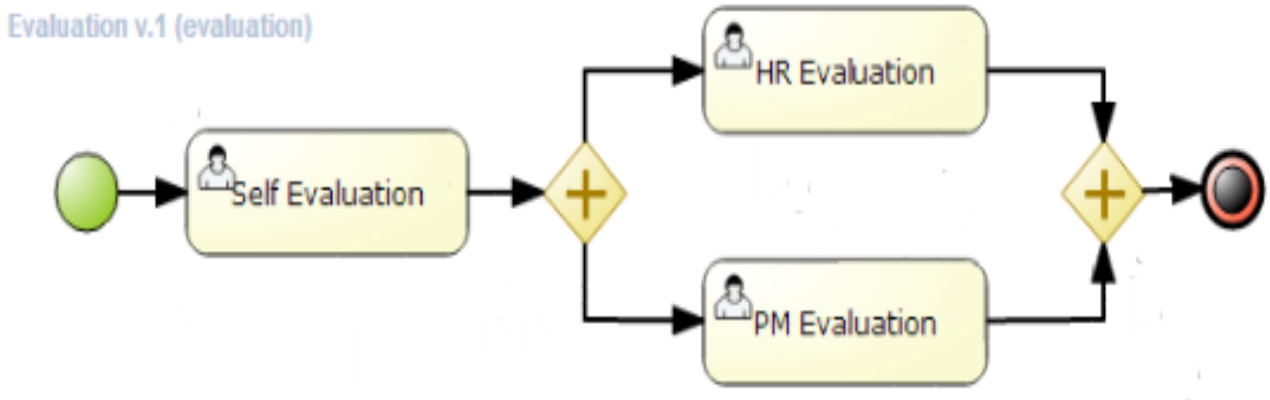
# jBPM & drools

**Business Process Management**
is a
***systematic approach***
to making an organization's
***workflow***
more effective, more efficient
and more capable of
***adapting***
to an ever-changing
environment.

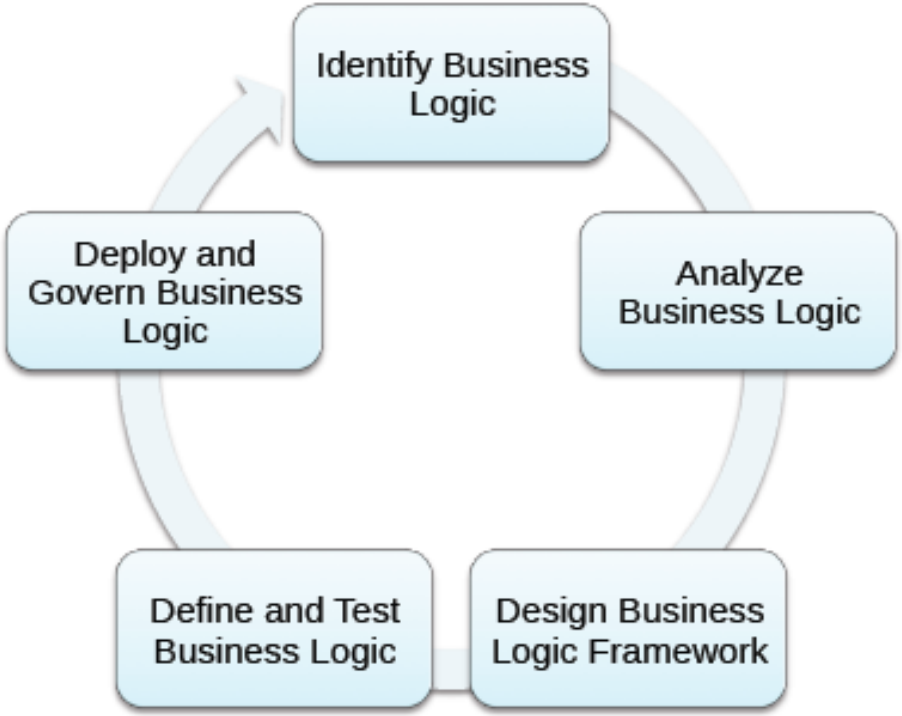
What is Workflow ?

A business process as an activity or set of activities that will accomplish a specific organizational goal.



# Process Overview

# Business Logic Development Process



- ✓ Service Development Iteration – main phases
  - ❖ Identify Business Logic
  - ❖ List Business Processes that are in scope
  - ❖ List associated Business Rules
  - ❖ Analyze Business Logic
  - ❖ Model Business Process
  - ❖ Analyze Rules
  - ❖ Design Business Logic Framework
  - ❖ Design Knowledge Packages
  - ❖ Create Fact Model
  - ❖ Create Domain Specific Languages, Decision Tables,
  - ❖ and / or Rule Templates
  - ❖ Define and Test Business Logic
  - ❖ Add technical implementation details to process definitions
  - ❖ Author and test Rules
  - ❖ Deploy and Govern Rules

- ✓ Project Scope – the scope of each iteration is defined as part of Identify Business Logic.

- ✓ Project Organization – the team should be small, composed of business analysts and IT architects and developers, and collocated to enhance communication.

# Identify Business Logic

## Identify Business Logic – Goal

- ✓ Identify business processes that are to be automated.
  Output: list of business processes

- ✓ Identify business rules associated with these processes.
  Output: list of business rules

## Identify Business Processes – Description

- ✓ Identify Business Processes is an iterative task.
- ✓ At this step, the processes can be captured in a Business Process List (text document that lists each business process by name)

## Identify Business Rules – Description

- ✓ Identifying Business Rules is an iterative task.

- ✓ At this step, the rules can be captured in a Business Rule List (text document that lists each rule)

## Business Rule List

If

the following conditions are met:

Then

perform the following actions:

For example;
Rule Account Replenishment

If

the account balance drops below the replenishment threshold

Then

replenish account by account replenishment amount

# Analyze Business Logic

## Analyze Business Logic – Goal

- ✓ Analyze and model business logic that is intended to be implemented using BPM and Rules technology.

  Model business processes
  Analyze rules

## Analyze Business Logic – Inputs

- ✓ Business Object Model
- ✓ Business Process List
- ✓ Business Rule List

## Analyze Business Logic – Outputs

- ✓ Java Fact Model or Guvnor Fact Model
- ✓ BPMN2 Process Models created in Web Designer or JBoss Developer Studio BPMN2 Visual Editor
- ✓ Business Rule List
- ✓ Business Rules prototyped in Guided Rule Editor or JBoss Developer Studio Drools IDE
- ✓ Guvnor Test Case or JUnit Test for prototyped rules

## Model Business Processes – Description

✓ Each process in the business process list should be drawn using BPMN2 Notation. Emphasis should be on the sequence of tasks, the type of task, and any conditional branching logic.

✓ Model “rule flows” as sub-processes that are “called” from the main process.

## Analyze Business Rules – Description

- ✓ The rules associated with each rule task within each business process should be further analyzed.
- ✓ The Business Rule List should be organized into preliminary rule package at the end of this task, where packages typically contain the rules the execute together within the same transaction and upon the same set of facts.

## Analyze Business Rules – Description

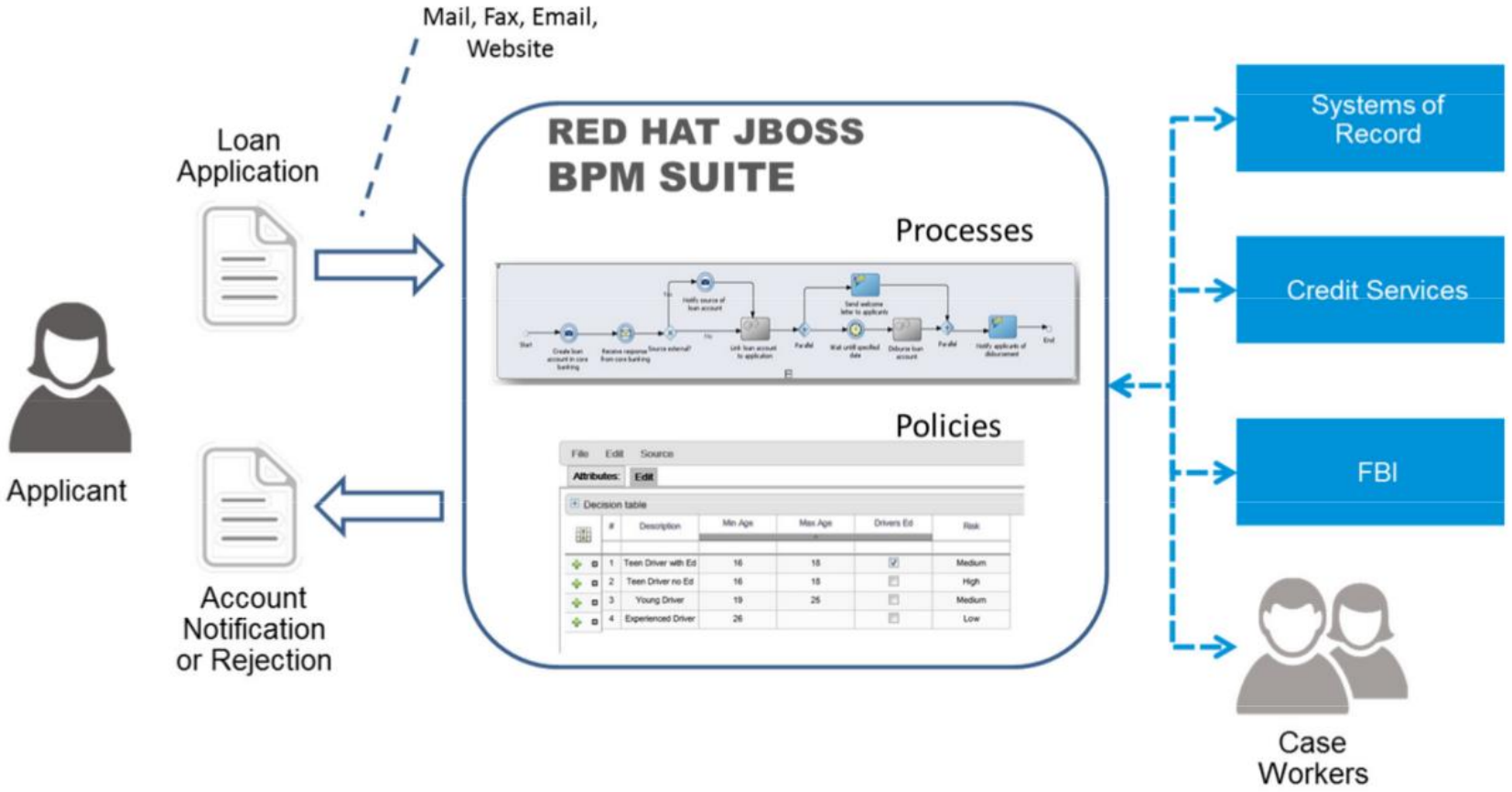
- ✓ Refine the Domain Object Model into a Fact Model.
- ✓ Transform the business rules in the Business Rule List /or Business Rule Table to use terms found in the Fact Model.
- ✓ Refine the rules so that they are atomic, standalone rules.
  i.e., so that they cannot be split into multiple rules without losing meaning or exhibiting great redundancy.

## Analyze Business Rules – Description

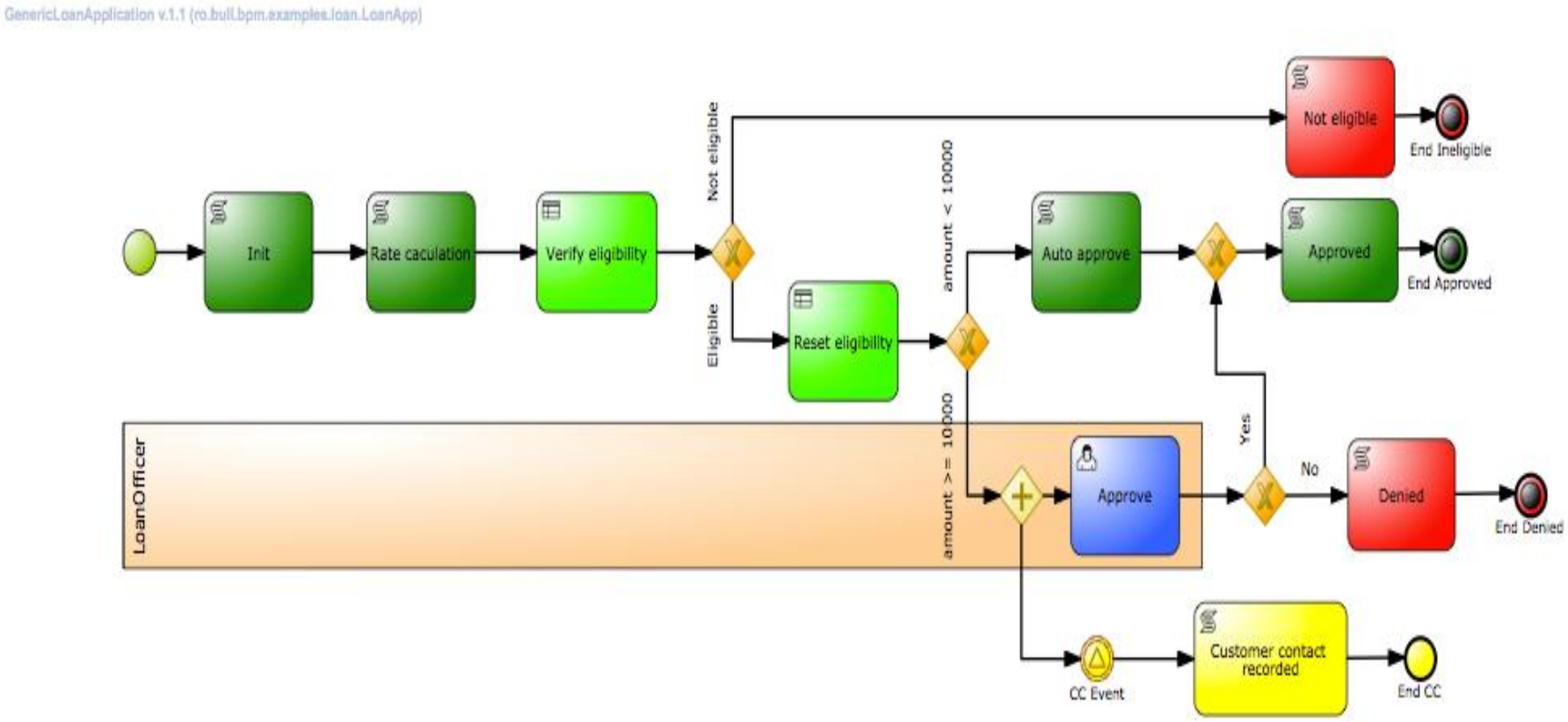
✓ Consider rule patterns, remove redundant rules, resolve overlaps and inconsistencies among rules, and ensure completeness among rules.

✓ Prototype several rules in either the Guided Rule Editor or the JBDS Drools IDE.

# Business Process Use cases




# Loan Processing System




Loan Processing System


Order Processing System

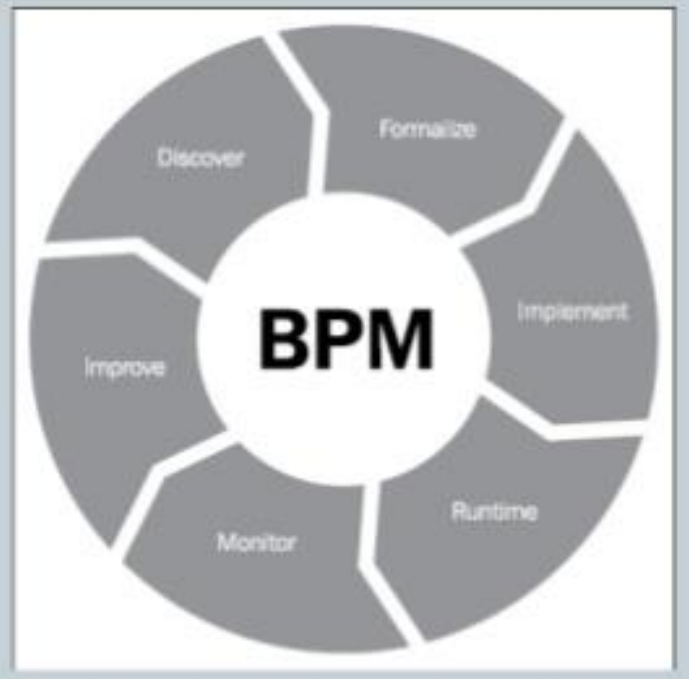
Common processing steps:

- Multiple Steps
- Different actors/players
- External Services
- Sub processes

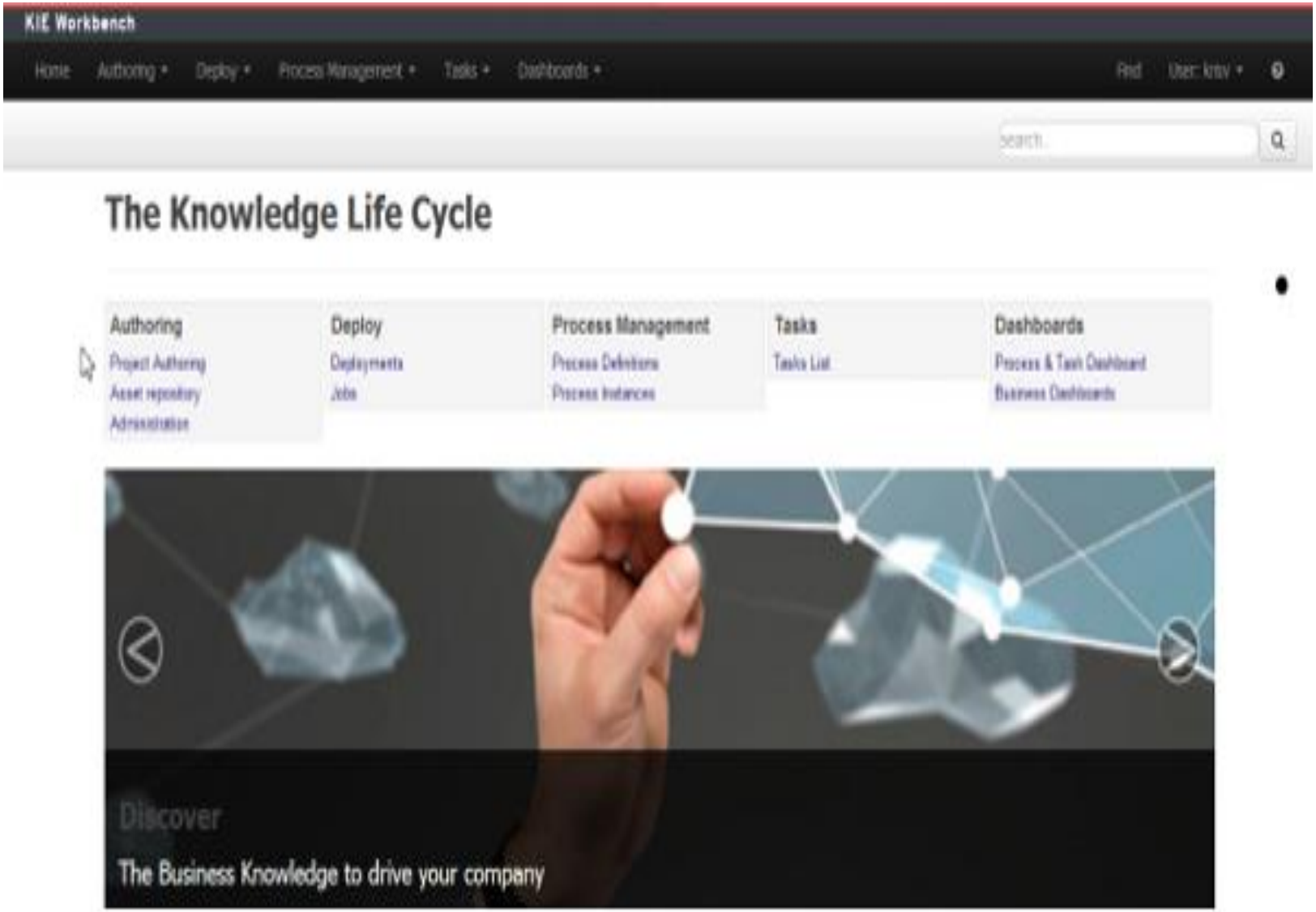
# What is Business Process Management

- Methodology
- Steps to make process better
- To provide us analytical data or reports.
- Improve and Audit Process Adherence.

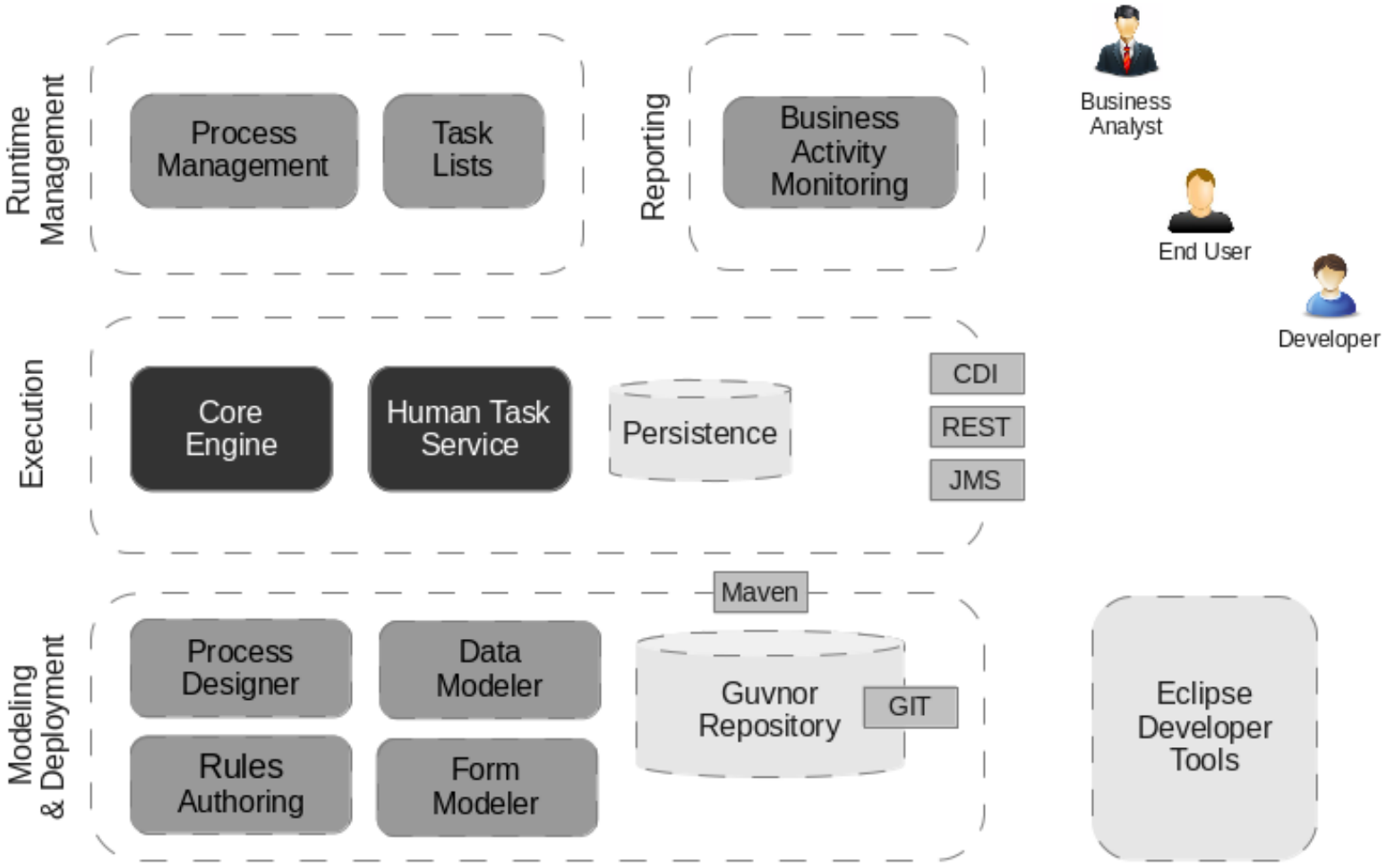
# What is Business Process Management




- Discover- Finding out tasks, people involved in current business process.
- Formalize- BPMN 2.0 is standard for process modeling. business analysts define processes /policies
- Implement- Implement processes to allow to test, validate, and simulate process behavior.
- Runtime- Deploy our business assets (processes, rules, and associated descriptions) to production environment and start training users
- Monitoring- Dashboard-like tools to monitor process execution and performance metrics.
- Improvements- Improvements by analyzing process execution and exceptional situations.




KIE Workbench




# Core Architecture

overview of the different components of the jBPM project.

✓ The core engine is the heart of the project and allows you to execute business processes in a flexible manner.

✓ It is a pure Java component that you can choose to embed as part of your application or deploy it as a service and connect to it through the web-based UI or remote APIs.

✓ An optional core service is the human task service that will take care of the human task life cycle if human actors participate in the process.

✓ Another optional core service is runtime persistence; this will persist the state of all your process instances and log audit information about everything that is happening at runtime.

✓ Applications can connect to the core engine by through it's Java API or as a set of CDI services, but also remotely through a REST and JMS API.

Web-based tools allows you to model, simulate and deploy your processes and other related artifacts (like data models, forms, rules, etc.):

✓ The process designer allows business users to design and simulate business processes in a web-based environment.

✓ The data modeler allows non-technical users to view, modify and create data models for use in your processes.

✓ A web-based form modeler also allows you to create, generate or edit forms related to your processes (to start the process or to complete one of the user tasks).

✓ Rule authoring allows you to specify different types of business rules (decision tables, guided rules, etc.) for combination with your processes.

✓ All assets are stored and managed on the Guvnor repository (exposed through GIT) and can be managed (versioning), built and deployed.

✓ The web-based management console allows business users to manage their runtime (manage business processes like start new processes, inspect running instances, etc.), to manage their task list and to perform Business Activity Monitoring (BAM) and see reports.

✓ The Eclipse-based developer tools are an extension to the Eclipse IDE, targeted towards developers, and allows you to create business processes using drag and drop, test and debug your processes, etc.

**Business process management**

- BPM involves the designing, modeling, executing, monitoring, and optimizing of business processes.

- A specialized software system that helps achieve these objectives completely is called Business Process Management System (BPMS).

# When To Use A BPM Suite?

✓ **Change processes frequently**

A BPMS coupled with a continuous improvement mindset, make a difference when it comes to processes that change frequently.

✓ **Automate and monitor processes across the value chain**

We have to look at it from the customer perspective, and analyze all the steps involved in the business process, including steps that happen outside the oranization--say at the suppliers, the business partners, the dealers, the franchisees, and the customer.

- ✓ **Support cross-functional processes**

Heart of BPM is about tackling end to end business processes, and that means we should look across functional departments, including the white space between functional groups where work gets lost.

- ✓ **Allow business users or customers to monitor the status of requests or work in progress**

- ✓ **Extend the life of legacy apps**

BPMS product can be implemented on top of the application(s) to become the layer that integrates apps or delivers additional value to the workers using the apps.

✓ **Collect work metrics or meet SLAs**

It is possible to monitor work in real time, or look at analyses of work that has already been processed. By setting key performance indicators, BPM suites can become powerful management tools for managers and executives

✓ **Improve inaccurate, inconsistent or laborious manual work**

✓ **Capture process knowledge**

A BPMS product is one way to model the process for process knowledge and also link it to an execution engine that makes it much easier to update the process version as it changes over time.

**Core elements**

The chief constituents of a BPMN diagram, BPMN elements, can be broadly classified into five categories:

**Flow Objects**

These objects define the behavior of a business process

**Data**

This represents the data associated in the business process

**Connecting Objects:**
These objects are used to connect the flow objects to each other

**Swimlanes:**
This is used to categorize the flow objects

**Artifacts:**
These provide additional information about the process

## Flow Objects

Flow objects are the main building blocks of BPMN and represent the concepts that are being modeled.

The notational guidelines dictate what they look like, so that the diagram clearly communicates the same process regardless of who created the diagram or who is viewing it.

Flow objects can be separated into three main areas:

- Events
- Activities
- Gateways

## Events

The standard way to represent an event is with a circle, and it represents that  something happens. You can insert icons inside of the event circles to further describe the type of event.

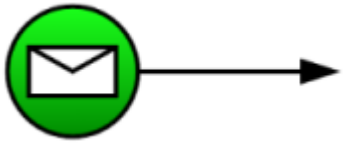
Events can be considered either "catching" or "throwing" information.

A start event acts as the impetus for a process, and although they’re all circles, only the start event has a *single narrow line encompassing the circle*.

The start event can't catch information since it represents a beginning process and no data or actions could have been generated by the process yet.

## Start Event

Start events are exactly what they sound like. Each process must begin with a starting event. If the process is catching information (for example, receiving an email), In the case of a start event type, the event type is always caught.



Start event catching example

An intermediate event represents anything that's happening in the process that is not the start or end event.

The intermediate event is symbolized by *a circle with a double line*. Because intermediate events are anything happening in the middle, they can catch or throw information.

For example, a task could cause an event that throws a signal to another section of the organization. Then, there could be another event waiting to catch that signal. Those would both be intermediate events.

## Intermediate Event

An intermediate event is any event that occurs between a start and an end event.

The intermediate event circle has a double line, and the event can catch or throw information.

Connecting objects indicate the directional flow, like whether the event is catching or throwing.

Users can find event types for intermediate events by using the pop-up context panel that appears when you add a new BPMN shape to the canvas.

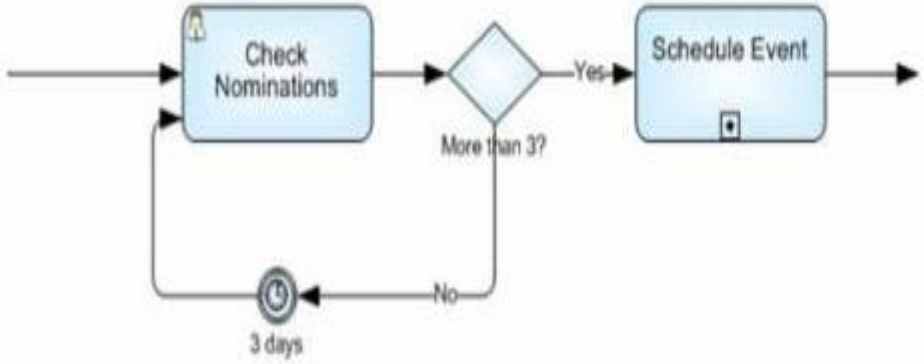
Intermediate event catching example

Intermediate event throwing example

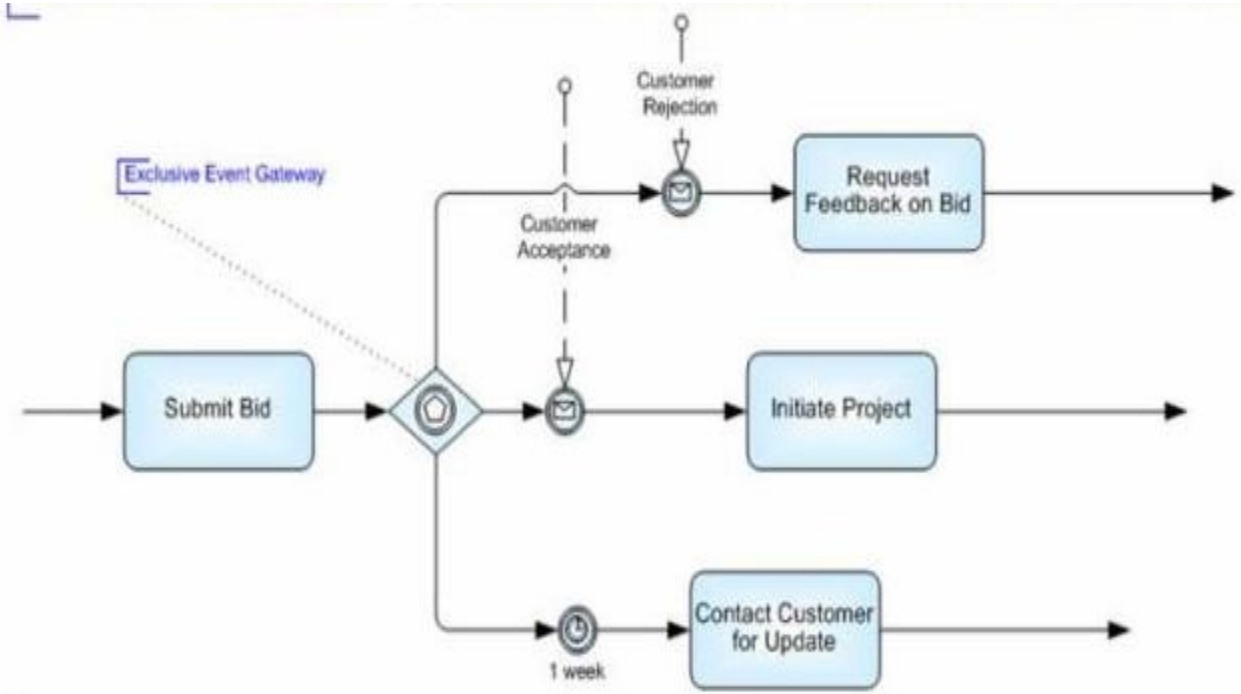
An end event represents the result of the process and is represented by a *circle with a thick line*.

It can only throw information.

# Intermediate Events - Examples




Delay Using a Timer Event - In this example, HR is waiting to see if it has received enough nominations for a training event before scheduling. The timer event introduces a delay of 3 days between each check. Once 4 or more nominations have been received, HR schedules the event.




Event Gateway – The gateway is initiated by which-ever following intermediate event occurs first. In this case, the bidder is expecting a response, either positive or negative, within a week. If no response is received within that time, the bidder contacts the customer for an update.

## End Event

End events are styled with a single thick black line to indicate that it’s an  end event. End events are always thrown because there is no process to catch after the final event.

End event throwing example

# Gateways

✓ Gateways are used to control how Sequence Flows interact as they converge and diverge within a Process

✓ If the flow does not need to be controlled, then a Gateway is not needed

✓ The term "Gateway" implies that there is a gating mechanism that either allows or disallows passage through the Gateway as tokens arrive at a Gateway, they can be:

- merged together on input and/or
- split apart on output as the Gateway mechanisms are invoked.

[split apart on inflow and merged on outflow ?]

✓Gateways, like Activities, are capable of consuming or generating additional tokens, effectively controlling the execution semantics of a given Process

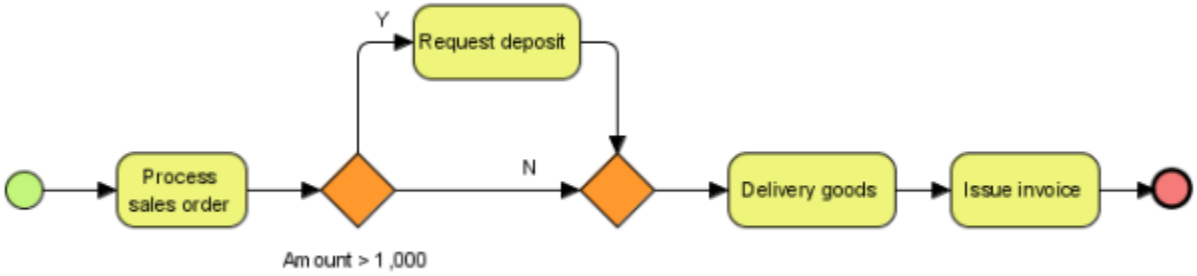
✓The main difference is that Gateways do not represent 'work' being done and they are considered to have zero effect on the operational measures of the Process being executed (cost, time, etc.).

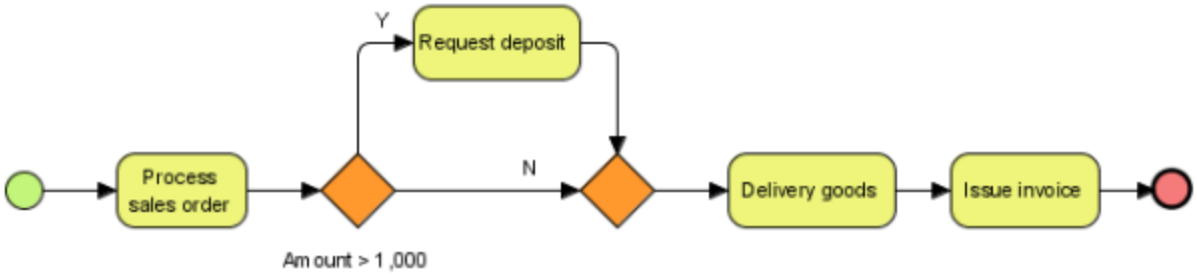
✓Gateways can define all the types of Business Process Sequence Flow behavior:

- Decisions/branching (exclusive, inclusive, complex)
- merging, forking, joining

## Exclusive Gateway

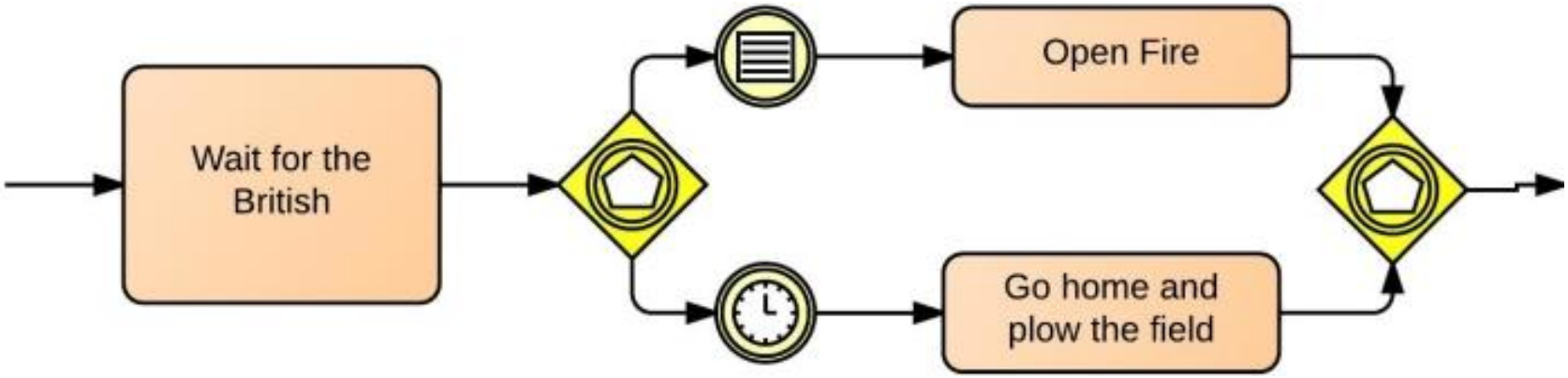
Exclusive gateway is a diversion point of a business process flow. For a given instance of the process, there is only one of the paths can be taken. See below business process diagram. If the order amount is greater than 1,000, we will request deposit from customer. If the order amount is 1,000 or less, we will deliver goods directly. There is only two possible path of the business process and these two possible paths will only execute one but not both.




Y
Request deposit
Process sales order
N
Delivery goods
Issue invoice
Amount > 1,000

**Event-Based Gateway**

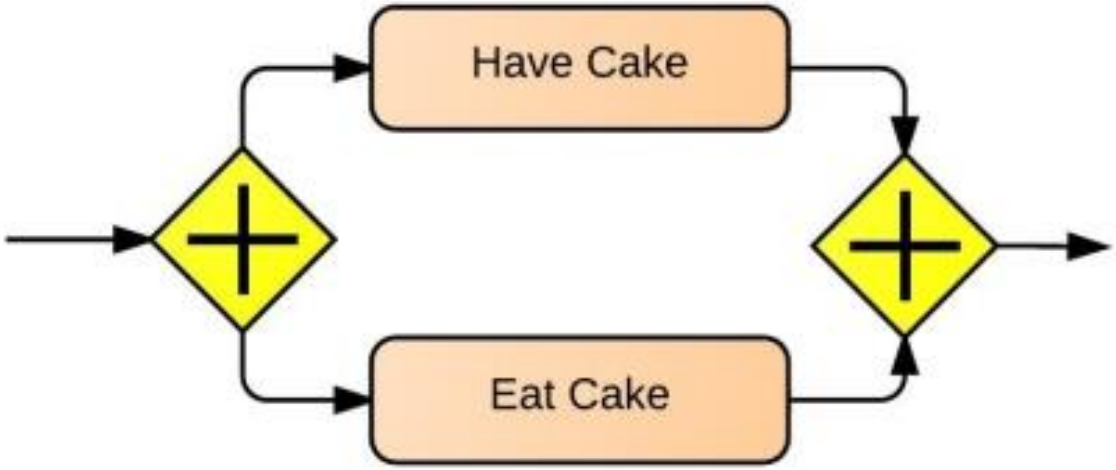
An event-based gateway is similar to a exclusive gateway because both involve one path in the flow. In the case of an event-based gateway, however, you are evaluating which event has occurred, not which condition is being met. An example of a event-based gateway is the decision to hold fire until your soldiers can see the whites of their enemies' eyes.
In this process flow, if a certain amount of time passes without the British coming, the soldiers will go home.



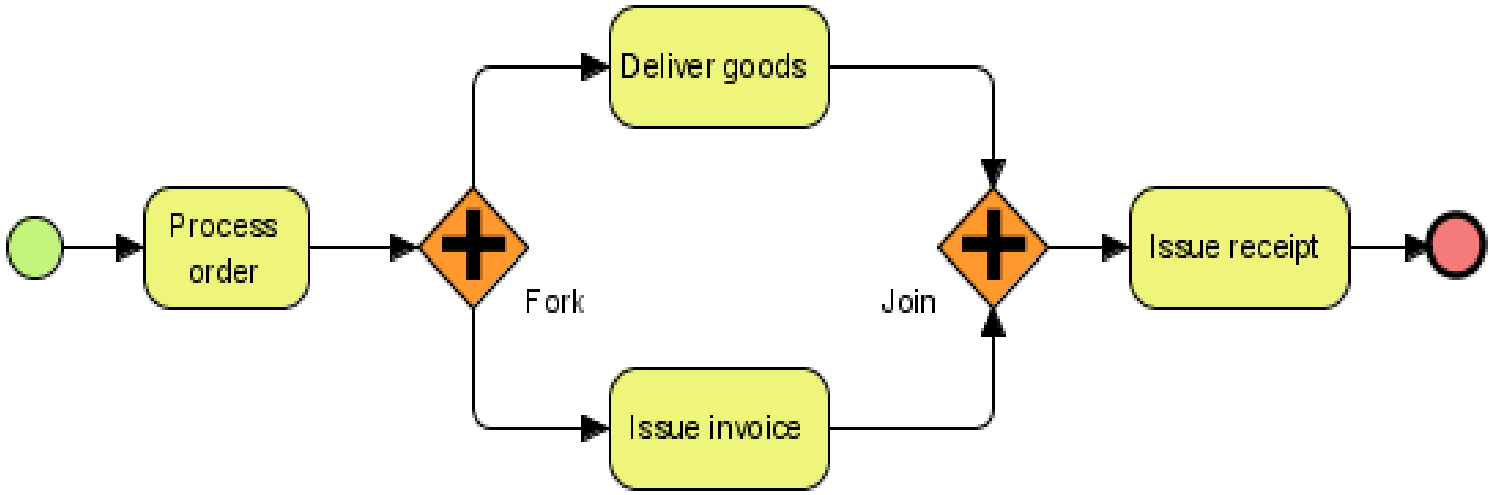
**Parallel Gateway**

A parallel gateway is very different than the previous gateways because you aren't evaluating any condition or event. Instead, parallel gateways are used to represent two concurrent tasks in a business flow.

In the example below, this business process is having its cake and eating it too.

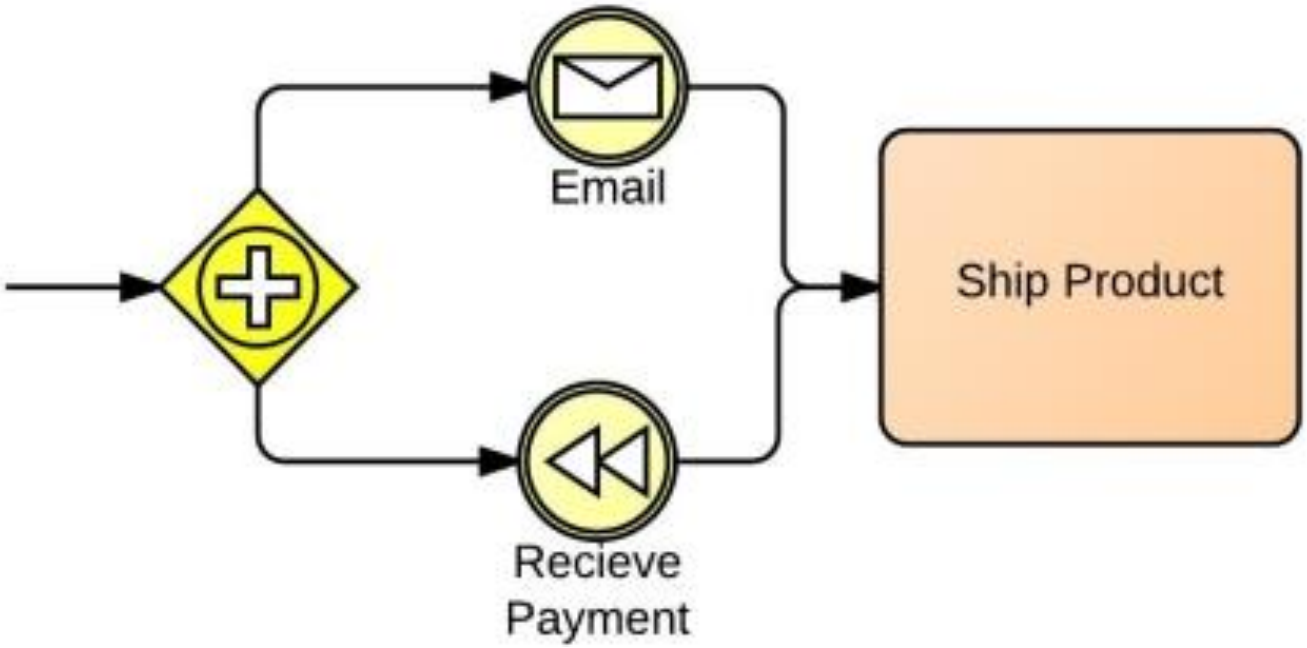


A parallel gateway is used to visualize concurrent execution of activities. We the process arrive to parallel gateway node, the work will split into multiple token and will merge when reach to the joining parallel gateway. Bellow in typical example of parallel gateway.



**Parallel Event-Based Gateway**

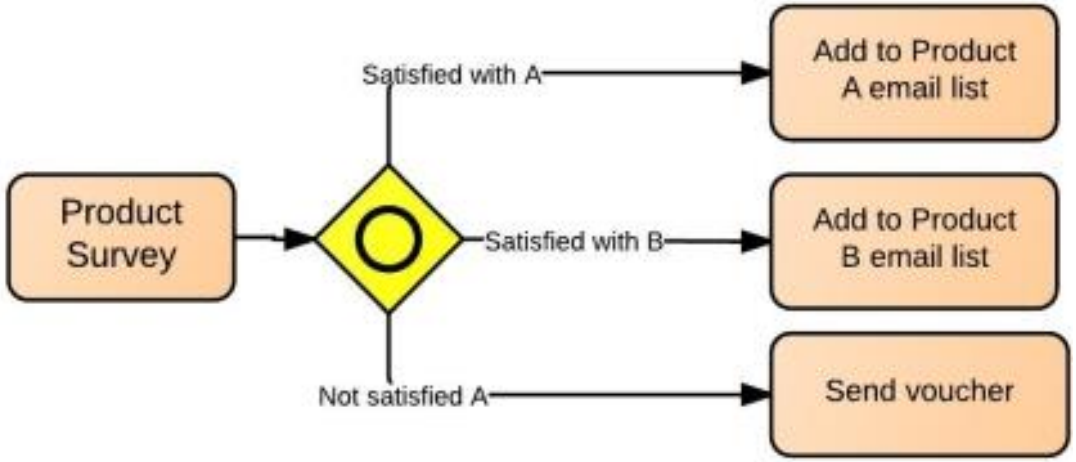
As the name suggest, this gateway is similar to a parallel gateway. It allows for Multiple processes to happen at the same time, but unlike the parallel gateway, the processes are event dependent. You can think of a parallel event-based gateway as a non-exclusive, event-based gateway where multiple events can trigger multiple processes, but the processes are still event dependent.



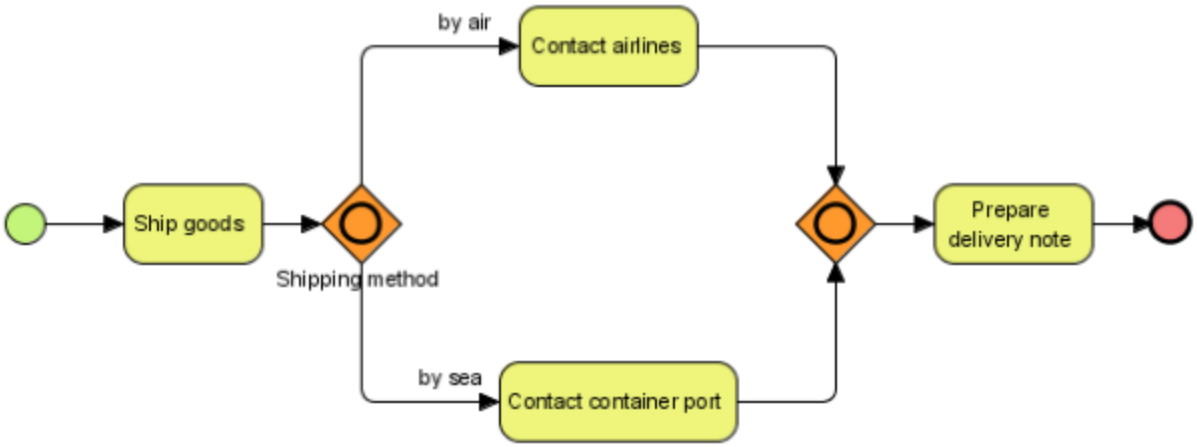
**Inclusive Gateway**

An inclusive gateway breaks the process flow into one or more flows. An example of a inclusive gateway is business actions taken based on survey results. In the example below, one process is triggered if the consumer is satisfied with product A. Another flow is triggered when the consumer indicates that they are satisfied with product B.

A third process is triggered if they aren't satisfied with A. There will be a minimal flow of one and a max of two.



Inclusive gateway is also a division point of the business process. Unlike the exclusive gateway, inclusive gateway may trigger more than 1 out-going paths. Since inclusive gateway may trigger more than 1 out-going paths, the condition checking process will have a little bit different then the exclusive gateway. All out-going conditions will be evaluated no matter has fulfilled out-going flow or not. Below business process diagram show a typical inclusive gateway usage.

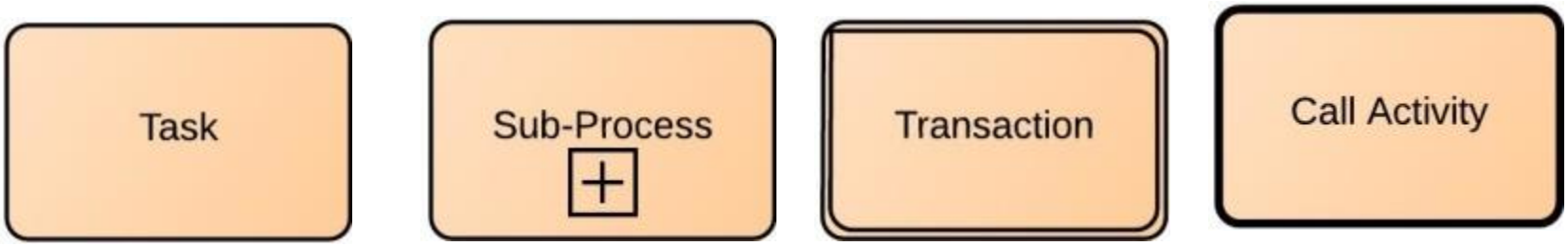


## Complex Gateway

As the name signifies, complex gateways are only used for the most complex flows in the business process. When possible, you should use a simpler gateway to do what needs to be done. If you need multiple gateways to describe the business flow, then that's an ideal case for the complex gateway.

Complex gateways need more descriptive text because you're using words in place of symbols.

## Activities



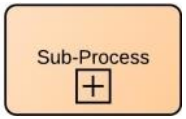

The activity symbol is a rectangle with rounded corners that is used to describe the kind of work being done in that part of the process.

There are four different kinds of activities: tasks, sub-processes, transactions, and call activities.


A task is the most granular level of a process. There are many types of symbols that signify the various task types

Task Types :

Normal Task
Manual Task
Receive Task
Script Task
Send Task
Service Task
User Task
Loop Task
Multiple Instance Task
Compensation Task
Compensation Loop Task
Business Rule Task Type




Even though the task and sub-process shapes are similar (a dotted line Delineates an event sub-process), an event sub-process represents one that was triggered by an event from the parent process.

Sub process Types:

Loop Sub-process
Multi-Instance Sub-process
Compensation Sub-process
Ad hoc Sub-process
Event Sub-process

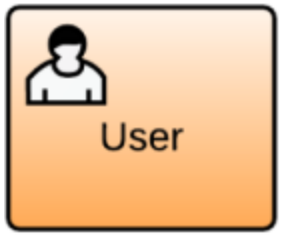

A transaction activity is a specialized sub-process symbol that represents payment processes. All transaction activities are contained by a double line. Transactions are unique because they must ensure that all participants have completed their parts of the transaction before the sub-process can be completed.


A call activity is a global process that is used whenever a certain process needs to be implemented. Whenever the call activity notation is used, control of the process is pushed to the global predefined process.


Script tasks are executed by a business process engine. The script is written in a language that the engine can parse, which is Java/MVEL etc.,


Business rule is an explicit type that was added with BPMN 2.0. Business rules are specific types of services maintained by a business working group, rather than an IT group. The rule shape is used to represent the implementation of a business rule.


A compensation task is a specialized version of a task that only happens when another specific task occurs previously. A compensation loop task is a task that happens over and over in sequential order and involves some sort of compensation.

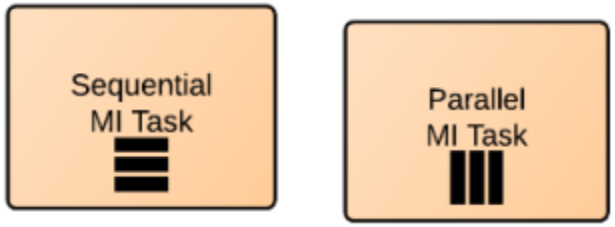



Indicates that the task is being performed by a person and cannot be easily broken down into simpler tasks.


A service task is any task that uses an automated application or web service to complete the task.


A send task is a task that sends a message to another process or lane. The task is completed once the message is sent.


A receive task indicates that the process is relying on an incoming message from a third party. Upon receiving a  message, the task has been performed.

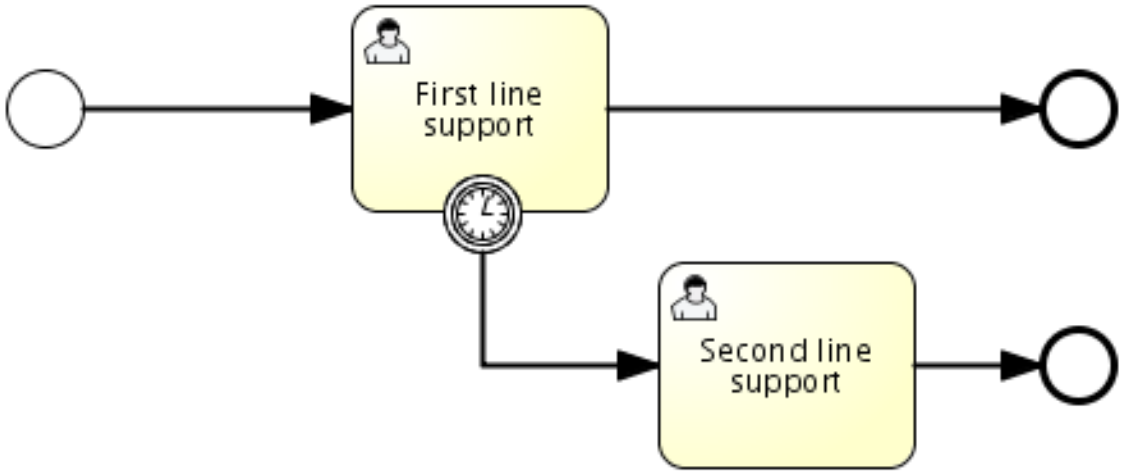


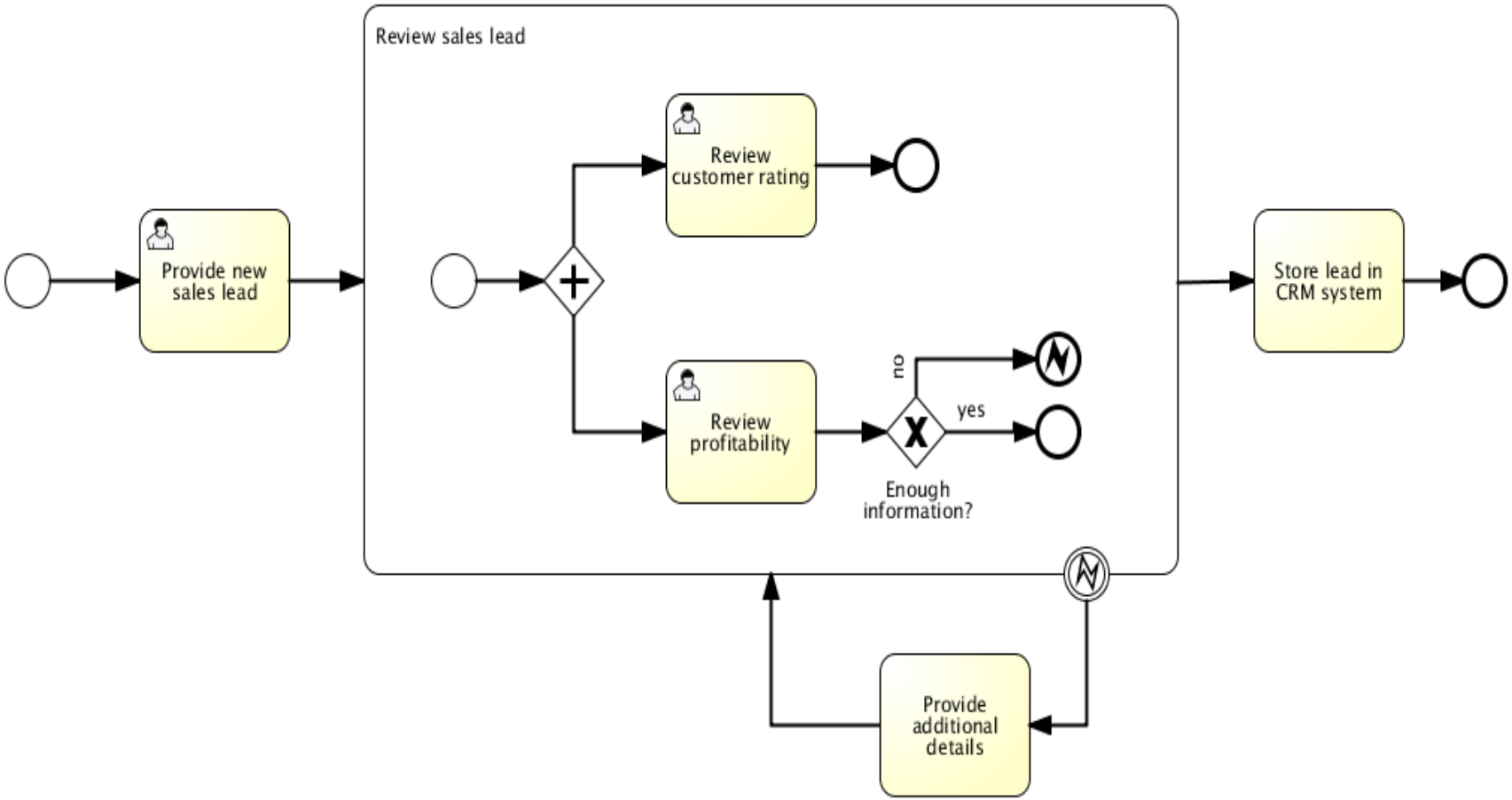

A multiple instance task is a task that happens multiple times. These instances can happen in parallel or sequentially.

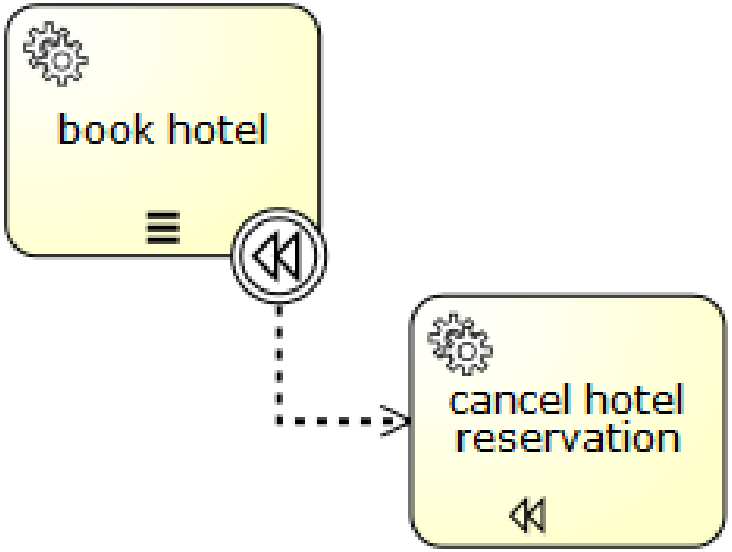
## Boundary Events

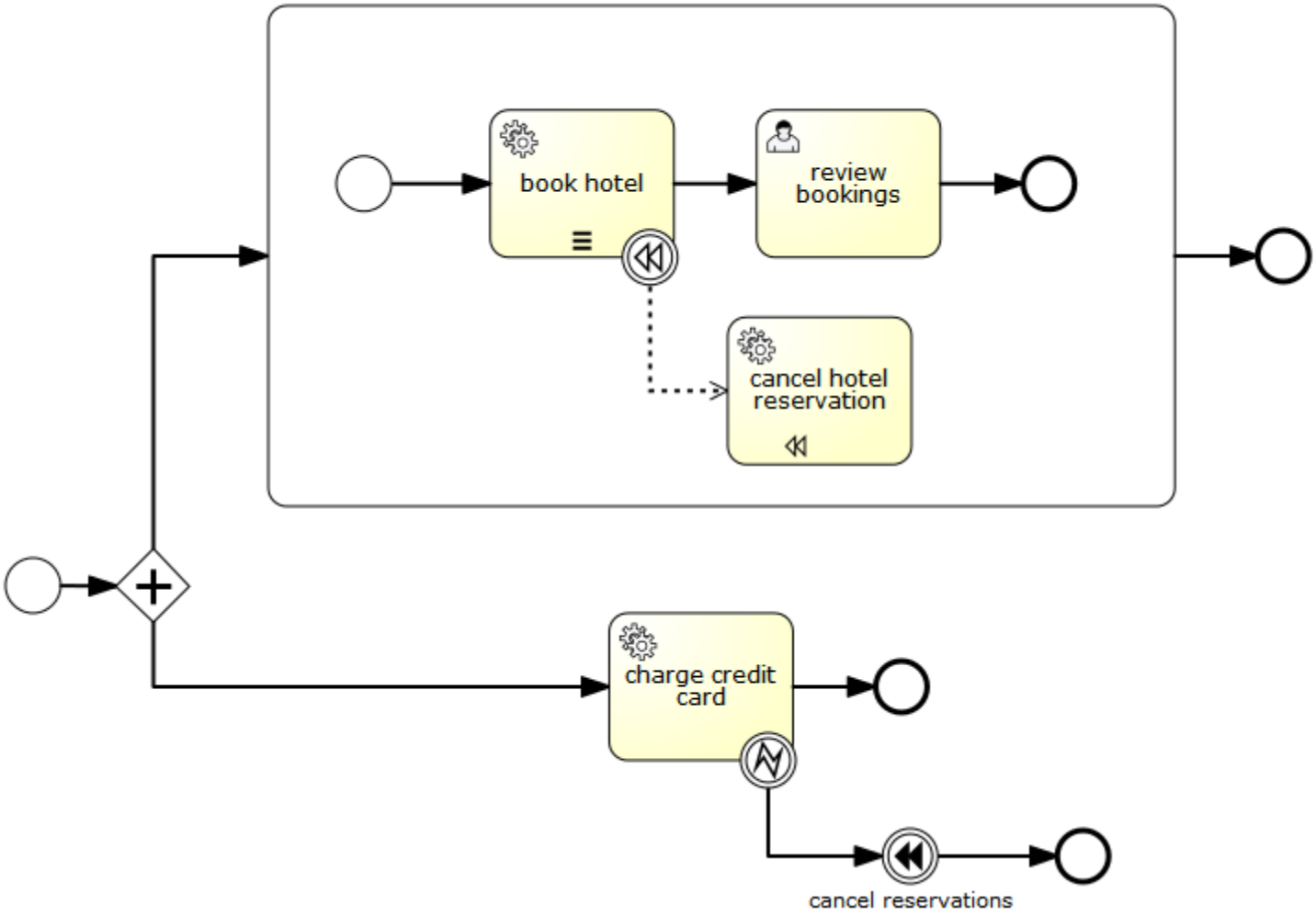
Boundary events are *catching* events that are attached to an activity (a boundary event can never be throwing).

This means that while the activity is running, the event is *listening* for a certain type of trigger. When the event is *caught*, the activity is interrupted and the sequence flow going out of the event are followed.


First line support
Second line support


Review sales lead
Provide new sales lead
Review customer rating
Review profitability
no
yes
Enough information?
Store lead in CRM system
Provide additional details

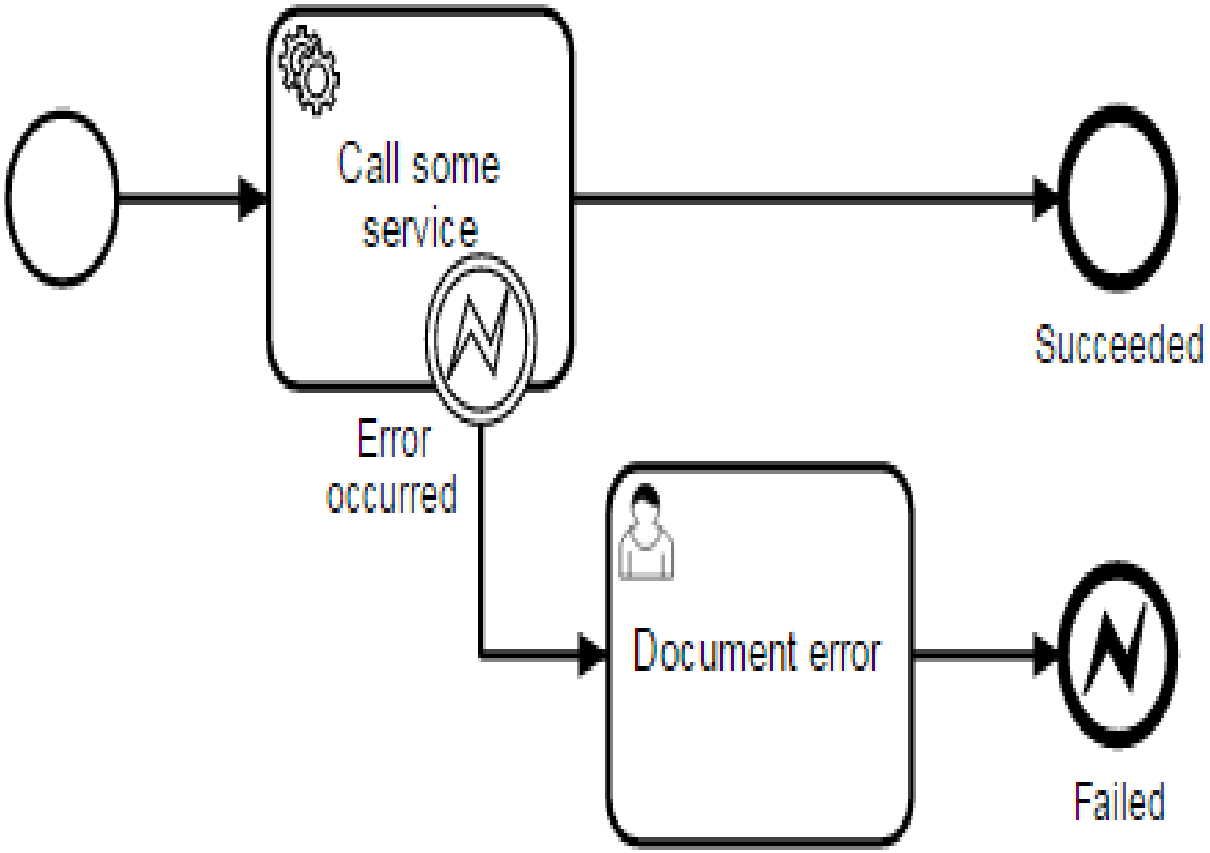

book hotel
cancel hotel reservation


book hotel
review bookings
cancel hotel reservation
charge credit card
cancel reservations

## Error Events

Error events are events which are triggered by a defined error.

SucceededFailedCall some serviceDocument errorErroroccurred


Call some service
Succeeded
Error occurred
Document error
Failed

Business Errors vs. Techincal Errors

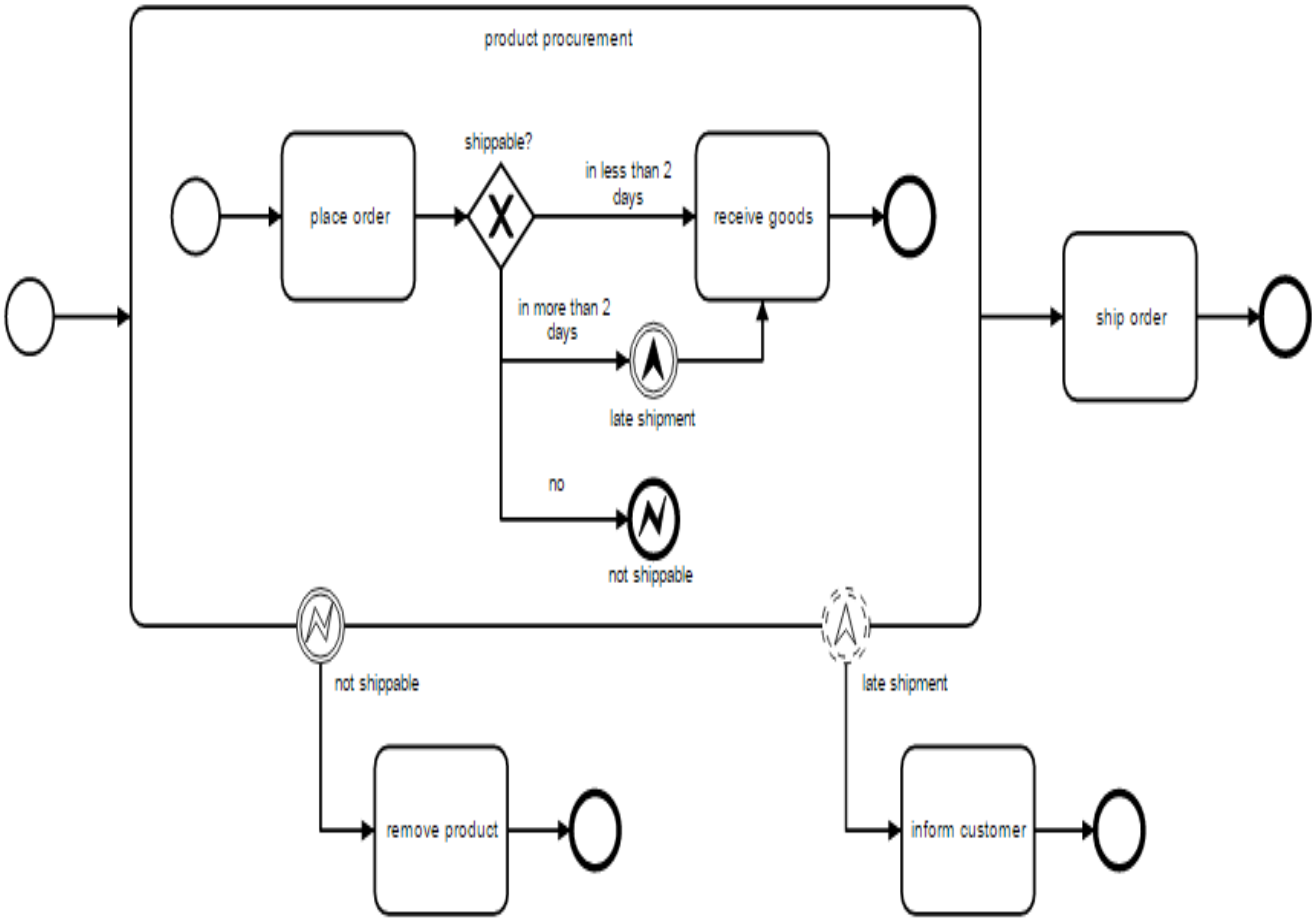
A BPMN error is meant for business errors - which are different than technical exceptions. So, this is different than Java exceptions - which are, by default, handled in their own way.

## Escalation Events

Escalation events are events which reference a named escalation.

They are mostly used to communicate from a subprocess to an upper process.

Unlike an error, an escalation event is non critical and execution continues at the location of throwing.

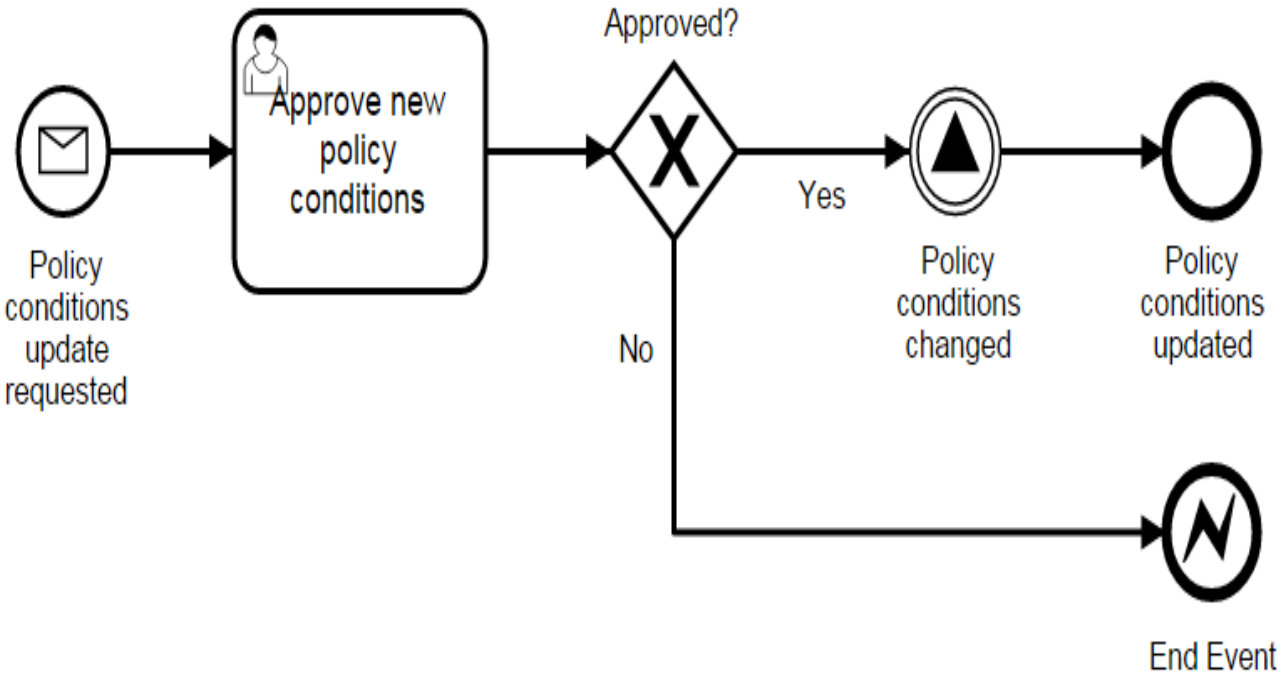

product procurement
place order
shippable?
in less than 2 days
receive goods
in more than 2 days
late shipment
no
not shippable
ship order
not shippable
remove product
late shipment
inform customer

## Signal Events

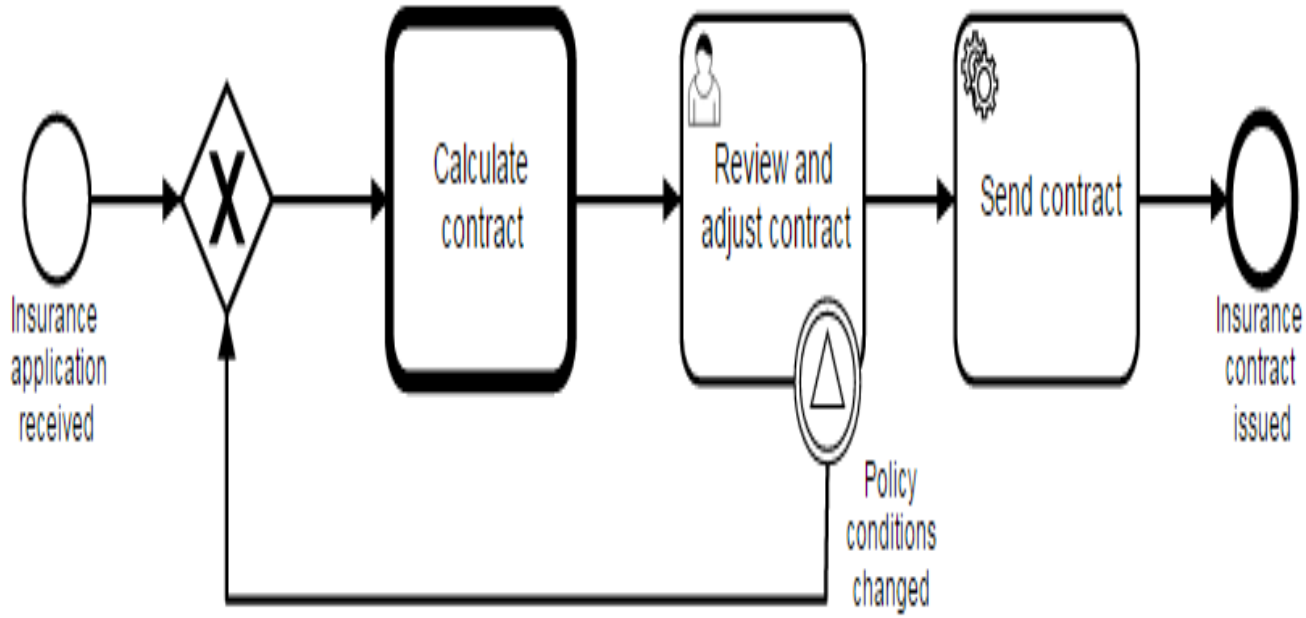
Signal events are events which reference a named signal. A signal is an event of global scope (broadcast semantics) and is delivered to all active handlers.

The following is an example of two separate processes communicating using signals.

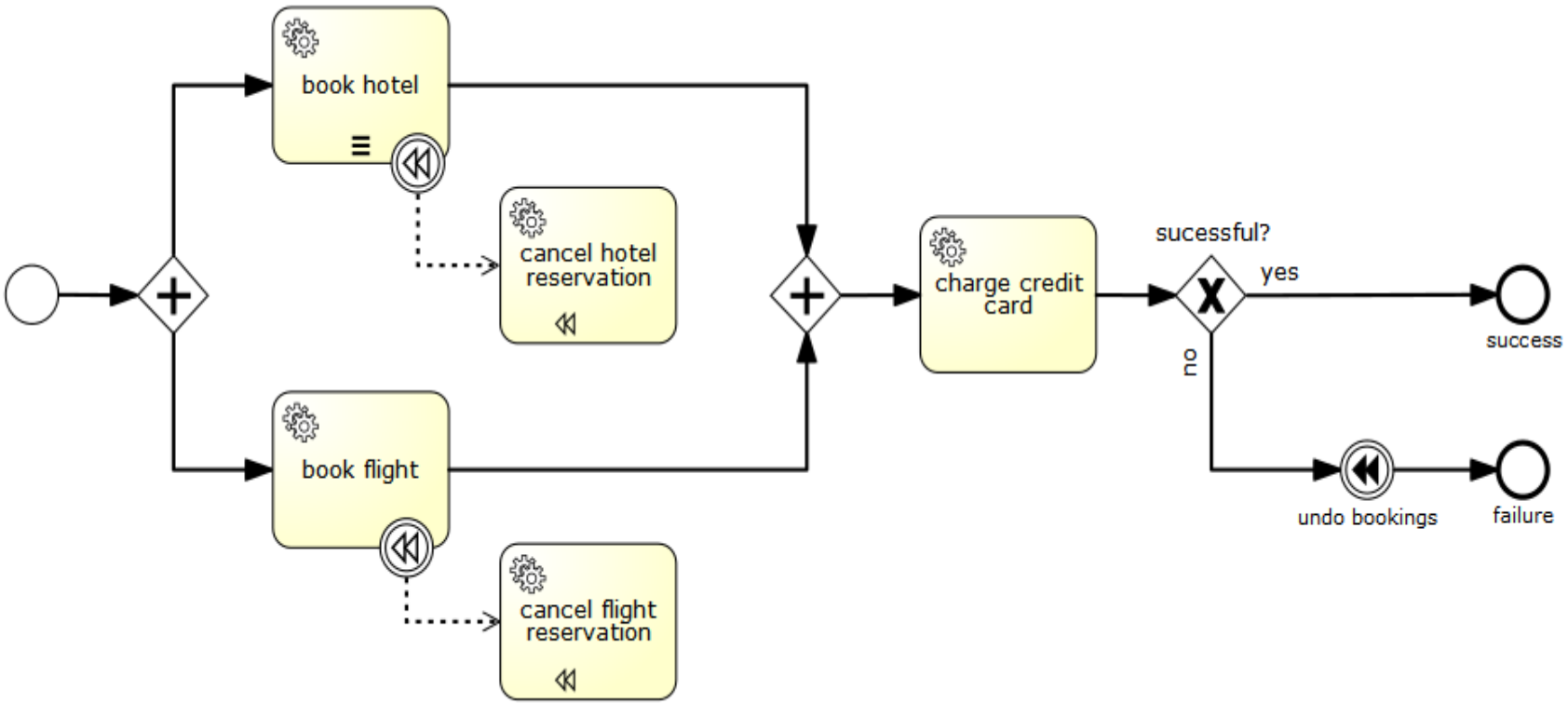
The first process is started if an insurance policy is updated or changed. After the changes have been reviewed by a human participant, a signal event is thrown, signaling that a policy has changed:


Approved?
Approve new policy conditions
Policy conditions update requested
Yes
No
Policy conditions changed
Policy conditions updated
End Event

This event can now be caught by all process instances which are interested. The following is an example of a process subscribing to the event.



# Compensation Event



## Cancel End Event

The cancel end event can only be used in combination with a transaction subprocess.

When the cancel end event is reached, a cancel event is thrown which must be caught by a cancel boundary event.

The cancel boundary event then cancels the transaction and triggers compensation.

## Cancel Boundary Event

An attached intermediate catching cancel event on the boundary of a transaction subprocess, or, for short, a cancel boundary event, is triggered when a transaction is canceled.

When the cancel boundary event is triggered, it first interrupts all executions active in the current scope. Next, it starts compensation of all active compensation boundary events in the scope of the transaction.

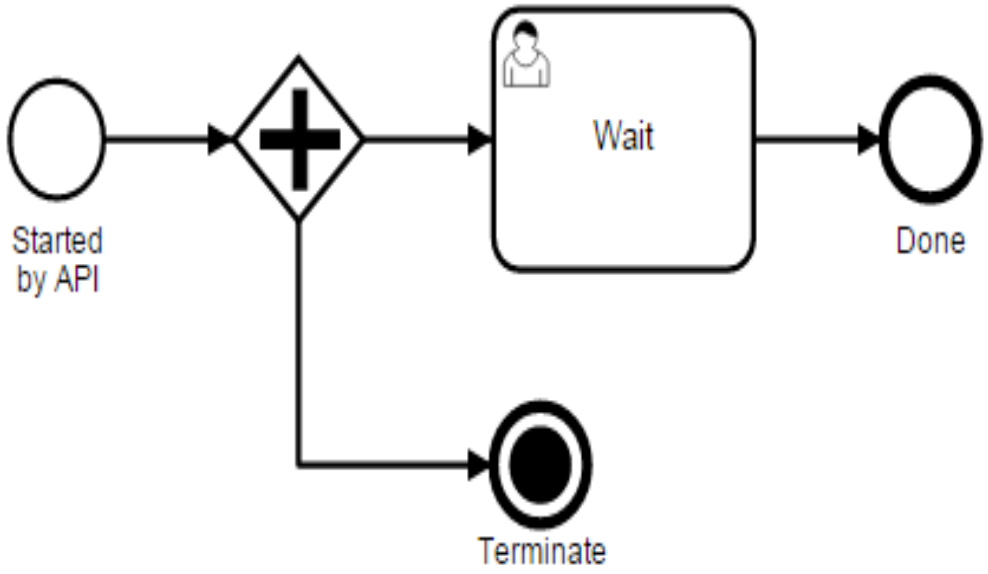
Compensation is performed synchronously, i.e. the boundary event waits before compensation is completed before leaving the transaction. When compensation is completed, the transaction subprocess is left using the sequence flow(s) running out of the cancel boundary event.

## Terminate Events

A terminate event ends the complete scope where the event is raised and all inner scopes.

It is useful if you had a parallel split in your process before and you want to consume all tokens that are currently available immediately.

If you use it on the process instance level, the whole process is terminated. On a subprocess level the current scope and all inner processes will be terminated.


Started
by API
Wait
Done
Terminate

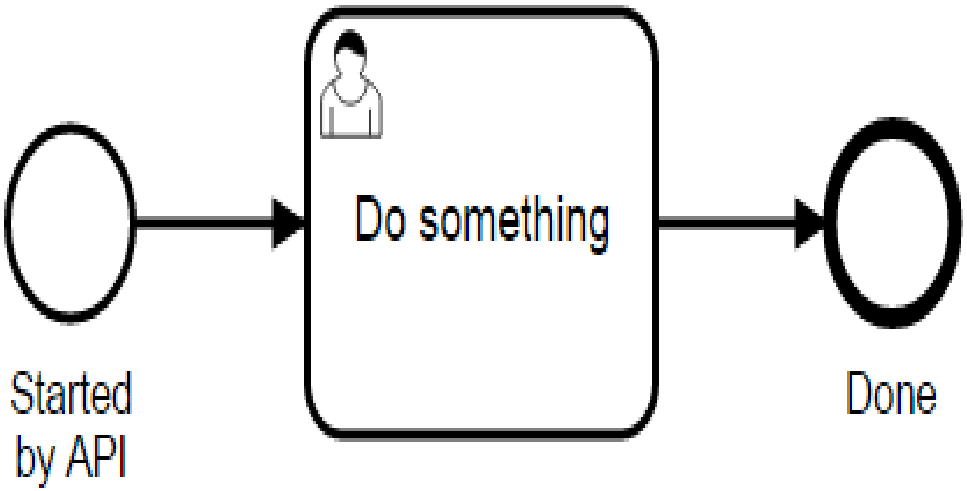
## None Events

None events are unspecified events, also called ‘blank’ events. For instance, a ‘none’ start event technically means that the trigger for starting the process instance is unspecified. This means that the engine cannot anticipate when the process instance must be started.

The none start event is used when the process instance is started through the API by calling one of the startProcessInstanceBy... methods.

```
ProcessInstance processInstance =
runtimeService.startProcessInstanceByKey('invoice');
```

Note: a subprocess must always have a none start event.


Do something
Started
by API
Done

## Interrupting Event vs  non-Interrupting Event

In the case of an Interrupting Event, the activity which was being performed will immediately be canceled.  An Interrupting Event is shown by a solid circle or two solid circles around the event icon depending on whether the event was a start event (one solid circle) or an intermediate event (two solid circles).

In the case of a Non-Interrupting Event, the activity which was being performed will continue in parallel along with the new flow that was initiated by the boundary event.

The current activity will NOT be cancelled (or interrupted).

A Non-Interrupting Event is shown by a dashed circle or two dashed circles around the event icon depending on whether the event was a start event (one dashed circle) or an intermediate event (two dashed circles).

# Remote Java API

Remote Java API

✓ The Remote Java API provides KieSession, TaskService and AuditService interfaces to the JMS and REST APIs.

✓ The interface implementations provided by the Remote Java API take care of the underlying logic needed to communicate with the JMS or REST APIs.

✓ These implementations will allow you to interact with a remote workbench instance

*To interacting with the remote runtime* is to use a RemoteRuntimeEngineFactory
static newRestBuilder(), newJmsBuilder() or newCommandWebServiceClientBuilder()
to create a builder instance.

Use the new builder instance to configure and
to create a RuntimeEngine instance to interact with the server.

# REST API

**REST API description**

Entry point for REST interface is at :

http://localhost:8080/jbpm-console/rest-api.jsp

1. Returns a list of all the available deployed instances in a JaxbDeploymentUnitList instance

http://localhost:8080/jbpm-console/rest/deployment [GET]

2. Returns a JaxbDeploymentUnit instance containing the information (including the configuration) of the deployment unit.

http://localhost:8080/jbpm-console/rest/deployment/{deploymentId}  [GET]

3. Deploys the deployment unit referenced by the deploymentId

[POST] /deployment/ {deploymentId} /deploy

4. Undeploys the deployment unit referenced by the deploymentId

[POST] /deployment/ {deploymentId} /undeploy

***These POST deployment calls are both asynchronous.

# History of REST calls

| URL Template | Type | Description |
|---|---|---|
| /history/clear/ | POST | delete all process, node and history records |
| /history/instances | GET | return the list of all process instance history records |
| /history/instance/{procInstId} | GET | return a list of process instance history records for a process instance |
| /history/instance/{procInstId}/child | GET | return a list of process instance history records for the subprocesses of the process instance |
| /history/instance/{procInstId}/node | GET | return a list of node history records for a process instance |
| /history/instance/{procInstId}/node/{nodeId} | GET | return a list of node history records for a node in a process instance |
| /history/instance/{procInstId}/variable | GET | return a list of variable history records for a process instance |

| | | |
|---|---|---|
| /history/instance/{procInstId}/variable/{variableId} | GET | return a list of variable history records for a variable in a process instance |
| /history/process/{procDefId} | GET | return a list of process instance history records for process instances using a given process definition |
| /history/variable/{varId} | GET | return a list of variable history records for a variable |
| /history/variable/{varId}/instances | GET | return a list of process instance history records for process instances that contain a variable with the given variable id |
| /history/variable/{varId}/value/{value} | GET | return a list of variable history records for variable(s) with the given variable id and given value |
| /history/variable/{varId}/value/{value}/instances | GET | return a list of process instance history records for process instances with the specified variable that contains the specified variable value |

## Task REST calls

| URL Template | Type | Description |
|---|---|---|
| /task/query | GET | return a TaskSummary list |
| /task/content/{contentID} | GET | returns the content of a task |
| /task/{taskID} | GET | return the task |
| /task/{taskID}/activate | POST | activate the task |
| /task/{taskID}/claim | POST | claim the task |
| /task/{taskID}/claimnextavailable | POST | claim the next available task |
| /task/{taskID}/complete | POST | complete the task (accepts query map paramaters) |
| /task/{taskID}/delegate | POST | delegate the task |
| /task/{taskID}/exit | POST | exit the task |
| /task/{taskID}/fail | POST | fail the task |
| /task/{taskID}/forward | POST | forward the task |

| | | |
|---|---|---|
| /task/{taskID}/nominate | POST | nominate the task |
| /task/{taskID}/release | POST | release the task |
| /task/{taskID}/resume | POST | resume the task (after suspending) |
| /task/{taskID}/skip | POST | skip the task |
| /task/{taskID}/start | POST | start the task |
| /task/{taskID}/stop | POST | stop the task |
| /task/{taskID}/suspend | POST | suspend the task |
| /task/{taskID}/content | GET | returns the content of a task |
| /task/{taskID}/showTaskForm | GET | returns a valid URL to the task form to be shown on a client aplication. |

| URL Template | Type | Description |
|---|---|---|
| /runtime/{deploymentId}/process/{procDefID}/start | POST | start a process instance based on the Process definition (accepts query map parameters) |
| /runtime/{deploymentId}/process/{procDefID}/startform | POST | returns a valid URL to the start process form to be shown on a client aplication. |
| /runtime/{deploymentId}/process/instance/{procInstanceID} | GET | return a process instance details |
| /runtime/{deploymentId}/process/instance/{procInstanceID}/abort | POST | abort the process instance |

| | | |
|---|---|---|
| /runtime/{deploymentId}/process/instance/{procInstanceID}/signal | POST | send a signal event to process instance (accepts query map parameters) |
| /runtime/{deploymentId}/process/instance/{procInstanceID}/variable/{varId} | GET | return a variable from a process instance |
| /runtime/{deploymentId}/signal/{signalCode} | POST | send a signal event to deployment |
| /runtime/{deploymentId}/workitem/{workItemID}/complete | POST | complete a work item (accepts query map parameters) |
| /runtime/{deploymentId}/workitem/{workItemID}/abort | POST | abort a work item |

| | | |
|---|---|---|
| /runtime/{deploymentId}/withvars/process/{procDefini tionID}/start | POST | start a process instance and return the process instance with its variables<br><br>Note that even if a passed variable is not defined in the underlying process definition, it is created and initialized with the passed value. |
| /runtime/{deploymentId}/withvars/process/instance/{pr ocInstanceID}/ | GET | return a process instance with its variables |
| /runtime/{deploymentId}/withvars/process/instance/{pr ocInstanceID}/signal | POST | send a signal event to the process instance (accepts query map parameters)<br><br>The following query parameters are accepted:<br><br>• The `signal` parameter specifies the name of the signal to be sent<br>• The `event` parameter specifies the (optional) value of the signal to be sent |

## Serialization: JAXB or JSON

Except for the Execute calls, all other REST calls described below can use either JAXB or JSON.

All REST calls, unless otherwise specified, will use JAXB serialization.

When using JSON, make sure to add the JSON media type ("application/json") to the ACCEPT header of your REST call.

# Transactions

## Transactions

The jBPM engine supports JTA transactions.

It also supports local transactions only when using Spring.

Whenever we do not provide transaction boundaries inside our application, the engine will automatically execute each method invocation on the engine in a separate transaction.

If this behavior is acceptable, we don't need to do anything else. we can, however, also specify the transaction boundaries. This allows use  to combine multiple commands into one transaction.

persistence.xml

```
<property name="hibernate.transaction.jta.platform"
value="org.hibernate.transaction.JBossTransactionManagerLookup"
/>
```

**Application code:**

```
UserTransaction ut = (UserTransaction) new InitialContext().lookup(
"java:comp/UserTransaction" );

ut.begin();

// perform multiple commands inside one transaction

ksession.insert( new Person( "John Doe" ) );

ksession.startProcess( "MyProcess" );

// commit the transaction

ut.commit();
```

## Container managed transaction (EJB/Spring)

To configure this transaction manager following must be done:

Insert transaction manager and persistence context manager into environment prior to creating/loading session

```
Environment env = EnvironmentFactory.newEnvironment();
env.set(EnvironmentName.ENTITY_MANAGER_FACTORY, emf);
env.set(EnvironmentName.TRANSACTION_MANAGER, new
ContainerManagedTransactionManager());
env.set(EnvironmentName.PERSISTENCE_CONTEXT_MANAGER,
new JpaProcessPersistenceContextManag
```