supported through the use of message events.

* Implicit termination pattern

End event.

* Explicit termination pattern

The process needs to end explicitly, and this will terminate all the remaining tasks/activities and cause the processes to end.

This event terminates the end event and the Error End Event.

Supported via a terminate end event.

* Cancellation pattern
  + Cancel multi-instance task pattern

We can use multi-instance subprocesses and boundary catch events. Boundary catch events can be attached to the multi-instance subprocess, and they can be configured to accept certain events (signals). These signals/events are meant to cancel the remaining instances of the multi-instance subprocess. When this signal arrives or this event is raised, then the remaining instances of the multi-instance will be cancelled.

3. Invocation patterns are discussed in this chapter:

* Web service pattern:
  + Asynchronous request callback pattern

Asynchronous service start operations are defined using the Message Start Event or Catch Event. Callback operations are defined using a Message Throw Event or a Message End Event. Correlation mechanisms should be implemented too. You can also use, Receive and Send Tasks to expose the BPM process operation as the asynchronous operation.

The service interface of an asynchronous BPM process shows a Start operation and a Callback operation. A BPM process input is defined using the Message Start Event, Catch Event, or Receive task, and the process output is defined using the Message End Event, Throw Event, or Send task. In the case of an asynchronous interface, there will be two ports: one port is for requests and one is for callbacks. The Message End Event or the Send task has to define a callback operation.

* + Synchronous request response pattern

You can configure a message start event, Message Catch Event, or Receive task event to create a service interface. This service interface can be defined with an asynchronous operation or with a synchronous operation. The BPM process input is defined using a Message Start Event, Catch Event, or Receive task, and the process output is defined using a Message End Event, Throw Event, or Send task, respectively. In the case of a synchronous interface, there will be one port for request and response. You can have the fault definition in the operation to return the error message to the service consumer. This can be achieved with the Business Exception implementation type, which would result in the generation of the fault definition in the operation. The Message Start Event can automatically set the conversation to initiate. In the case of the receive task, it should be capable enough to create instances, and a Receive task will always follow the None Start Event.

Timeout handling.

public static Map<String, BlockingQueue> queues;

myRestCall() {

var pi = camunda.startProcessInstanceByKey();

var queue = new BlockingQueue();

queues.put(pi.getId(), queue);

var x = queue.poll( 3, SECONDS);

if (x!=null) {

// process instance finished :slight\_smile:

return xxx;

} else {

// not yet done - what to do?

}

}

And then you can add a listener:

NotifyListener implements ExecutionListener {

public voud execute() {

Client.queues.get(pi.getId()).put("yeah - here you can transfer some result")

}

So as a next thing you have to either poll the engine to query when the process instance ends (e.g. via the HistoryService) and then load some process variables from it (again, e.g via the HistoryService).

And if you want to avoid the polling you need some in-memory way to notify the waiting thread that a process instance ends.

@RequestMapping(method=RequestMethod.POST)

public OrderResponse placeOrderPOST(String orderId, int amount) throws Exception {

ProcessInstance instance = camunda.getRuntimeService().startProcessInstanceByKey(//...

OrderResponse response = new OrderResponse();

for {

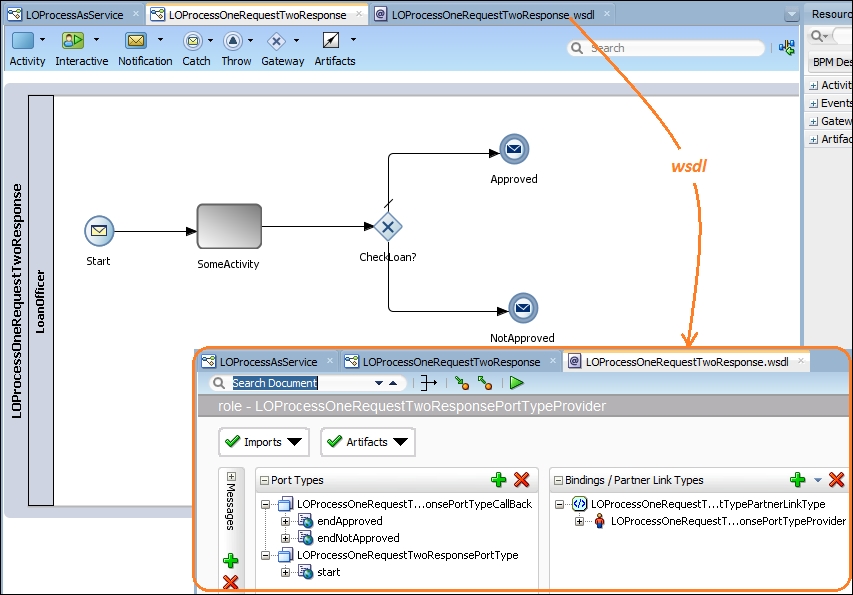
// query - and if count > 0 then use it to read variables

camunda.getHistoryService().createProcessInstanceQuery().endedI()...

}

return response;

* + One request, one of two possible responses pattern



* + Two request pattern

We can define a BPM process interface that can have two start events. This can be achieved by a BPM interface that not only exposes the Message Start Event, but also exposes the Message Catch Event. When you try to execute this process interface, the Message Start Event should always be executed first. Also, correlation should be enabled before you plan to expose the service interface of a BPM process with the start event and catch event messages together.

* One-way invocation pattern:
  + Implementing one-way invocation using a timer

A Timer Start Event can raise the BPM process instance either on a specific date (time/date) or periodically (the timer cycle), and you can specify the date or interval using a fixed value or by using an expression through an expression builder. An instance gets created each time the timer condition in the Timer Start Event gets evaluated to tru e. Similarly, in the start cycle case, the Timer Start Event is configured to use a cycle; hence, the process instance gets created periodically.

* + Implementing one-way invocation using e-mail

In the message start events and for the UMS configuration in SOA/BPM.

The UMS adapter is configured for inbound interaction; hence, the adapter is able to asynchronously receive messages/notifications from UMS. The UMS adapter is configured as a listener and has initiated threads to process it. The UMS adapter will act as a proxy between SOA and the external world. The use-case-adapter will be able to receive messages from the inbox of the loan officer, as the adapter was able to retrieve e-mails from the Gmail IMAP server supporting SSL.

* Publish/subscribe pattern
* Multievent instantiation
* Human task initiator pattern
* Guaranteed delivery pattern

# 4. Human Task

* Milestone pattern
* Routing pattern
* Assignment patterns
* List builder patterns
* Parallel routing pattern
* Serial routing pattern
* Single routing pattern
* FYI pattern
* Task aggregation pattern
* Dispatching pattern
* Escalation pattern
* Rule-based reassignment and delegation pattern
* Ad hoc routing pattern
* Request info feature
* Reassignment and delegation pattern
* Force completion pattern
* Routing rule pattern
* Error assignees and reviewers
* Deadline
* Escalation, expiry, and renewal
* Exclusion
* Error assignee and reviewer
* Notification
* Content access policy and task actions
* Enterprise content management for task documents

5. Interaction pattern

* Conversation pattern

In BPM, a conversation can be defined for a Send/Throw/End message and Receive/Catch/Start message, using the conversation property. The BPM engine uses the WS-addressing correlation or message-based correlation, and default/advance-defined WS-conversation ID for each conversation.

* Asynchronous interaction pattern

The Message Throw Event or Send Task, when used to invoke a BPM process/service, essentially initiates a conversation.

* Synchronous interaction pattern

When you need to design a synchronous interaction, use a service task. Service task invokes processes and services, synchronously. When a service tasks invoke a process or service, the token waits at the service task until a response is returned. After the response is received, the token continues to the next sequence flow in the process. You use process data objects to assign input data to an invoked service's input, and the service output is assigned back to PDOs.

* Subprocess interaction pattern
  + Reusable processes interaction pattern
  + Embedded subprocess interaction pattern
* Event-driven interaction pattern

# 6. Correlation Patterns

* Message-based correlation pattern
* Cancel instance pattern
* Update task pattern
* Query pattern
* Suspend process pattern
* Suspend activity pattern
* Cancel activity pattern

**7. Exception Handling Patterns**

* Reassigned Exception Handling Pattern
* Allocated Exception Handling Pattern
* Force-Terminate Exception Handling Pattern
* Force-Error Exception Handling Pattern
* Force-Complete Exception Handling Pattern
* Invoked Exception Handling Pattern
* Invoked State Exception Handling Pattern
* Continue Execution Exception Handling Pattern
* Force-Terminate Execution Exception Handling Pattern
* Force-Error Execution Exception Handling Pattern
  + External Exception Handling Pattern
  + Internal Exception Handling Pattern
    - Internal Complete Exception Handling Pattern
    - Internal Terminate Exception Handling Pattern
    - Internal Error Exception Handling Pattern
  + Reallocated Exception Handling Pattern
* External Exception Handling Pattern
* Process Level Exception Handling Pattern
* System Level Exception Handling Pattern
* External Triggers

# Patterns

In process-aware information systems various perspectives can be distinguished. The [control-flow perspective](http://www.workflowpatterns.com/patterns/control/index.php) captures aspects related to control-flow dependencies between various tasks (e.g. parallelism, choice, synchronization etc). Originally twenty patterns were proposed for this perspective, but in the latest iteration this has grown to over forty patterns. The [data perspective](http://www.workflowpatterns.com/patterns/data/index.php) deals with the passing of information , scoping of variables, etc, while the [resource perspective](http://www.workflowpatterns.com/patterns/resource/index.php) deals with resource to task allocation, delegation, etc. Finally the patterns for the [exception handling perspective](http://www.workflowpatterns.com/patterns/exception/index.php) deal with the various causes of exceptions and the various actions that need to be taken as a result of exceptions occurring.

Process data extracted from information systems for the purpose of process analysis needs to be of high data-quality for analysis results to be accurate and trustworthy. [Event Log Imperfection](http://www.workflowpatterns.com/patterns/logimperfection/index.php) patterns capture some frequently occurring data quality issues that commonly found in event log data used for process analysis.

# Control-Flow Patterns

### Downloads of the original and revised control-flow patterns papers:

N. Russell, A.H.M. ter Hofstede, W.M.P. van der Aalst, and N. Mulyar. [**Workflow Control-Flow Patterns: A Revised View**](http://www.workflowpatterns.com/documentation/documents/BPM-06-22.pdf). (PDF, 1.04Mb)BPM Center Report BPM-06-22 , BPMcenter.org, 2006.

W.M.P van der Aalst, A.H.M. ter Hofstede, B. Kiepuszewski, and A.P. Barros.[**Workflow Patterns**](http://www.workflowpatterns.com/documentation/documents/wfs-pat-2002.pdf)**.** (PDF, 718 Kb).Distributed and Parallel Databases, 14(3), pages 5-51, July 2003

## Introduction

The Workflow Patterns Initiative was established with the aim of delineating the fundamental requirements that arise during business process modelling on a recurring basis and describe them in an imperative way. The first deliverable of this research project was a set of twenty patterns describing the control-flow perspective of workflow systems. Since their release, these patterns have been widely used by practitioners, vendors and academics alike in the selection, design and development of workflow systems [[vdAtHKB03](http://www.workflowpatterns.com/patterns/control/bibliography.php#vdAtHKB03)]. This body of work presents the first systematic review of the original twenty control-flow patterns and provides a formal description of each of them in the form of a Coloured Petri-Net (CPN) model. It also identifies twenty three new patterns relevant to the control-flow perspective. Detailed context conditions and evaluation criteria are presented for each pattern and their implementation is assessed in fourteen commercial offerings including workflow and case handling systems, business process modelling formalisms and business process execution languages.

## Revisiting the Original Patterns

Here we present a revised description of the original twenty control-flow patterns previously presented in [[vdAtHKB03](http://www.workflowpatterns.com/patterns/control/bibliography.php#vdAtHKB03)]. Although this material is motivated by earlier research conducted as part of the Workflow Patterns Initiative, the descriptions for each of these patterns have been thoroughly revised and a new set of evaluations have been undertaken. In several cases, detailed review of a pattern has indicated that there are potentially several distinct ways in which the original pattern could be interpreted and implemented. In order to resolve these ambiguities, we have taken the decision to base the revised definition of the original pattern on the most restrictive interpretation of its operation and to delineate this from other possible interpretations that could be made. In several situations, a substantive case exists for consideration of these alterative operational scenarios and where this applies, these are presented in the form of new control-flow patterns.

## New Control Flow Patterns

Review of the patterns associated with the control-flow perspective over the past few years has led to the recognition that there are a number of distinct modelling constructs that can be identified during process modelling that are not adequately captured by the original set of twenty patterns. Here we present twenty three new control-flow patterns that augment the existing range of patterns described above and elsewhere [[vdABtHK00](http://www.workflowpatterns.com/patterns/control/bibliography.php#vdABtHK00), [vdAtHKB03](http://www.workflowpatterns.com/patterns/control/bibliography.php#vdAtHKB03)]. In an attempt to describe the operational characteristics of each pattern more rigourously, we also present a formal model in Coloured Petri-Net (CPN) format for each of them. In fact the explicit modelling of the original patterns using CPN Tools helped identify a number of new patterns as well as delineating situations where some of the original patterns turned out to be collections of patterns.

## Basic Control Flow Patterns

This class of pattern captures elementary aspects of process control and are similar to the definitions of these concepts initially proposed by the Workflow Management Coalition (WfMC) [[Wor99](http://www.workflowpatterns.com/patterns/control/bibliography.php#Wor99)].

1. [Sequence](http://www.workflowpatterns.com/patterns/control/basic/wcp1.php)2. [Parallel Split](http://www.workflowpatterns.com/patterns/control/basic/wcp2.php)3. [Synchronization](http://www.workflowpatterns.com/patterns/control/basic/wcp3.php)4. [Exclusive Choice](http://www.workflowpatterns.com/patterns/control/basic/wcp4.php)5. [Simple Merge](http://www.workflowpatterns.com/patterns/control/basic/wcp5.php)

## Advanced Branching and Synchronization Patterns

Here we present a series of patterns which characterise more complex branching and merging concepts which arise in business processes. Although relatively commonplace in practice, these patterns are often not directly supported or even able to be represented in many commercial offerings. The original control-flow patterns identified four of these patterns: Multi-Choice, Synchronizing Merge, Multi-Merge and Discriminator.

In this revision, the Multi-Choice and Multi-Merge have been retained in their previous form albeit with a more formal description of their operational semantics. For the other patterns however, it has been recognized that there are a number of distinct alternatives to the manner in which they can operate. The original Synchronizing Merge now provides the basis for three patterns: the Structured Synchronizing Merge (WCP7), the Acyclic Synchronizing Merge (WCP37) and the General Synchronizing Merge (WCP38).

In a similar vein, the original Discriminator pattern is divided into six (6) distinct patterns: the Structured Discriminator (WCP9), the Blocking Discriminator (WCP28), the Cancelling Discriminator (WCP29), the Structured Partial Join (WCP30), the Blocking Partial Join (WCP31) and the Cancelling Partial Join (WCP32). One other addition has been the Generalized AND-Join (WCP33) which identifies a more flexible AND-join useful in concurrent processes.

Of these patterns, the original descriptions for the Synchronizing Merge and the Discriminator are superseded by their structured definitions.

6. [Multi-Choice](http://www.workflowpatterns.com/patterns/control/advanced_branching/wcp6.php)7. [Structured Synchronizing Merge](http://www.workflowpatterns.com/patterns/control/advanced_branching/wcp7.php)8. [Multi-Merge](http://www.workflowpatterns.com/patterns/control/advanced_branching/wcp8.php)9. [Structured Discriminator](http://www.workflowpatterns.com/patterns/control/advanced_branching/wcp9.php)28. [Blocking Discriminator](http://www.workflowpatterns.com/patterns/control/new/wcp28.php)29. [Cancelling Discriminator](http://www.workflowpatterns.com/patterns/control/new/wcp29.php)30. [Structured Partial Join](http://www.workflowpatterns.com/patterns/control/new/wcp30.php)31. [Blocking Partial Join](http://www.workflowpatterns.com/patterns/control/new/wcp31.php)32. [Cancelling Partial Join](http://www.workflowpatterns.com/patterns/control/new/wcp32.php)33. [Generalised AND-Join](http://www.workflowpatterns.com/patterns/control/new/wcp33.php)37. [Local Synchronizing Merge](http://www.workflowpatterns.com/patterns/control/new/wcp37.php)38. [General Synchronizing Merge](http://www.workflowpatterns.com/patterns/control/new/wcp38.php)41. [Thread Merge](http://www.workflowpatterns.com/patterns/control/new/wcp41.php)42. [Thread Split](http://www.workflowpatterns.com/patterns/control/new/wcp42.php)

## Multiple Instance Patterns

Multiple instance patterns describe situations where there are multiple threads of execution active in a process model which relate to the same activity (an hence share the same implementation definition). Multiple instances can arise in three situations:

1. An activity is able to initiate multiple instances of itself when triggered (we denote this form of activity a multiple instance activity);
2. A given activity is initiated multiple times as a consequence of it receiving several independent triggerings (e.g. as part of a loop or in a process instance in which there are several concurrent threads of execution as might result from a Multi-Merge for example; and
3. Two or more activities in a process share the same implementation definition. This may be the same activity definition in the case of a multiple instance activity or a common sub-process definition in the case of a block activity. Two (or more) of these activities are triggered such that their executions overlap (either partially or wholly).

Although all of these situations potentially involve multiple concurrent instances of an activity or sub-process, it is the first of them in which we are most interested as they require the triggering and synchonization of multiple concurrent activity instances. This group of patterns focusses on the various ways in which these events can occur.

Similar to the differentiation introduced in the Advanced Branching and Synchronization Patterns to capture the distinction between the Discriminator and the Partial Join pattern variants, three new patterns have been introduced to recognize alternative operational semantics for multiple instances. These are the the Static Partial Join for Multiple Instances (WCP34), the Cancelling Static Partial Join for Multiple Instances (WCP35) and the Dynamic Partial Join for Multiple Instances (WCP36).

12. [Multiple Instances without Synchronization](http://www.workflowpatterns.com/patterns/control/multiple_instance/wcp12.php)13. [Multiple Instances with a Priori Design-Time Knowledge](http://www.workflowpatterns.com/patterns/control/multiple_instance/wcp13.php)14. [Multiple Instances with a Priori Run-Time Knowledge](http://www.workflowpatterns.com/patterns/control/multiple_instance/wcp14.php)15. [Multiple Instances without a Priori Run-Time Knowledge](http://www.workflowpatterns.com/patterns/control/multiple_instance/wcp15.php)34. [Static Partial Join for Multiple Instances](http://www.workflowpatterns.com/patterns/control/new/wcp34.php)35. [Cancelling Partial Join for Multiple Instances](http://www.workflowpatterns.com/patterns/control/new/wcp35.php)36. [Dynamic Partial Join for Multiple Instances](http://www.workflowpatterns.com/patterns/control/new/wcp36.php)

## State-based Patterns

State-based patterns reflect situations for which solutions are most easily accomplished in process languages that support the notion of state. In this context, we consider the state of a process instance to include the broad collection of data associated with current execution including the status of various activities as well as process-relevant working data such as activity and case data elements.

The original patterns include three patterns in which the current state is the main determinant in the course of action that will be taken from a control-flow perspective. These are: Deferred Choice (WCP16), where the decision about which branch to take is based on interaction with the operating environment, Interleaved Parallel Routing (WCP 17), where two or more sequences of activities are undertaken on an interleaved basis such that only one activity instance is executing at any given time and Milestone (WCP18), where the enabling of a given activity only occurs where the process is in a specific state.

In recognition of further state-based modelling scenarios, four new patterns have also been identified. These are: Critical Section (WCP39), which provides the ability to prevent concurrent execution of specific parts of a process, Interleaved Routing (WCP40), which denotes situations where a group of activities can be executed sequentially in any order, and Thread Merge (WCP41) and Thread Split (WCP42) which provide for coalescence and divergence of distinct threads of control along a single branch.

16. [Deferred Choice](http://www.workflowpatterns.com/patterns/control/state/wcp16.php)17. [Interleaved Parallel Routing](http://www.workflowpatterns.com/patterns/control/state/wcp17.php)18. [Milestone](http://www.workflowpatterns.com/patterns/control/state/wcp18.php)39. [Critical Section](http://www.workflowpatterns.com/patterns/control/new/wcp39.php)40. [Interleaved Routing](http://www.workflowpatterns.com/patterns/control/new/wcp40.php)

## Cancellation and Force Completion Patterns

Several of the patterns above (e.g. (WCP6) Structured Synchronizing Merge and (WCP9) Structured Discriminator) have variants that utlize the concept of activity cancellation where enabled or active activity instance are withdrawn. Various forms of exception handling in processes are also based on cancellation concepts. This section presents two cancellation patterns - Cancel Task (WCP19) and Cancel Case (WCP20).

Three new cancellation patterns have also been identified Cancel Region (WCP25), Cancel Multiple Instance Activity (WCP26) and Complete Multiple Instance Activity (WCP27).

19. [Cancel Task](http://www.workflowpatterns.com/patterns/control/cancellation/wcp19.php)20. [Cancel Case](http://www.workflowpatterns.com/patterns/control/cancellation/wcp20.php)25. [Cancel Region](http://www.workflowpatterns.com/patterns/control/new/wcp25.php)26. [Cancel Multiple Instance Activity](http://www.workflowpatterns.com/patterns/control/new/wcp26.php)27. [Complete Multiple Instance Activity](http://www.workflowpatterns.com/patterns/control/new/wcp27.php)

## Iteration Patterns

The following patterns deal with capturing repetitive behaviour in a workflow.

10. [Arbitrary Cycles](http://www.workflowpatterns.com/patterns/control/structural/wcp10.php)21. [Structured Loop](http://www.workflowpatterns.com/patterns/control/new/wcp21.php)22. [Recursion](http://www.workflowpatterns.com/patterns/control/new/wcp22.php)

## Termination Patterns

The following patterns deal with the circumstances under which a workflow is considered to be completed.

11. [Implicit Termination](http://www.workflowpatterns.com/patterns/control/structural/wcp11.php)43. [Explicit Termination](http://www.workflowpatterns.com/patterns/control/new/wcp43.php)

## Trigger Patterns

The following patterns deal with the external signals that may be required to start certain tasks.

23. [Transient Trigger](http://www.workflowpatterns.com/patterns/control/new/wcp23.php)24. [Persistent Trigger](http://www.workflowpatterns.com/patterns/control/new/wcp24.php)

## Disclaimer

We, the authors and the associated institutions, assume no legal liability or responsibility for the accuracy and completeness of any product-specific information contained in this body of work. All possible efforts have been make to ensure that the results presented are, to the best of our knowledge, up to date and correct.

# Workflow Data Patterns

### Download of the data patterns paper:

N. Russell, A.H.M. ter Hofstede, D. Edmond, and W.M.P. van der Aalst.  
[**Workflow Data Patterns**](http://www.workflowpatterns.com/documentation/documents/data_patterns%20BETA%20TR.pdf)**.** (PDF, 423 Kb).  
QUT Technical report, FIT-TR-2004-01, Queensland University of Technology, Brisbane, 2004.

## Introduction

Workflow systems seek to provide an implementation vehicle for complex, recurring business processes. Notwithstanding this common objective, there are a variety of distinct features offered by commercial workflow management systems. These differences result in significant variations in the ability of distinct tools to represent and implement the plethora of requirements that may arise in contemporary business processes. Many of these requirements recur quite frequently during the requirements analysis activity for workflow systems and abstractions of these requirements serve as a useful means of identifying the key components of workflow languages. Previous work has identified a number of Workflow Control-flow Patterns which characterise the range of control flow constructs that might be encountered when modelling and analysing workflow. In this web site you will find a series of Workflow Data Patterns that aim to capture the various ways in which data is represented and utilised in workflows. By delineating these Patterns in a form that is independent of specific workflow technologies and modelling languages, we are able to provide a comprehensive treatment of the workflow data perspective and we subsequently use these Patterns as the basis for a detailed comparison of a number of commercially available workflow management systems and business process modelling languages.

Before you view the different data patterns, you may wish to examine one of the following options to gain a better understanding of the work presented to you:

* To get a better understanding of the data characteristics that occur repeatedly in workflow systems, go to the [data characterisation](http://www.workflowpatterns.com/patterns/data/data_characterisation.php) web page.
* To get a better understanding of the basic terms and concepts used in this body of work, go to the [workflow structure](http://www.workflowpatterns.com/patterns/data/workflow_structure.php) web page.

## Data Visibility

Within the context of a process, there are a variety of distinct ways in which data elements can be defined and utilized. Typically these variations relate to the process construct to which they are anchored and the scope in which they are accessible. More importantly, they directly influence the way in which the data element may be used, e.g. to capture production information, to manage control data or for communication with the external environment. Here we consider each of the potential contexts in which a data construct can be defined and utilized.

1. [Task Data](http://www.workflowpatterns.com/patterns/data/visibility/wdp1.php)
2. [Block Data](http://www.workflowpatterns.com/patterns/data/visibility/wdp2.php)
3. [Scope Data](http://www.workflowpatterns.com/patterns/data/visibility/wdp3.php)
4. [Multiple Instance Data](http://www.workflowpatterns.com/patterns/data/visibility/wdp4.php)
5. [Case Data](http://www.workflowpatterns.com/patterns/data/visibility/wdp5.php)
6. [Folder Data](http://www.workflowpatterns.com/patterns/data/visibility/wdp6.php)
7. [Workflow Data](http://www.workflowpatterns.com/patterns/data/visibility/wdp7.php)
8. [Environment Data](http://www.workflowpatterns.com/patterns/data/visibility/wdp8.php)

## Data Interaction

Here we examine the various ways in which data elements can be passed between components in a process and how the characteristics of the individual components can influence the manner in which the trafficking of data elements occurs. Of particular interest is the distinction between the communication of data between components within a process as against the data-oriented interaction of a process element with the external environment.

### Internal Data Interaction

1. [Data Interaction - Task to Task](http://www.workflowpatterns.com/patterns/data/internal/wdp9.php)
2. [Data Interaction - Block Task to Sub-Workflow Decomposition](http://www.workflowpatterns.com/patterns/data/internal/wdp10.php)
3. [Data Interaction - Sub-Workflow Decomposition to Block Task](http://www.workflowpatterns.com/patterns/data/internal/wdp11.php)
4. [Data Interaction - to Multiple Instance Task](http://www.workflowpatterns.com/patterns/data/internal/wdp12.php)
5. [Data Interaction - from Multiple Instance Task](http://www.workflowpatterns.com/patterns/data/internal/wdp13.php)
6. [Data Interaction - Case to Case](http://www.workflowpatterns.com/patterns/data/internal/wdp14.php)

### External Data Interaction

1. [Data Interaction - Task to Environment - Push-Oriented](http://www.workflowpatterns.com/patterns/data/external/wdp15.php)
2. [Data Interaction - Environment to Task - Pull-Oriented](http://www.workflowpatterns.com/patterns/data/external/wdp16.php)
3. [Data Interaction - Environment to Task - Push-Oriented](http://www.workflowpatterns.com/patterns/data/external/wdp17.php)
4. [Data Interaction - Task to Environment - Pull-Oriented](http://www.workflowpatterns.com/patterns/data/external/wdp18.php)
5. [Data Interaction - Case to Environment - Push-Oriented](http://www.workflowpatterns.com/patterns/data/external/wdp19.php)
6. [Data Interaction - Environment to Case - Pull-Oriented](http://www.workflowpatterns.com/patterns/data/external/wdp20.php)
7. [Data Interaction - Environment to Case - Push-Oriented](http://www.workflowpatterns.com/patterns/data/external/wdp21.php)
8. [Data Interaction - Case to Environment - Pull-Oriented](http://www.workflowpatterns.com/patterns/data/external/wdp22.php)
9. [Data Interaction - Workflow to Environment - Push-Oriented](http://www.workflowpatterns.com/patterns/data/external/wdp23.php)
10. [Data Interaction - Environment to Workflow - Pull-Oriented](http://www.workflowpatterns.com/patterns/data/external/wdp24.php)
11. [Data Interaction - Environment to Workflow - Push-Oriented](http://www.workflowpatterns.com/patterns/data/external/wdp25.php)
12. [Data Interaction - Workflow to Environment - Pull-Oriented](http://www.workflowpatterns.com/patterns/data/external/wdp26.php)

## Data Transfer Patterns

Here we consider the manner in which the actual transfer of data elements occurs between one process component and another. These patterns serve as an extension to those presented in the Data Interaction patterns and aim to capture the various mechanisms by which data elements can be passed across the interface of a process component.

The specific style of data passing that is used in a given scenario depends on a number of factors including whether the two components share a common address space for data elements, whether it is intended that a distinct copy of an element is passed as against a reference to it and whether the component receiving the data element can expect to have exclusive access to it. These variations give rise to a number of distinct patterns as described below.

1. [Data Transfer by Value - Incoming](http://www.workflowpatterns.com/patterns/data/mechanisms/wdp27.php)
2. [Data Transfer by Value - Outgoing](http://www.workflowpatterns.com/patterns/data/mechanisms/wdp28.php)
3. [Data Transfer - Copy In/Copy Out](http://www.workflowpatterns.com/patterns/data/mechanisms/wdp29.php)
4. [Data Transfer by Reference - Unlocked](http://www.workflowpatterns.com/patterns/data/mechanisms/wdp30.php)
5. [Data Transfer by Reference - With Lock](http://www.workflowpatterns.com/patterns/data/mechanisms/wdp31.php)
6. [Data Transformation - Input](http://www.workflowpatterns.com/patterns/data/mechanisms/wdp32.php)
7. [Data Transformation - Output](http://www.workflowpatterns.com/patterns/data/mechanisms/wdp33.php)

## Data-based Routing

Whereas the above sections have examined characteristics of data elements in isolation from other process perspectives (i.e. control, resource, etc.), the following patterns capture the various ways in which data elements can interact with other perspectives and influence the overall operation of a process instance.

1. [Task Precondition - Data Existence](http://www.workflowpatterns.com/patterns/data/routing/wdp34.php)
2. [Task Precondition - Data Value](http://www.workflowpatterns.com/patterns/data/routing/wdp35.php)
3. [Task Postcondition - Data Existence](http://www.workflowpatterns.com/patterns/data/routing/wdp36.php)
4. [Task Postcondition - Data Value](http://www.workflowpatterns.com/patterns/data/routing/wdp37.php)
5. [Event-based Task Trigger](http://www.workflowpatterns.com/patterns/data/routing/wdp38.php)
6. [Data-based Task Trigger](http://www.workflowpatterns.com/patterns/data/routing/wdp39.php)
7. [Data-based Routing](http://www.workflowpatterns.com/patterns/data/routing/wdp40.php)

## Disclaimer

We, the authors and the associated institutions, assume no legal liability or responsibility for the accuracy and completeness of any product-specific information contained in this body of work. All possible efforts have been make to ensure that the results presented are, to the best of our knowledge, up to date and correct.

# Workflow Resource Patterns

### Download of the resource patterns paper:

N. Russell, A.H.M. ter Hofstede, D. Edmond, and W.M.P. van der Aalst.  
[**Workflow Resource Patterns**](http://www.workflowpatterns.com/documentation/documents/Resource%20Patterns%20BETA%20TR.pdf)**.** (PDF, 418 Kb).  
BETA Working Paper Series, WP 127, Eindhoven University of Technology, Eindhoven, 2004.

## Introduction

Workflow systems seek to provide an implementation vehicle for complex, recurring business processes. Notwithstanding this common objective, there are a variety of distinct features offered by commercial workflow management systems. These differences result in significant variations in the ability of distinct tools to represent and implement the plethora of requirements that may arise in contemporary business processes. Many of these requirements recur quite frequently during the requirements analysis activity for workflow systems and abstractions of these requirements serve as a useful means of identifying the key components of workflow languages. Previous work has identified a number of Workflow Control-flow Patterns and Workflow Data Patterns, which characterize the range of control flow and data constructs that might be encountered when modelling and analysing workflows. In this body of work, we describe a series of Workflow Resource Patterns that aim to capture the various ways in which resources are represented and utilized in workflows. By delineating these Patterns in a form that is independent of specific workflow technologies and modelling languages, we are able to provide a comprehensive treatment of the resource perspective and we subsequently use these Patterns as the basis for a detailed comparison of a number of commercially available workflow management systems and business process modelling languages.

Before you view the different resource patterns, you may wish to examine one of the following options to gain a better understanding of the work presented to you:

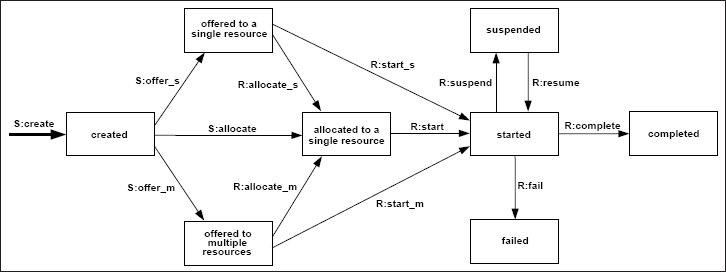
* For a general explanation of terms, go to the [resource modelling](http://www.workflowpatterns.com/patterns/resource/resource_modelling.php) web page.
* For an introduction to the concepts used in this body of work, go to the [workflow structure](http://www.workflowpatterns.com/patterns/resource/workflow_structure.php) web page.
* To read about the lifecycle of a work item, go to the [work distribution of resources](http://www.workflowpatterns.com/patterns/resource/work_distribution.php) web page.

## Creation Patterns

Creation Patterns correspond to limitations on the manner in which a work item may be executed. They are specified at design time, usually in relation to a task, and serve to restrict the range of resources that can undertake work items that correspond to the task. They also influence the manner in which a work item can be matched with a resource that is capable of undertaking it.

The essential rationale for creation patterns is that they provide a degree of clarity about how a work item should be handled after creation during the offering and allocation stages prior to it being executed. This ensures that the operation of a process conforms with its intended design principles and operates as efficiently and deterministically as possible.

In terms of the work item lifecycle, creation patterns come into effect at the time a work item is created. This state transition occurs at the beginning of the work item lifetime and is illustrated by the bold arrow in Figure [3](http://www.workflowpatterns.com/patterns/resource/index.php#fig3). For all of these patterns it is assumed that there is an associated organizational model which allows resources to be uniquely identified and that there is a mechanism to distribute work items to specific resources identified in the organizational model. As creation patterns are specified at design time, they usually form part of the process model which describes a business process.



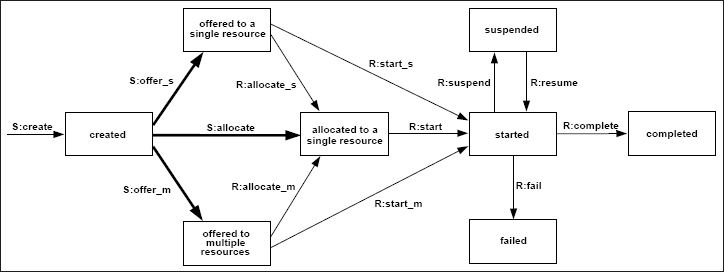
**Figure 3**: Creation Patterns

1. [Direct Distribution](http://www.workflowpatterns.com/patterns/resource/creation/wrp1.php)
2. [Role-Based Distribution](http://www.workflowpatterns.com/patterns/resource/creation/wrp2.php)
3. [Deferred Distribution](http://www.workflowpatterns.com/patterns/resource/creation/wrp3.php)
4. [Authorization](http://www.workflowpatterns.com/patterns/resource/creation/wrp4.php)
5. [Separation of Duties](http://www.workflowpatterns.com/patterns/resource/creation/wrp5.php)
6. [Case Handling](http://www.workflowpatterns.com/patterns/resource/creation/wrp6.php)
7. [Retain Familiar](http://www.workflowpatterns.com/patterns/resource/creation/wrp7.php)
8. [Capability-Based Distribution](http://www.workflowpatterns.com/patterns/resource/creation/wrp8.php)
9. [History-Based Distribution](http://www.workflowpatterns.com/patterns/resource/creation/wrp9.php)
10. [Organisational Distribution](http://www.workflowpatterns.com/patterns/resource/creation/wrp10.php)
11. [Automatic Execution](http://www.workflowpatterns.com/patterns/resource/creation/wrp11.php)

## Push Patterns

Push Patterns characterise situations where newly created work items are proactively offered or allocated to resources by the system. These may occur indirectly by advertising work items to selected resources via a shared work list or directly with work items being allocated to specific resources. In both situations however, it is the system that takes the initiative and causes the distribution process to occur. Figure [4](http://www.workflowpatterns.com/patterns/resource/index.php#fig4) illustrates (as bold arcs) the potential state transitions associated with push-based distribution:

* S:offer\_s corresponds to a work item being offered to a single resource
* S:offer\_m corresponds to a work item being offered to multiple resources (one of which will ultimately execute it)
* S:allocate corresponds to a work item being directly allocated to a resource immediately after it has been created



**Figure 4:** Push Patterns

Nine push Patterns have been identified. These divide into three distinct groups. The first three Patterns identify the actual manner of work distribution - whether the workflow system offers the work item to a single resource, to multiple resources or whether it allocates it directly to a single resource (footnote [1).](http://www.workflowpatterns.com/patterns/resource/footnotes.php" \l "foot363) These patterns correspond directly to the bold arcs in Figure [4](http://www.workflowpatterns.com/patterns/resource/index.php#fig4).

The second group of patterns relate to the means by which a resource is selected to undertake a work item where there are multiple possible resources identified. Three possible strategies are described - random allocation, round robin allocation and shortest queue. These patterns correspond to alternate ways in which the S:offer\_s and S:allocate transitions may occur.

The final three patterns identify the timing of the distribution process and in particular the relationship between the availability of a work item for offering/allocation to resources and the time at which it commences execution. Three variants are possible - work items are offered/allocated before they have commenced (early distribution), after they have commence (late distribution) or the two events are simultaneous (distribution on enablement). These Patterns do not have a direct analogue in Figure [4](http://www.workflowpatterns.com/patterns/resource/index.php#fig4) but relate to the time at which the S:offer\_s, S:offer\_m and S:allocate transitions may occur with respect to the work item's readiness to be executed (i.e. already started, immediate start or subsequent start).

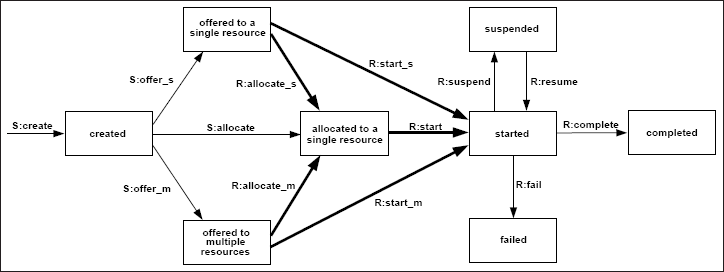
1. [Distribution by Offer - Single Resource](http://www.workflowpatterns.com/patterns/resource/push/wrp12.php)
2. [Distribution by Offer - Multiple Resources](http://www.workflowpatterns.com/patterns/resource/push/wrp13.php)
3. [Distribution by Allocation - Single Resource](http://www.workflowpatterns.com/patterns/resource/push/wrp14.php)
4. [Random Allocation](http://www.workflowpatterns.com/patterns/resource/push/wrp15.php)
5. [Round Robin Allocation](http://www.workflowpatterns.com/patterns/resource/push/wrp16.php)
6. [Shortest Queue](http://www.workflowpatterns.com/patterns/resource/push/wrp17.php)
7. [Early Distribution](http://www.workflowpatterns.com/patterns/resource/push/wrp18.php)
8. [Distribution on Enablement](http://www.workflowpatterns.com/patterns/resource/push/wrp19.php)
9. [Late Distribution](http://www.workflowpatterns.com/patterns/resource/push/wrp20.php)

## Pull Patterns

Pull Patterns correspond to the situation where individual resources are made aware of specific work items, that require execution, either via a direct offer from the system or indirectly through a shared work list. The commitment to undertake a specific task is initiated by the resource itself rather than the system. Generally this results in the work item being placed on the specific work list for the individual resource for later execution although in some cases, the resource may elect to commence execution on the work item immediately. The various state transitions associated with pull patterns are illustrated in Figure [5](http://www.workflowpatterns.com/patterns/resource/index.php#fig5):

* R:allocate\_s corresponds to the situation where a work item has been offered to a single resource and the resource has indicated it will commit to executing at some future time
* R:allocate\_m corresponds to the situation where a work item has been offered to multiple resources and one of the resources has indicated it will commit to executing at some future time. The work item is deemed to be allocated to that resource and is no longer available to the other resources to which it was offered
* R:start\_s corresponds to the situation where a work item which has been offered to a single resource being started by that resource
* R:start\_m corresponds to the situation where a work item which has been offered to multiple resources is started by one of those resources
* R:start corresponds to the situation where a work item which has been allocated to a single resource being started by that resource.

Six pull patterns have been identified. These divide into two distinct groups. The first three patterns identify the specifics of the actual "pull" action initiated by the resource, with a particular focus on the work item state before and after the interaction. These patterns correspond to the bold arcs in Figure [5](http://www.workflowpatterns.com/patterns/resource/index.php#fig5). The second group of patterns focus on the sequence in which the work items are presented to the resource and the ability of the system and the individual resource to influence the sequence and manner in which they are displayed. The final pattern in this group illustrates the degree of freedom that the resource has in selecting the next work item to execute. These patterns do not have a direct analogue in Figure [5](http://www.workflowpatterns.com/patterns/resource/index.php#fig5) but apply to all of the "pull" transitions illustrated as bold arcs.



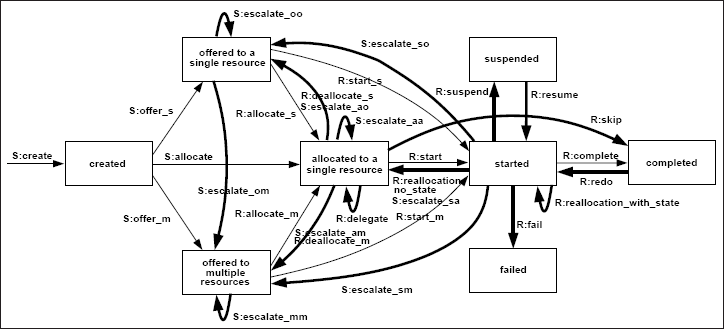
**Figure 5:** Pull Patterns

Note that the distinction between push and pull patterns is identified by the initiator of the various transitions. For the push patterns in Figure [4](http://www.workflowpatterns.com/patterns/resource/index.php#fig4), the state transitions for work items are all triggered by the system, whereas in Figure [5](http://www.workflowpatterns.com/patterns/resource/index.php#fig5) which denotes pull patterns, the transitions are initiated by individual resources. Other characteristics of interest which ultimately lead to additional pull patterns, relate to whether the resource has the ability to reorder their own work sequence or it is determined by the system, and whether a resource can select which work item they wish to commence next from those on its work queue.

1. [Resource-Initiated Allocation](http://www.workflowpatterns.com/patterns/resource/pull/wrp21.php)
2. [Resource-Initiated Execution - Allocated Work Item](http://www.workflowpatterns.com/patterns/resource/pull/wrp22.php)
3. [Resource-Initiated Execution - Offered Work Item](http://www.workflowpatterns.com/patterns/resource/pull/wrp23.php)
4. [System-Determined Work Queue Content](http://www.workflowpatterns.com/patterns/resource/pull/wrp24.php)
5. [Resource-Determined Work Queue Content](http://www.workflowpatterns.com/patterns/resource/pull/wrp25.php)
6. [Selection Autonomy](http://www.workflowpatterns.com/patterns/resource/pull/wrp26.php)

## Detour Patterns

Detour Patterns refer to situations where work item distributions that have been made for resources are interrupted either by the system or at the instigation of the resource. As a consequence of this event, the normal sequence of state transitions for a work item is varied. The range of possible scenarios for detour patterns are illustrated in Figure [6](http://www.workflowpatterns.com/patterns/resource/index.php#fig6).



**Figure 6:** Detour Patterns

There are a number of possible impacts on a work item, depending on its current state of progression and whether the detour was initiated by the resource with which the work item was associated or by the system. These include:

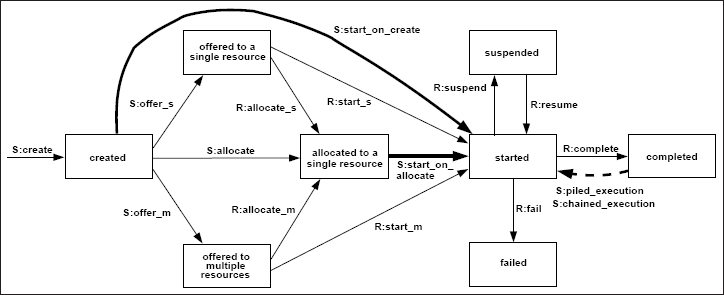
* delegation - where a resource allocates a work item previous allocated to it to another resource
* escalation - where the system attempts to progress a work item that has stalled by offering or allocating it to another resource
* deallocation - where a resource makes a previously allocated or started work item available for offer and subsequent allocation
* stateful reallocation - where a resource allocates a work item that it has started to another resource and the current state of the work item is retained
* stateless reallocation - where a resource allocates a work item that it has started to another resource but the current state is not retained (i.e. the work item is restarted)
* suspension/resumption - where a resource temporarily suspends execution of a work item or recommences execution of a previously suspended work item
* skipping - where a resource elects to skip the execution of a work item allocated to it
* redo - where a resource repeats execution of a work item completed earlier
* pre-do - where a resource executes a work item that is ahead of the current execution point in a case

Each of these actions relate to one or more transitions in Figure [6](http://www.workflowpatterns.com/patterns/resource/index.php#fig6) and corresponds to a specific patterns below.

1. [Delegation](http://www.workflowpatterns.com/patterns/resource/detour/wrp27.php)
2. [Escalation](http://www.workflowpatterns.com/patterns/resource/detour/wrp28.php)
3. [Deallocation](http://www.workflowpatterns.com/patterns/resource/detour/wrp29.php)
4. [Stateful Reallocation](http://www.workflowpatterns.com/patterns/resource/detour/wrp30.php)
5. [Stateless Reallocation](http://www.workflowpatterns.com/patterns/resource/detour/wrp31.php)
6. [Suspension-Resumption](http://www.workflowpatterns.com/patterns/resource/detour/wrp32.php)
7. [Skip](http://www.workflowpatterns.com/patterns/resource/detour/wrp33.php)
8. [Redo](http://www.workflowpatterns.com/patterns/resource/detour/wrp34.php)
9. [Pre-Do](http://www.workflowpatterns.com/patterns/resource/detour/wrp35.php)

## Auto-Start Patterns

Auto-start patterns relate to situations where execution of work items is triggered by specific events in the lifecycle of the work item or the related process definition. Such events may include the creation or allocation of the work item, completion of another instance of the same work item or a work item that immediately precedes the one in question. The state transitions associated with these patterns are illustrated by bold arcs in Figure [7](http://www.workflowpatterns.com/patterns/resource/index.php#fig7).



**Figure 7:** Auto-start Patterns

1. [Commencement on Creation](http://www.workflowpatterns.com/patterns/resource/autostart/wrp36.php)
2. [Commencement on Allocation](http://www.workflowpatterns.com/patterns/resource/autostart/wrp37.php)
3. [Piled Execution](http://www.workflowpatterns.com/patterns/resource/autostart/wrp38.php)
4. [Chained Execution](http://www.workflowpatterns.com/patterns/resource/autostart/wrp39.php)

## Visibility Patterns

Visibility Patterns classify the various scopes in which work item availability and commitment are able to be viewed by resources. They give a indication of how open to scrutiny the operation of a PAIS is.

1. [Configurable Unallocated Work Item Visibility](http://www.workflowpatterns.com/patterns/resource/visibility/wrp40.php)
2. [Configurable Allocated Work Item Visibility](http://www.workflowpatterns.com/patterns/resource/visibility/wrp41.php)

## Multiple Resource Patterns

Up to this point, the focus of this catalogue of patterns has been on situations where there is a one-to-one correspondence between the resources and work items in a given allocation or execution. In other words, resources cannot work on different work items simultaneously and it is not possible that multiple resources work on the same work item. In situations where people are not restricted by information technology, there is often a many-to-many correspondence between the resources and work items in a given allocation or execution. Therefore, it may be desirable to support this using process technology. Here we discuss patterns relaxing the one-to-one correspondence between resources and work items that we have assumed previously.

Let us first consider the one-to-many situation, i.e., resources can work on different work items simultaneously. This is a fairly simple requirement, supported by most systems.

1. [Simultaneous Execution](http://www.workflowpatterns.com/patterns/resource/multiple_resources/wrp42.php)
2. [Additional Resources](http://www.workflowpatterns.com/patterns/resource/multiple_resources/wrp43.php)

Simultaneous Execution is easy to support and contemporary systems support this one-to-many correspondence between the resources and work items in a given allocation or execution. Unfortunately, it is more difficult to support a many-to-one correspondence, i.e., multiple resources working on the same work item. This is a pity since for more complicated activities people tend to work in teams and collaborate to jointly executed work items. Moveover, there is also a lack of consideration for work items that require access to multiple non-human resources (e.g. plant and equipment, fuel, consumables etc.) in order to proceed. Given the limited support of today's PAIS, only one pattern is proposed which implies a many-to-one correspondence.

## Disclaimer

We, the authors and the associated institutions, assume no legal liability or responsibility for the accuracy and completeness of any product-specific information contained in this body of work. All reasonable efforts have been make to ensure that the results presented are, to the best of our knowledge, up to date and correct.

# Exception Handling Patterns

### Download of the exception handling patterns paper:

N. Russell, W.M.P. van der Aalst, and A.H.M. ter Hofstede. [**Exception Handling Patterns in Process-Aware Information Systems**](http://www.workflowpatterns.com/documentation/documents/BPM-06-04.pdf). (PDF, 247 Kb)BPM Center Report BPM-06-04 , BPMcenter.org, 2006.

This body of work presents a classification framework for exception handling in process-aware information systems (PAIS) based on patterns. This framework is independent of specific modelling approaches or technologies and as such provides an objective means of delineating the exception-handling capabilities of specific workflow and process-aware information systems. It is subsequently used to assess the level of exceptions support provided by eight commercial workflow systems and business process modelling and execution languages. On the basis of these investigations, we propose a graphical, tool-independent language for defining exception handling strategies in process-aware information systems.

### Click on one of the following links:

[Introduction](http://www.workflowpatterns.com/patterns/exception/introduction.php)

[A Framework for Exception Handling](http://www.workflowpatterns.com/patterns/exception/framework.php)

[Exception Types](http://www.workflowpatterns.com/patterns/exception/exceptiontypes.php)

[Exception Handling at Work Item Level](http://www.workflowpatterns.com/patterns/exception/exception_workitemlevel.php)

[Exception Handling at Case Level](http://www.workflowpatterns.com/patterns/exception/exception_caselevel.php)

[Recovery Action](http://www.workflowpatterns.com/patterns/exception/recoveryaction.php)

[Characterising Exception Handling Strategies](http://www.workflowpatterns.com/patterns/exception/characterisingstrategies.php)

[Survey of Exception Handling Capabilities](http://www.workflowpatterns.com/patterns/exception/survey.php)

[Considerations for a Workflow Exception Language](http://www.workflowpatterns.com/patterns/exception/considerations.php)

[Related Work](http://www.workflowpatterns.com/patterns/exception/relatedwork.php)

[Conclusions](http://www.workflowpatterns.com/patterns/exception/conclusion.php)

[References](http://www.workflowpatterns.com/patterns/exception/bibliography.php)

[Evaluations](http://www.workflowpatterns.com/patterns/exception/evaluations.php)

## Disclaimer

We, the authors and the associated institutions, assume no legal liability or responsibility for the accuracy and completeness of any product-specific information contained in this body of work. All possible efforts have been make to ensure that the results presented are, to the best of our knowledge, up to date and correct.

# Event Log Imperfection Patterns

### Downloads of the original event log imperfection patterns paper:

S. Suriadi, R. Andrews, M. Wynn and A.H.M. ter Hofstede. [**Event Log Imperfection Patterns: Towards a Systematic Approach to Cleaning Event Logs**](http://www.workflowpatterns.com/documentation/documents/ELP-17-01.pdf). (PDF, 1.29Mb)Information Systems , 64(1), pages 132-150, Elsevier, 2017.

## Introduction

The quality of the data presented to process modeling algorithms is critical to the success of any process mining exercise. Such modelling tools are generally data quality indifferent, i.e. provided the input can be parsed by the tool, its underlying algorithm(s) will generate output. Thus the veracity, realism, precision of the analysis results will be affected by input data quality.

Pre-processing (cleaning) event logs to address quality issues prior to conducting a process mining analysis is a necessary, but generally tedious and ad hoc task. Clearly, a systematic approach to identifying and remedying (commonly occurring) event log data quality issues is desirable.

It turns out that there are many data quality issues commonly found in process mining event logs or encountered while preparing event logs from raw data sources that can be described using patterns. Thus, systematically checking for the signatures of these patterns in an event log can reveal the existence of the associated quality issues and inform appropriate remedial action(s).

In these pages, we present a set of **event log imperfection patterns**, distilled from our experiences in conducting process mining analyses. The patterns relate to data quality issues that have been observed in multiple event logs (or raw data sources) by multiple practitioners/researchers. We do not claim that this collection of patterns is complete (i.e. captures every possible problem that may afflict an event log). We maintain however, that the patterns provide a way to check for some commonly occurring, and from a process mining perspective, high priority issues, which, when systematically and routinely addressed, provides a ground level quality assurance and allows researchers and practitioners to devote effort to uncovering domain and log specific quality issues.

## Background

Event logs used in process mining have a structure designed to allow representation of key attributes of events that occurred over multiple executions of a given process. These event logs may then be presented to a process mining algorithm with the aim of analysing and mapping the process (discovery), determining the extent to which actual execution of the process agrees with the intended execution (conformance) or quantifying various aspects of process behaviour (performance).

Irrespective of the specific type of analysis, it is true that the success of the analysis depends on the quality of the data used in the analysis. In these pages we discuss patterns as a mechanism for describing commonly encountered problems and solutions and introduce our pattern language as a structured method for describing our event log imperfection patterns. We also describe the characteristics of an event log including the key components of an event log and some definitions we will use later when defining our patterns. We also provide some links to existing works that discuss general notions of data quality, and, recognising, that an event log, as a data structure, has some unique features (relating to temporal ordering of records), we describe a data quality framework [(J.C. Bose et al., 2013)](http://ieeexplore.ieee.org/abstract/document/6597227/) suitable for event logs that allows us to easily categorise the event log data quality issues to which each of our patterns relate.

## Patterns

In A Pattern Language: Towns, Buildings, Construction [[AIS77](http://www.workflowpatterns.com/patterns/logimperfection/bibliography.php#AIS77)], Christopher Alexander provides the rationale for using patterns as a means of describing problems (and their solutions) commonly faced in our built environment.

"Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over without ever doing it the same way twice".

Subsequently, patterns as a means of understanding and communicating the characteristics of an apparently chaotic domain have been adopted in other disciplines including business analysis, object-oriented programming, enterprise architecture, workflow functionality and system security.

In these pages we illustrate the use of patterns as a novel way of considering data quality issues that may be encountered in process mining event logs. We discuss the notion at a conceptual level, seeking to strike a balance between generality and precision, with the aim making it accessible to a broad cross-section of the process mining community. We adopt a textual/graphical representation as a convenient, language and implementation independent representation that promotes awareness and makes the patterns easily understandable without overwhelming the reader with technical considerations. Thus, we describe each pattern as a pattern language that uses a set of components:

* **Description**: outline of the pattern and how and where the pattern may be introduced into a log
* **Affect**: consequence of the existence of the pattern on the outcomes of a process mining analysis
* **Data Quality Issues**: type of data error and the event log entities affected by the pattern
* **Manifestation and Detection**: strategy to detect the presence of the pattern in a log
* **Remedy**: how the pattern may be removed from a log
* **Side-effects of Remedy**: possible, undesirable consequences of application of the remedy

## Data Quality Framework

As with other consumed products, the quality of data (used as input for an analysis) can be assessed in terms of it’s fitness for purpose. Data quality is frequently discussed in the literature as a multi-dimensional concept, meaning that multiple facets of a given data object (where data objects include, but are not limited to item, record, database, etc.) need to be considered to determine the quality. Frequently mentioned quality dimensions include, accuracy, relevance, completeness, consistency, reliability, etc. The following are useful links when considering various notions of data quality:

* [ISO/IEC 25012](https://www.iso.org/obp/ui/#iso:std:iso-iec:25012:ed-1:v1:en) standard aims to define a "general data quality model for data retained in a structured format within a computer system".
* [Battini and Scannapieco(2006)](https://link.springer.com/book/10.1007%2F3-540-33173-5) provide a comprehensive general data quality framework.
* [Wang and Strong (1996)](http://www.tandfonline.com/doi/abs/10.1080/07421222.1996.11518099) discuss the importance of data quality to consumers and provide a framework, the dimensions of which are still frequently used today. Importantly, the authors validate their dimensions empirically with data consumer input (i.e. using surveys).

Event logs, as data objects peculiar to process mining analyses, have some unique features that ‘standard’ data quality frameworks do not adequately address, thus making these frameworks inadequate to assess event log quality. There are only a few quality frameworks specifically targeted at event logs.

* [Process Mining Manifesto](https://link.springer.com/chapter/10.1007%2F978-3-642-28108-2_19)(van der Aalst et al.,2011) defines a star-rating (1 to 5) that defines the maturity of the log (readiness for process mining).
* [Bose et al., 2013](http://bpmcenter.org/wpcontent/uploads/reports/2013/BPM-13-02.pdf), categorise data quality issues according to the event attribute affected and whether the data is Missing, Incorrect, Imprecise or Irrelevant and the event log (see Table 1).
* [Mans et al., 2012](https://link.springer.com/chapter/10.1007/978-3-642-36438-9_10), describes log quality as a two-dimensional spectrum with the first dimension concerned with the level of abstraction of the events and the second dimension relating to the accuracy of the timestamp (as measured by granularity, directness of recording and correctness).

We currently adopt the data quality framework proposed by J.C. Bose, R. Mans and W.M.P. van der Aalst as a means of categorising data quality issues that affect event logs.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | **Event Log Entities** | | | | | | | | |
| **Event Log**  **Quality Issues** |  | case | event | relationship | case attrs. | position | activity name | timestamp | resource | event attrs. |
| Missing data | I1 | I2 | I3 | I4 | I5 | I6 | I7 | I8 | I9 |
| Incorrect data | I10 | I11 | I12 | I13 | I14 | I15 | I16 | I17 | I18 |
| Imprecise data |  |  | I19 | I20 | I21 | I22 | I23 | I24 | I25 |
| Irrelevant data | I26 | I27 |  |  |  |  |  |  |  |

Table 1- Manifestation of quality issues in event log attributes

A description/example of each of these data quality issues may be found [here](http://bpmcenter.org/wpcontent/uploads/reports/2013/BPM-13-02.pdf).

## Links to Individual Imperfection Patterns

1. [Form-based Event Capture](http://www.workflowpatterns.com/patterns/logimperfection/elp1.php)  
2. [Inadvertent Time Travel](http://www.workflowpatterns.com/patterns/logimperfection/elp2.php)  
3. [Unanchored Event](http://www.workflowpatterns.com/patterns/logimperfection/elp3.php)  
4. [Scattered Event](http://www.workflowpatterns.com/patterns/logimperfection/elp4.php)  
5. [Elusive Case](http://www.workflowpatterns.com/patterns/logimperfection/elp5.php)  
6. [Scattered Case](http://www.workflowpatterns.com/patterns/logimperfection/elp6.php)  
7. [Collateral Events](http://www.workflowpatterns.com/patterns/logimperfection/elp7.php)  
8. [Polluted Label](http://www.workflowpatterns.com/patterns/logimperfection/elp8.php)  
9. [Distorted Label](http://www.workflowpatterns.com/patterns/logimperfection/elp9.php)  
10. [Synonymous Labels](http://www.workflowpatterns.com/patterns/logimperfection/elp10.php)  
11. [Homonymous Label](http://www.workflowpatterns.com/patterns/logimperfection/elp11.php)