Application Delivery Fundamentals

Parameswari Ettiappan

consulting | technology | outsourcing

Goals

- BPMN
- DMN Training
- BPM & Process Automation
- Process Engine
- Process Applications
- Camunda BPM and Microservices
- Tools & Infrastructure
- Camunda BPM Docker / Open shift Deployment

Business Process Management Platforms

Comindware®

opentext™

nintex

ORACLE

ProcessMaker®
Workflow Simplified

appian

software AG

IBM

Cloud Pak Automation

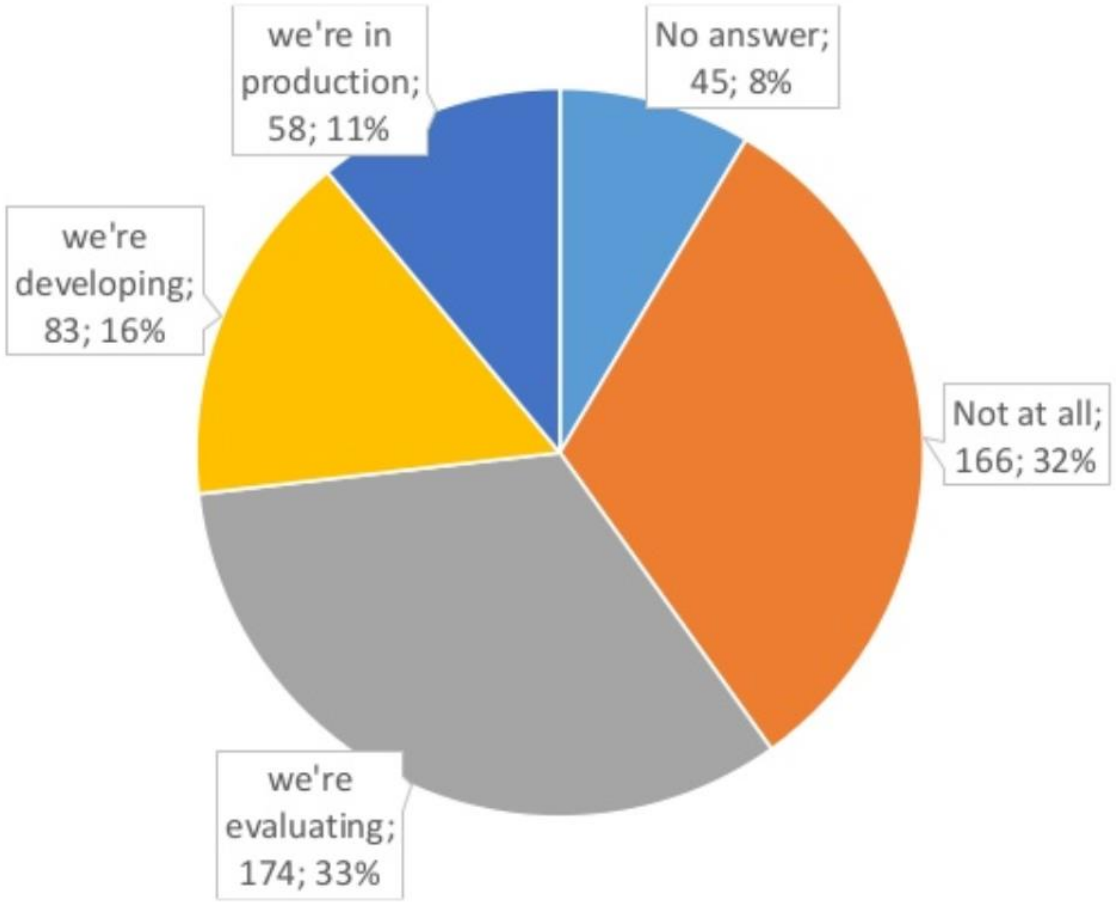
Bizagi

ProcessMaker®
Workflow Simplified

Bonitasoft

BIZFLOW

Do you already use Camunda?



Why do you use Camunda?

Reasons:

- Executes BPMN 2.0
- Best option for Java
- is Open Source

Advantages:

- Stanardized technologies and Skills
- More flexible
- Easier, quicker development
- cost-effective

Also trusted by:

Camunda

- Camunda BPM platform is a flexible framework for workflow and process automation.
- It has a core native BPMN 2.0 process engine that runs inside the Java Virtual Machine.
- It can be embedded inside any Java application and any Runtime Container and does not require a lot of processing power.

Camunda Overview

- Camunda Platform is a complete process automation tech stack with powerful execution engines for BPMN workflows and DMN decisions paired with essential applications for modeling, operations, and analytics.
- With a clear vision to automate any process, anywhere, Camunda is reinventing process automation for the digital enterprise.
- Featuring a developer-friendly, open-source approach, enterprises can use Camunda Platform to breakthrough technological, organizational, and infrastructure...

Pega Overview

- Pega BPM is a low-code application platform.
- It is a world-class Case Management and BPM tool trusted by many major business enterprises.
- It provides an extensive number of features for diverse technical and business solutions.
- The go-to-market (GTM) using Pega software is faster compared to traditional software development processes.
- Pega Robotic Process Automation (RPA) plays a significant role to increase ROI.

Case Management

Case Management		
Case Design	Easy to Design	Easy to use
SLA configuration	Easy to configure during design	Requires multiple steps to configure the SLAs
Automated Decisioning (Process Automation) based on data	PEGA provides various types of rules to achieve the automation and decisioning, this requires a technical expertise	Process automation is possible and provides the workflow engine to design the flows and connect different rules to achieve the automation.
Audit trail of each actions	OOTB and can be customized based on need wi technical expertise	OOTB and can be customized based on need without technical expertise
Activity tracking	OOTB	OOTB
Bulk/Batch processing and case creation	Available in Pega	Available. Technical expertise is required to implement
3rd party integration for external rules and data	Extensive list of connectors available, using the to build integration to 3rd party system require technicalm expertise and requires multiple independent steps to complete an integration.	Comes with industry standard REST, HTTP and SOAP connector to integrate with other 3rd party systems
Automated correspondense	Available. Building correspondense requires technical expertise	Email notification comes OOTB. Other types of notifications can be customized by a technical person
Survey	Available	Not available, but can integrate with 3rd party survey

- Case Management feature is strongest in Pega BPM because it provides a seamless process to capture any new case types, design different stages, steps in each stage, design data model, design user interface, and test it.
- Camunda requires a good level of technical expertise to implement a real enterprise business use case.

Business User Friendliness

Business/End User

Configuring rules	Technical expertise is required	Available but if we are planning to build a complex rule driven solution then Drools can be used as a dedicated rule engine to achieve the same capability
Configuring data	Technical expertise is required	Technical expertise is required
Flexibility to modify work flow	Technical expertise is required	Depends on the complexity of the workflow
Adding New Application users	Easy to Add	Easy to add
Updating user access	Easy to Update	Easy to Update
Defining new access levels	Technical expertise is required	Easy to define
KPI Metrics about the workforce	Some default reports available. Customization is hard and required technical team	Some default reports/dashboards are available. Customization requires technical team assistance
Developing realtime insights/reports	Some default reports available. Customization is hard and required technical team	Some default reports/dashboards are available. Customization requires technical team assistance
Dashboard configuration	Easy to choose from the available reports in a dashboard, but building new insights requires technical expertise	Easy to choose from the availabe reports in dashboards
Work/Assignment routing	OOTB	Has to be developed
Configuring SLAs	Configuring the SLAs are easy but the action to be taken on different SLA requires technical expertise	Configuring SLAs and managing them is little complex than PEGA & Decisions
Accesing & configuring Audit trail	Technical expertise is required	Technical expertise is required
Accessing & Configuring user activities	Technical expertise is required	Technical expertise is required
Maintaining templates for email/Mail correspondence	Technical expertise is required	Business user friendly
Creating New templates for correspondence	Technical expertise is required	Business user friendly

Business User Friendliness

- PEGA BPM comes second place since this requires technical expertise to implement a solution.
- Camunda is a developer-friendly tool and limited.
- Java expertise comes in handy to design and develop an end-to-end enterprise solution.

Architecture/Developer/Designer perspective

Designer/Developer/Architect

	Pega	camunda
Developing workflows	It is easy	It is easy
Developing rules	It is developer friendly	It is developer friendly, for complex business rule driven use cases, it requires Drools to make it more efficient and faster
Developing Integrations with 3rd party system	It can be done by Wizard	Requires technical expertise to develop integrations
Exposing workflow through APIs	It requires good amount of technical work	OOTB services are available but custom developed applications require technical expertise to expose them as a API
Designing Data model	Developer Friendly	Developer friendly
Designing User Interface	Developer Friendly	Developer friendly
Maintaining versions of rules & workflow	Well defined and easy for deployment	Not OOTB, can be implemented using GIT
Designing Access roles and levels	Well defined access control	Simple but multiple steps to setup
Built-in connectors	Standard Connectors available	Standard connectors(REST/HTTP/SOAP)
Email automation	Limited OOTB feature is available	Limited
File process automation (Listener)	OOTB feature is available	Not OOTB
Desinging communications	OOTB to design communications like Email/Mail	Email is OOTB
Channel integrations (SMS, Social media, etc)	Available but licensed separately	3rd Party integration
Realtime reporting	Due to the data structure used in PEGA, realtime reporting is a challenge. Data has to be extracted via BIX to a 3rd party system to build reports	Tableau, Power BI, etc can be connected directly to Camunda data source to do a realtime reporting. Stores data in a Relational database
Data purging/Archiving	Pega OOTB Feature	Custom Agent needed
Batch/Bulk processing design and development	Pega OOTB Feature	Available but limited
Data mapping between UI and process	Developer Friendly	Developer friendly
Developing Portal for external users (Internet facing)	Needs Development effort	Can be integrated via any frontend frameworks
Developing mobile app	Available but licensed separately, also limitted with built-in feature	Not Available
Applying AI and Datascience for decisioning	Available but licensed separately	Available through 3rd party API integrations
Process automation	OOTB	OOTB
Reusing rules and apps for different BUs	It is developer friendly	It depends on the design. Uses BPMN and OOPS concepts, the designer should be well aware of the use case and the design of the app to make it more reusable

Architecture/Developer/Designer perspective

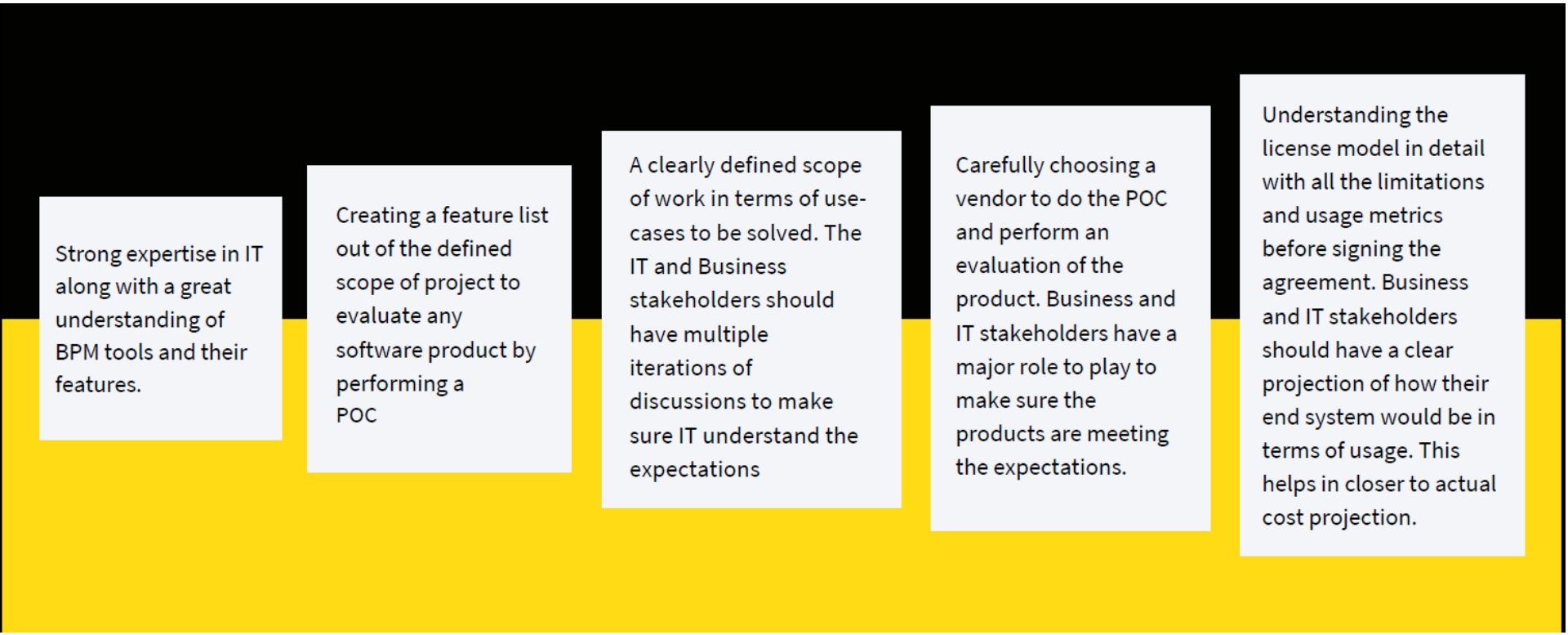
- Pega BPM, though it has extensive features provided, it requires individual expertise and multiple levels to manage a project.
- It adds complexity which in turn needs a developer to invest more time in learning.
- Developing PEGA BPM resources would be much costlier than Camunda.
- Camunda would take 2nd place as compared to PEGA BPM.

General Features

General Features

	Pega	camunda
Flexibility in BPM	95%	85%
Flexible BRE(Business Rules Engine)	90%	70%
Configuring External authentication	OOTB	OOTB
Flexible Authorization	Well defined	Multiple steps involved but manageable
Branding the process for different Business Units	Well defined	Provides the platform to achieve
Branding the templates of communication	Its developer Friendly	Developer friendly
External Portal	Need Development effort	Integrates via API with any frontend frameworks
Realtime reporting	Due to the data structure used in PEGA, realtime reporting is a challenge. Data has to be extracted via BIX to a 3rd party system to build reports	Tableau, Power BI, etc can be connected directly to Camunda data source to do realtime reporting. Data is stored in a Relational database
Custom feature development	Java Code	Java
Robotics Process Automation	Available but licensed separately	Not OOTB
Industry Solutions (Healthcare, Finance, Banking, etc)	Multiple industry solutions available	Not Available
AI driven decisioning	Available but licensed separately	It can be achieved through Third party Integration
Mobile app development	Available but licensed separately	Not Available
Flexible in re-using the rules and workflow	Yes - But it would be tightly coupled with the code	Yes, little harder than Decisions and PEGA
Bulk process or Batch process automation	Pega OOTB Feature	Available but limitted
Hosting	On-Prem and Cloud and Container Managed (Kubernets, etc)	On-Prem and cloud
Pricing	Defined by PEGA team per enterprise, limitations are applied based on the usage of the software mostly per case licensing model	Open Source & Enterprise
Documentation	Detailed PEGA academy with Rich training contents available	Compare to PEGA Camunda requires more detailed guides on documentations
Training	PEGA Academy, University Academy Program provides the candidates the opportunity to learn PEGA	Through Partner program and designated training programs and Self-Learning videos

A Successful Investment in BPM Needs



Camunda Sample Customers

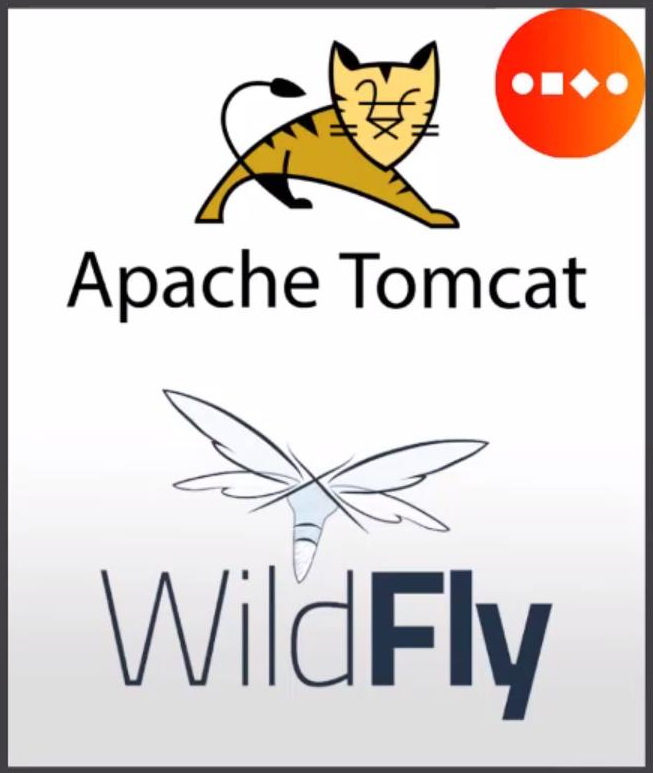
- 24 Hour Fitness, Accruent, Allianz Indonesia, AT&T Inc., Atlassian, CSS Insurance, Deutsche Telekom, Generali, Provinzial NordWest Insurance Services, Swisscom AG, U.S. Department of Veterans Affairs...

Pega Sample Customers

- The State of Maine, ANZ, Coca-Cola, Department for Environment, Food & Rural Affairs (DEFRA), ME, New South Wales, OptumRx, Texas Department of Transportation, UnitedHealthcare Medicare & Retirement.

Camunda

How to Run Camunda



OR

5

CAMUNDA

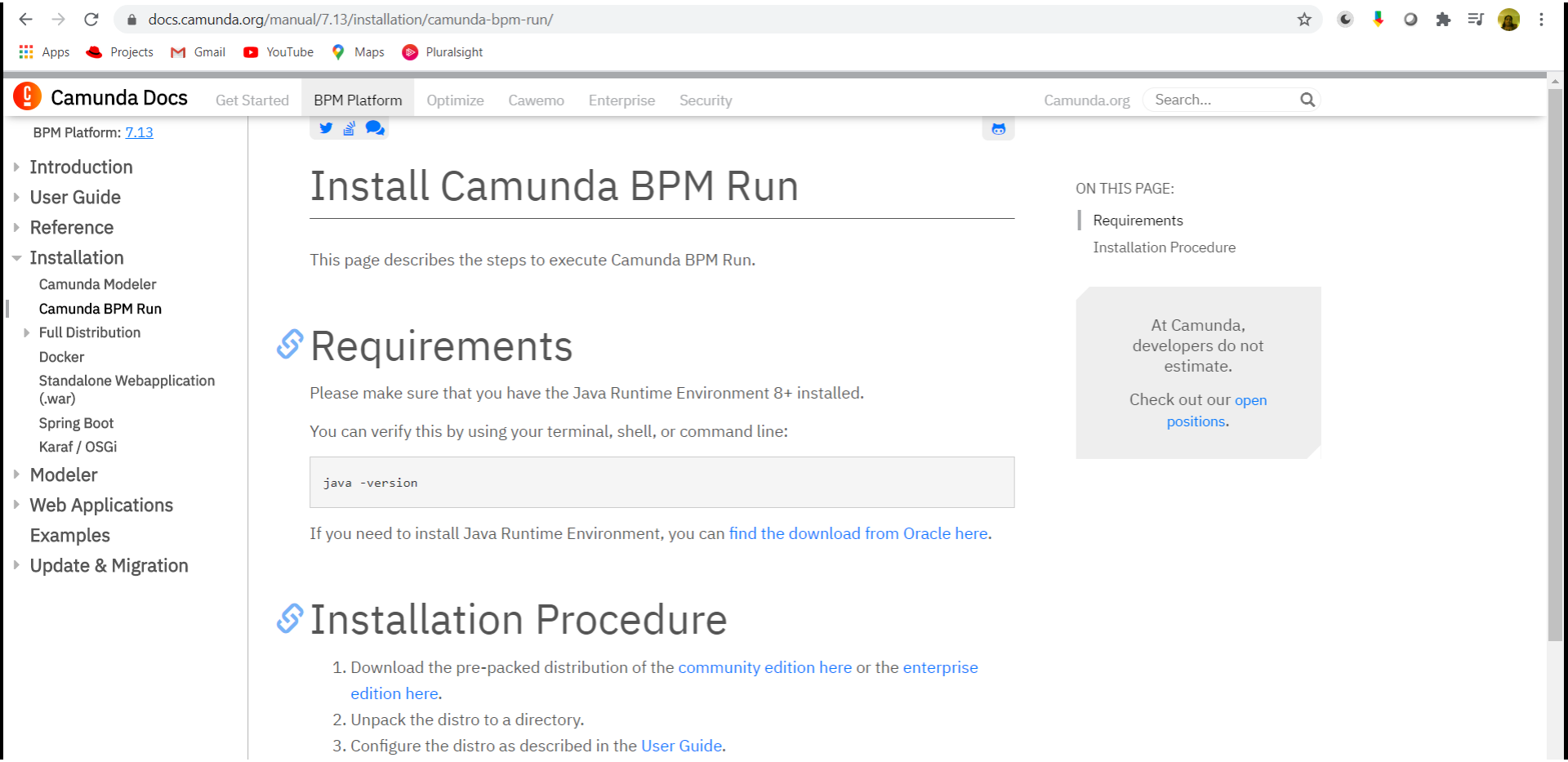
Camunda

Running Camunda BPM Run

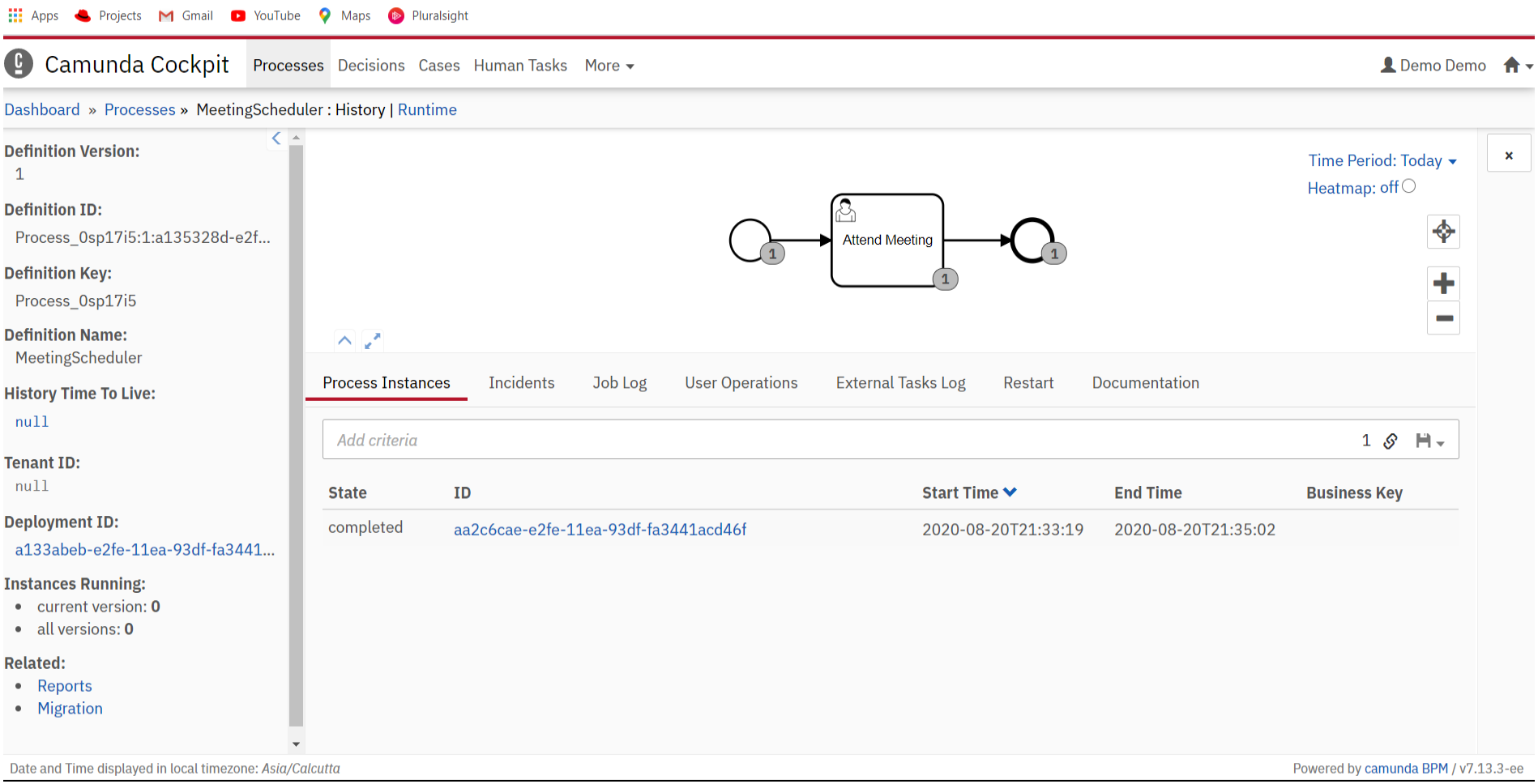
- Self Contained Distribution
- Fast Startup
- Easy Configuration
- Easy to Customize
- No need to know any Java.

Camunda BPM Run

Camunda Installation

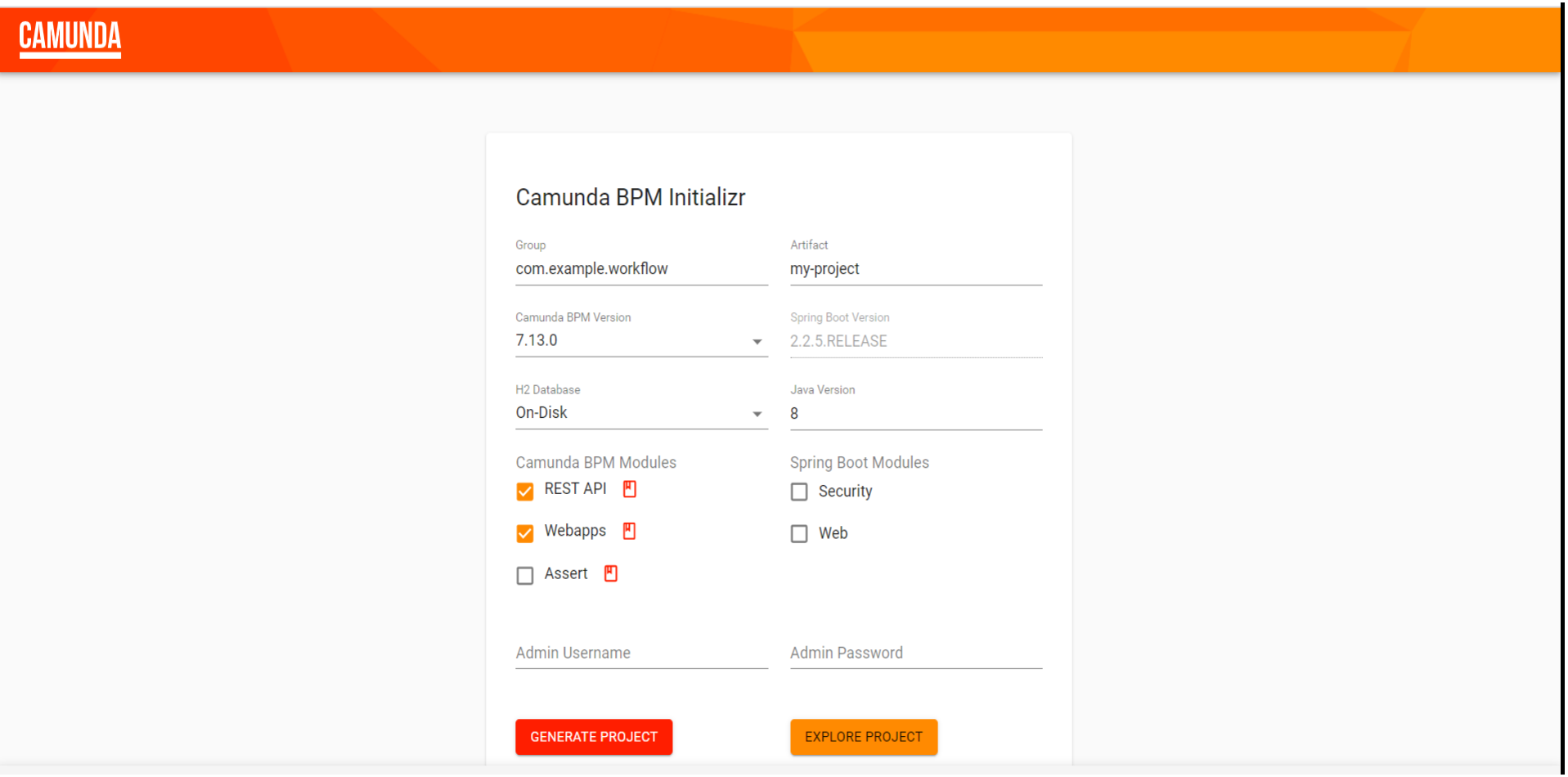


Camunda Installation

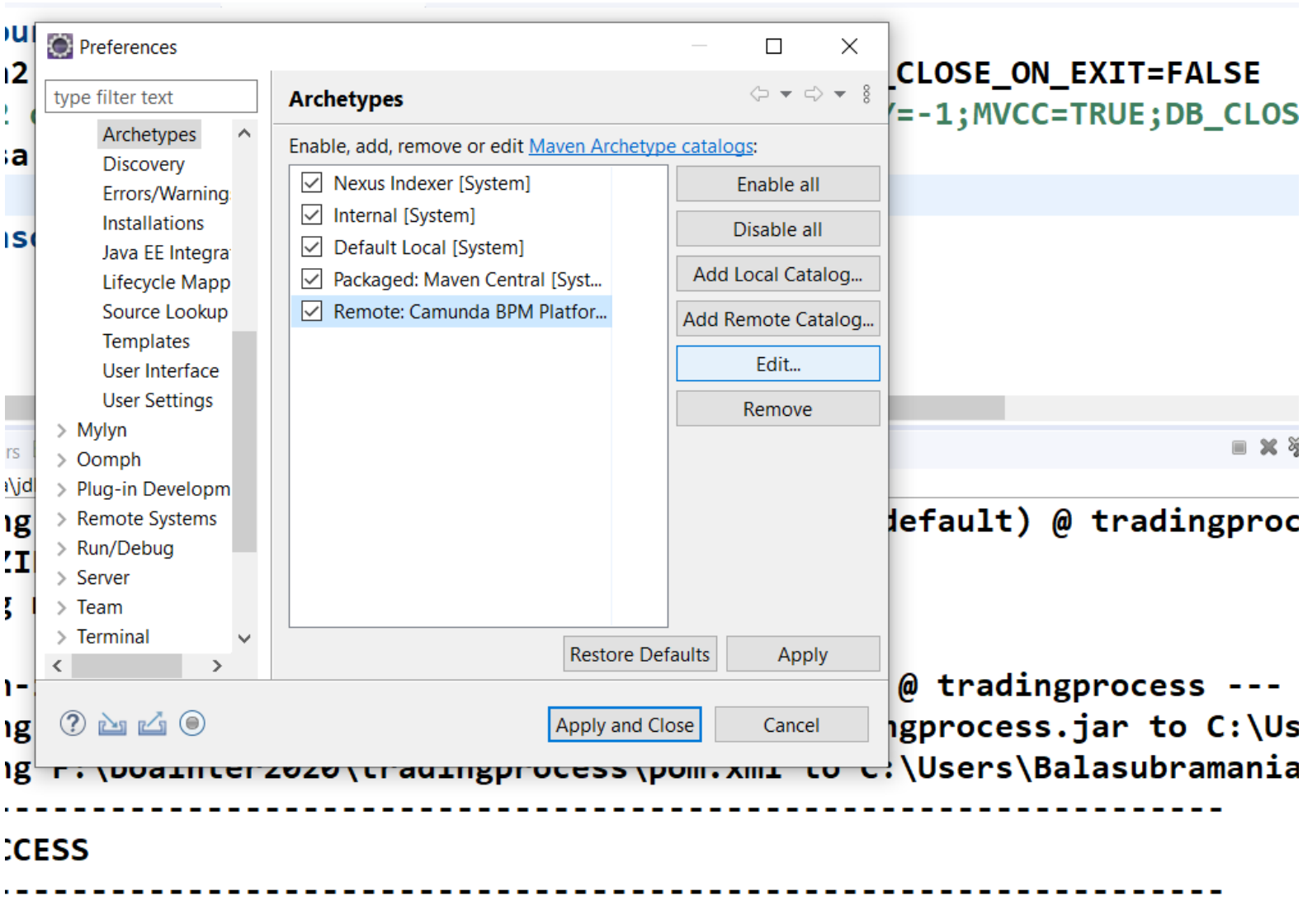


https://start.camunda.com/

Spring Boot Integration

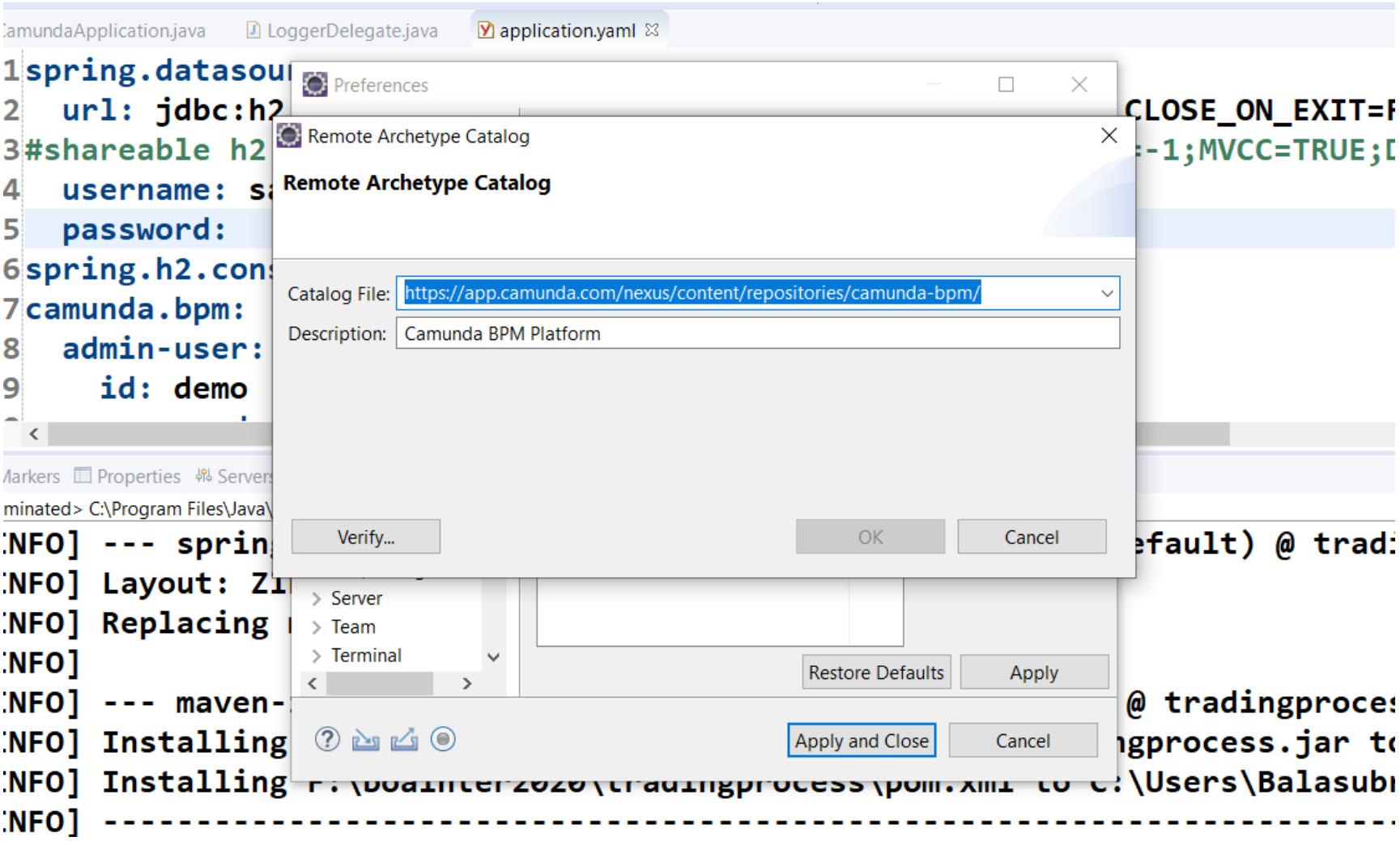


Usage in Eclipse IDE



Usage in Eclipse IDE

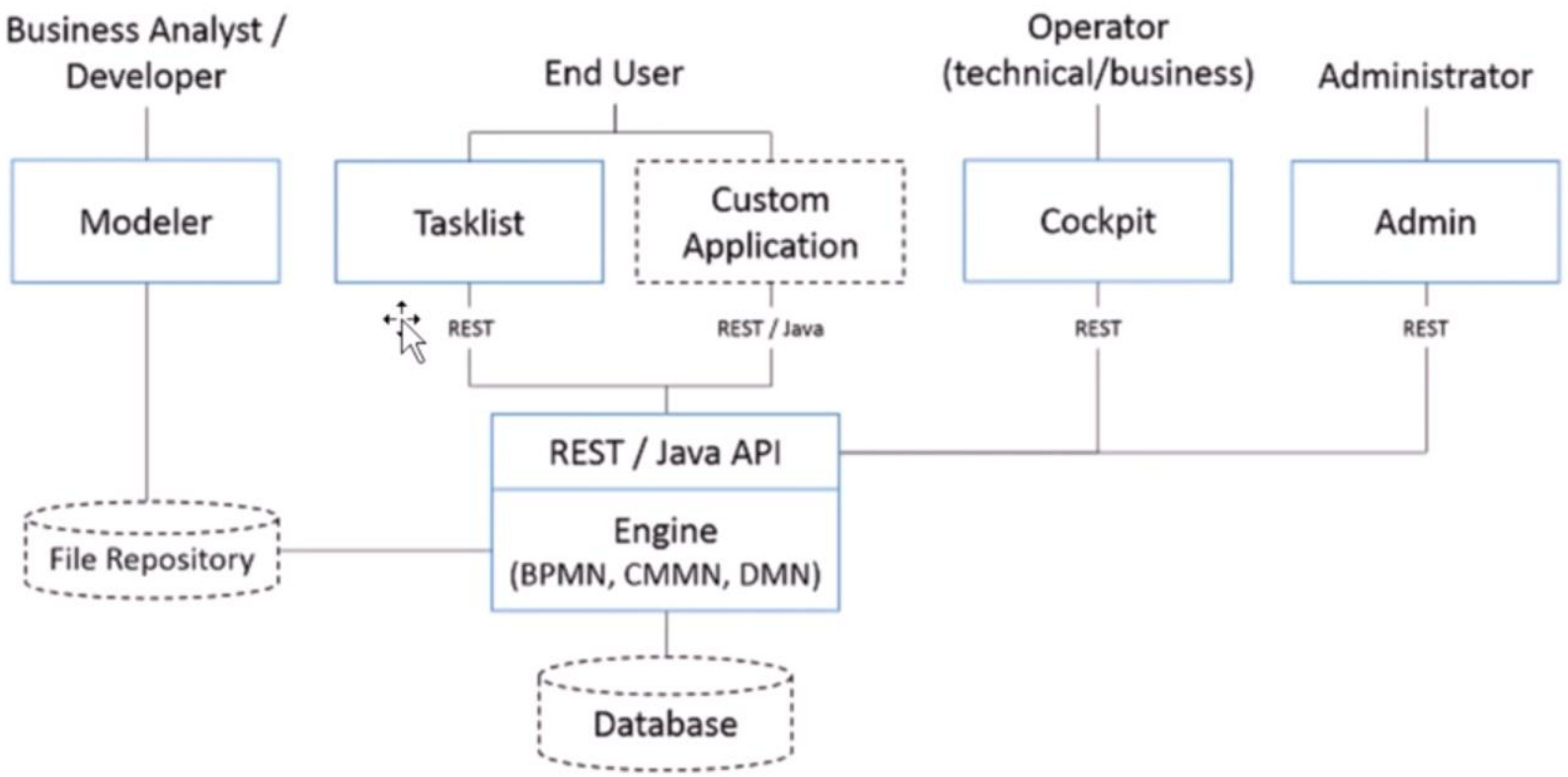
https://app.camunda.com/nexus/content/repositories/camunda-bpm/



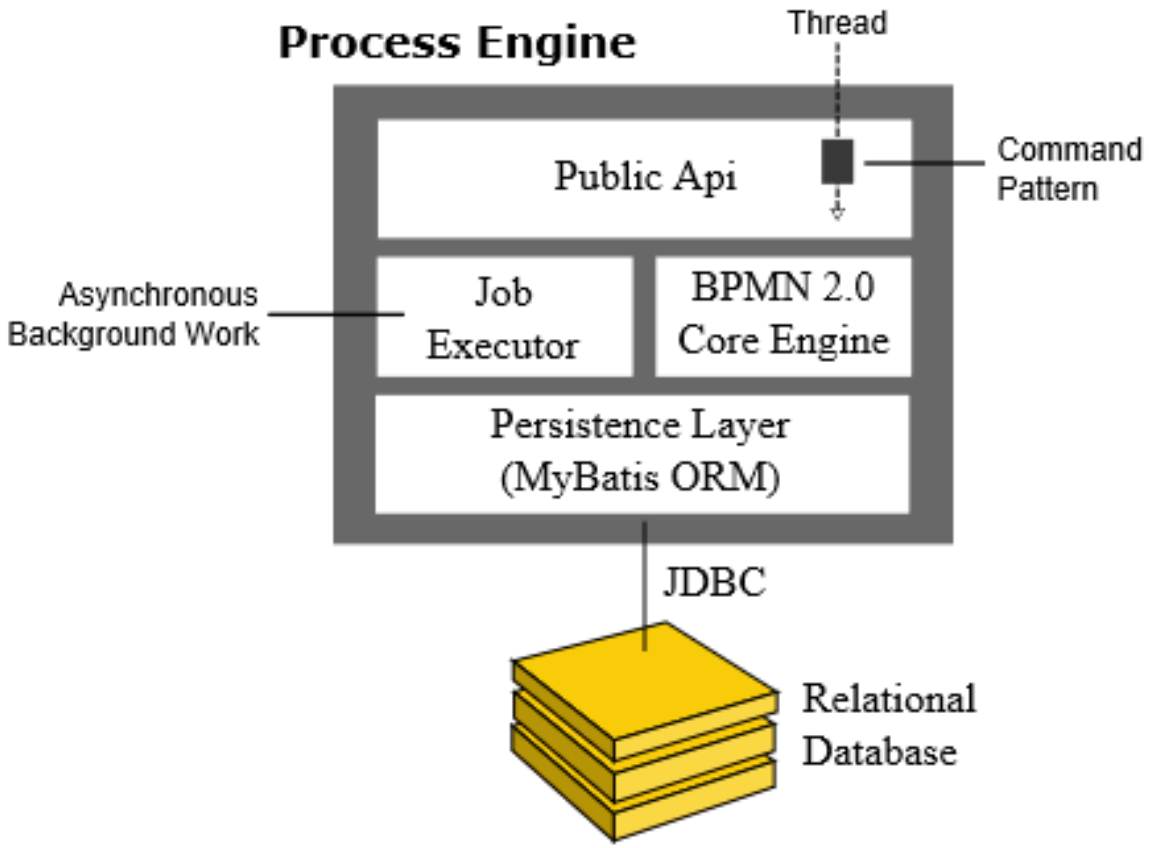
Architecture Overview

- Camunda BPM is a Java-based framework.
- The main components are written in Java.
- It provides Java developers the tools that they need for designing, implementing and running business processes and workflows on the JVM.
- It is also providing Camunda to non java developers using Camunda BPM REST API.
- It allows you to build applications connecting to a remote process engine.

Camunda BPM Components



Process Engine Architecture



BPMN 2.0

- BPMN 2.0 Core Engine: This is the core of the process engine.
- It features a lightweight execution engine for graph structures (PVM - Process Virtual Machine).
- A BPMN 2.0 parser which transforms BPMN 2.0 XML files into Java Objects and a set of BPMN Behavior implementations.

Process Engine Public API

- Process Engine Public API: Service-oriented API allowing Java applications to interact with the process engine.
- The different responsibilities of the process engine (i.e., Process Repository, Runtime Process Interaction, Task Management, …) are separated into individual services.
- The public API features a command-style access pattern:
- Threads entering the process engine are routed through a Command Interceptor which is used for setting up Thread Context such as Transactions.

Job Executor

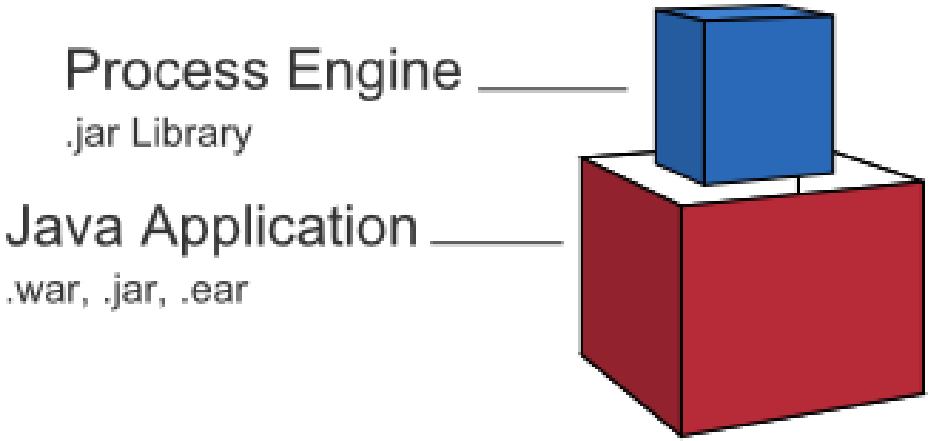
- Job Executor: The Job Executor is responsible for processing asynchronous background work such as Timers or asynchronous continuations in a process.
- The Persistence Layer: The process engine features a persistence layer responsible for persisting process instance state to a relational database.
- It uses MyBatis mapping engine for object relational mapping.

Camunda BPM Platform Architecture

- Camunda BPM platform is a flexible framework which can be deployed in different scenarios.
  - Embedded Process Engine
  - Shared, Container-Managed Process Engine
  - Standalone (Remote) Process Engine Server
  - Clustering Model

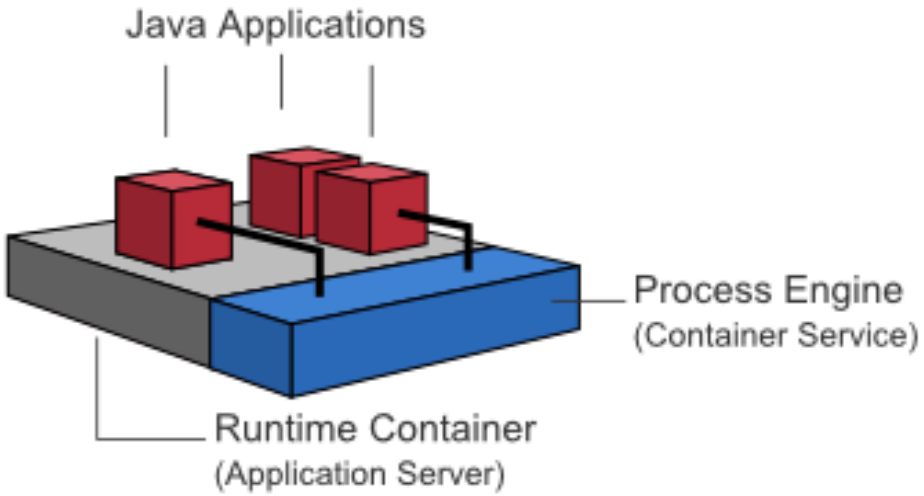
Embedded Process Engine

- The process engine is added as an application library to a custom application.
- This way, the process engine can easily be started and stopped with the application lifecycle.
- It is possible to run multiple embedded process engines on top of a shared database.



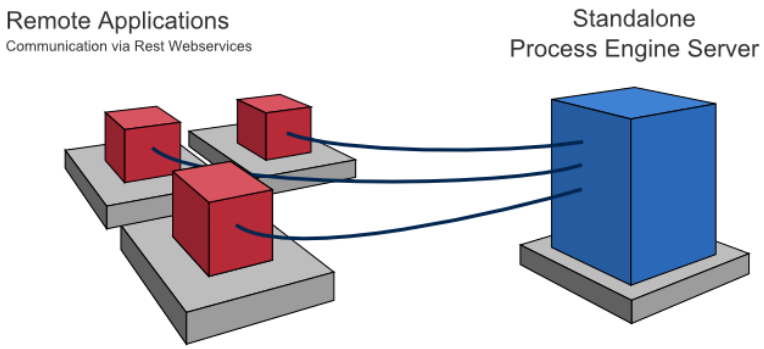
Shared, Container-Managed Process Engine

- In this case, the process engine is started inside the runtime container (Servlet Container, Application Server, …).
- The process engine is provided as a container service and can be shared by all applications deployed inside the container.
- The process engine keeps track of the process definitions deployed by an application and delegates execution to the application.



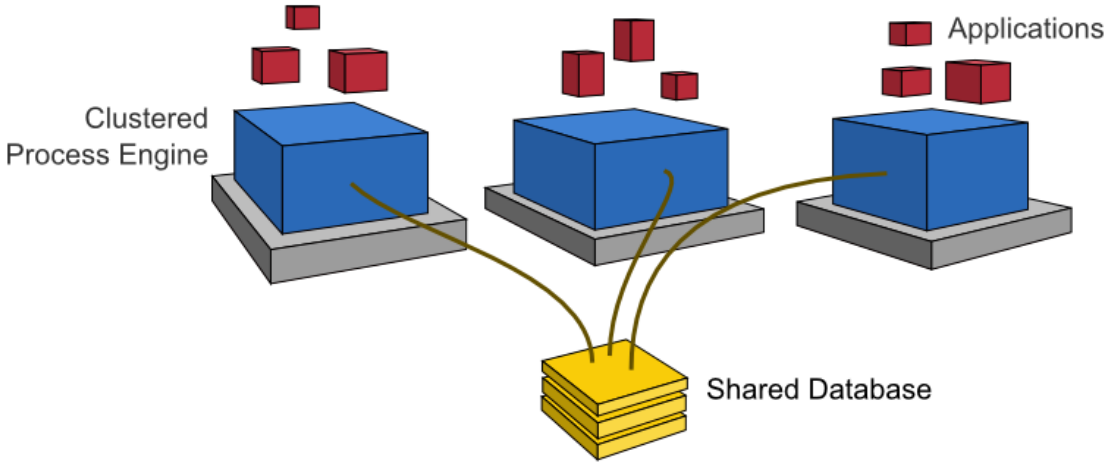
Standalone (Remote) Process Engine Server

- The process engine is provided as a network service.
- Different applications running on the network can interact with the process engine through a remote communication channel.
- The easiest way to make the process engine accessible remotely is to use the built-in REST API.
- Different communication channels such as SOAP Web services or JMS are possible but need to be implemented by users.



Clustering Model

- In order to provide scale-up or fail-over capabilities, the process engine can be distributed to different nodes in a cluster.
- Each process engine instance must then connect to a shared database.



Clustering Model

- The individual process engine instances do not maintain session state across transactions.
- Whenever the process engine runs a transaction, the complete state is flushed out to the shared database.
- This makes it possible to route subsequent requests which do work in the same process instance to different cluster nodes.
- As far as the process engine is concerned, there is no difference between setups for scale-up and setups for fail-over (as the process engine keeps no session state between transactions).

Session State in a Clustered Environment

- Camunda BPM doesn’t provide load-balancing capabilities or session replication capabilities out of the box.
- The load-balancing function would need to be provided by a third-party system.
- Session replication would need to be provided by the host application server.

Session State in a Clustered Environment

- “Sticky sessions” could be configured and enabled within your load balancing solution.
- This would ensure that all requests from a given user are directed to the same instance over a configurable period of time.
- Session sharing can be enabled in your application server such that the application server instances share session state.
- This would allow users to connect to multiple instances in the cluster without being asked to login multiple times.

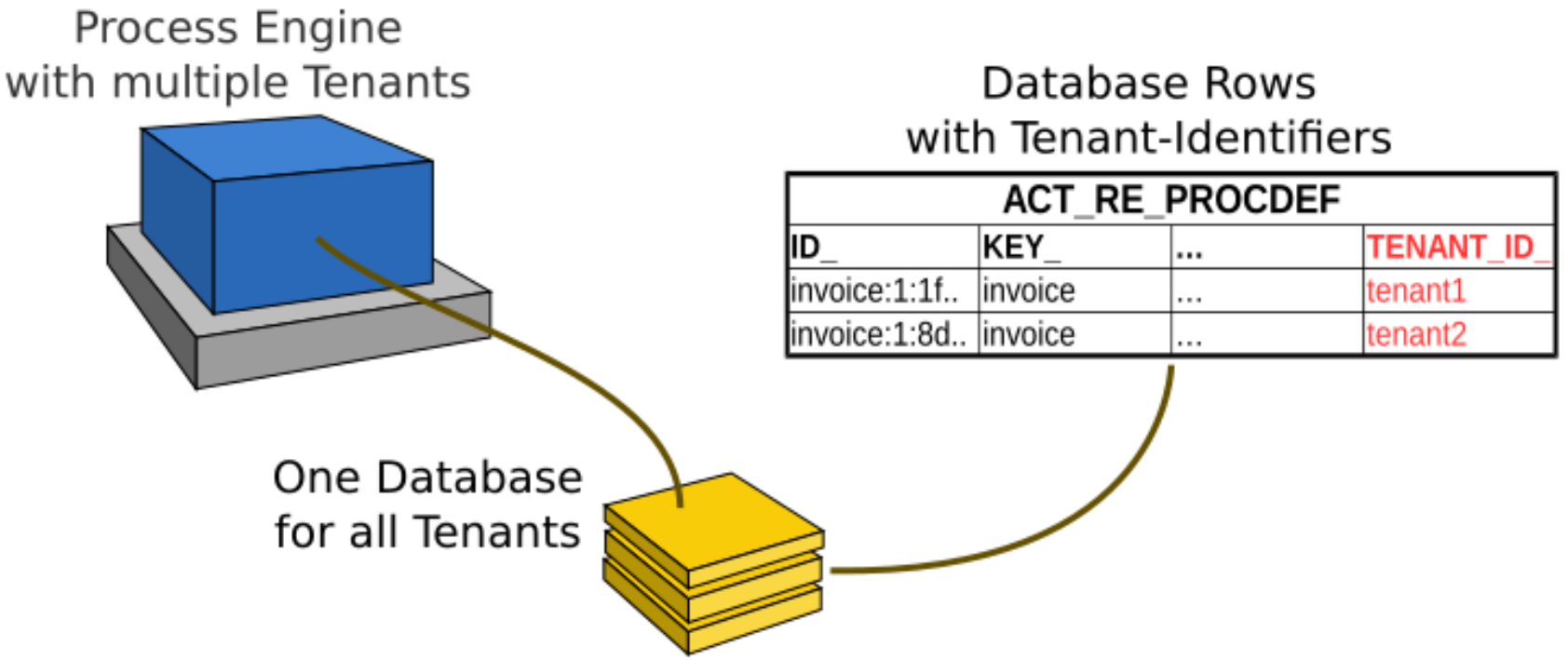
Multi-Tenancy

- Multi-Tenancy regards the case in which a single camunda installation should serve more than one tenant.
- For each tenant, certain guarantees of isolation should be made.
- For example, one tenant’s process instances should not interfere with those of another tenant.

Multi-Tenancy

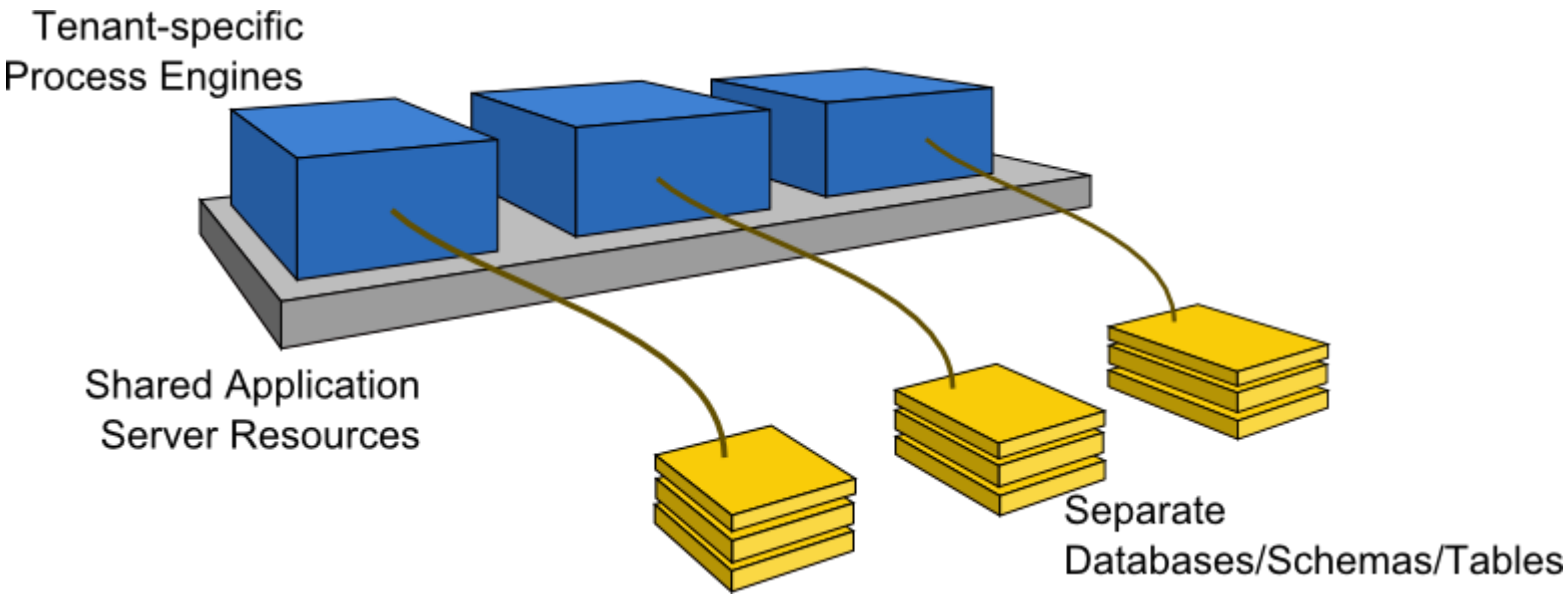
- Multi-Tenancy can be achieved in two different ways.
- One way is to use one process engine per tenant.
- The other way is to use just one process engine and associate the data with tenant identifiers.
- The two ways differ from each other in the level of data isolation, the effort of maintenance and the scalability.
- A combination of both ways is also possible.

Single Process Engine With Tenant-Identifiers

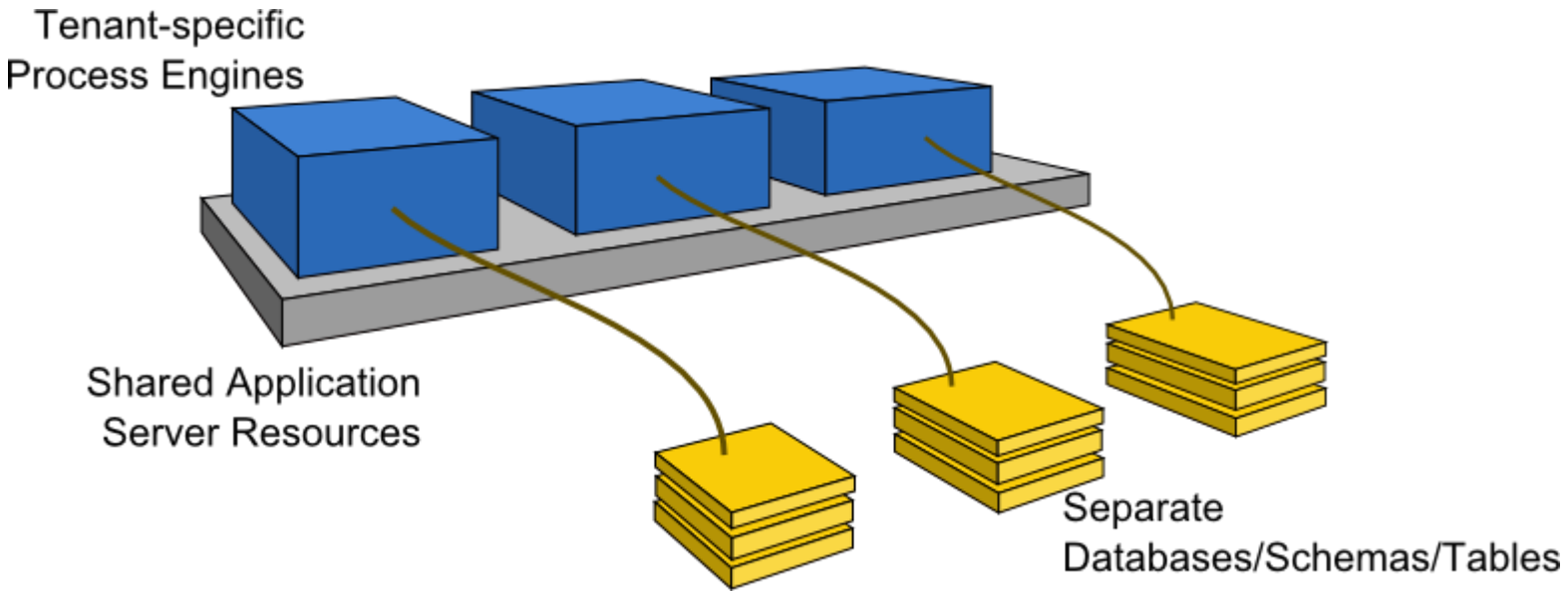


ACT_RE_PROCDEF			
ID_	KEY_	...	TENANT_ID_
invoice:1:1f..	invoice	...	tenant1
invoice:1:8d..	invoice	...	tenant2

One Process Engine Per Tenant

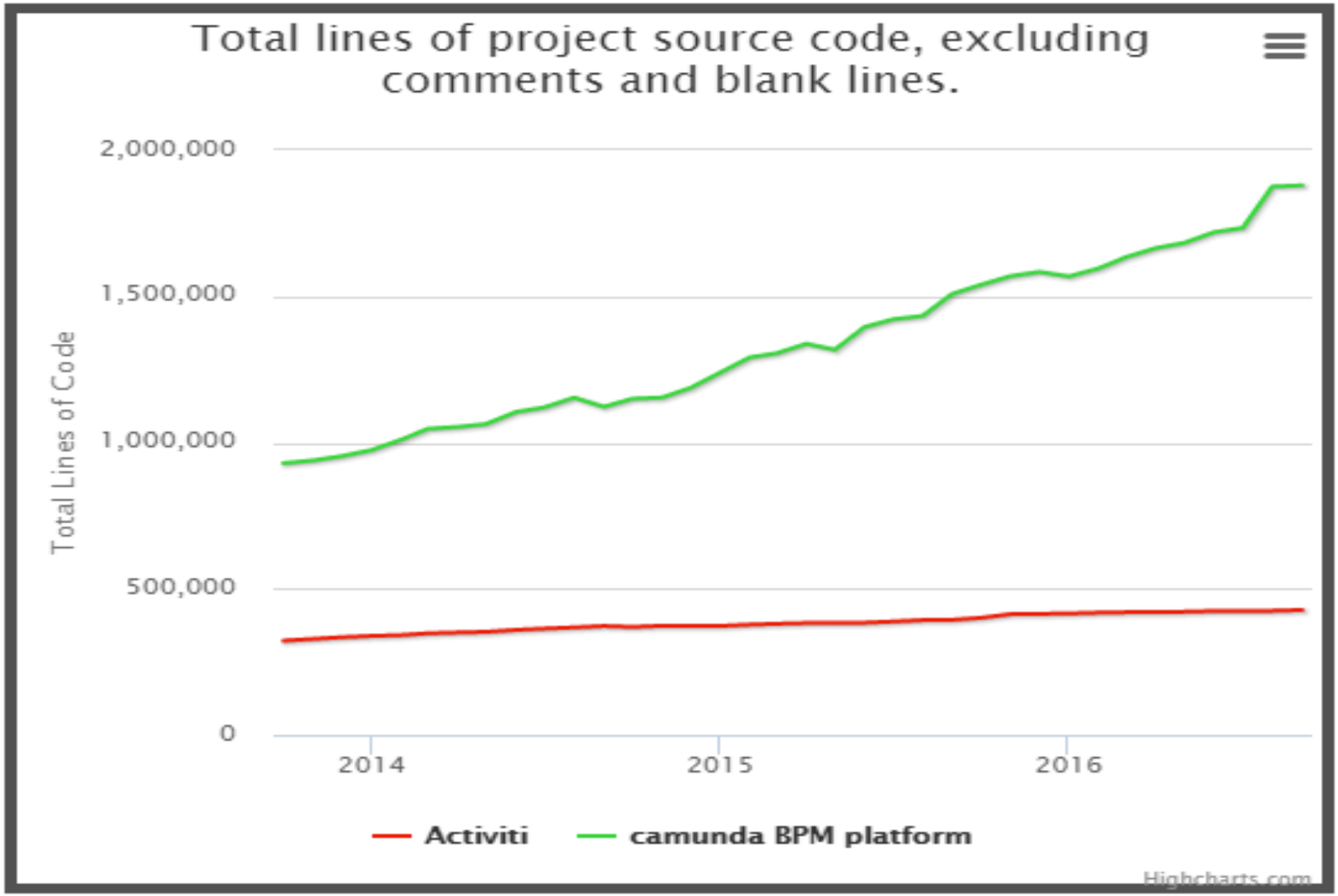


Multi-Tenancy Model



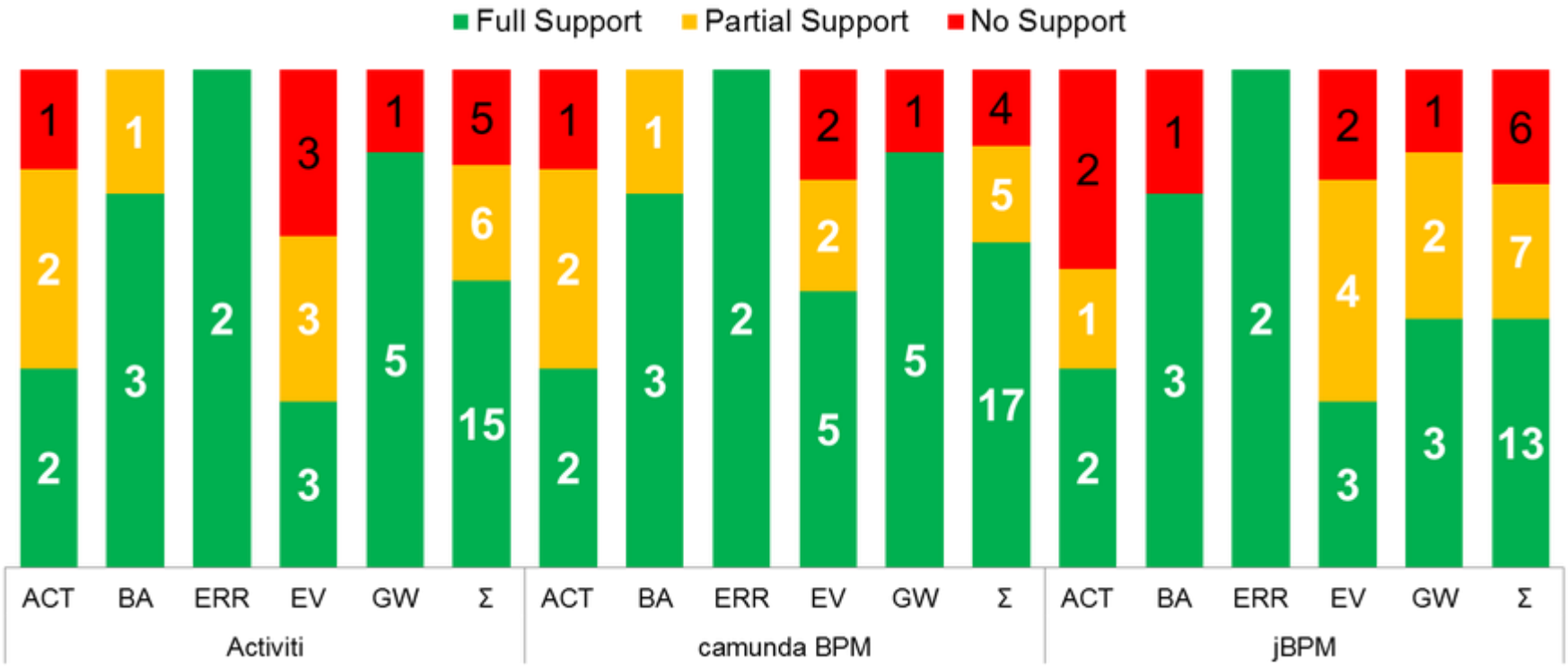
- To serve multiple, independent parties with one Camunda installation, the process engine supports multi-tenancy.
- The following multi tenancy models are supported:
  - Table-level data separation by using different database schemas or databases
  - Row-level data separation by using a tenant marker

Camunda vs Activiti

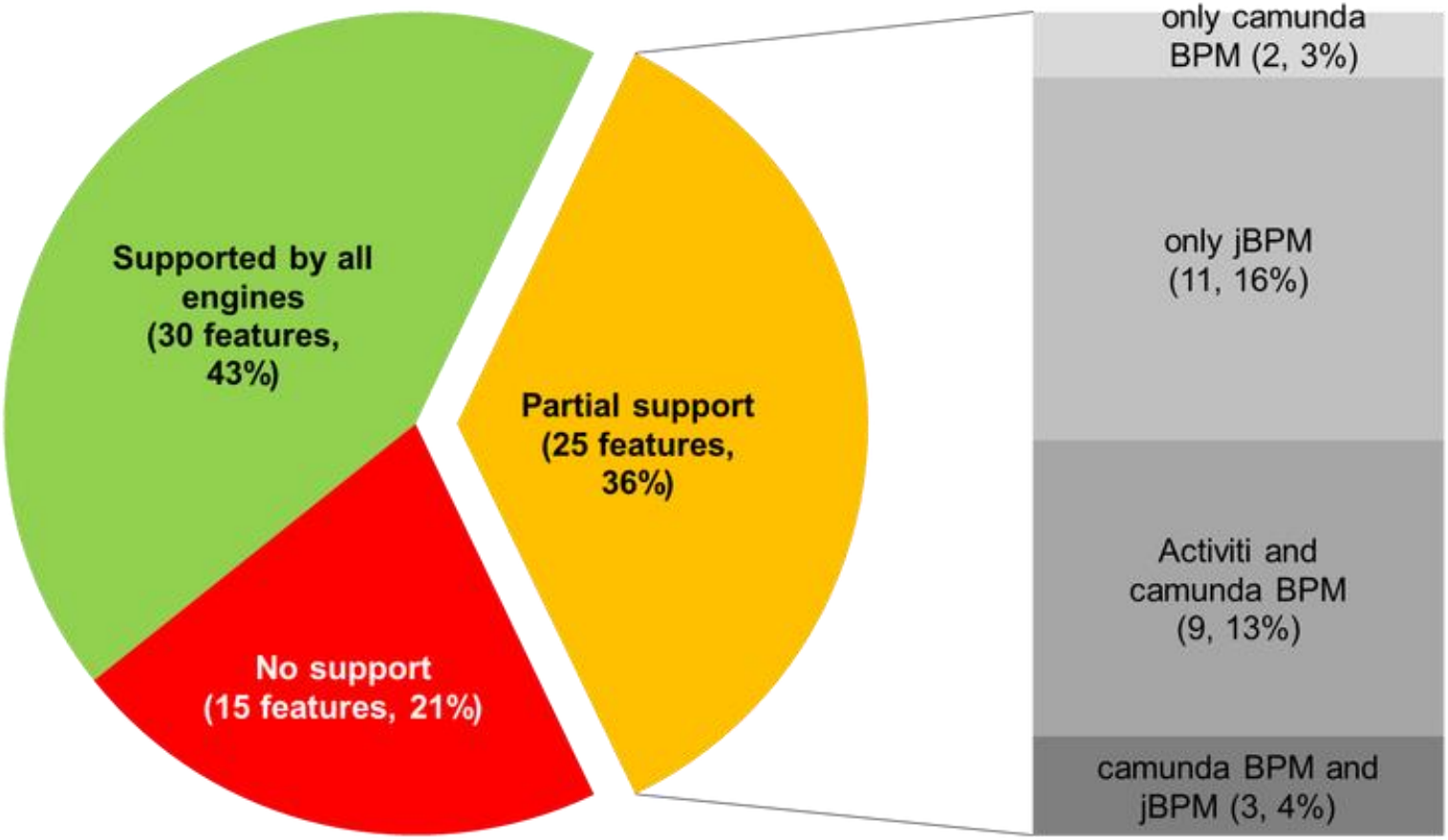


Camunda vs Activiti

ACT stands for actitivities (task, subprocesses, …), BA for basics (sequenceFlows, lanes, participants, …), ERR for errors

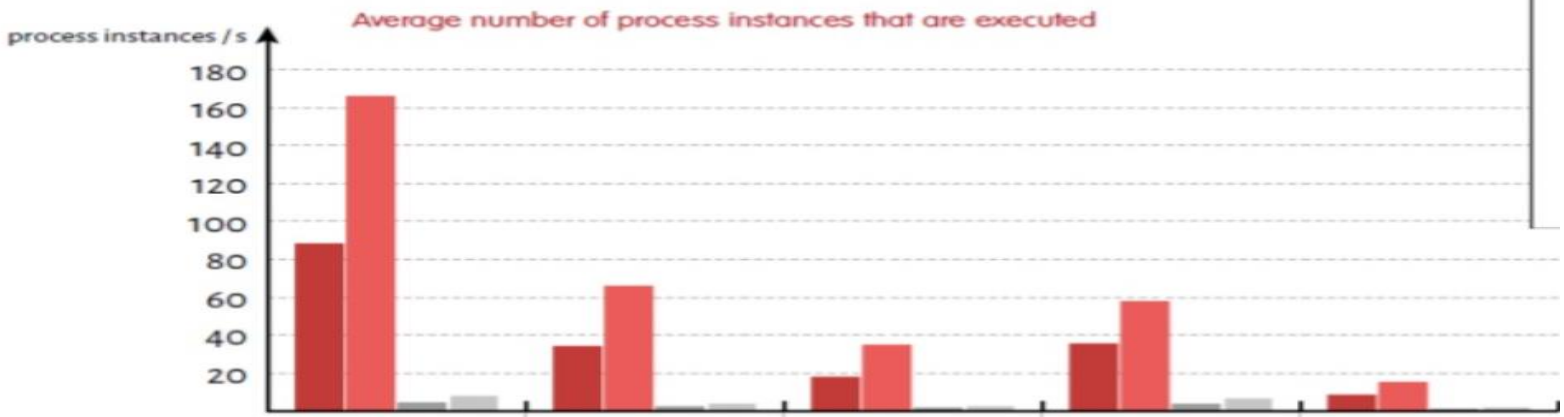


Camunda vs Activiti vs jBPM



Camunda vs jBPM

We are much faster than competition



	Process 1: Service Task	Process 2: User Task	Process 3: Parallel Tasks	Process 4: Credit Card Contract "fast"	Process 4: Credit Card Contract "loop"
Camunda 1 Thread	88.04	33.85	17.94	35.25	7.97
Camunda 2 Threads	166.27	66.09	34.82	57.65	14.87
jBPM 1 Thread	4.29	2.01	1.26	3.30	0.78
jBPM 2 Threads	7.50	3.66	2.19	6.29	1.48

Selecting the Process Engine Mode

Selecting the **Process Engine Mode**

	Container-Managed Engine	Embedded Engine	Remote Engine
	Run the engine as a service preconfigured in your Java EE container.	Use the process engine as a simple library right within your own application, typically started via Spring Boot GREENFIELD .	Run the engine as an isolated BPM server only communicating with it via Web Services.
Engine Bootstrap / Lifecycle Management	Out-of-the-box	Out-of-the-box for Spring Boot, otherwise do-it-yourself (see options below)	
Camunda Webapps work in all use-cases	✔	See limitations below	
Camunda REST API works in all use-cases	✔	See options below	
Multiple Process Applications can share a central engine	✔	Doable with a shared database, but requires custom development and	

Selecting the Process Engine Mode

Camunda Best Practices

Changelog

Multiple Engines can share resources (e.g. share the Job Executor)	✔		The Remote Engine is bootstrapped either as embedded or container-managed engine
One application WAR/EAR can include the process engine		✔	
Supports untouched ("vanilla") containers		✔	
Runs in every Java environment	On Supported Containers	✔	
Development Effort	Minimal	Depends	
Responsibility for Engine Installation and Configuration	Operations or Application Developer	Application Developer	
Application point of view on process engine	Library	Library	Remote Server
Possible communication types with services	Java InVM, Remote	Java InVM, Remote	Remote

Selecting the Process Engine Mode

Camunda Best Practices

Changelog

Use when	You use a supported application server and prefer to seperate engine installation from application development	You use Spring Boot GREENFIELD .	Your architecture or applications are not Java based.
		You want a single deployment including the engine.	You cannot touch your core application.
		You cannot make changes to your container.	For security, your BPM platform is separated from applications.
		You use a container we do not support.	

Selecting the Process Engine Mode

Camunda Best Practices

Changelog

Understanding **Embedded Engine** Specifics

Using Spring Boot

The Camunda Spring Boot Starter is a clean way of controlling the embedded engine easily, so you don't have to think about the specifics mentioned below in this section. This makes Spring Boot the GREENFIELD choice for Camunda projects.

Bootstrapping the Engine and Managing its **Lifecycle**

When running the engine in embedded mode you have to control the **lifecycle** of the engine yourself, basically **starting up** and **shutting down** the engine - and providing access to the API whenever a client needs it. You have several options to do that.

	Spring Boot	**Spring Application Context**	**processes.xml**	**Programmatic**
	Configure, start and stop the engine via Spring Boot Starter	Configure, start and stop the engine via Spring Beans defined in your Application Context.	Configure, start and stop the engine via Camunda's processes.xml descriptor and a ProcessApplication class.	Configure, start and stop the engine yourself programmatically by using Java code.
Use when	You target Spring Boot as runtime environment.	You already use Spring.	You do not want to introduce a Spring dependency just for Camunda.	You need full control over the engine or want to do advanced customizations.
Unlimited Configuration Options	✔	✔		✔
Development Effort	Low	Medium	Low	High

Selecting the Process Engine Mode

Camunda Best Practices

Changelog

Choosing a **Programming Framework**

	Spring	Spring Boot GREENFIELD	CDI	OSGI / Blueprint
	The Java EE alternative to write JVM based systems.	Spring applications with convention over configuration.	Contexts and Dependency Injection (CDI) for Java EE.	A Dynamic Module System for Java.
Use to Control Embedded Engine Lifecycle	✔	✔		
Use Named Beans in Expression Language	✔	✔	✔	✔
Required Skill Level	Spring basics to get started	Spring Boot basics to get started	CDI basics to get started	Big experience with OSGI, introduces complexity many customers struggle with
Use when	Use the framework **you are already experienced with**!			
Supported in Enterprise Edition	✔	✔	✔	Community Extension

Selecting the Process Engine Mode

Camunda Best Practices

Changelog

Choosing a **Container / Application Server**

		Java SE containers		Wildfly	Other Java EE containers	Unsupported Containers	OSGI Containers
		Tomcat (see supported Java SE environments)	Spring Boot Starter GREENFIELD	JBoss AS, Wildfly, Glassfish, IBM WAS, Oracle WLS (see supported Java EE environments)			Apache Karaf
(Out-of-the-box) JTA Transaction Integration			(✔)	✔		Maybe	
Java EE Features (JPA, JSF, CDI, ...)			(✔)	✔		Maybe	
Engine Mode	**Container-Managed Engine**	✔		✔			Community Extension
	Embedded Engine	✔	✔	✔		✔	✔
Supported with Enterprise Edition		✔	✔	✔		Engine only	Engine only
Use when		Use the container **you already operate or are experienced with**!					
		Tomcat	**Spring Boot**	**Wildfly, Glassfish, IBM WAS, Oracle WLS**			**Apache Karaf**

Selecting the Process Engine Mode

Camunda Best Practices

Changelog

Considering the **User Interface**

Understanding the **Use Cases**

n the User Interface you might have different components, for which you might even decide differently.

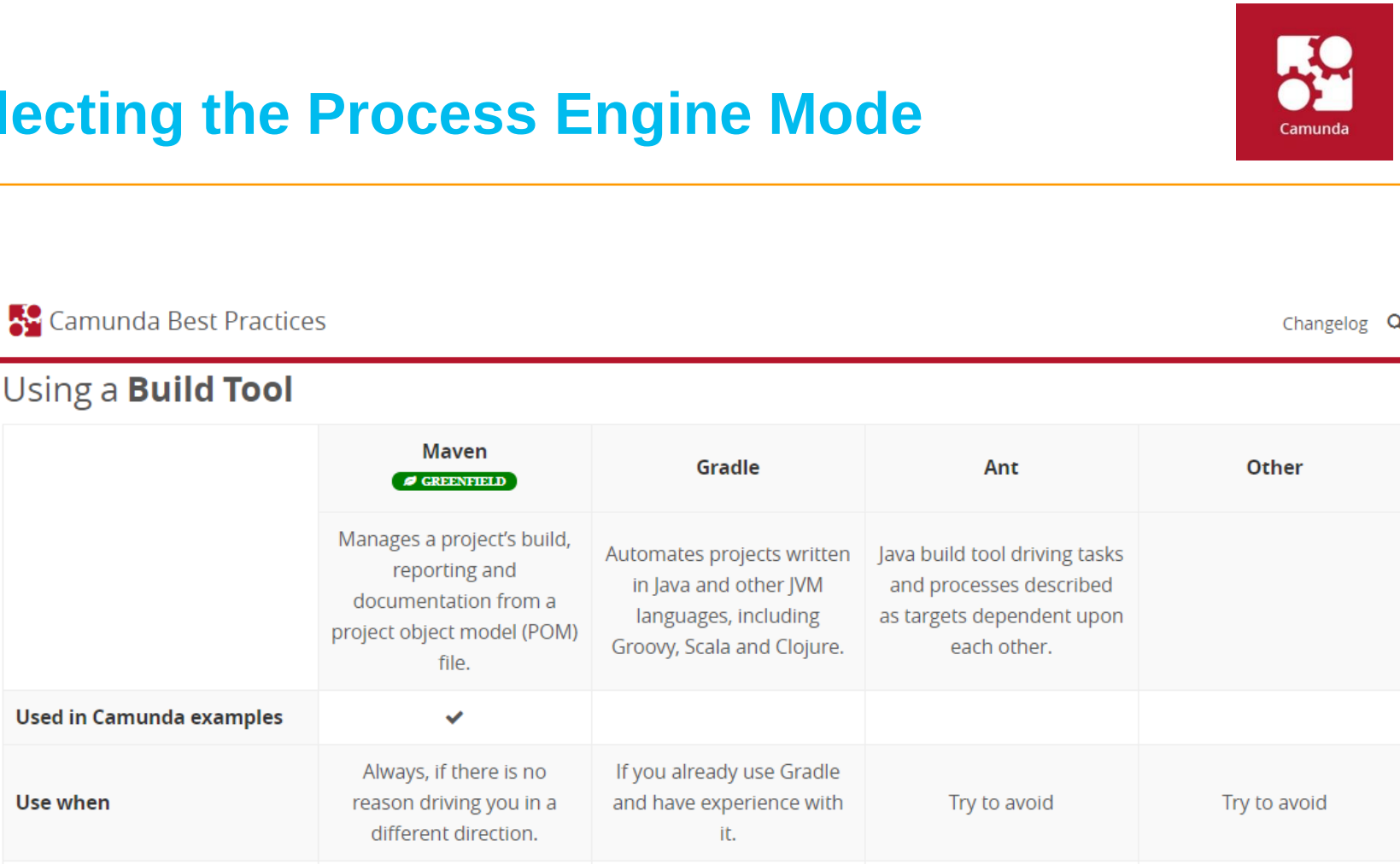
	Organisation Internal			Public
	Tasklist Application	**Custom Business Application**	**Monitoring and/or Reporting Application**	**Internet Website**
	A (central) dedicated tasklist, showing tasks from potentially different contexts.	Your business application includes use cases steering the process engine.	An application showing state of processes, based on instances or as statistics.	Your internet website e.g kicks of process instances when a user submits an order.
Target Group	All employees	Specific employees	Operators (Business/Technical)	Customers
Published in	Intranet			Internet
Required Usability	Generic tasklist, customizable for roles	Tailored to a specific business use case	Different for business or technical operations	Easy-to-use modern web application
Required Technology		Use technology of business application		Use modern HTML5 stack
Possibility to Reuse	✔		✔	

Selecting the Process Engine Mode

Choosing a **User Interface Technology**

	HTML5	Java UI Frameworks	Non Java Technology	Portal
	e.g. JavaScript, AngularJS	e.g. JSF, Vaadin, Wicket	e.g. PHP	e.g. Portlets, Liferay
Learning Curve for Java Developers	high	low	medium	depends on portal
Typical Use Cases	Single Page Applications	Java-based Applications and Form-based Applications	Standalone Web Applications	Company wide portals
Use when	Use the framework **you are experienced with or want to build up experience with**! (Don't forget about the learning curve!)			

Selecting the Process Engine Mode



Camunda Best Practices

Changelog

Using a **Build Tool**

	Maven GREENFIELD	Gradle	Ant	Other
	Manages a project's build, reporting and documentation from a project object model (POM) file.	Automates projects written in Java and other JVM languages, including Groovy, Scala and Clojure.	Java build tool driving tasks and processes described as targets dependent upon each other.	
Used in Camunda examples	✔			
Use when	Always, if there is no reason driving you in a different direction.	If you already use Gradle and have experience with it.	Try to avoid	Try to avoid

Selecting the Process Engine Mode

Camunda Best Practices

Changelog

	Learn More	Learn More	Learn More

Choosing a **Database**

Camunda requires a **relational database** for persistence.

Even if the persistence provider can be plugged and e.g. exchanged by some **NoSQL** persistence this is neither recommended nor supported. Therefore, if you have use cases for this discuss them with Camunda beforehand!

	PostgreSQL GREENFIELD (Prod)	Oracle	H2 GREENFIELD (Test)	Other Database
	PostgreSQL is an open source object-relational database system.	Oracle Database is a commercial object-relational database system.	H2 is a Java SQL database with in-memory mode and a small footprint.	
Best Performance Observations	✔	✔		
In-Memory Mode			✔	
No installation required			✔	
Recommended for Production Use	✔	✔		✔ (if supported)

Web Application Architecture

- The Camunda BPM web applications are based on a RESTful architecture.
- Frameworks used:
  - JAX-RS based Rest API
  - AngularJS
  - RequireJS
  - jQuery
  - Twitter Bootstrap
  - camunda-bpmn.js
  - ngDefine
  - angular-data-depend

Container/Application Server for Runtime Components

- Application-Embedded Process Engine
  - All Java application servers
  - Camunda Spring Boot Starter: Embedded Tomcat
- Container-Managed Process Engine and Camunda Cockpit, Tasklist, Admin
  - Apache Tomcat 7.0 / 8.0 / 9.0
  - JBoss EAP 6.4 / 7.0 / 7.1 / 7.2
  - Wildfly Application Server 10.1 / 11.0 / 12.0 / 13.0 / 14.0 / 15.0 / 16.0 / 17.0 / 18.0
  - IBM WebSphere Application Server 8.5 / 9.0 (Enterprise Edition only)
  - Oracle WebLogic Server 12c (12R1,12R2) (Enterprise Edition only) and Deployment scenarios).

Container/Application Server for Runtime Components

- Databases
- Supported Database Products
  - MySQL 5.6 / 5.7
  - MariaDB 10.0 / 10.2 / 10.3
  - Oracle 11g / 12c / 18c
  - IBM DB2 10.5 / 11.1 (excluding IBM z/OS for all versions)
  - PostgreSQL 9.4 / 9.6 / 10.4 / 10.7 / 11.1 / 11.2
  - Amazon Aurora PostgreSQL compatible with PostgreSQL 9.6 / 10.4 / 10.7
  - Microsoft SQL Server 2012/2014/2016/2017
  - H2 1.4

Container/Application Server for Runtime Components

- Web Browser
  - Google Chrome latest
  - Mozilla Firefox latest
  - Internet Explorer 11
  - Microsoft Edge
- Java
  - Java 8 / 9 / 10 / 11 / 12 / 13 (if supported by your application server/container)

Container/Application Server for Runtime Components

- Java Runtime
  - Oracle JDK 8 / 9 / 10 / 11 / 12 / 13
  - IBM JDK 8 (with J9 JVM)
  - OpenJDK 8 / 9 / 10 / 11 / 12 / 13, including builds of the following products:
  - Oracle OpenJDK
  - AdoptOpenJDK (with HotSpot JVM)
  - Amazon Corretto
  - Azul Zulu

Container/Application Server for Runtime Components

- Camunda Modeler
- Supported on the following platforms:
  - Windows 7 / 10
  - Mac OS X 10.11
  - Ubuntu LTS (latest)

- Reported to work on
  - Ubuntu 12.04 and newer
  - Fedora 21
  - Debian 8

Container/Application Server for Runtime Components

- List of Community Extensions
- The following is a list of current (unsupported) community extensions:
  - Apache Camel Integration
  - Camunda Docker Images
  - Custom Batch
  - DMN Scala Extension
  - Elastic Search Extension
  - Email Connectors
  - FEEL Scala Extension
  - Grails Plugin

Container/Application Server for Runtime Components

- GraphQL API
- Keycloak Identity Provider Plugin
- Migration API
- Mockito Testing Library
- Needle Testing Library
- OSGi Integration
- PHP SDK
- Process Test Coverage
- Reactor Event Bus
- REST Client Spring Boot
- Scenario Testing Library
- Single Sign On for JBoss
- Tasklist Translations
- Wildfly Swarm

Container/Application Server for Runtime Components

- Definition of Public API
  - The Camunda BPM public API is limited to the following items:
  - Java API:
  - All non-implementation Java packages (package name does not contain impl) of the following modules.
    - camunda-engine
    - camunda-engine-spring
    - camunda-engine-cdi
    - camunda-engine-dmn
    - camunda-bpmn-model

Container/Application Server for Runtime Components

- camunda-cmn-model
- camunda-dmn-model
- camunda-spin-core
- camunda-connect-core
- camunda-commons-typed-valuesSwarm

Process Engine Bootstrapping

- Application Managed Process Engine
  - Programmatically via Java API
  - Via XML configuration
  - Via Spring
- Shared, Container Managed Process Engine
  - A container of your choice (e.g., Tomcat, JBoss or IBM WebSphere) manages the process engine.
  - The configuration is carried out in a container specific way(Runtime Container Integration).

Process Engine Configuration Bean

- The Camunda engine uses the Process Engine Configuration bean to configure and construct a standalone Process Engine.
- The following classes are currently available:
  - org.camunda.bpm.engine.impl.cfg.StandaloneProcessEngineConfiguration
    - The process engine is used in a standalone way.
    - The engine itself will take care of the transactions.
    - By default, the database will only be checked when the engine boots.
    - An exception is thrown if there is no database schema, or the schema version is incorrect.

Process Engine Configuration Bean

- org.camunda.bpm.engine.impl.cfg.StandaloneInMemProcessEngineConfiguration
  - This is a convenience class for unit testing purposes. The engine itself will take care of the transactions.
  - An H2 in-memory database is used by default.
  - The database will be created and dropped when the engine boots and shuts down.
  - When using this, probably no additional configuration is needed
  - Except, for example, when using the job executor or mail capabilities.

Process Engine Configuration Bean

- org.camunda.bpm.engine.spring.SpringProcessEngineConfiguration
  - To be used when the process engine is used in a Spring environment. See the Spring integration section for more information.
- org.camunda.bpm.engine.impl.cfg.JtaProcessEngineConfiguration
  - To be used when the engine runs in standalone mode, with JTA transactions.

Bootstrap a Process Engine Using the Java API

- Configure the process engine programmatically by creating the right Process Engine Configuration object or by using some pre-defined one:

```
ProcessEngineConfiguration.createStandaloneProcessEngineConfiguration();
ProcessEngineConfiguration.createStandaloneInMemProcessEngineConfiguration();
```

Now you can call the `buildProcessEngine()` operation to create a Process Engine:

```
ProcessEngine processEngine = ProcessEngineConfiguration.createStandaloneInMemProcessEngineConf
  .setDatabaseSchemaUpdate(ProcessEngineConfiguration.DB_SCHEMA_UPDATE_FALSE)
  .setJdbcUrl("jdbc:h2:mem:my-own-db;DB_CLOSE_DELAY=1000")
  .setJobExecutorActivate(true)
  .buildProcessEngine();
```

Bootstrap a Process Engine Using the Java API

This will look for a `camunda.cfg.xml` file on the classpath and construct an engine based on the configuration in that file. The following snippet shows an example configuration:

```
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframewo

  <bean id="processEngineConfiguration" class="org.camunda.bpm.engine.impl.cfg.StandaloneProces

    <property name="jdbcUrl" value="jdbc:h2:mem:camunda;DB_CLOSE_DELAY=1000" />
    <property name="jdbcDriver" value="org.h2.Driver" />
    <property name="jdbcUsername" value="sa" />
    <property name="jdbcPassword" value="" />

    <property name="databaseSchemaUpdate" value="true" />

    <property name="jobExecutorActivate" value="false" />

    <property name="mailServerHost" value="mail.my-corp.com" />
    <property name="mailServerPort" value="5025" />
  </bean>
```

Configure Process Engine Using camunda cfg XML

The easiest way to configure your Process Engine is through an XML file called `camunda.cfg.xml`. Using that you can simply do:

```
ProcessEngine processEngine = ProcessEngines.getDefaultProcessEngine()
```

The `camunda.cfg.xml` must contain a bean that has the id `processEngineConfiguration`, select the `ProcessEngineConfiguration` class best suited to your needs:

```
<bean id="processEngineConfiguration" class="org.camunda.bpm.engine.impl.cfg.StandaloneProcessE
```

Configure Process Engine in the bpm-platform.xml

- ProcessEngineConfiguration.createProcessEngineConfigurationFromResourceDefault();
- ProcessEngineConfiguration.createProcessEngineConfigurationFromResource(String resource);
- ProcessEngineConfiguration.createProcessEngineConfigurationFromResource(String resource, String beanName);
- ProcessEngineConfiguration.createProcessEngineConfigurationFromInputStream(InputStream inputStream);
- ProcessEngineConfiguration.createProcessEngineConfigurationFromInputStream(InputStream inputStream, String beanName);

Configure Process Engine in the bpm-platform.xml

- The bpm-platform.xml file is used to configure the Camunda Platform in the following distributions:
  - Apache Tomcat
  - IBM WebSphere Application Server
  - Oracle WebLogic Application Server

Configure Process Engine in the bpm-platform.xml

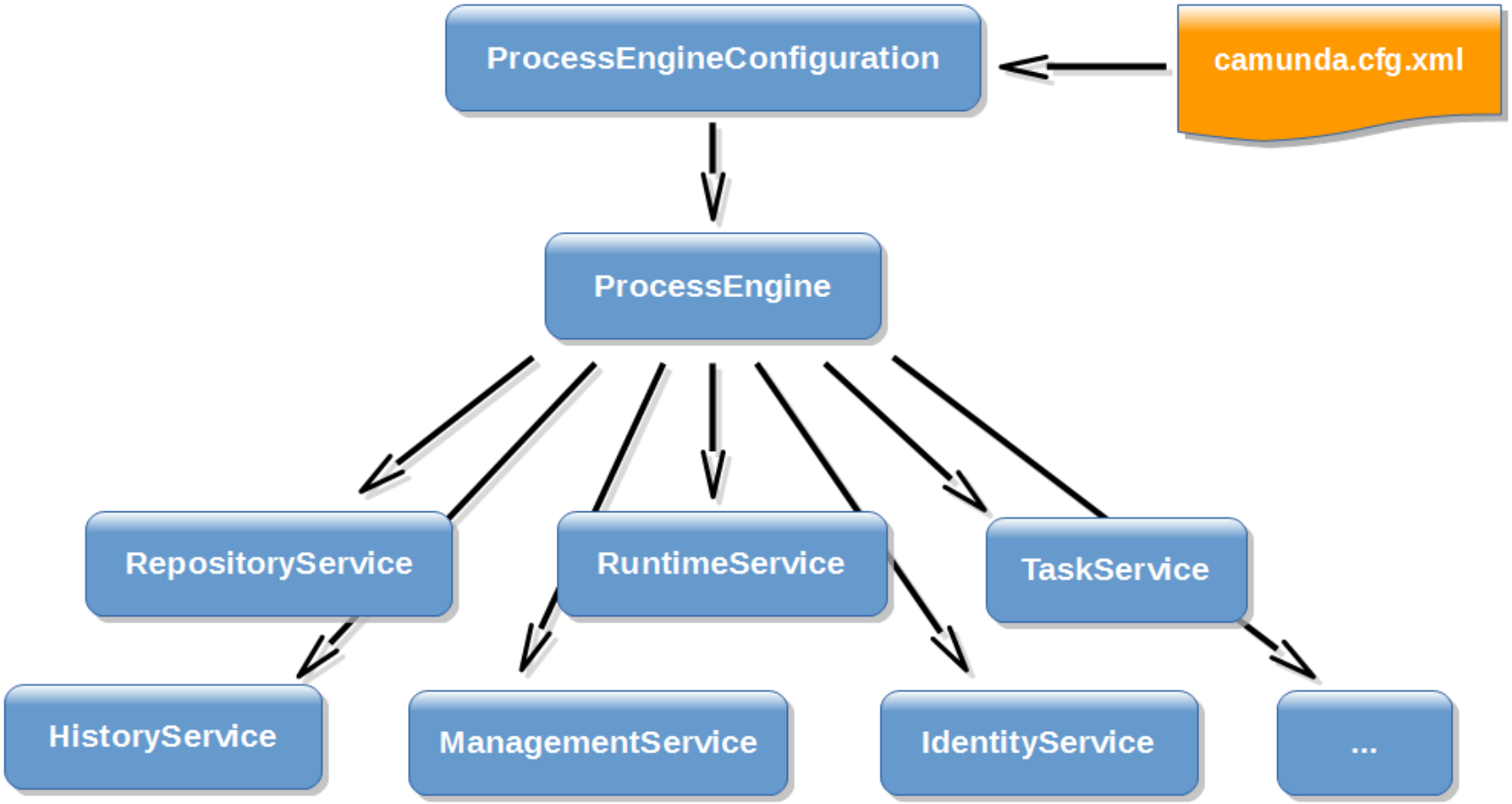
The <process-engine ... /> xml tag allows you to define a process engine:

```
<?xml version="1.0" encoding="UTF-8"?>
<bpm-platform xmlns="http://www.camunda.org/schema/1.0/BpmPlatform"
              xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
              xsi:schemaLocation="http://www.camunda.org/schema/1.0/BpmPlatform http://www.camu

  <job-executor>
    <job-acquisition name="default" />
  </job-executor>

  <process-engine name="default">
    <job-acquisition>default</job-acquisition>
    <configuration>org.camunda.bpm.engine.impl.cfg.StandaloneProcessEngineConfiguration</config
    <datasource>java:jdbc/ProcessEngine</datasource>

    <properties>
      <property name="history">full</property>
      <property name="databaseSchemaUpdate">true</property>
      <property name="authorizationEnabled">true</property>
    </properties>

  </process-engine>
</bpm-platform>
```

Process Engine Concepts

- A process definition defines the structure of a process.
- Camunda Platform uses BPMN 2.0 as its primary modeling language for modeling process definitions.
- In Camunda Platform you can deploy processes to the process engine in BPMN 2.0 XML format.
- The XML files are parsed and transformed into a process definition graph structure. This graph structure is executed by the process engine.

Process Engine API



Process Engine API


```
ProcessEngine processEngine = ProcessEngines.getDefaultProcessEngine();

RepositoryService repositoryService = processEngine.getRepositoryService();
RuntimeService runtimeService = processEngine.getRuntimeService();
TaskService taskService = processEngine.getTaskService();
IdentityService identityService = processEngine.getIdentityService();
FormService formService = processEngine.getFormService();
HistoryService historyService = processEngine.getHistoryService();
ManagementService managementService = processEngine.getManagementService();
FilterService filterService = processEngine.getFilterService();
ExternalTaskService externalTaskService = processEngine.getExternalTaskService();
CaseService caseService = processEngine.getCaseService();
DecisionService decisionService = processEngine.getDecisionService();
```

Process Engine API

- ProcessEngines.getDefaultProcessEngine() will initialize and build a process engine the first time it is called and afterwards always returns the same process engine.
- Proper creation and closing of all process engines can be done with ProcessEngines.init() and ProcessEngines.destroy().

Process Engine API

- The ProcessEngines class will scan for all camunda.cfg.xml and activiti.cfg.xml files.
- For all camunda.cfg.xml files, the process engine will be built in the typical way:

```
ProcessEngineConfiguration
  .createProcessEngineConfigurationFromInputStream(inputStream)
  .buildProcessEngine()
```

Process Engine API

- **The RepositoryService** is probably the first service needed when working with the Camunda engine.
- This service offers operations for managing and manipulating deployments and process definitions.
- **RuntimeService** is dynamic. It deals with starting new process instances of process definitions.
  - A process instance is one execution of such a process definition.
  - For each process definition there are typically many instances running at the same time.
- **The RuntimeService** is also the service which is used to retrieve and store process variables.
  - This is data specific to the given process instance and can be used by various constructs in the process (e.g., an exclusive gateway often uses process variables to determine which path is chosen to continue the process).
  - **The RuntimeService** also allows to query on process instances and executions. Executions are a representation of the ‘token’ concept of BPMN 2.0.

Process Engine API

- **The IdentityService** is simple.
  - It allows the management (creation, update, deletion, querying, …) of groups and users.
  - It is important to understand that the core engine doesn't do any checking on users at runtime.
  - For example, a task could be assigned to any user, but the engine does not verify if that user is known to the system.
  - This is because the engine can also be used in conjunction with services such as LDAP, Active Directory, etc.

Process Engine API

- **The FormService is an optional service.**
  - This service introduces the concept of a start form and a task form.
  - A start form is a form that is shown to the user before the process instance is started, while a task form is the form that is displayed when a user wants to complete a task.
  - This service exposes this data in an easy way to work with.
  - But again, this is optional as forms don’t need to be embedded in the process definition.

Process Engine API

- The **History Service** exposes all historical data gathered by the engine.
  - When executing processes, a lot of data can be kept by the engine (this is configurable) such as process instance start times, who did which tasks, how long it took to complete the tasks, which path was followed in each process instance, etc.
  - This service exposes mainly query capabilities to access this data.

Process Engine API

- The **Management Service** is typically not needed when coding custom applications.
  - It allows to retrieve information about the database tables and table metadata.
  - Furthermore, it exposes query capabilities and management operations for jobs.
  - Jobs are used in the engine for various things such as timers, asynchronous continuations, delayed suspension/activation, etc.

Process Engine API

- The **Filter Service** allows to create and manage filters.
  - Filters are stored queries like task queries.
  - For example filters are used by Tasklist to filter user tasks.
- The **External TaskService** provides access to external task instances.
  - External tasks represent work items that are processed externally and independently of the process engine.

Process Engine API

- The **Case Service** is like the RuntimeService but for case instances.
  - It deals with starting new case instances of case definitions and managing the lifecycle of case executions.
  - The service is also used to retrieve and update process variables of case instances.
- The **Decision Service** allows to evaluate decisions that are deployed to the engine.
  - It is an alternative to evaluate a decision within a business rule task that is independent from a process definition.

Query API

- To query data from the engine there are multiple possibilities:
  - Java Query API: Fluent Java API to query engine entities (like ProcessInstances, Tasks, …).
  - REST Query API: REST API to query engine entities (like ProcessInstances, Tasks, …).
  - Native Queries: Provide own SQL queries to retrieve engine entities (like ProcessInstances, Tasks, …) if the Query API lacks the possibilities you need (e.g., OR conditions).
  - Custom Queries: Use completely customized queries and an own MyBatis mapping to retrieve own value objects or join engine with domain data.
  - SQL Queries: Use database SQL queries for use cases like Reporting.

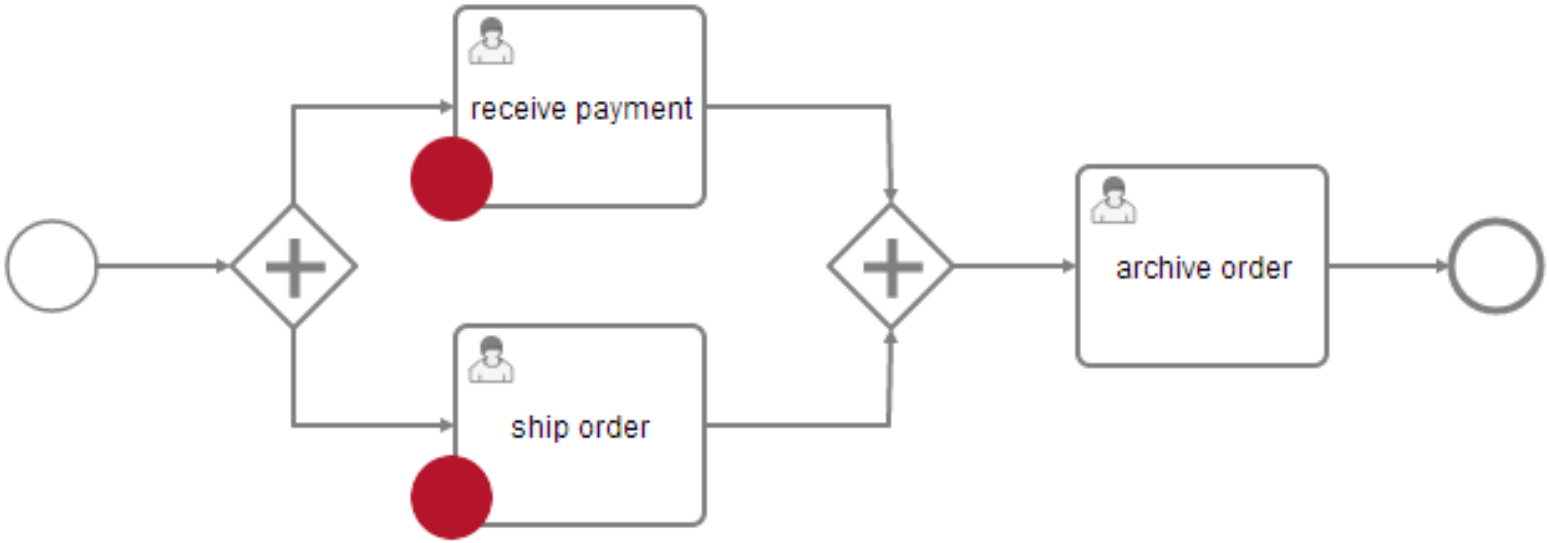
Process Variables

- Variables can be used to add data to process runtime state or, more particular, variable scopes.
- Various API methods that change the state of these entities allow updating of the attached variables.
- In general, a variable consists of a name and a value.
- The name is used for identification across process constructs.
- For example, if one activity sets a variable named var, a follow-up activity can access it by using this name.
- The value of a variable is a Java object.

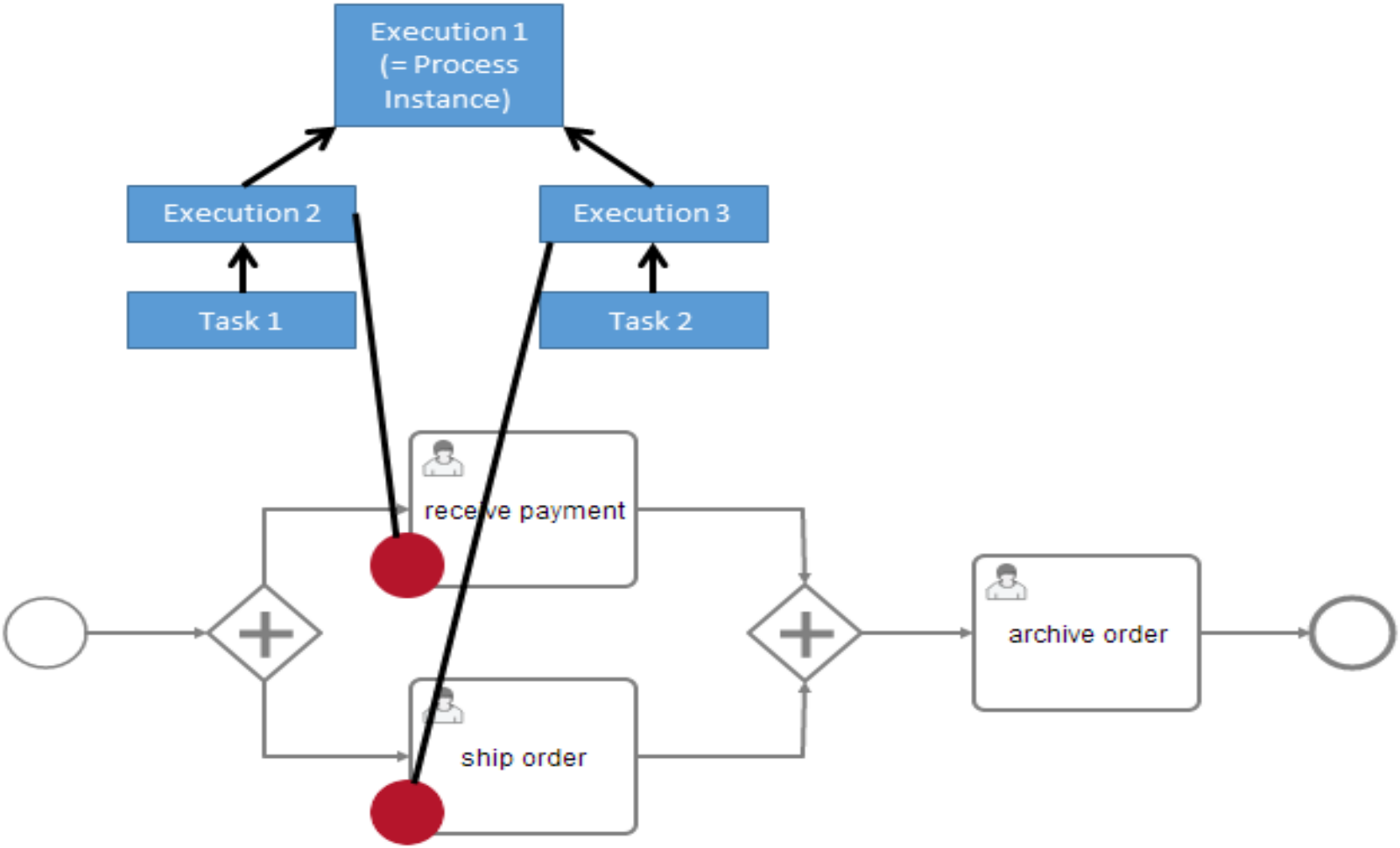
Variable Scopes and Variable Visibility

- In general, variables are accessible in the following cases:
  - Instantiating processes
  - Delivering messages
  - Task lifecycle transitions, such as completion or resolution
  - Setting/getting variables from outside
  - Setting/getting variables in a Delegate
  - Expressions in the process model
  - Scripts in the process model
  - (Historic) Variable queries

Variable Scopes and Variable Visibility



Variable Scopes and Variable Visibility

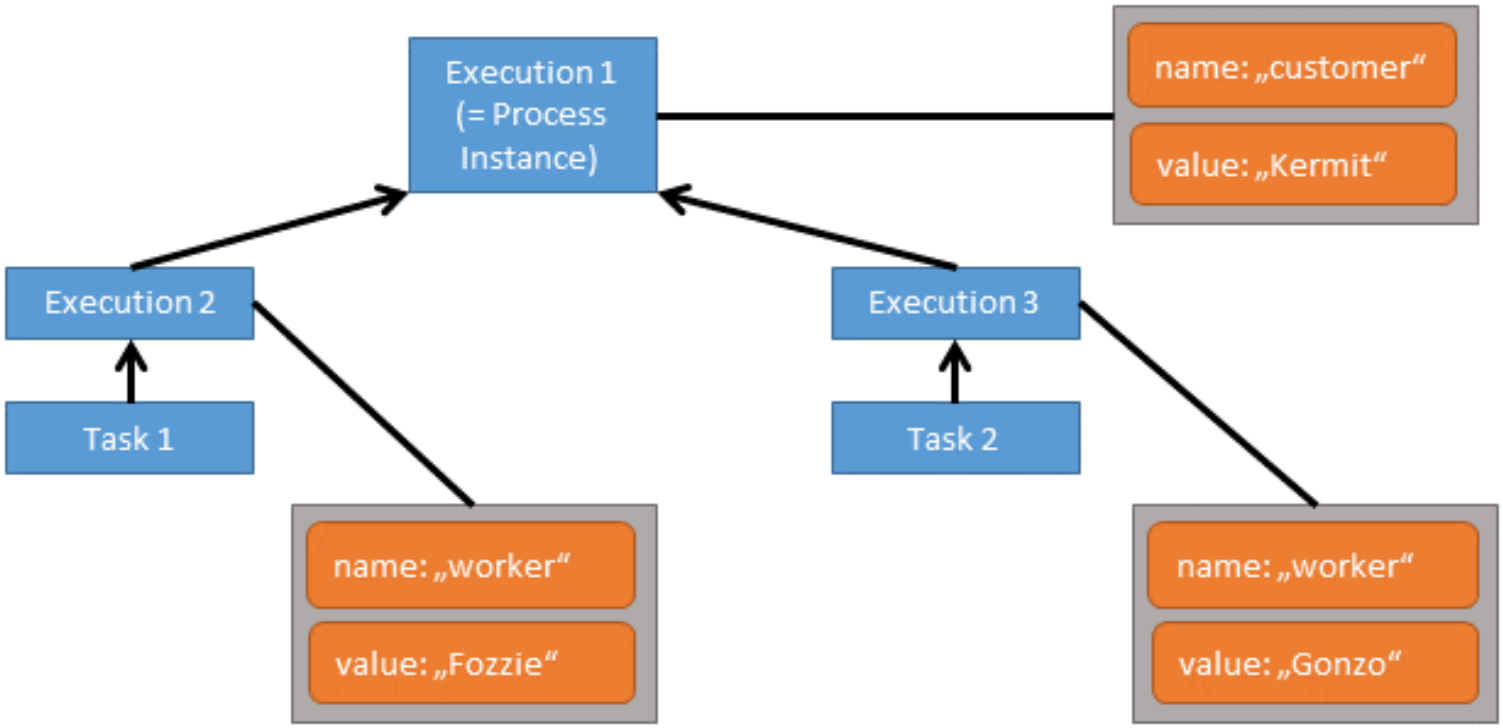


Variable Scopes and Variable Visibility

- There is a process instance with two child executions, each of which has created a task.
- All these five entities are variable scopes, and the arrows mark a parent-child relationship.
- A variable that is defined on a parent scope is accessible in every child scope unless a child scope defines a variable of the same name.
- The other way around, child variables are not accessible from a parent scope.
- Variables that are directly attached to the scope in question are called local variables.

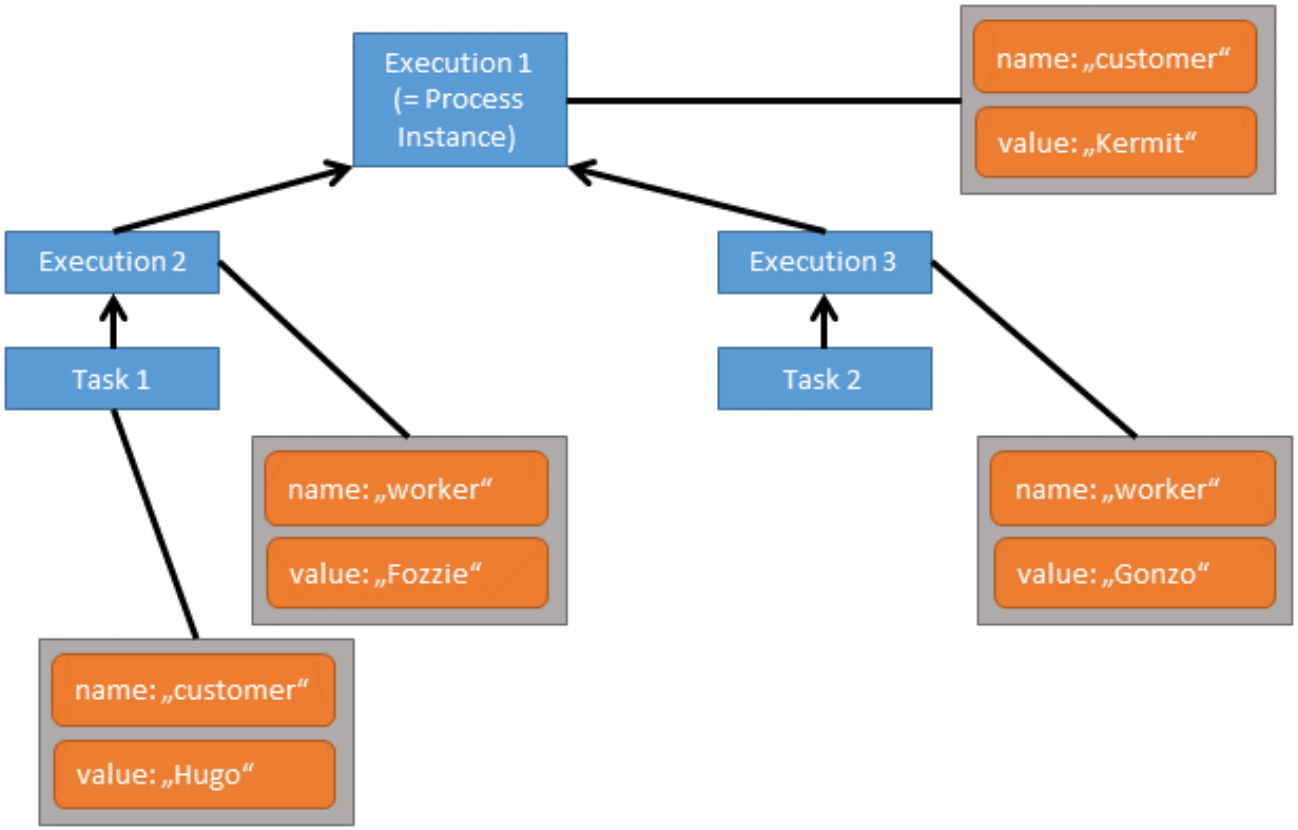
Variable Scopes and Variable Visibility

Consider the following assignment of variables to scopes:

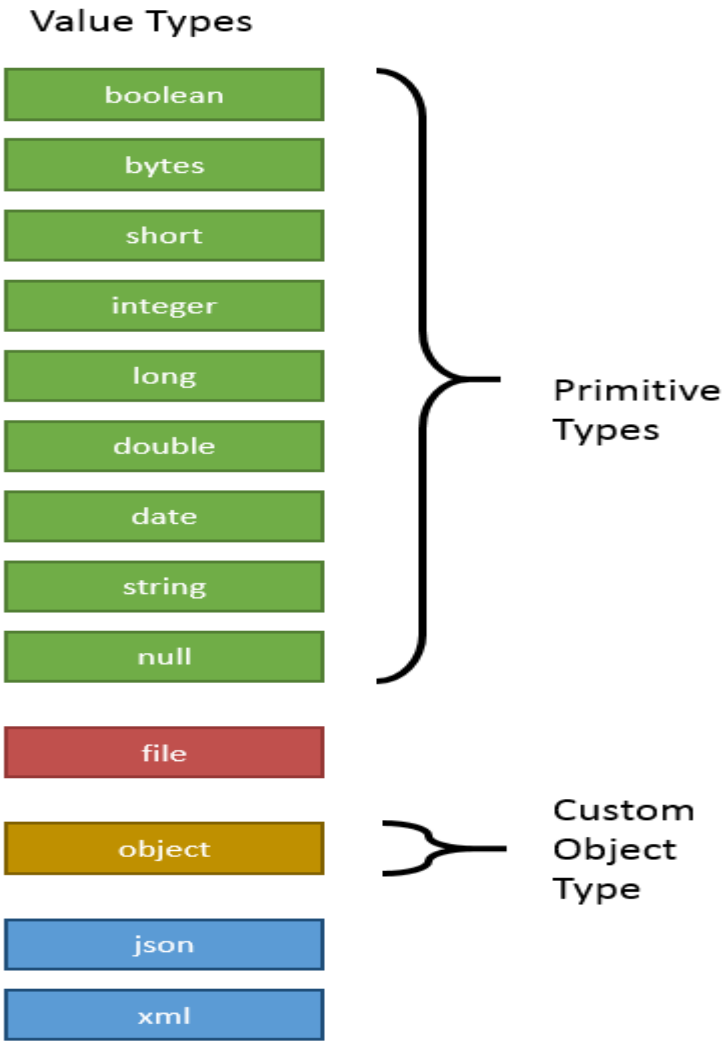


Variable Scopes and Variable Visibility

Consider the following assignment of variables to scopes:



Supported Variable Values

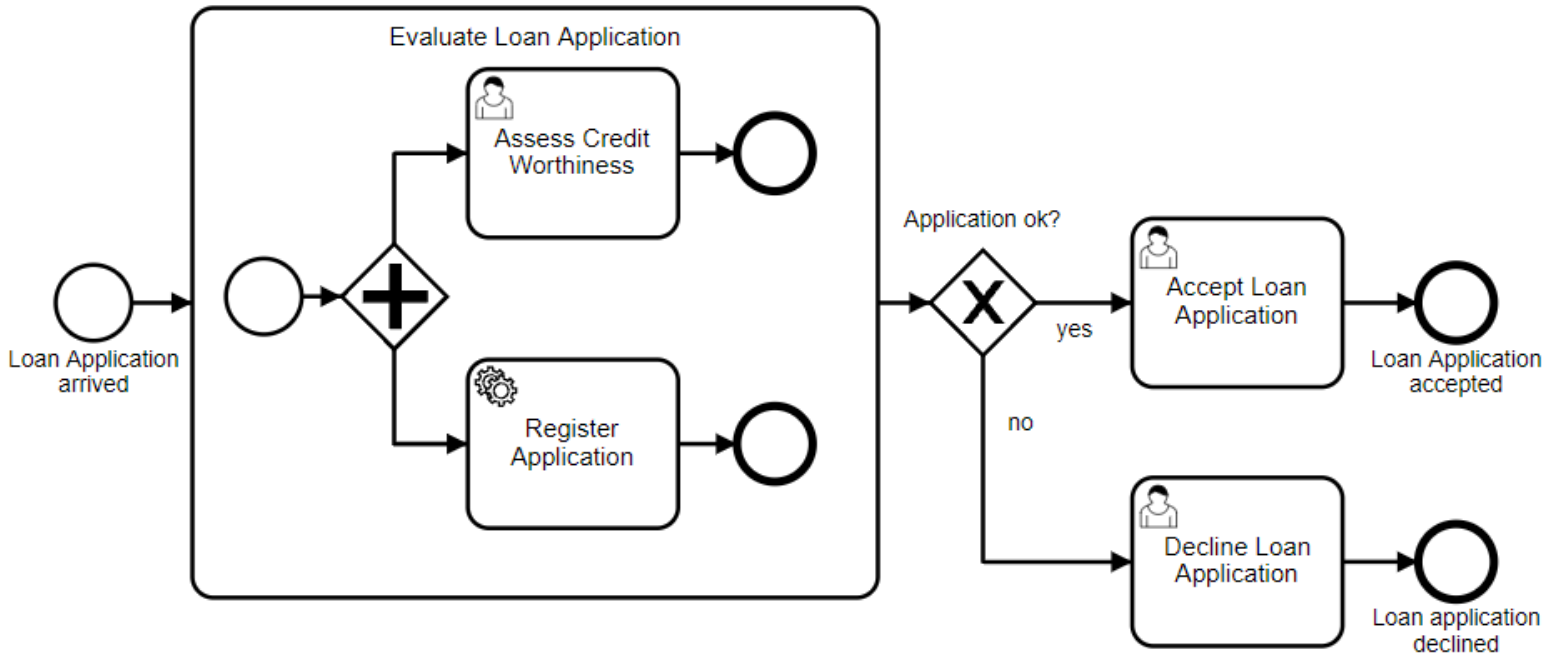


Variable Scopes and Variable Visibility

- // Java Object API: Get Variable
- Integer val1 = (Integer) execution.getVariable("val1");
- // Typed Value API: Get Variable
- IntegerValue typedVal2 = execution.getVariableTyped("val2");
- Integer val2 = typedVal2.getValue();
- Integer diff = val1 - val2;
- // Java Object API: Set Variable
- execution.setVariable("diff", diff);
- // Typed Value API: Set Variable
- IntegerValue typedDiff = Variables.integerValue(diff);
- execution.setVariable("diff", typedDiff);

Process Instance Modification by Example

As an example, consider the following process model:




```
ProcessInstance processInstance = runtimeService.createProcessInstanceQuery().singleResult();
runtimeService.createProcessInstanceModification(processInstance.getId())
  .startBeforeActivity("acceptLoanApplication")
  .cancelAllForActivity("declineLoanApplication")
  .execute();
```

Process Instance Modification by Example


```
ProcessInstance processInstance = runtimeService.createProcessInstanceQuery().singleResult();
runtimeService.createProcessInstanceModification(processInstance.getId())
  .startBeforeActivity("acceptLoanApplication")
  .setVariable("approver", "joe")
  .cancelAllForActivity("declineLoanApplication")
  .execute();
```

to start at the start event of the subprocess:

```
ProcessInstance processInstance = runtimeService.createProcessInstanceQuery().singleResult();
runtimeService.createProcessInstanceModification(processInstance.getId())
  .cancelAllForActivity("declineLoanApplication")
  .startBeforeActivity("subProcessStartEvent")
  .execute();
```

to start the subprocess itself:

```
ProcessInstance processInstance = runtimeService.createProcessInstanceQuery().singleResult();
runtimeService.createProcessInstanceModification(processInstance.getId())
  .cancelAllForActivity("declineLoanApplication")
  .startBeforeActivity("evaluateLoanApplication")
  .execute();
```

Process Instance Modification by Example

to start the process' start event:

```
ProcessInstance processInstance = runtimeService.createProcessInstanceQuery().singleResult();
runtimeService.createProcessInstanceModification(processInstance.getId())
  .cancelAllForActivity("declineLoanApplication")
  .startBeforeActivity("processStartEvent")
  .execute();
```

Delegation Code

- Delegation Code allows you to execute external Java code, evaluate expressions or scripts when certain events occur during process execution.
- There are different types of Delegation Code:
  - Java Delegates can be attached to a BPMN ServiceTask.
  - Execution Listeners can be attached to any event within the normal token flow, e.g. starting a process instance or entering an activity.
  - Task Listeners can be attached to events within the user task lifecycle, e.g. creation or completion of a user task.

Java Delegate


```
  public class ToUppercase implements JavaDelegate {

    public void execute(DelegateExecution execution) throws Exception {
      String var = (String) execution.getVariable("input");
      var = var.toUpperCase();
      execution.setVariable("input", var);
    }

}
```

Field Injection

- It is possible to inject values into the fields of the delegated classes. The following types of injection are supported:
  - Fixed string values
  - Expressions

```
public class ReverseStringsFieldInjected implements JavaDelegate {

  private Expression text1;
  private Expression text2;

  public void execute(DelegateExecution execution) {
    String value1 = (String) text1.getValue(execution);
    execution.setVariable("var1", new StringBuffer(value1).reverse().toString());

    String value2 = (String) text2.getValue(execution);
    execution.setVariable("var2", new StringBuffer(value2).reverse().toString());
  }
}
```

Database Schema

- The database schema of the process engine consists of multiple tables.
  - The table names all start with ACT.
  - The second part is a two-character identification of the use case of the table.

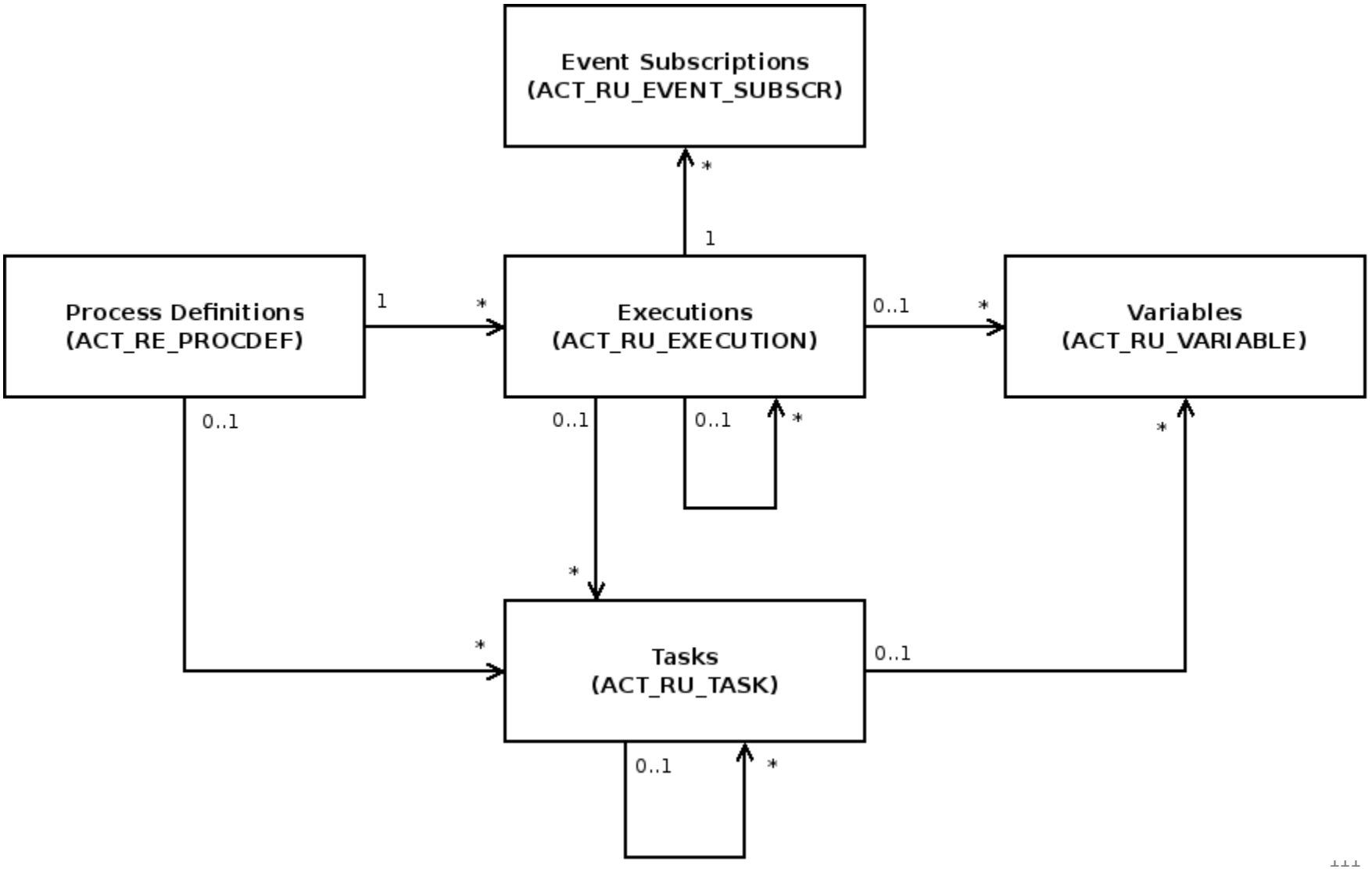
Database Schema

- **ACT_RE_*: RE** stands for repository.
  - Tables with this prefix contain ‘static’ information such as process definitions and process resources (images, rules, etc.).
- **ACT_RU_*: RU** stands for runtime.
  - These are the runtime tables that contain the runtime data of process instances, user tasks, variables, jobs, etc.
  - The engine only stores the runtime data during process instance execution and removes the records when a process instance ends.
  - This keeps the runtime tables small and fast.

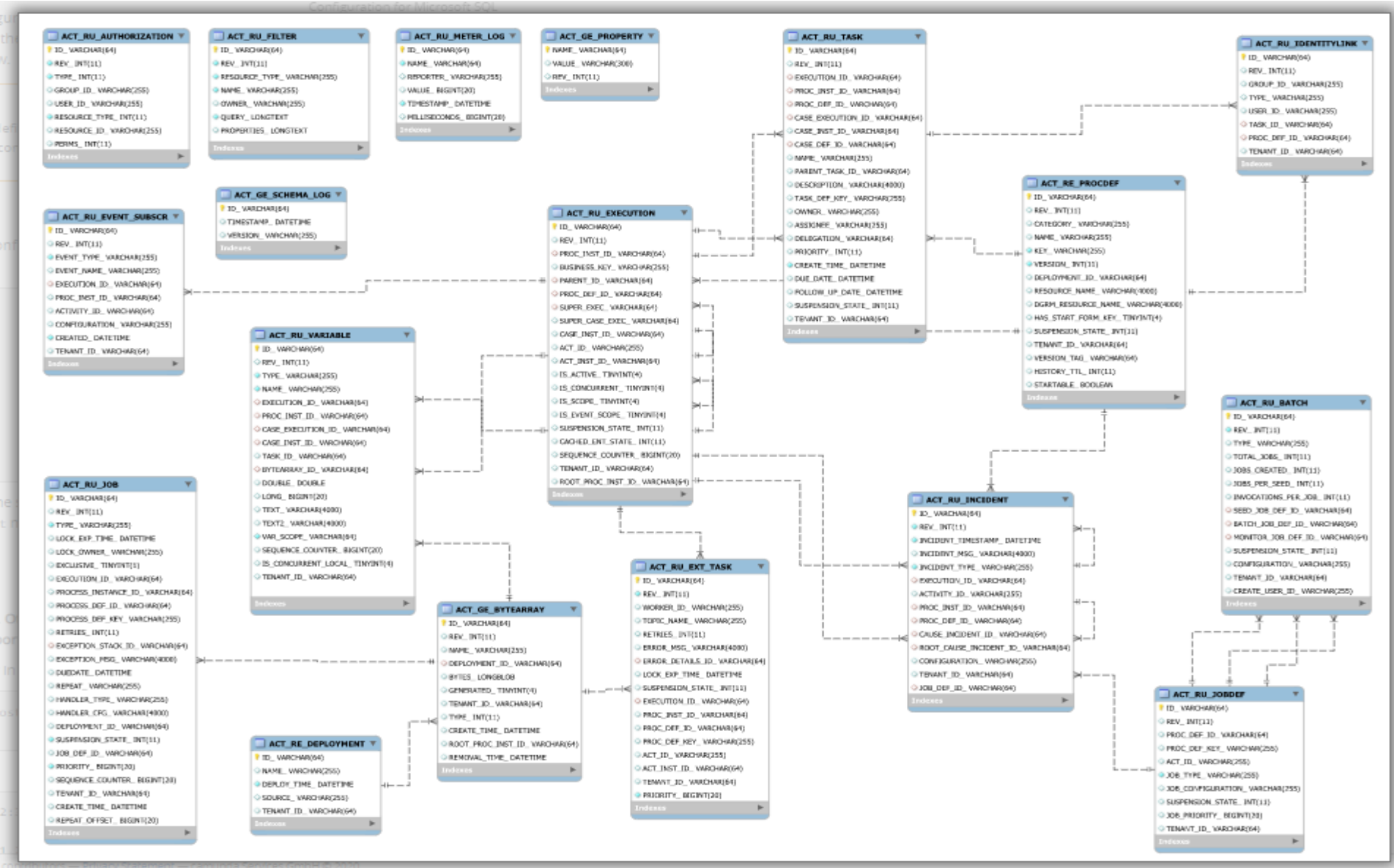
Database Schema

- **ACT_ID_*: ID stands for identity.**
  - These tables contain identity information such as users, groups, etc.
- **ACT_HI_*: HI stands for history.**
  - These are the tables that contain historical data such as past process instances, variables, tasks, etc.
- **ACT_GE_*:** General data, which is used in various use cases.

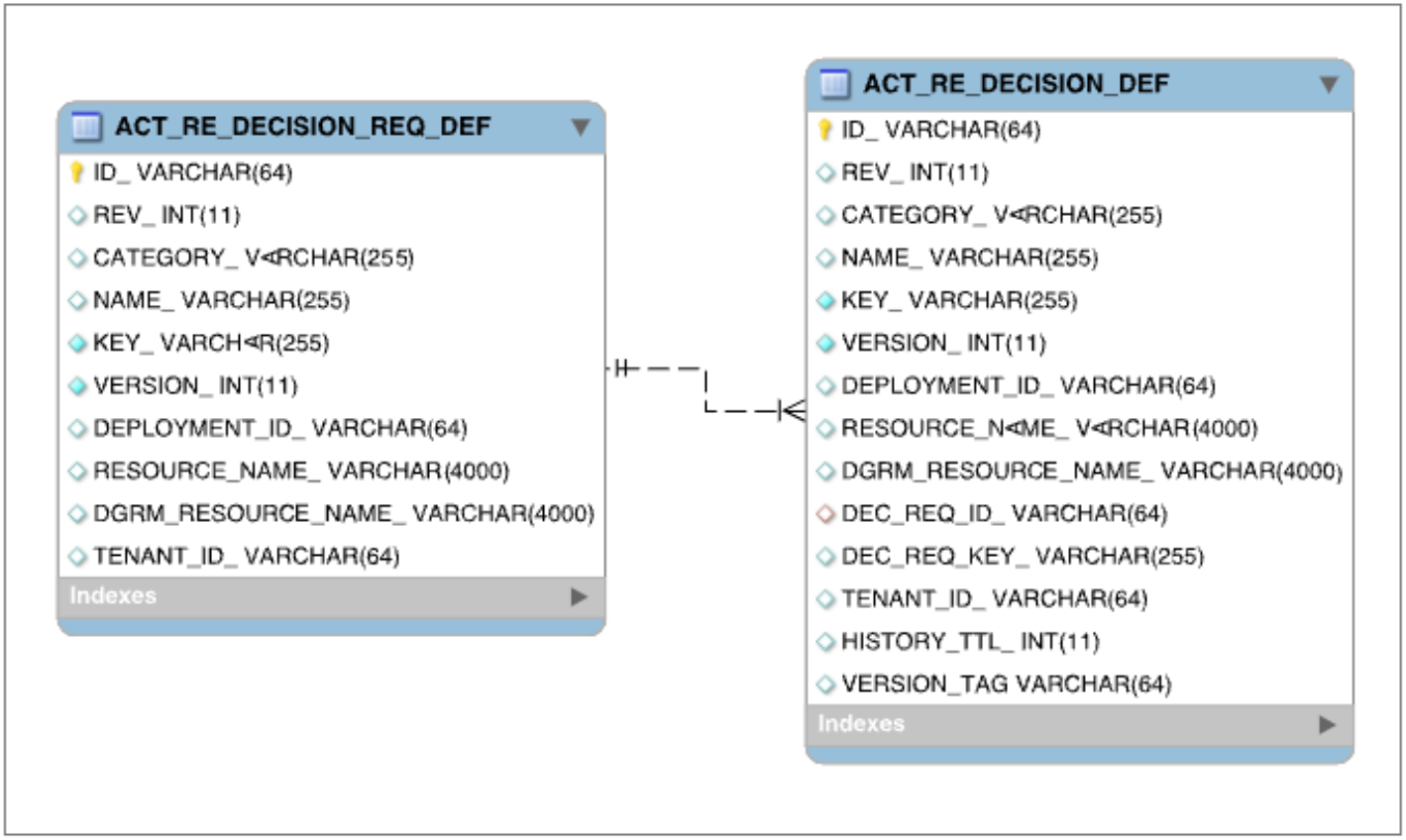
Database Schema



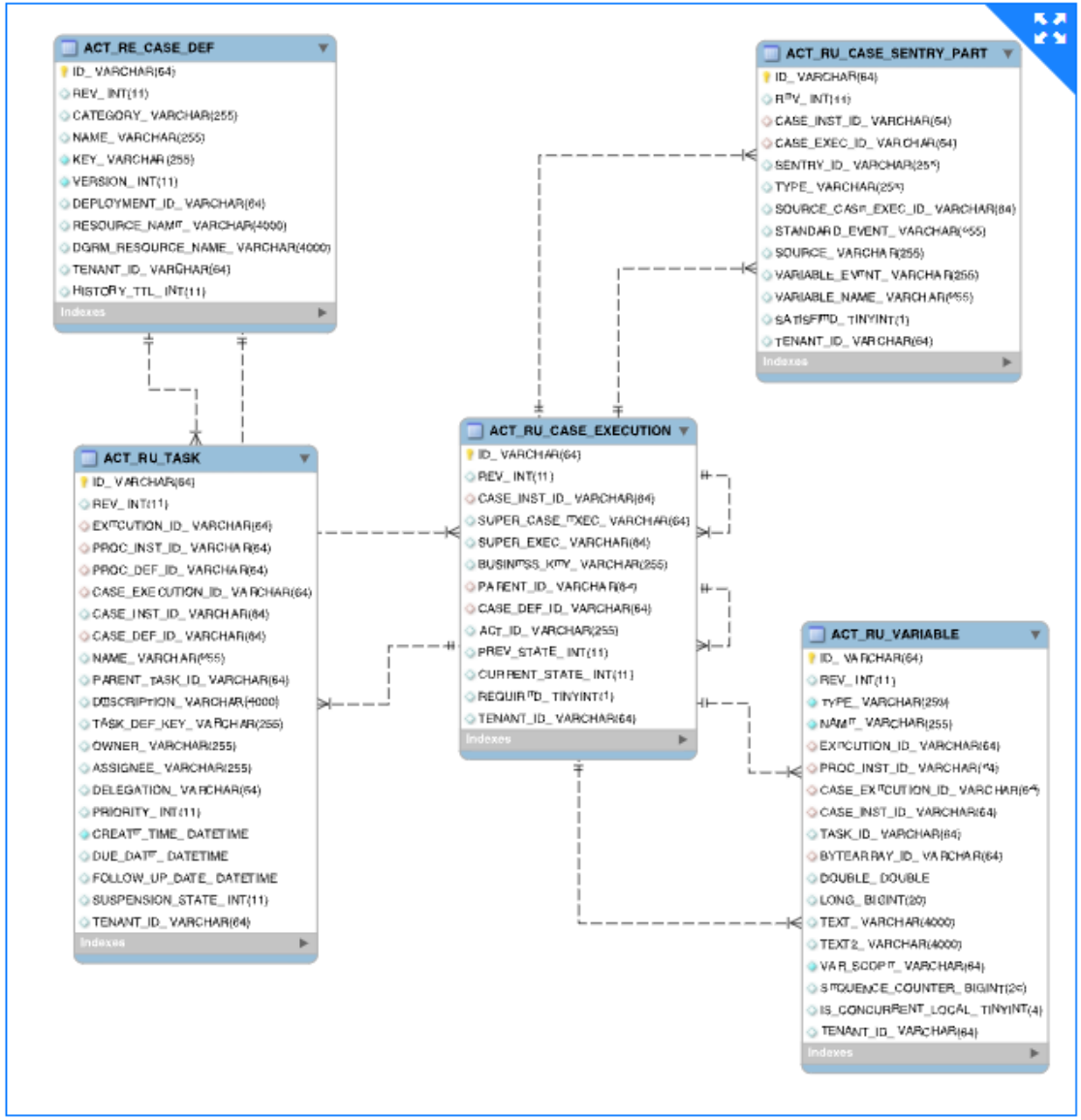
DB Structure (Engine BPMN)



DB Structure (Engine DMN)



DB Structure (Engine CMMN)



Database Configuration

Example database configuration

```
<property name="jdbcUrl" value="jdbc:h2:mem:camunda;DB_CLOSE_DELAY=1000" />
<property name="jdbcDriver" value="org.h2.Driver" />
<property name="jdbcUsername" value="sa" />
<property name="jdbcPassword" value="" />
```

Alternatively, a `javax.sql.DataSource` implementation can be used (e.g., DBCP from Apache Commons):

```
<bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource" >
  <property name="driverClassName" value="com.mysql.jdbc.Driver" />
  <property name="url" value="jdbc:mysql://localhost:3306/camunda?sendFractionalSeconds=false"
  <property name="username" value="camunda" />
  <property name="password" value="camunda" />
  <property name="defaultAutoCommit" value="false" />
</bean>

<bean id="processEngineConfiguration" class="org.camunda.bpm.engine.impl.cfg.StandaloneProcessE

    <property name="dataSource" ref="dataSource" />
    ...
```

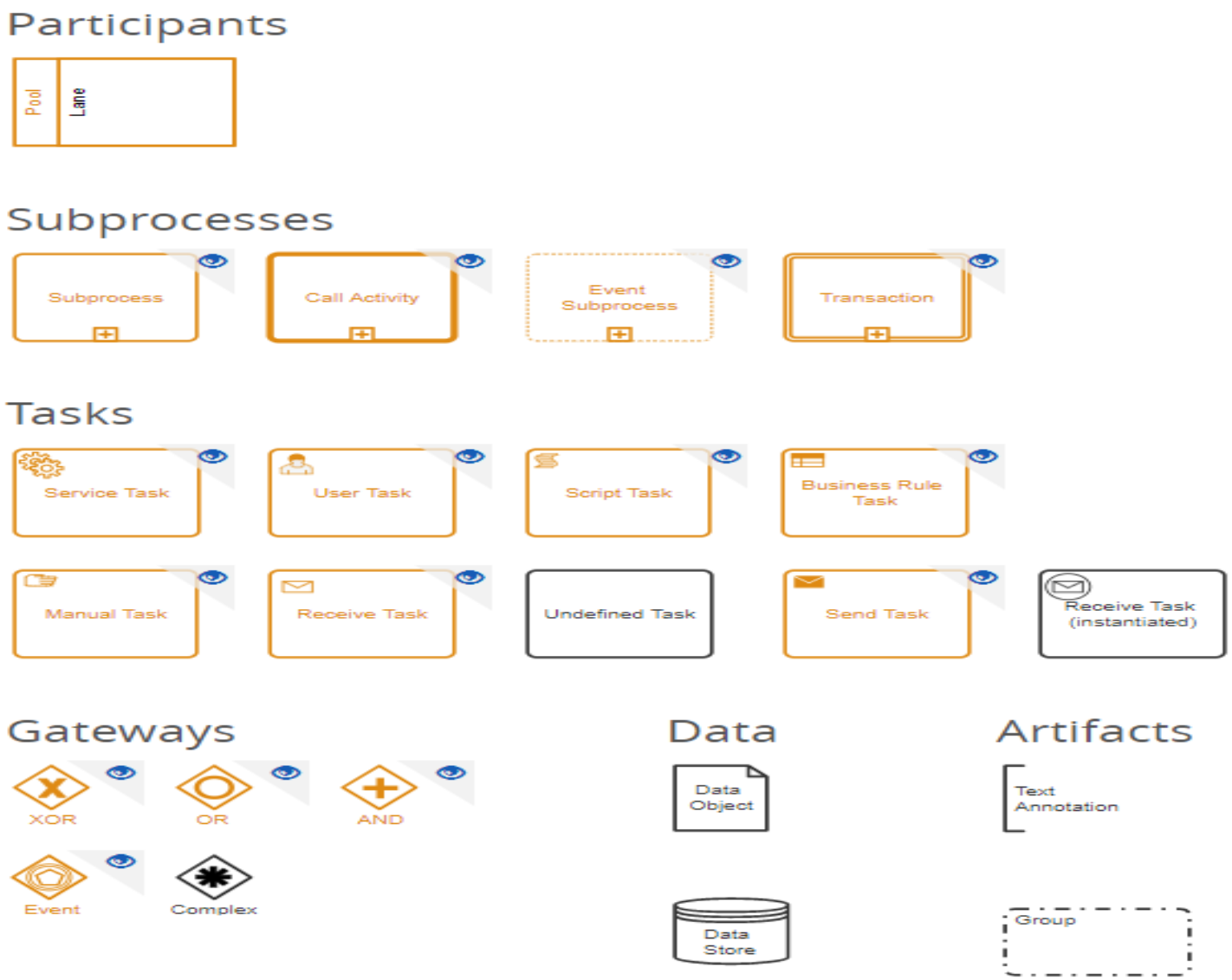
Model API

- BPMN Model API
- CMMN Model API
- DMN Model API

BPMN

- Business Process Model and Notation (BPMN) is a standard for Workflow and Process Automation. Camunda supports the 2.0 version of BPMN.

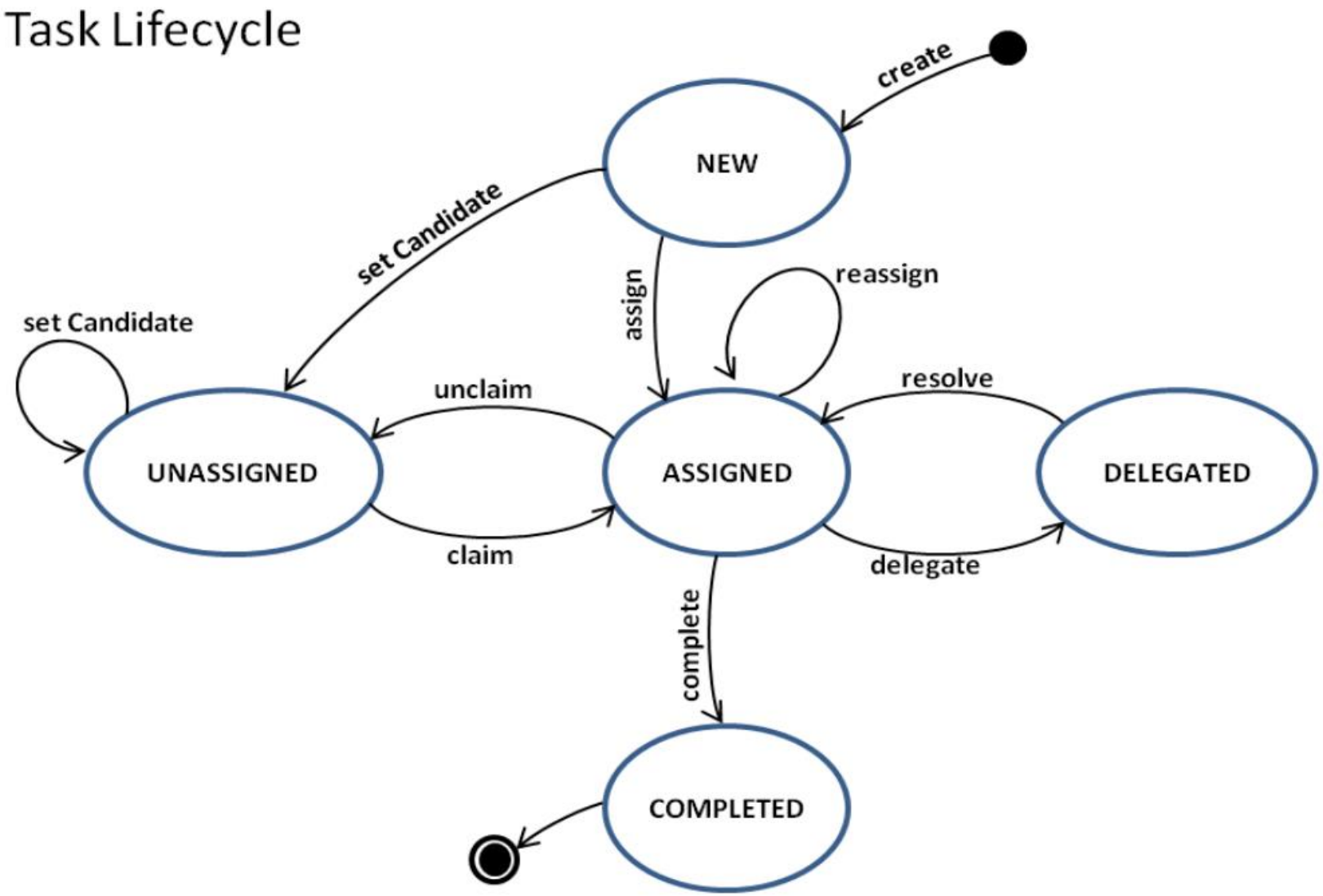
BPMN Symbols



Tasks

- Service Task
  - Invoke or execute business logic.
- Send Task
  - Send a message.
- User Task
  - A task performed by a human participant.
- Business Rule Task
  - Execute an automated business decision.
- Script Task
  - Execute a Script.
- Receive Task
  - Wait for a message to arrive.
- Manual Task
  - A task which is performed externally.
- Task Markers
  - Markers control operational aspects like repetition.

Task Lifecycle



Service Task

- A Service Task is used to invoke services.
- In Camunda this is done by calling Java code or providing a work item for an external worker to complete asynchronously.

Service Task

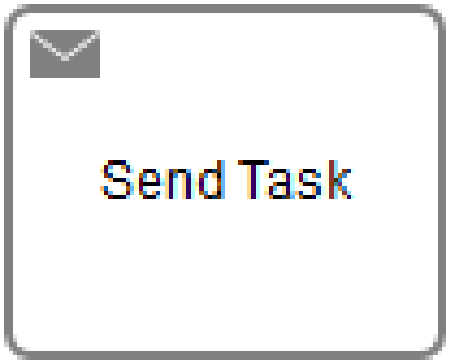
- Calling Java Code
- There are four ways of declaring how to invoke Java logic:
  - Specifying a class that implements a Java Delegate or Activity Behavior
  - Evaluating an expression that resolves to a delegation object
  - Invoking a method expression
  - Evaluating a value expression

External Tasks

- When a Service Task is declared external, the process engine offers a work item to workers that independently poll the engine for work to do.
- This decouples the implementation of tasks from the process engine and allows to cross system and technology boundaries.
- Calling Java code, where the process engine synchronously invokes Java logic, External Task contrast to this.

Send Task

- A Send Task is used to send a message. In Camunda this is done by calling Java code.
- The Send Task has the same behavior as a Service Task.
- <sendTask id="sendTask" camunda:class="org.camunda.bpm.MySendTaskDelegate" />



User Task

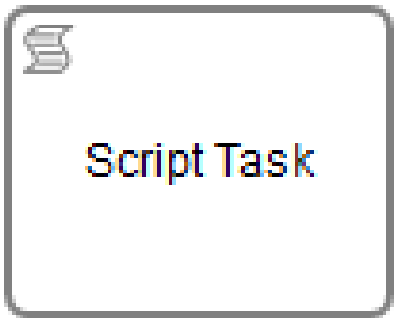
- A User Task is used to model work that needs to be done by a human actor.
- When the process execution arrives at such a User Task, a new task is created in the task list of the user(s) or group(s) assigned to that task.
- <userTask id="theTask" name="Important task" />

Business Rule Task

- A Business Rule Task is used to synchronously execute one or more rules.
- <businessRuleTask id="businessRuleTask"
- camunda:decisionRef="myDecision"
- camunda:decisionRefBinding="version"
- camunda:decisionRefVersion="12" />

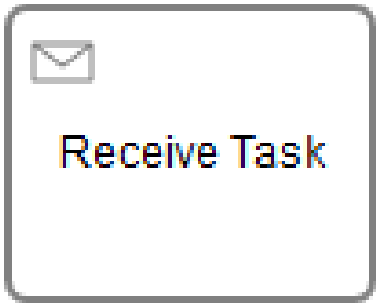
Script Task

- A Script Task is an automated activity. When a process execution arrives at the Script Task, the corresponding script is executed.
- <scriptTask id="theScriptTask" name="Execute script" scriptFormat="groovy">
- <script>
- sum = 0
- for (i in inputArray) {
- sum += i
- }
- </script>
- </scriptTask>



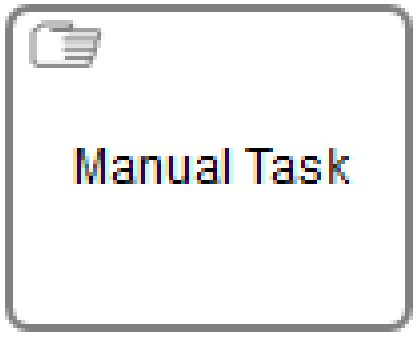
Receive Task

- A Receive Task is a simple task that waits for the arrival of a certain message.
- When the process execution arrives at a Receive Task, the process state is committed to the persistence storage.
- This means that the process will stay in this wait state until a specific message is received by the engine, which triggers continuation of the process beyond the Receive Task.



Manual Task

- A Manual Task defines a task that is external to the BPM engine.
- It is used to model work that is done by somebody who the engine does not need to know of and is there no known system or UI interface.
- For the engine, a manual task is handled as a pass-through activity, automatically continuing the process at the moment the process execution arrives at it.

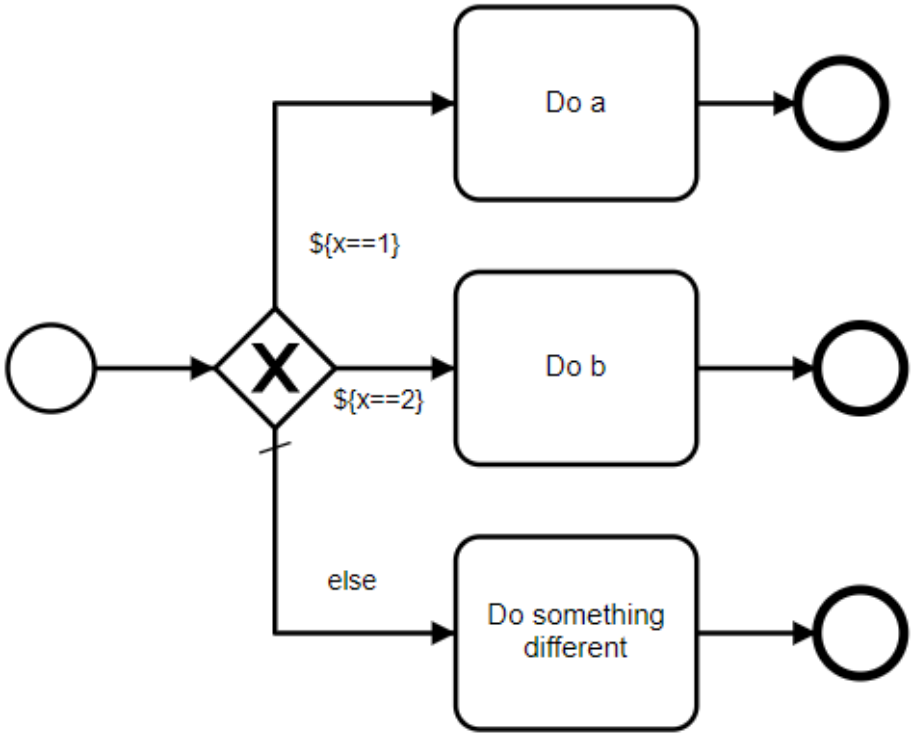



```
<manualTask id="myManualTask" name="Manual Task" />
```

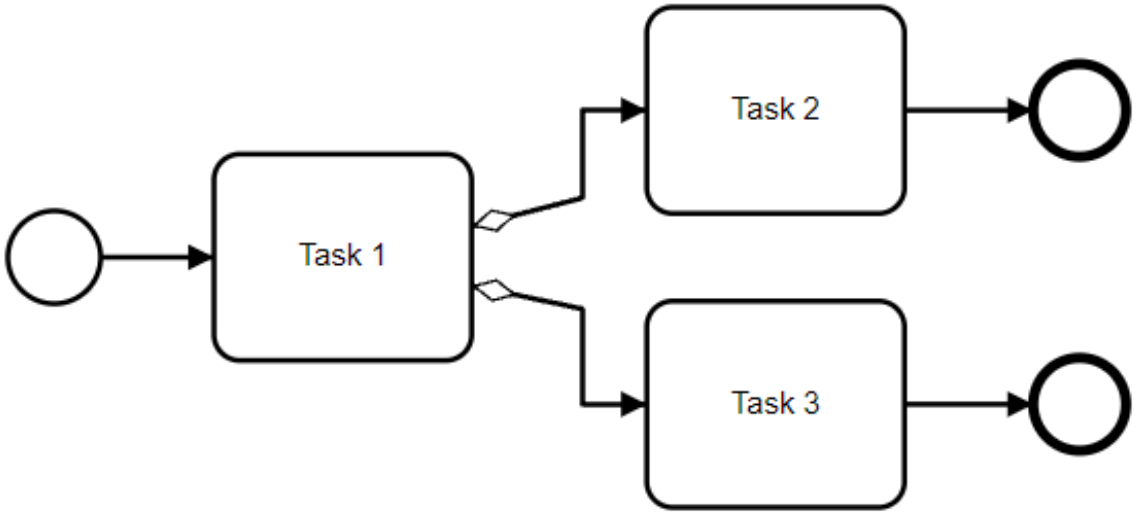
Gateways

- Gateways control token flow in a process.
- They allow modeling decisions based on data and events as well as fork / join concurrency.
  - Data-based Exclusive Gateway (XOR)
    - Model decisions based on data.
  - Conditional and Default Sequence Flows
    - Concurrency and decisions without Gateways.
  - Parallel Gateway
    - Model fork / join concurrency.
  - Inclusive Gateway
    - Model conditional fork / join concurrency.
  - Event-based Gateway
    - Model decisions based on events.

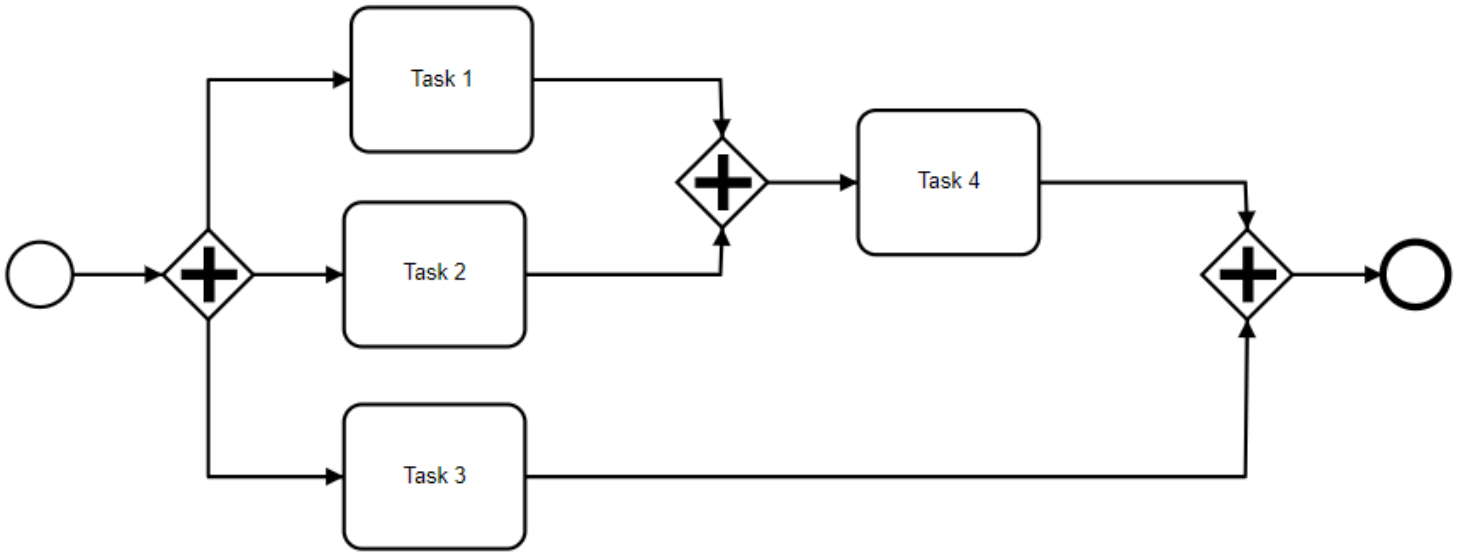
Data-based Exclusive Gateway (XOR)



Conditional and Default Sequence Flows



Parallel Gateway



Inclusive Gateway

- The Inclusive Gateway can be seen as a combination of an exclusive and a parallel gateway.
- Like an exclusive gateway, you can define conditions on outgoing sequence flows and the inclusive gateway will evaluate them.
- However, the main difference is that the inclusive gateway can receive more than one sequence flow, like a parallel gateway.

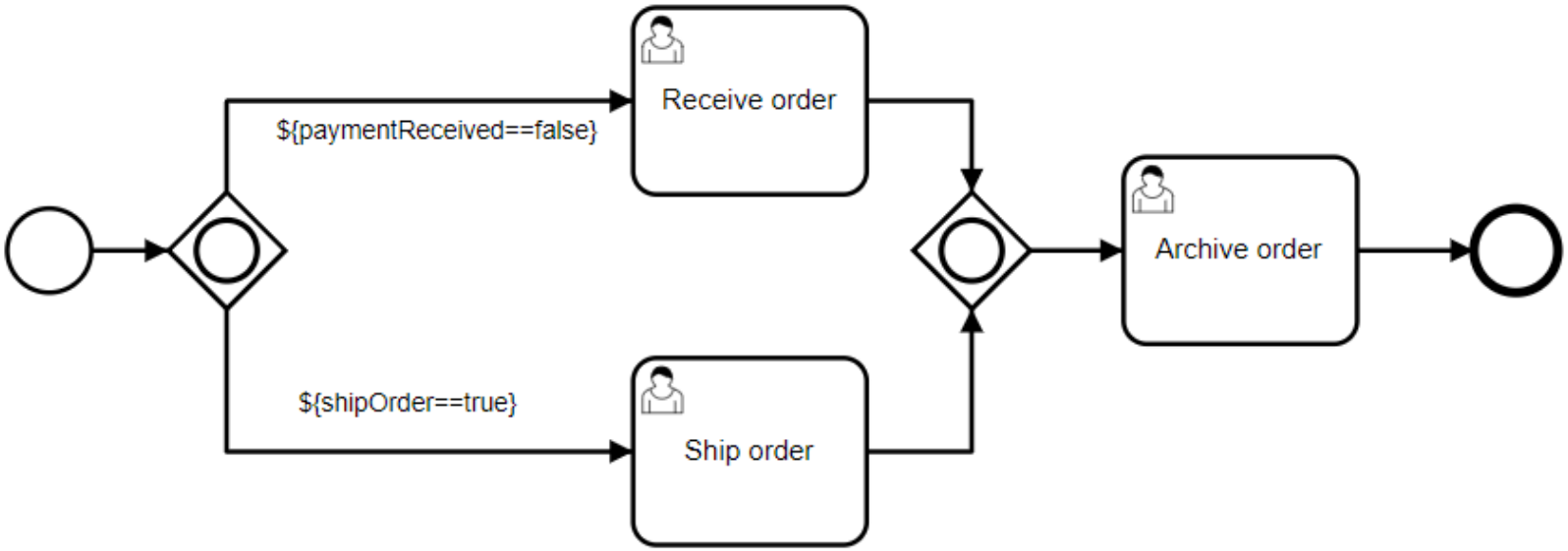
Inclusive Gateway

- The functionality of the inclusive gateway is based on the incoming and outgoing sequence flows:
  - fork: all outgoing sequence flow conditions are evaluated and for the sequence flow conditions that evaluate to ‘true’, the flows are followed in parallel, creating one concurrent execution for each sequence flow.

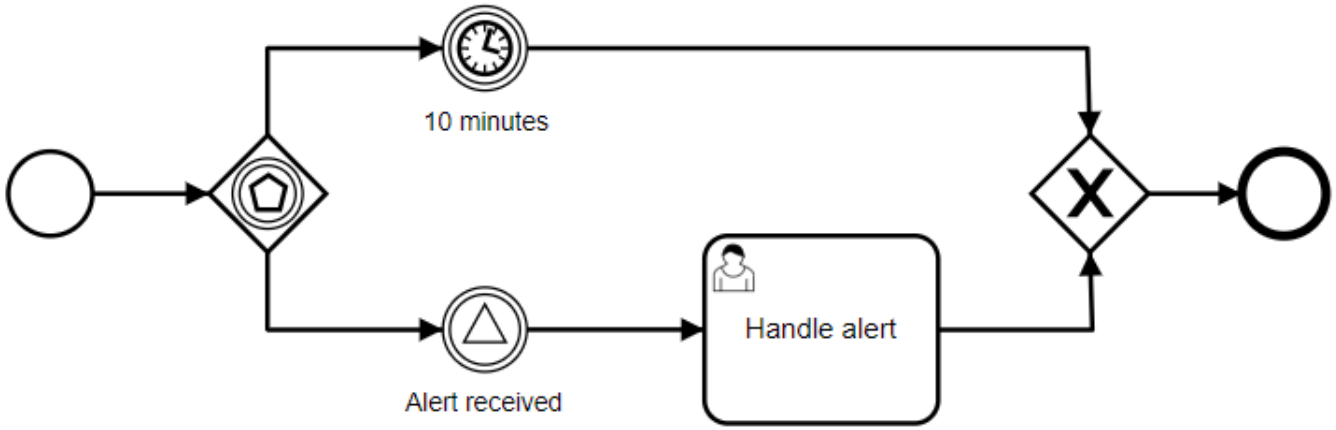
Inclusive Gateway

- The functionality of the inclusive gateway is based on the incoming and outgoing sequence flows:
  - join: all concurrent executions arriving at the inclusive gateway wait at the gateway until an execution has arrived for each of the incoming sequence flows that have a process token.
  - This is an important difference to the parallel gateway.
    - the inclusive gateway will only wait for the incoming sequence flows that are executed.
    - After the join, the process continues past the joining inclusive gateway.

Inclusive Gateway



Event-based Gateway



Events

Type	Start			Intermediate				End
	Normal	Event Subprocess	Event Subprocess non-interrupt	catch	boundary	boundary non-interrupt	throw	
None								
Message								
Timer								
Conditional								
Link								
Signal								
Error								
Escalation								

Events

Termination								
Compensation								
Cancel								
Multiple								
Multiple Parallel								

Events

- Start events define where a Process or Sub Process starts.
- The process engine supports different types of start events:
  - Blank
  - Timer
  - Message
  - Signal
  - Conditional

None Events (Blank)

- 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');

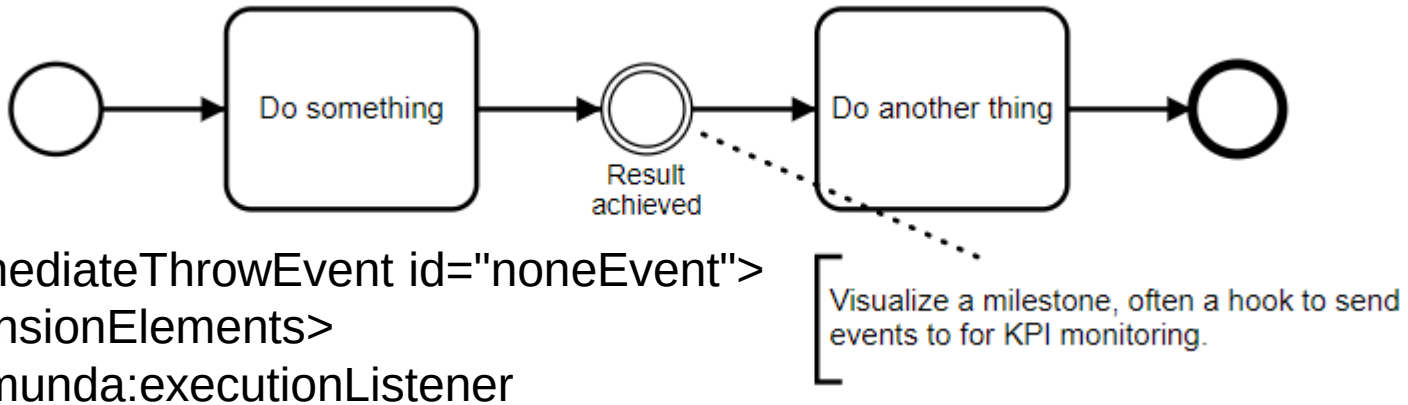
None Events (Blank)

None End Event

- A ‘none’ end event means that the result thrown when the event is reached is unspecified.
- As such, the engine will not do anything besides ending the current path of execution.
- The XML representation of a none end event is the normal end event declaration, without any sub-element (other end event types all have a sub-element declaring the type).
- <endEvent id="end" name="my end event" />

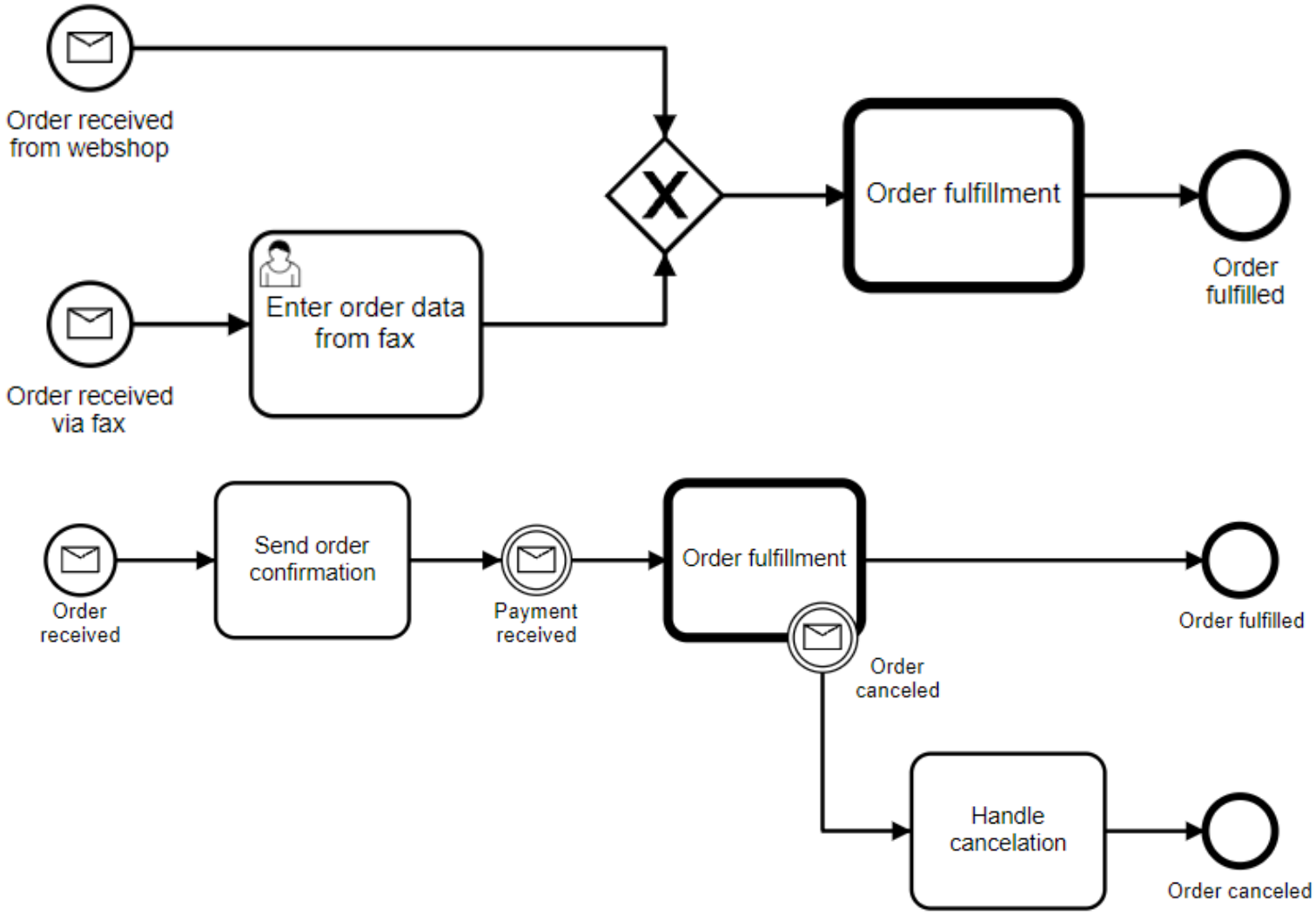
Intermediate None Event (throwing)

- The following process diagram shows a simple example of an intermediate none event, which is often used to indicate some state achieved in the process.

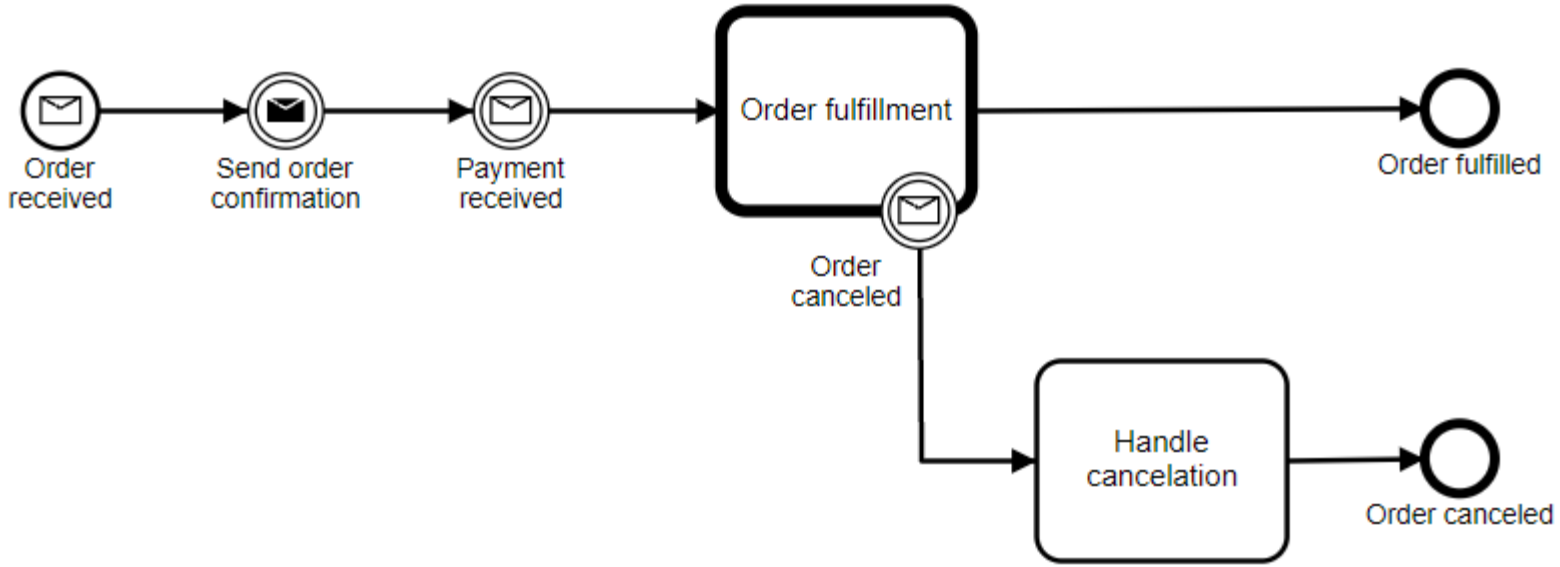



```
<intermediateThrowEvent id="noneEvent">
  <extensionElements>
    <camunda:executionListener
class="org.camunda.bpm.engine.test.bpm
n.event.IntermediateNoneEventTest$MyEx
ecutionListener" event="start" />
  </extensionElements>
</intermediateThrowEvent>
```

Message Events

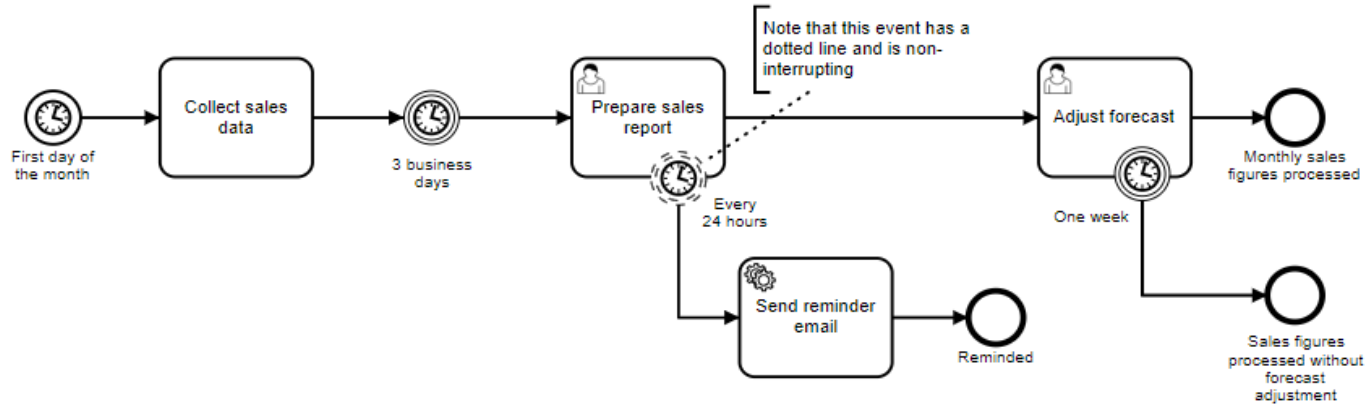


Message Events



Timer Events

- Timer events are events which are triggered by a defined timer.
- They can be used as start event, intermediate event or boundary event. Boundary events can be interrupting or not.



Defining a Timer

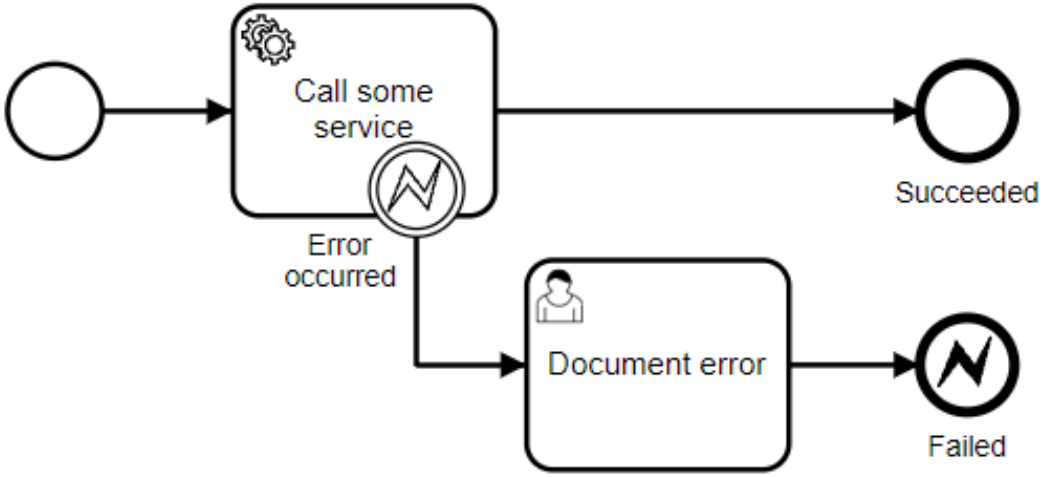
- <timerEventDefinition>
- <timeDate>2011-03-11T12:13:14Z</timeDate>
- </timerEventDefinition>

Time Duration

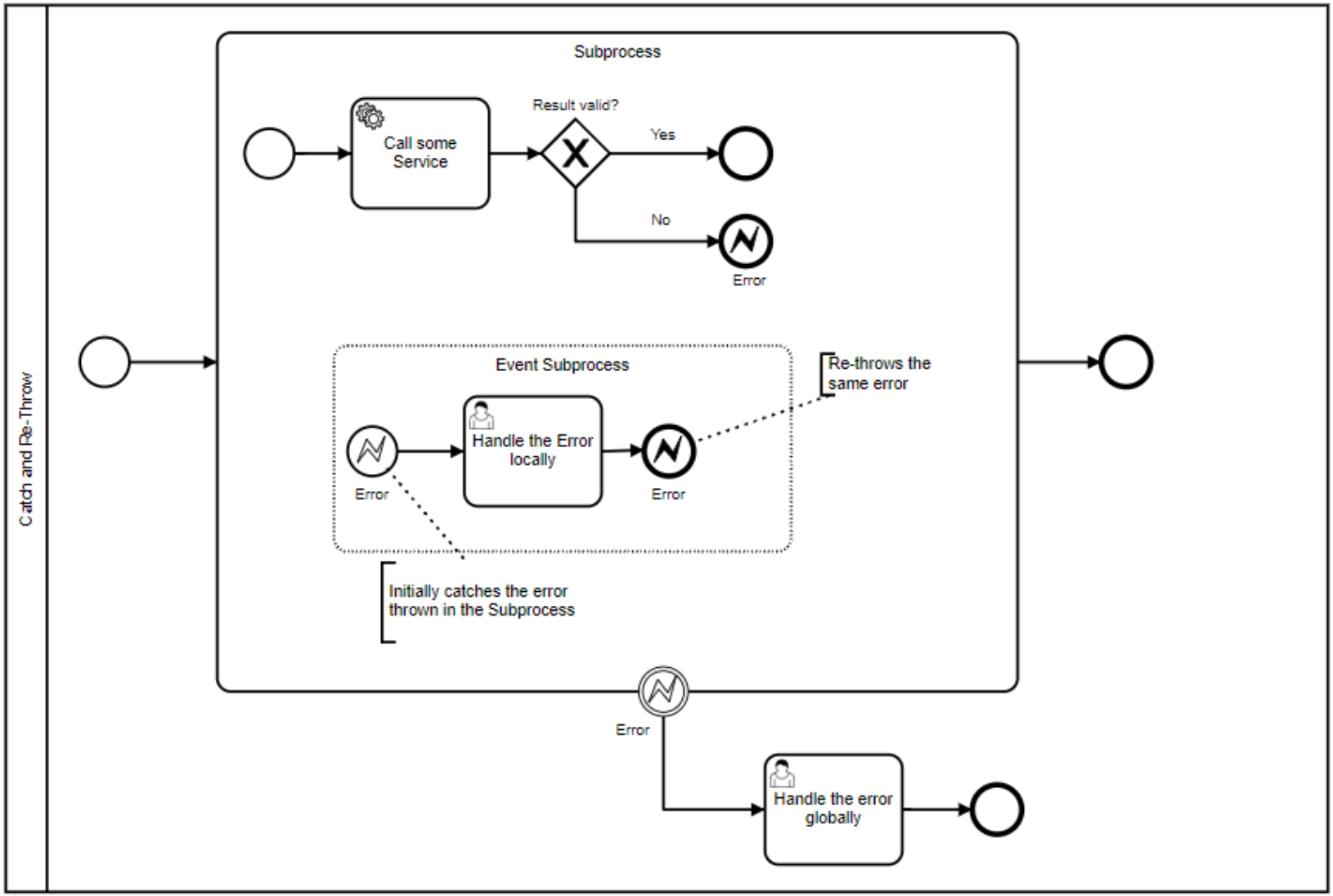
- To specify how long the timer should run before it is fired, a timeDuration can be specified as a sub-element of timerEventDefinition.
- It is possible to define the duration in two different ISO 8601 Durations formats:
- PnYnMnDTnHnMnS
- PnW
- <timerEventDefinition>
- <timeDuration>P10D</timeDuration>
- </timerEventDefinition>

Error Events

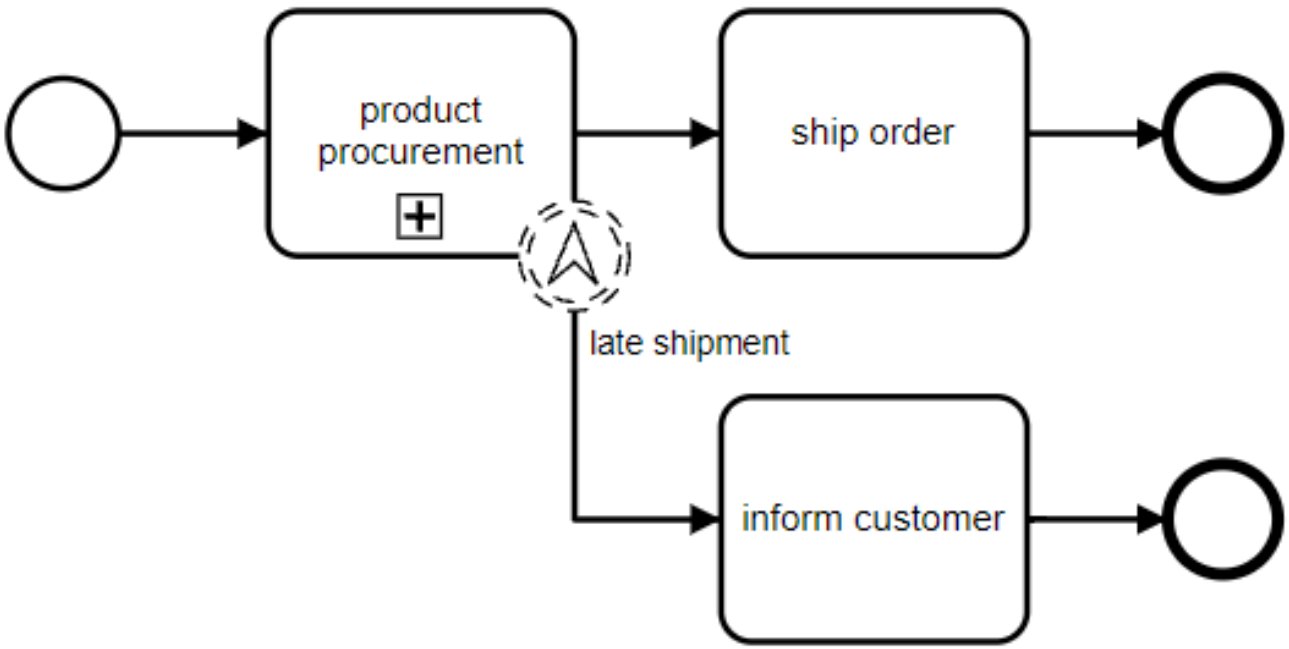
Error events are events which are triggered by a defined error.



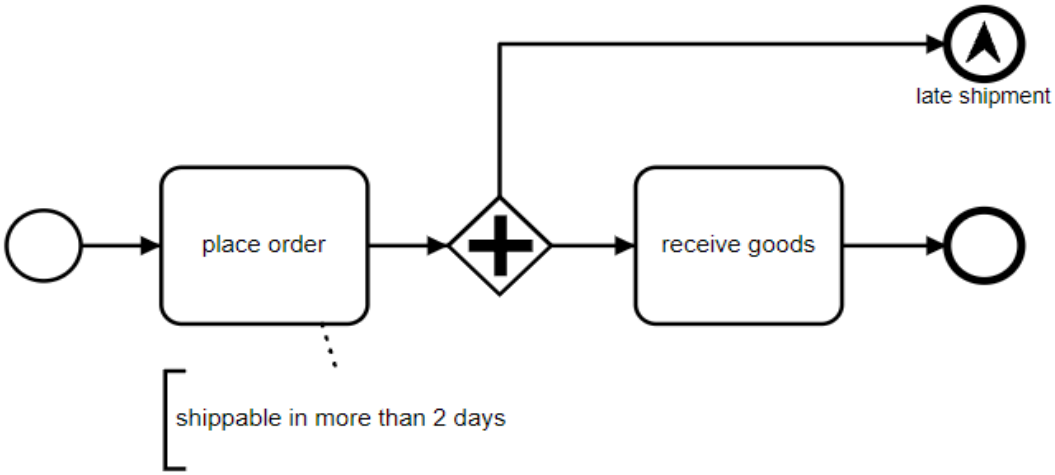
Error Events



Escalation Events



Escalation Events




```
<endEvent id="throwEscalation" name="late shipment">
  <escalationEventDefinition escalationRef="lateShipment" />
</endEvent>
```

Subprocess

- A subprocess in Camunda allows modeling based on reusability and grouping.
- Below are the different types of subprocesses supported by Camunda.
- **Embedded Subprocess**
  - Group a set of flow nodes into a scope.
- **Call Activity**
  - Call a process from another process.
- **Event Subprocess**
  - Event-driven subprocess.
- **Transaction Subprocess**
  - Model Business Transactions.

Embedded Subprocess

- A subprocess is an activity that contains other activities, gateways, events, etc., which itself forms a process that is part of a bigger process.
- A subprocess is completely defined inside a parent process.

Embedded Subprocess

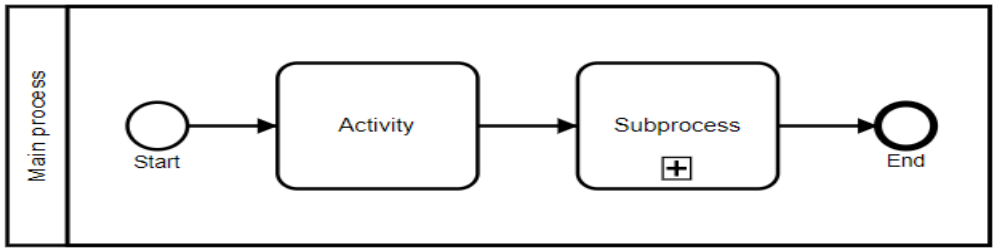
- Subprocesses have two major use cases:
  - Subprocesses allow hierarchical modeling.
  - Many modeling tools allow that subprocesses can be collapsed, hiding all the details of the subprocess and displaying a high-level, end-to-end overview of the business process.
  - A subprocess creates a new scope for events.
  - Events that are thrown during execution of the subprocess can be caught by a boundary event on the boundary of the subprocess, thus creating a scope for that event, limited to the subprocess.

Embedded Subprocess

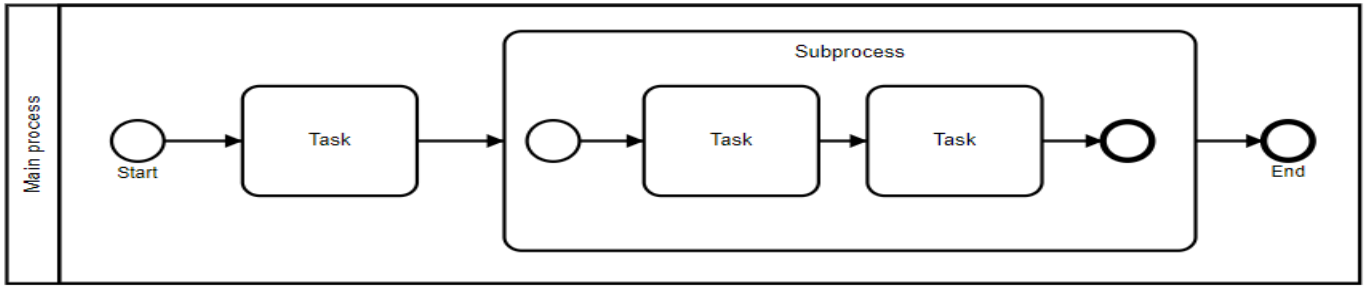
- Using a subprocess does impose some constraints:
  - A subprocess can only have one none start event; no other start event types are allowed.
  - A subprocess must have at least one end event.
  - Note that the BPMN 2.0 specification allows to omit the start and end events in a subprocess, but the current engine implementation does not support this.
  - Sequence flows can not cross subprocess boundaries.

Embedded Subprocess

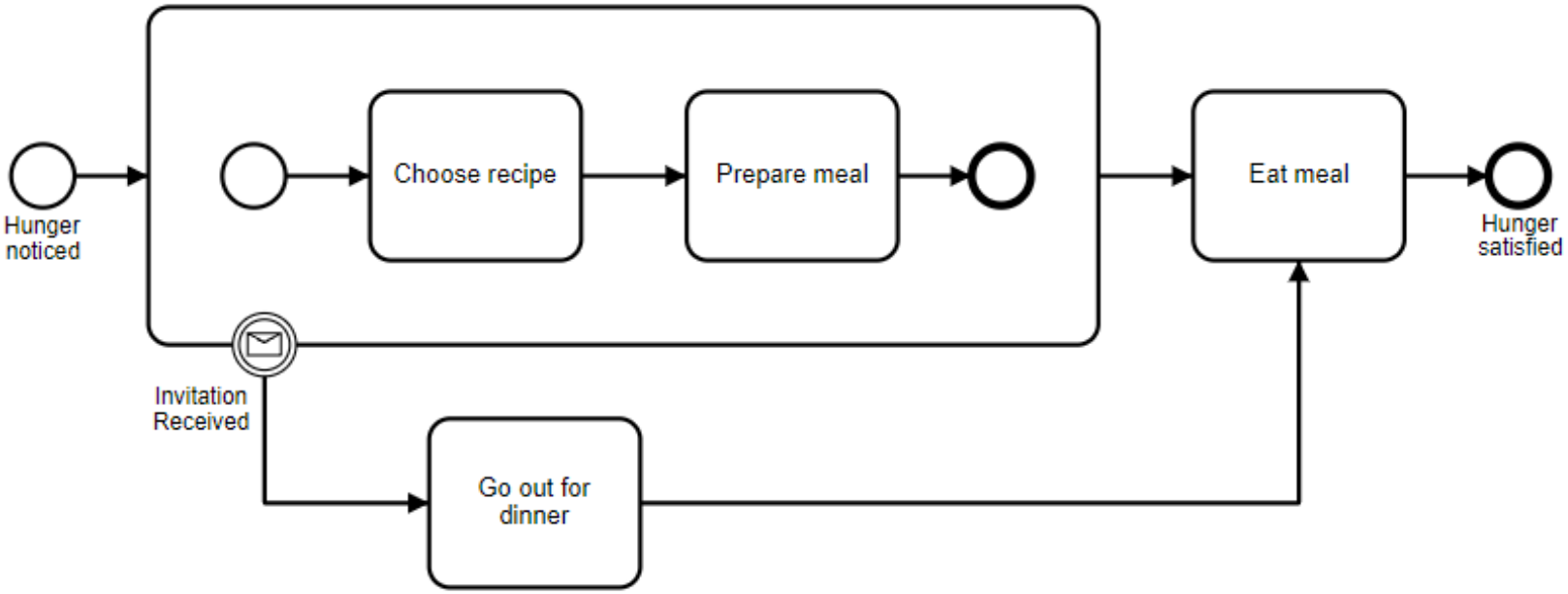
- A subprocess is visualized as a typical activity, i.e., a rounded rectangle.
- In case the subprocess is collapsed, only the name and a plus-sign are displayed, giving a high-level overview of the process:



In case the subprocess is expanded, the steps of the subprocess are displayed within the subprocess boundaries:



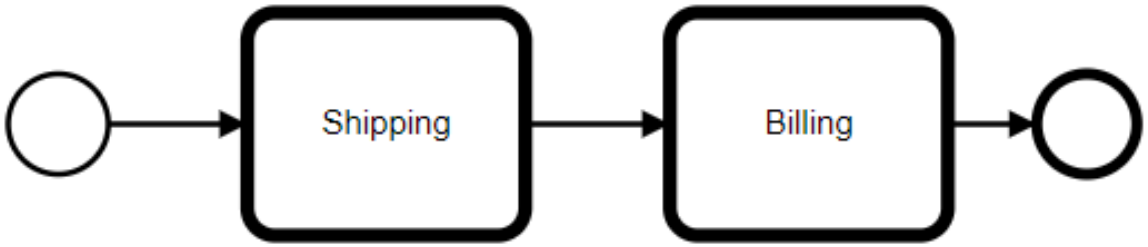
Embedded Subprocess



Call Activity

- A call activity is visualized the same way as a collapsed embedded subprocess, however with a thick border.
- Depending on the modeling tool, a call activity can also be expanded, but the default visualization is the collapsed representation.
- A call activity is a regular activity that requires a called Element which references a process definition by its key.

Call Activity




```
<callActivity id="callCheckCreditProcess" name="Check credit"
calledElement="checkCreditProcess" />
```

```
<startEvent id="theStart" />
<sequenceFlow id="flow1" sourceRef="theStart"
targetRef="shipping" /> <callActivity id="shipping"
name="Shipping" calledElement="shippingProcess" />
<sequenceFlow id="flow2" sourceRef="shipping"
targetRef="billing" />
<callActivity id="billing" name="Billing"
calledElement="billingProcess" />
<sequenceFlow id="flow3" sourceRef="billing" targetRef="end"
/>
<endEvent id="end" />
```

Transaction Subprocess

- A transaction subprocess is an embedded subprocess which can be used to group multiple activities to a transaction.
- A transaction is a logical unit of work which allows grouping of a set of individual activities, so that they either succeed or fail collectively.

Transaction Subprocess

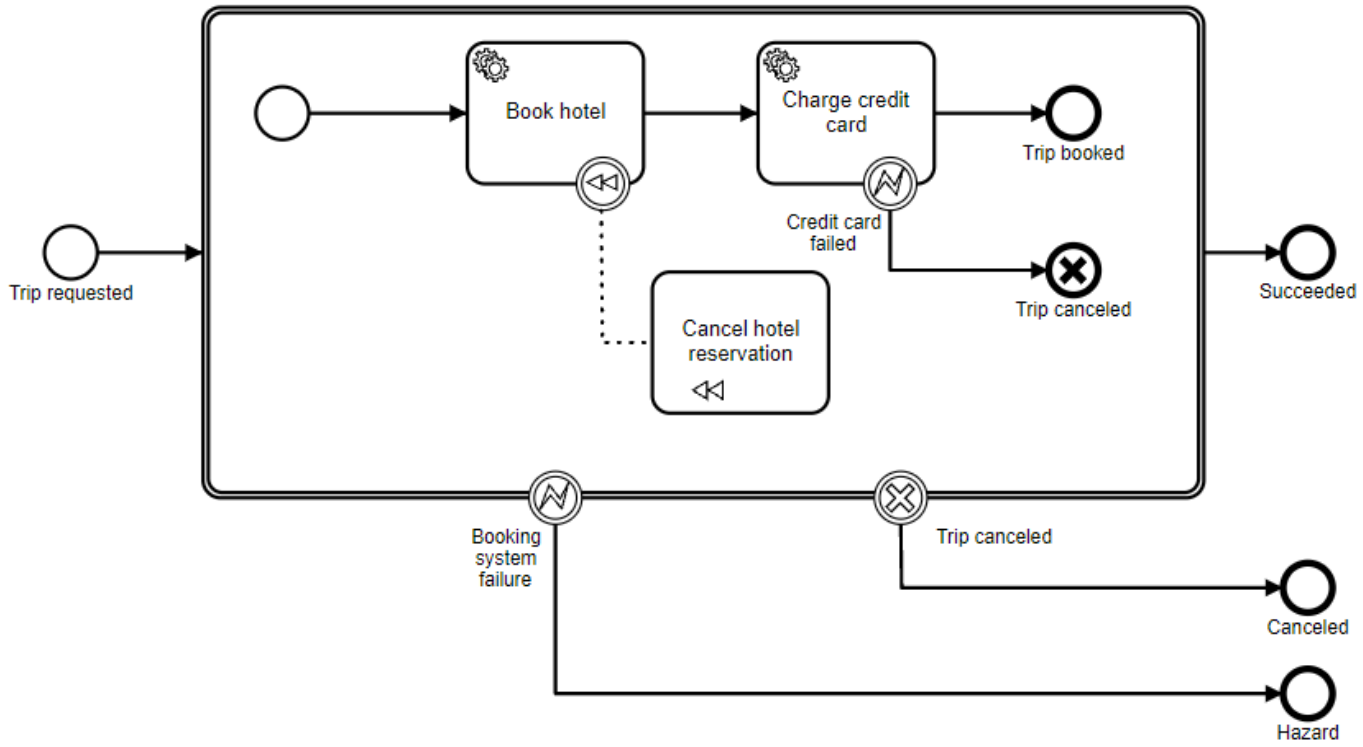
- A transaction can have three possible outcomes:
  - A transaction is successful if it is neither canceled nor terminated by a hazard.
  - If a transaction subprocess is successful, it is left using the outgoing sequence flow(s).
  - A successful transaction might be compensated if a compensation event is thrown later in the process.

Transaction Subprocess

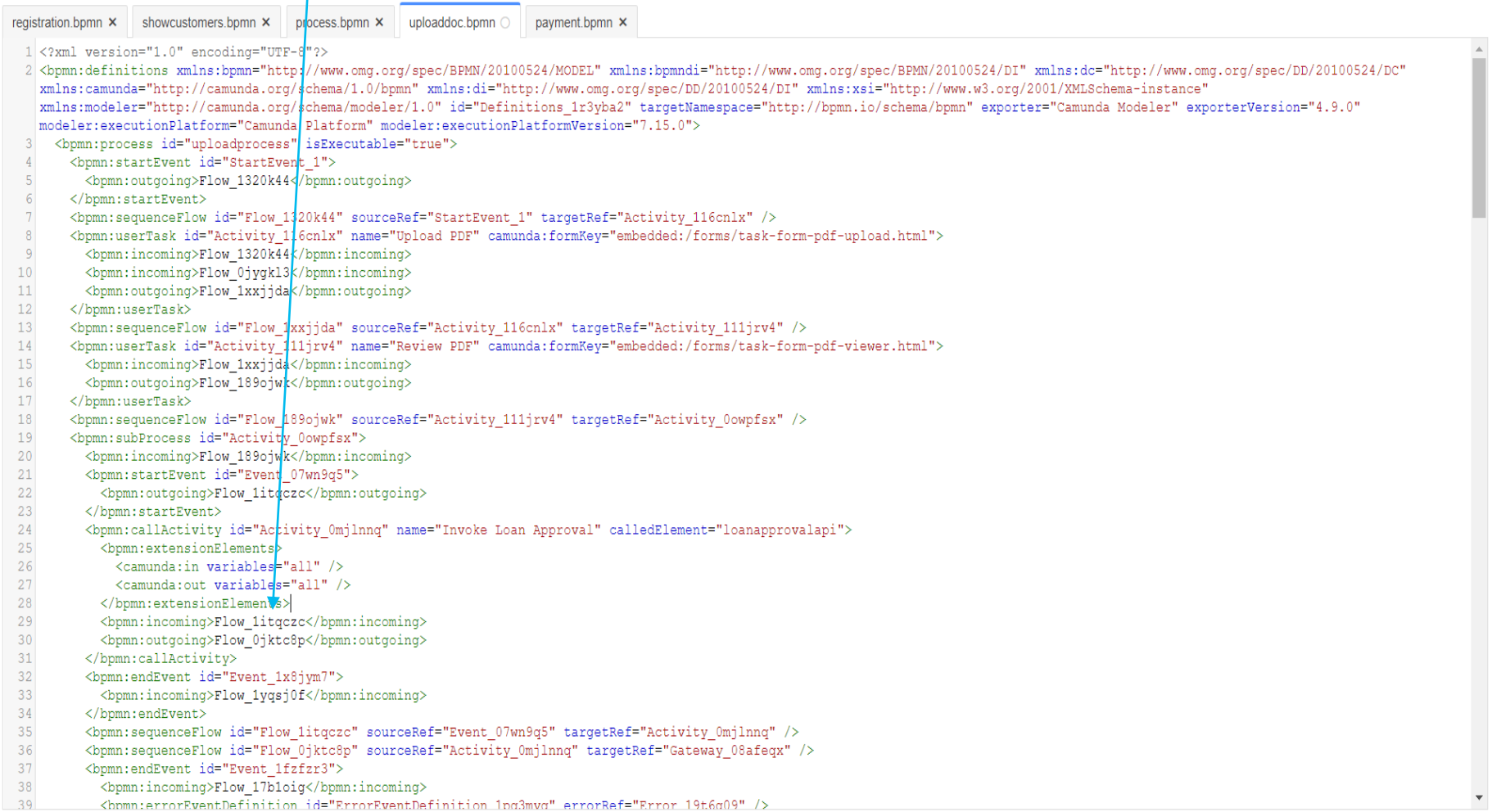
- A transaction can have three possible outcomes:
  - A transaction is canceled if an execution reaches the cancel end event.
  - In that case, all executions are terminated and removed. A single remaining execution is then set to the cancel boundary event, which triggers compensation.
  - After compensation is completed, the transaction subprocess is left, using the outgoing sequence flow(s) of the cancel boundary event.

Transaction Subprocess

- A transaction can have three possible outcomes:
  - A transaction is ended by a hazard if an error event is thrown which is not caught within the scope of the transaction subprocess.



Call Activity



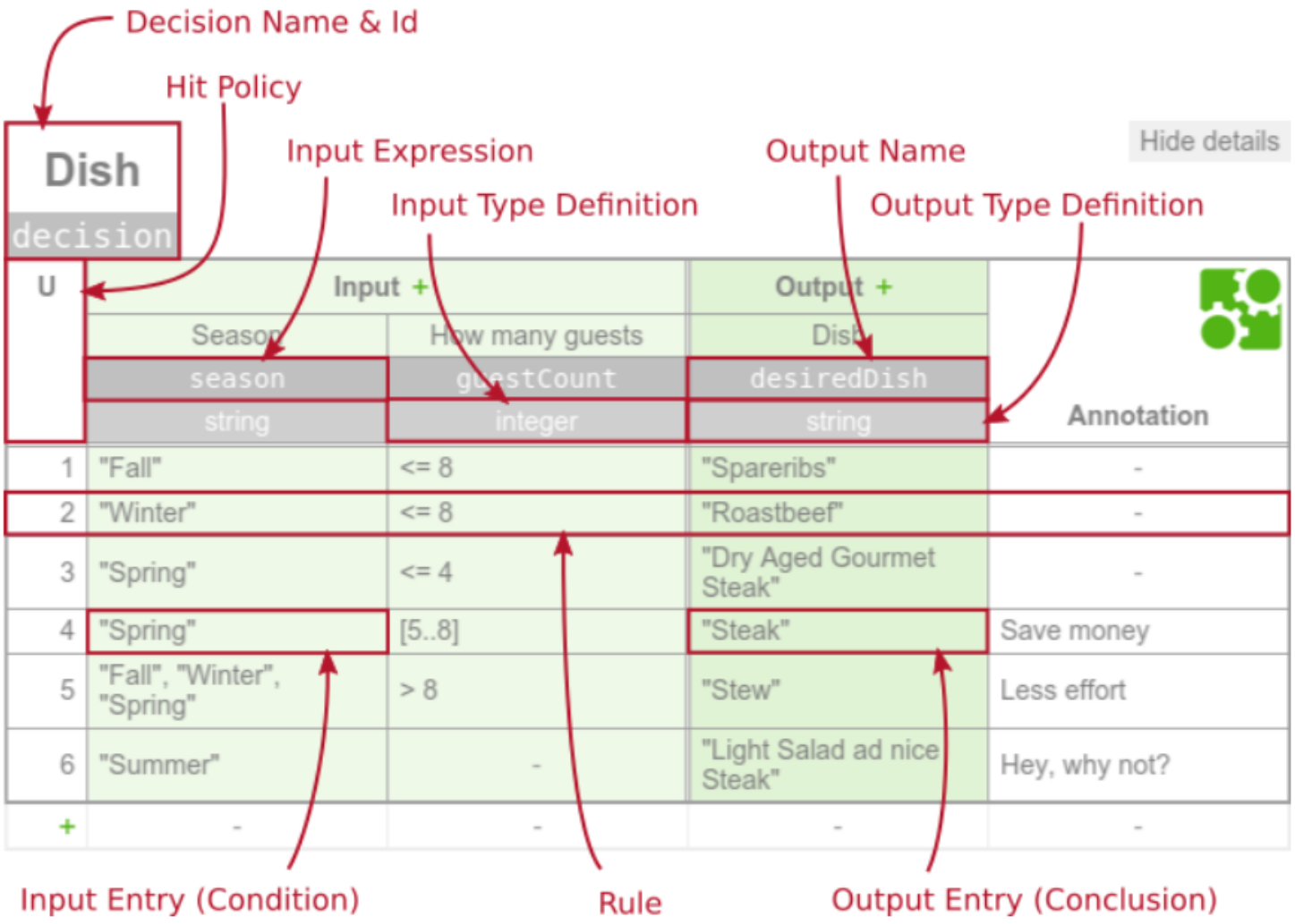
Spring Boot Query History

- http://localhost:7070/rest/history/process-instance
- http://localhost:7070/rest/history/process-instance?finishedAfter=2013-01-01T00:00:00.000+0200&finishedBefore=2013-04-01T23:59:59.000+0200&executedActivityAfter=2013-03-23T13:42:44.000+0200

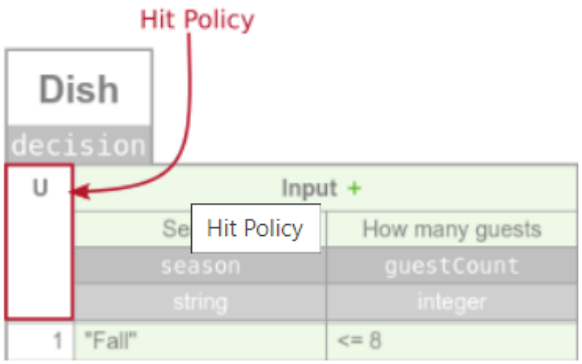
DMN

- Decision Model and Notation (DMN) is a standard for Business Decision Management.
- Currently the Camunda DMN engine partially supports DMN 1.1, including Decision Tables, Decision Literal Expressions, Decision Requirements Graphs and the Friendly Enough Expression Language (FEEL).
- https://docs.camunda.org/manual/7.15/reference/dmn/decision-table/hit-policy/

DMN Decision Table



DMN Hit Policy



A decision table has a hit policy that specifies what the results of the evaluation of a decision table consist of.

The hit policy is set in the `hitPolicy` attribute on the `decisionTable` XML element. If no hit policy is set, then the default hit policy `UNIQUE` is used.

```
<definitions xmlns="http://www.omg.org/spec/DMN/20151101/dmn.xsd" id="definitions" name="defin
  <decision id="dish" name="Dish">
    <decisionTable id="decisionTable" hitPolicy="RULE ORDER">
      <!-- .. -->
    </decisionTable>
  </decision>
</definitions>
```

DMN

Visual representation	XML representation
U	UNIQUE
A	ANY
F	FIRST
R	RULE ORDER
C	COLLECT

Unique Hit Policy

Only a single rule can be satisfied. The decision table result contains the output entries of the satisfied rule.

If more than one rule is satisfied, the Unique hit policy is violated.

See the following decision table.

Depending on the current season the dish should be chosen. Only one dish can be chosen, since only one season can exist at the same time.

Camunda

Any Hit Policy

Multiple rules can be satisfied. However, all satisfied rules must generate the same output. The decision table result contains only the output of one of the satisfied rules.

If multiple rules are satisfied which generate different outputs, the hit policy is violated.

See the following example:

Leave apply

Hit Policy: ANY

	Input +		Output +
	Vacation Days	State	result
	vacationDays	state	applicationResult
	integer	string	string
1	0	-	refused
2	> 0	probation period	refused
3	> 0	no probation period	applied
+	-	-	-

This is a decision table for the leave application. If the applier has no vacation days left or is currently in the probation period, the application will be refused. Otherwise the application is applied.

Camunda

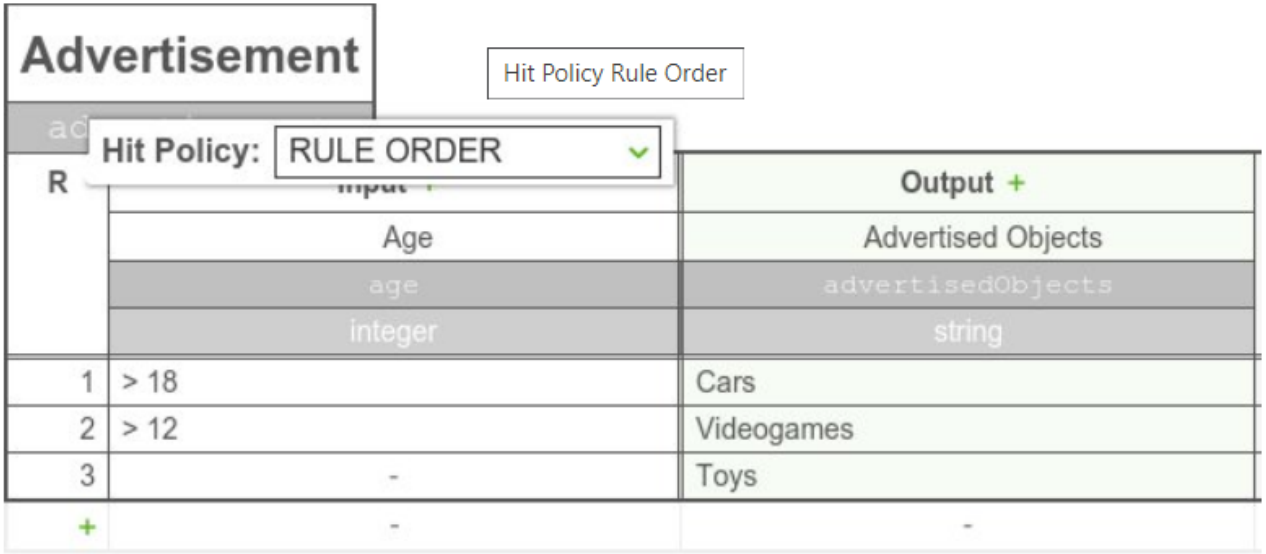
First Hit Policy

Multiple rules can be satisfied. The decision table result contains only the output of the first satisfied rule.

See the above decistion table for advertisement. Regarding the current age of the user, which advertisement should be shown is decided. For example, the user is 19 years old. All the rules will match, but since the hit policy is set to first only, the advertisement for Cars is used.

Rule Order Hit Policy

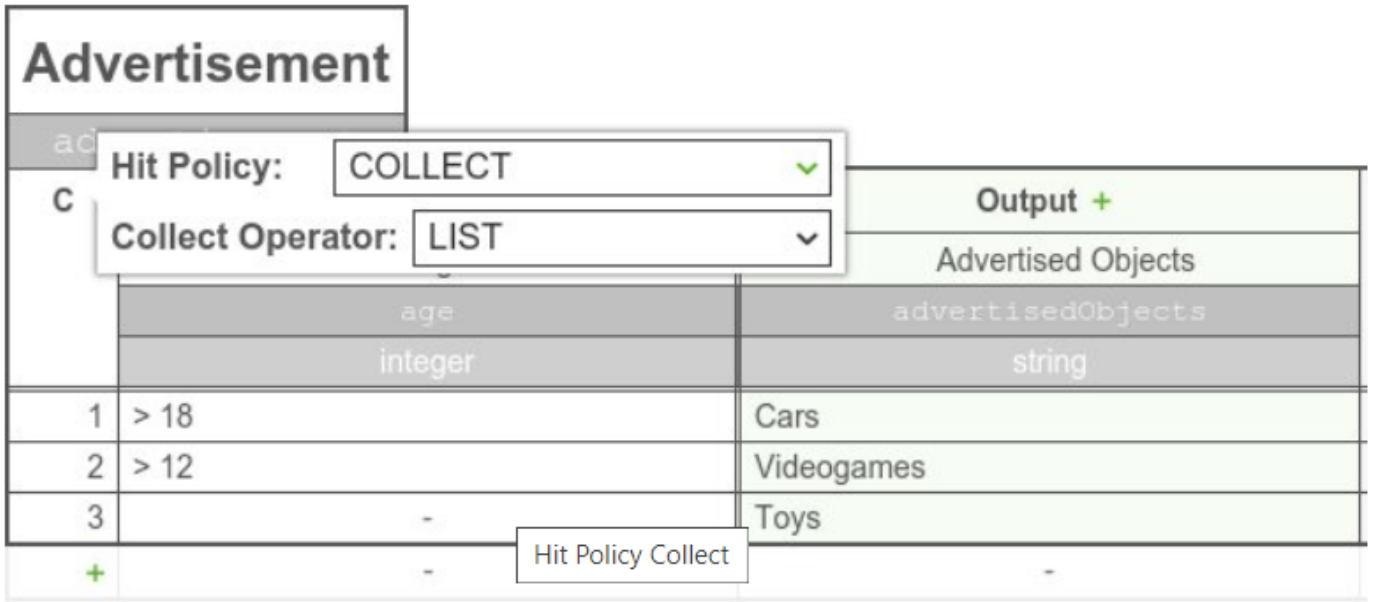
Multiple rules can be satisfied. The decision table result contains the output of all satisfied rules in the order of the rules in the decision table.



Again, see the advertisement example with the rule order policy. Say we have a user at the age of 19 again. All rules are satisfied so all outputs are given, ordered by the rule ordering. It can perhaps be used to indicate the priority of the displayed advertisements.

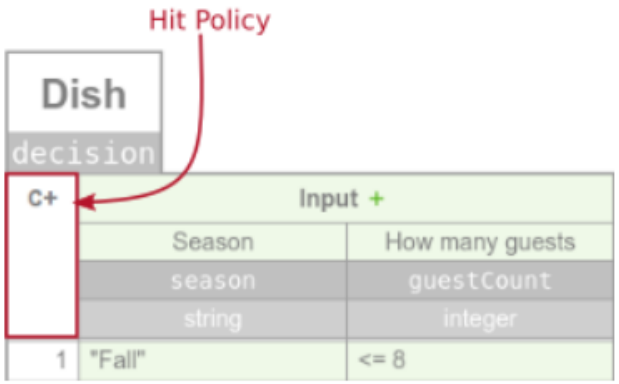
Collect Hit Policy

Multiple rules can be satisfied. The decision table result contains the output of all satisfied rules in an arbitrary order as a list.



With this hit policy, the output list has no ordering. So the advertisement will be arbitrary if, for example, the age is 19.

Aggregators for Collect Hit Policy



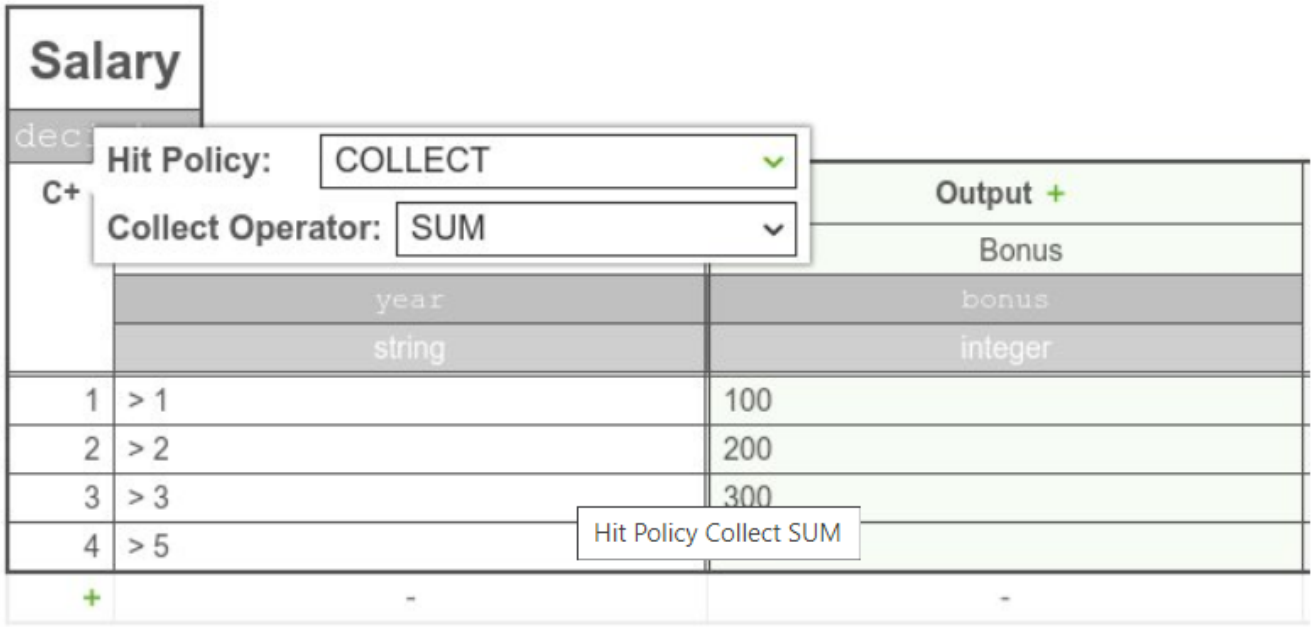
In the visual representation of the decision table the aggregator is specified by a marker after the hit policy. The following aggregators are supported by the Camunda DMN engine:

Visual representation	XML representation	Result of the aggregation
+	SUM	the sum of all output values
<	MIN	the smallest value of all output values
>	MAX	the largest value of all output values
#	COUNT	the number of output values

DMN

SUM aggregator

The SUM aggregator sums up all outputs from the satisfied rules.



The showed decision table can be used to sum up the salary bonus for an employee. For example, the employee has been working in the company for 3.5 years. So the first, second and third rule will match and the result of the decision table is 600, since the output is summed up.

DMN

MIN aggregator

The MIN aggregator can be used to return the smallest output value of all satisfied rules. See the following example of a car insurance. After years without a car crash the insurance fee will be reduced.

Car insurance

Hit Policy: COLLECT
Collect Operator: MIN

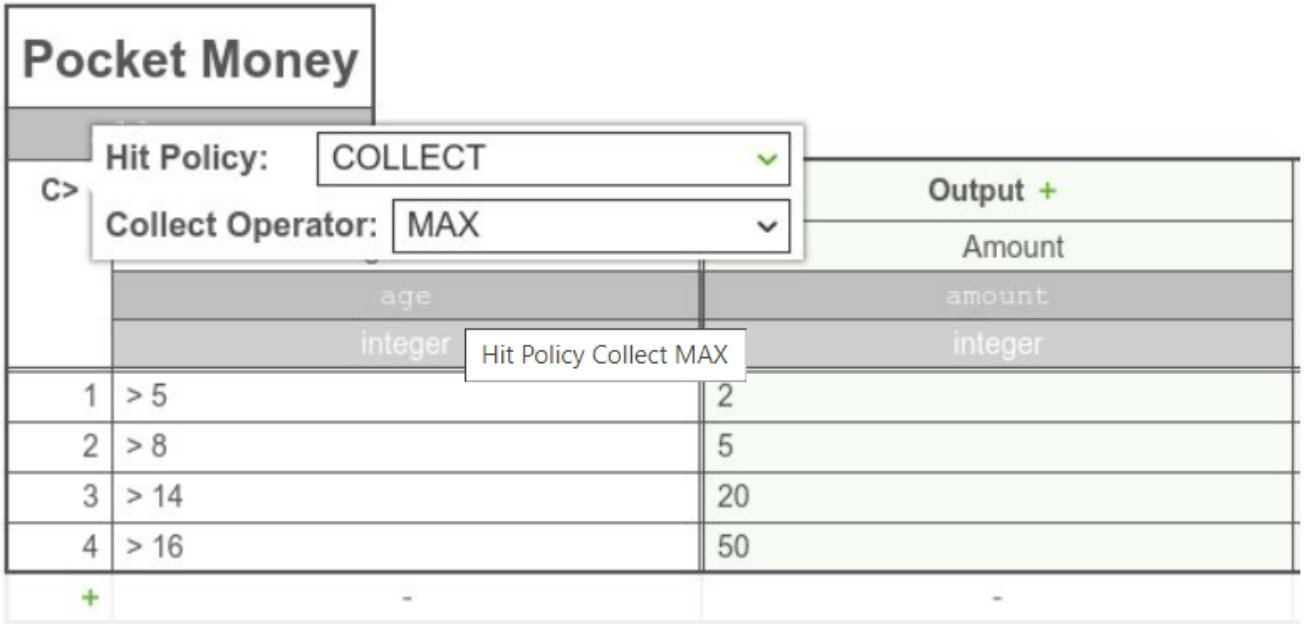
	C<	Output +
		Insurance fee
	year	fee
	integer	double
1	-	205.43
2	> 2	150.21
3	> 3	98.83
4	> 4	64.32
+	-	-

For example, if the input for the decision table is 3.5 years, the result will be 98.83, since the first three rules match but the third rule has the minimal output.

DMN

MAX aggregator

The MAX aggregator can be used to return the largest output value of all satisfied rules.

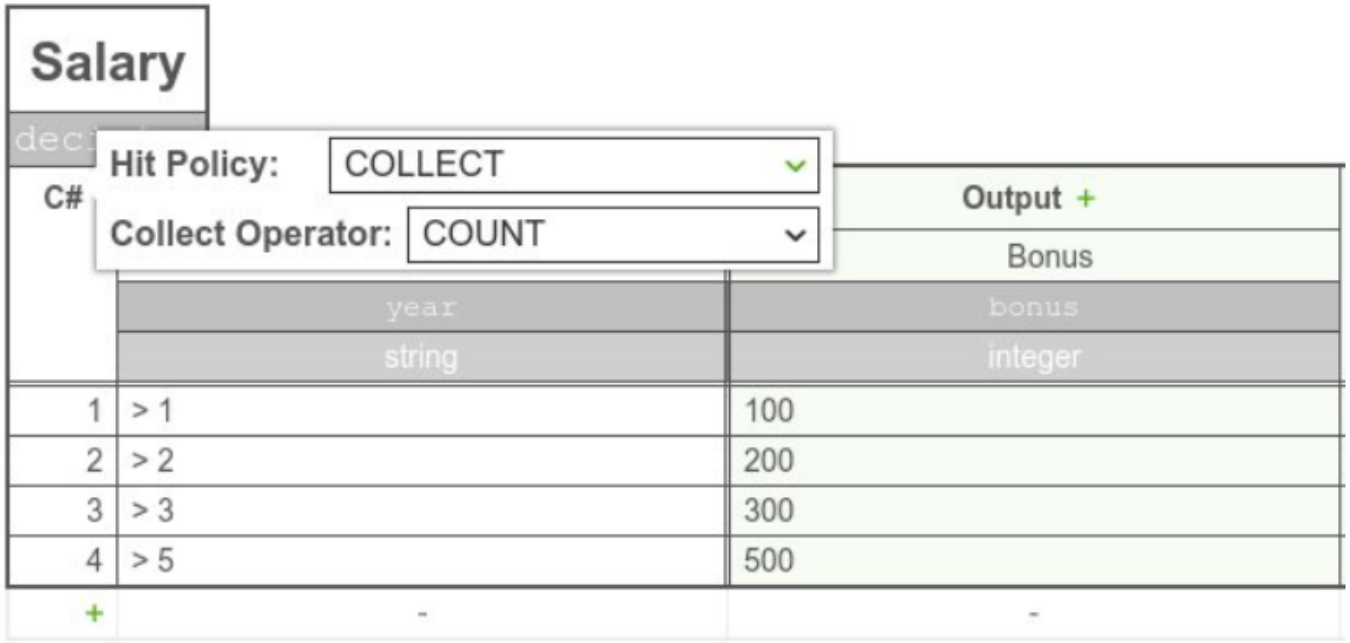


This decision table represents the decision for the amount of pocket money for a child. Depending of the age, the amount grows. For example, an input of 9 will satisfy the first and second rules. The output of the second rule is larger then the output of the first rule, so the output will be 5. A child at the age of 9 will get 5 as pocket money.

DMN

COUNT aggregator

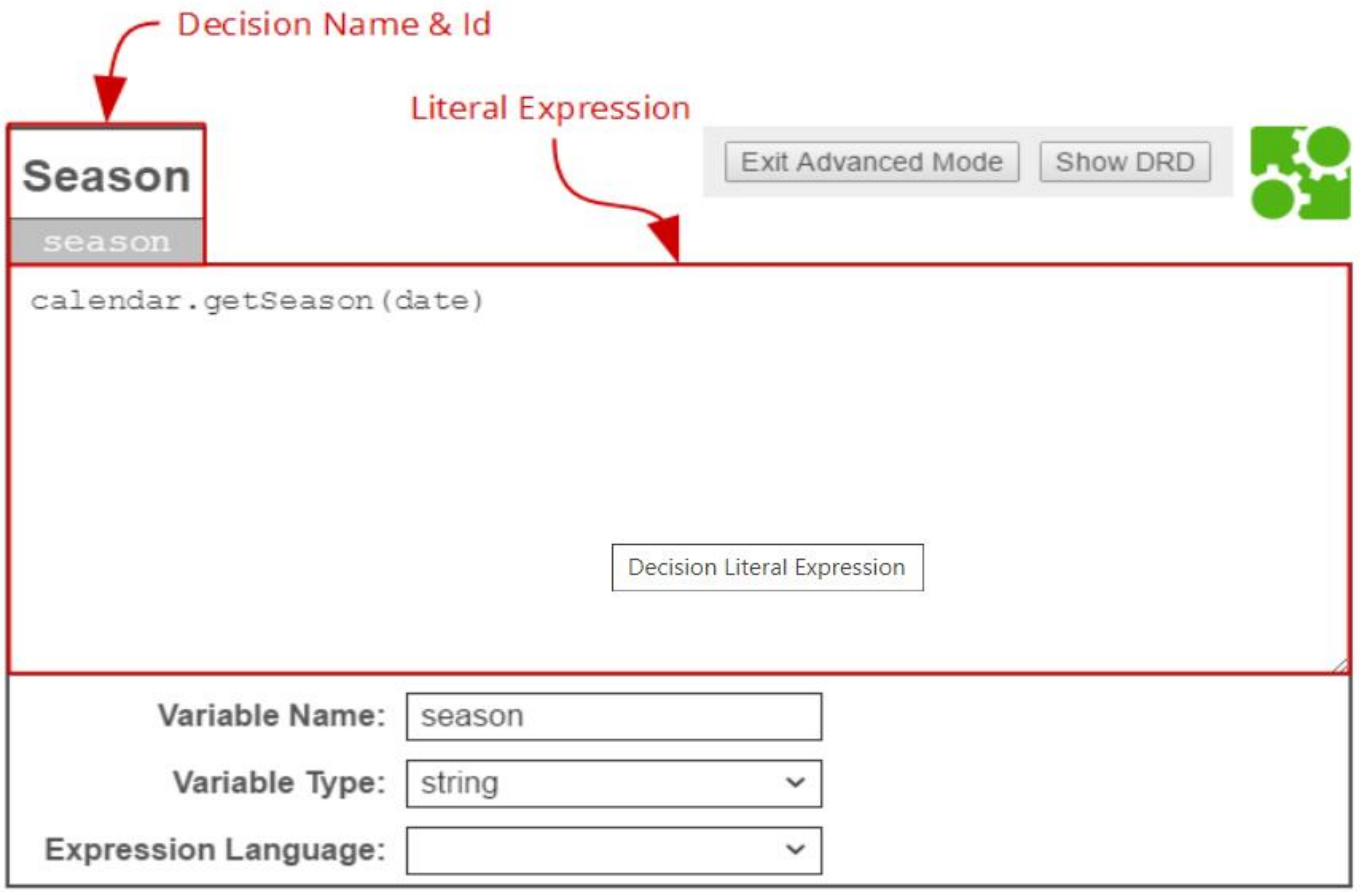
The COUNT aggregator can be use to return the count of satisfied rules.



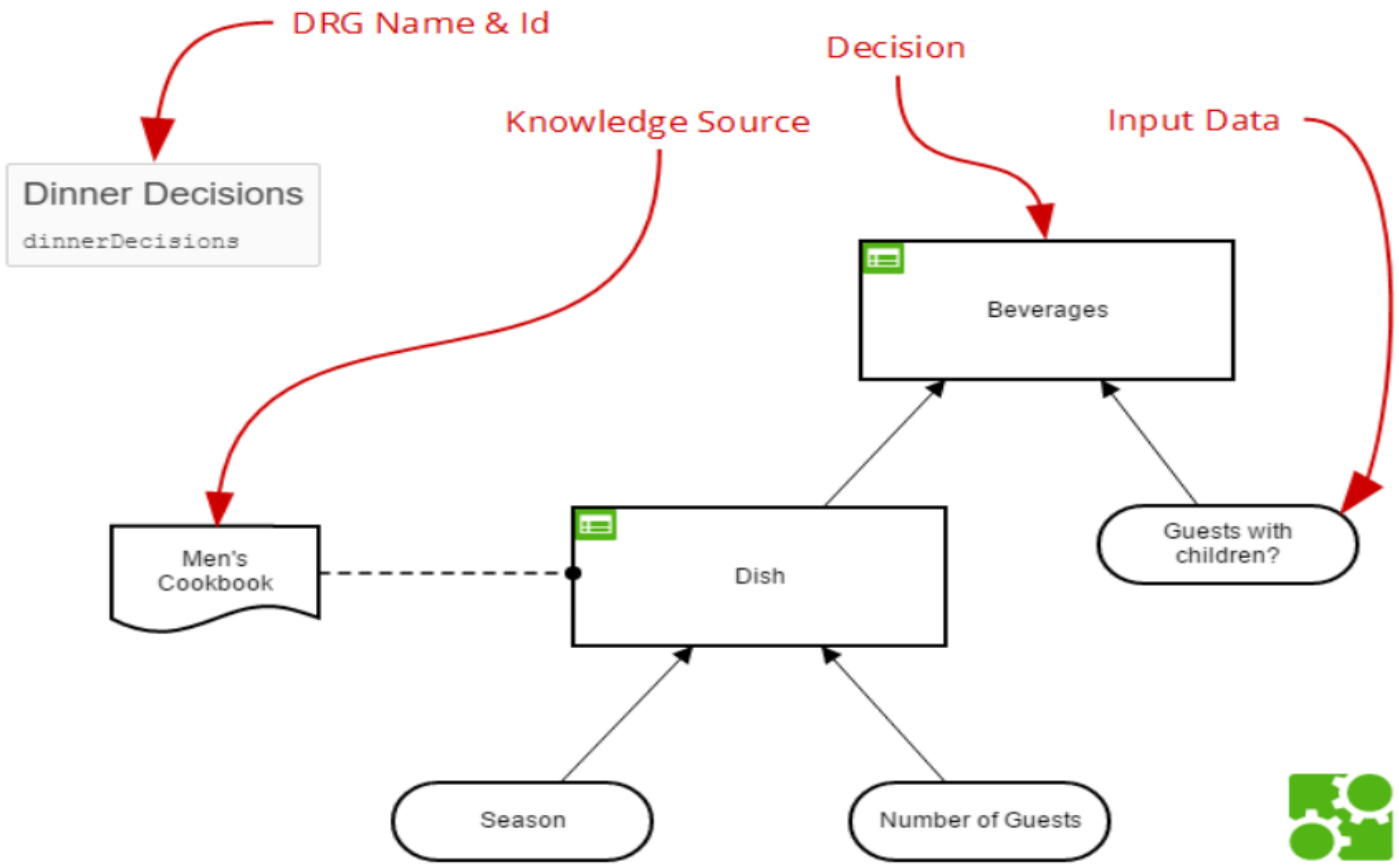
C#	year	Output + Bonus
	string	integer
1	> 1	100
2	> 2	200
3	> 3	300
4	> 5	500
+	-	-

For example, see the salary bonus decision table again, this time with the COUNT aggregator. With an input of 4, the first three rules will be satisfied. Therefore, the result from the decision table will be 3, which means that after 4 years the result of the decision table is 3 salary bonuses.

DMN Decision Literal Expression



Decision Requirements Graph



Friendly Enough Expression Language (FEEL)

- Decision Model and Notation (DMN) defines a Friendly Enough Expression Language (FEEL).
- It can be used to evaluate expressions in a decision table.
- The Camunda DMN engine only supports FEEL for input entries of a decision table.
- This corresponds to FEEL simple unary tests.

Friendly Enough Expression Language (FEEL)

FEEL Data Types

The Camunda DMN engine supports the following FEEL data types.

String

Input +
Typed Input
input
string
"Fall"

FEEL supports Strings. They must be encapsulated in double quotes. They support only the equal comparison operator.

Numeric Types

Input +
Typed Input
input

Friendly Enough Expression Language (FEEL)

Boolean

Input +
Typed Input
input
boolean
true

FEEL supports the boolean value `true` and `false`. The boolean type only supports the equal comparison operator.

Date

Input +
Typed Input
input
date
date and time("2015-11-30T12:00:00")

FEEL supports date types. In the Camunda DMN engine the following date types are available:

- date and time

To create a date and time value, the function `date and time` has to be used with a single String parameter. The parameter specifies the date and time in the format `yyyy-MM-dd'T'HH:mm:ss`.

Date types support all comparison operators and ranges.

FEEL Language Elements

Comparison

FEEL simple unary tests support the following comparison operators. Please note that the equals operator is empty and *not* `=`. Also, a non equal operator such as `!=` does *not* exist. To express this, negation has to be used.

Name	Operator	Example	Description
Equal		`"Steak"`	Test that the input value is equal to the given value.
Less	`<`	`< 10`	Test that the input value is less than the given value.
Less or Equal	`<=`	`<= 10`	Test that the input value is less than or equal to the given value.
Greater	`>`	`> 10`	Test that the input value is greater than the given value.
Greater or Equal	`>=`	`>= 10`	Test that the input value is greater than or equal to the given value.

FEEL Language Elements

Range

Some FEEL data types, such as numeric types and date types, can be tested against a range of values. These ranges consist of a start value and an end value. The range specifies if the start and end value is included in the range.

Start	End	Example	Description
include	include	`[1..10]`	Test that the input value is greater than or equal to the start value and less than or equal to the end value.
exclude	include	`]1..10]` or `(1..10]`	Test that the input value is greater than the start value and less than or equal to the end value.
include	exclude	`[1..10[` or `[1..10)`	Test that the input value is greater than or equal to the start value and less than the end value.
exclude	exclude	`]1..10[` or `(1..10)`	Test that the input value is greater than the start value and less than the end value.

FEEL Language Elements

Disjunction

A FEEL simple unary test can be specified as conjunction of expressions. These expressions have to either have comparisons or ranges. The test is `true` if at least one of conjunct expressions is `true`.

Examples:

- `3,5,7`: Test if the input is either 3, 5 or 7
- `<2,>10`: Test if the input is either less than 2 or greater than 10
- `10,[20..30]`: Test if the input is either 10 or between 20 and 30
- `"Spareribs","Steak","Stew"`: Test if the input is either the String Spareribs, Steak or Stew
- `date and time("2015-11-30T12:00:00"),date and time("2015-12-01T12:00:00")`: Test if the input is either the date November 30th, 2015 at 12:00:00 o'clock or December 1st, 2015 at 12:00:00 o'clock
- `>customer.age,>21`: Test if the input is either greater than the `age` property of the variable `customer` or greater than 21

FEEL Language Elements

Negation

A FEEL simple unary test can be negated with the `not` function. This means if the containing expression returns `true`, the test will return `false`. Please *note* that only one negation as first operator is allowed but it can contain a disjunction.

Examples:

- `not("Steak")`: Test if the input is not the String Steak
- `not(>10)`: Test if the input is not greater than 10, which means it is less than or equal to 10
- `not(3,5,7)`: Test if the input is neither 3, 5 nor 7
- `not([20..30])`: Test if the input is not between 20 and 30

FEEL Language Elements

Qualified Names

FEEL simple unary tests can access variables and object properties by qualified names.

Examples:

- `x`: Test if the input is equal to the variable `x`
- `>= x`: Test if the input is greater than or equal to the variable `x`
- `< customer.age`: Test if the input is less than the `age` property of the variable `customer`

Date Functions

FEEL simple unary tests provide functions to create date types. The Camunda DMN engine supports the following date functions:

- `date and time("...")`: Creates a date and time value from a String with the format `yyyy-MM-dd'T'HH:mm:ss`

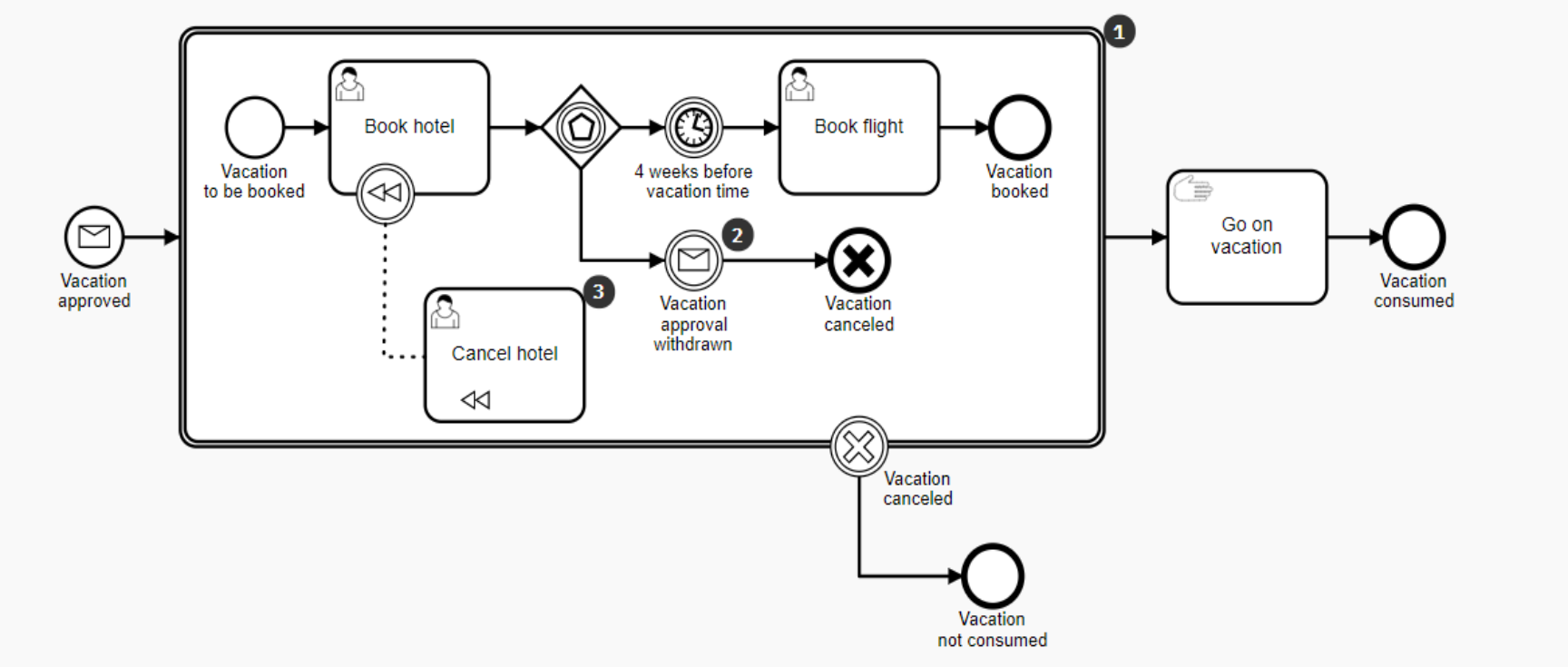
Examples:

- `date and time("2015-11-30T12:00:00")`: Test if the input is the date November 30th, 2015 at 12:00:00 o'clock
- `[date and time("2015-11-30T12:00:00")..date and time("2015-12-01T12:00:00")]`: Test if the input is between the date November 30th, 2015 at 12:00:00 o'clock and December 1st, 2015 at 12:00:00 o'clock

Camunda BPM Platform Docker Images

- docker pull camunda/camunda-bpm-platform:latest
- docker run -d --name camunda -p 8080:8080 camunda/camunda-bpm-platform:latest
- # open browser with url: http://192.168.99.101:8080/camunda-welcome/index.html

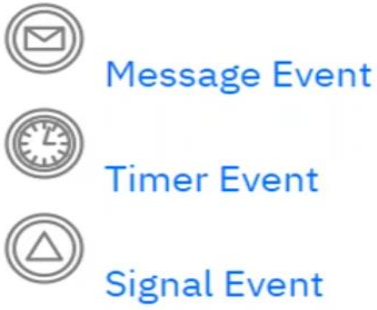
Business Transaction



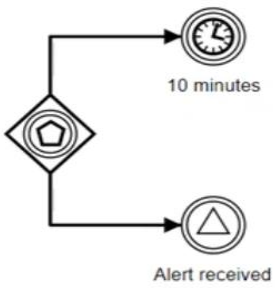
Transaction

Wait States

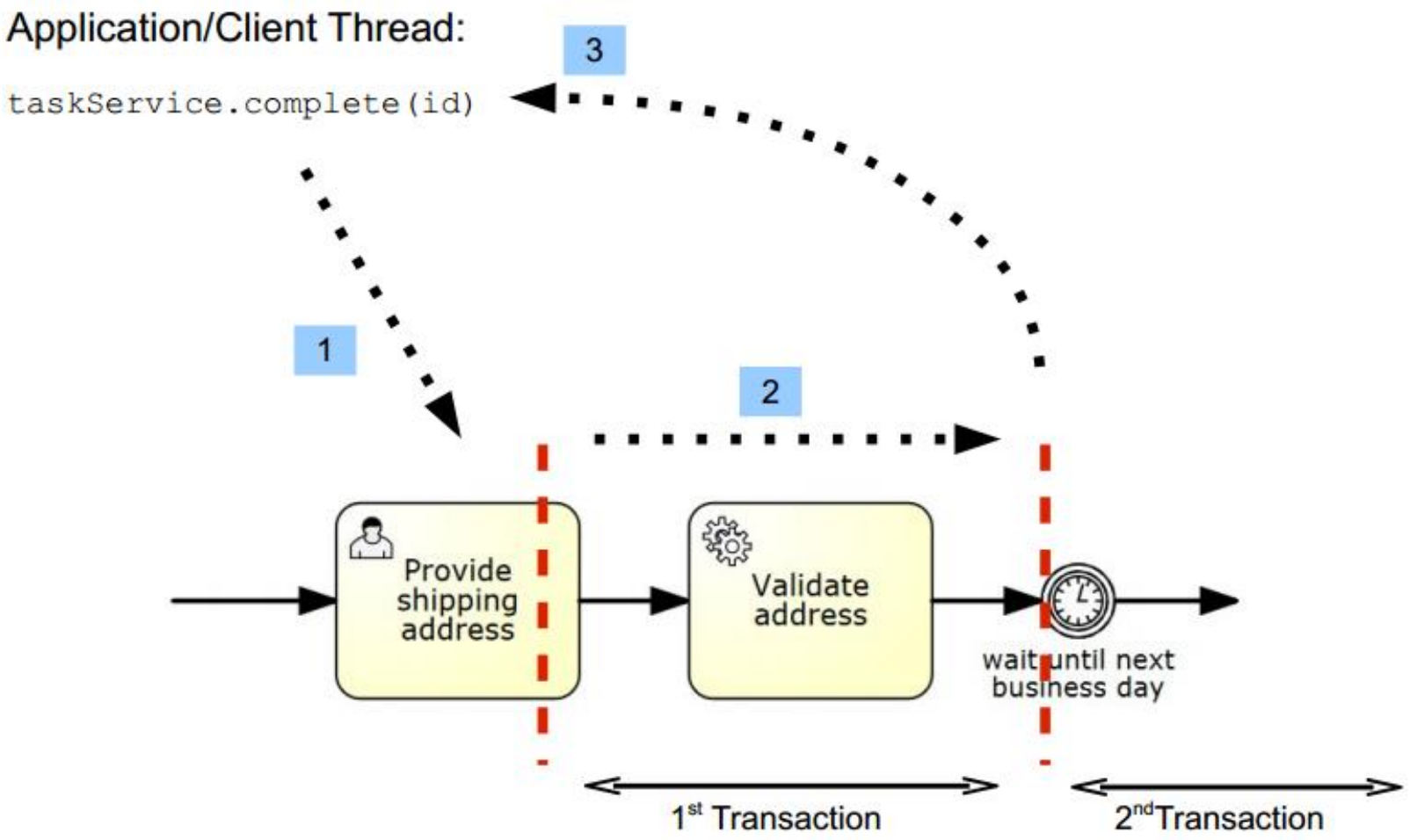
A wait state is a task which is performed *later*, which means that the engine persists the current execution to the database and waits to be triggered again



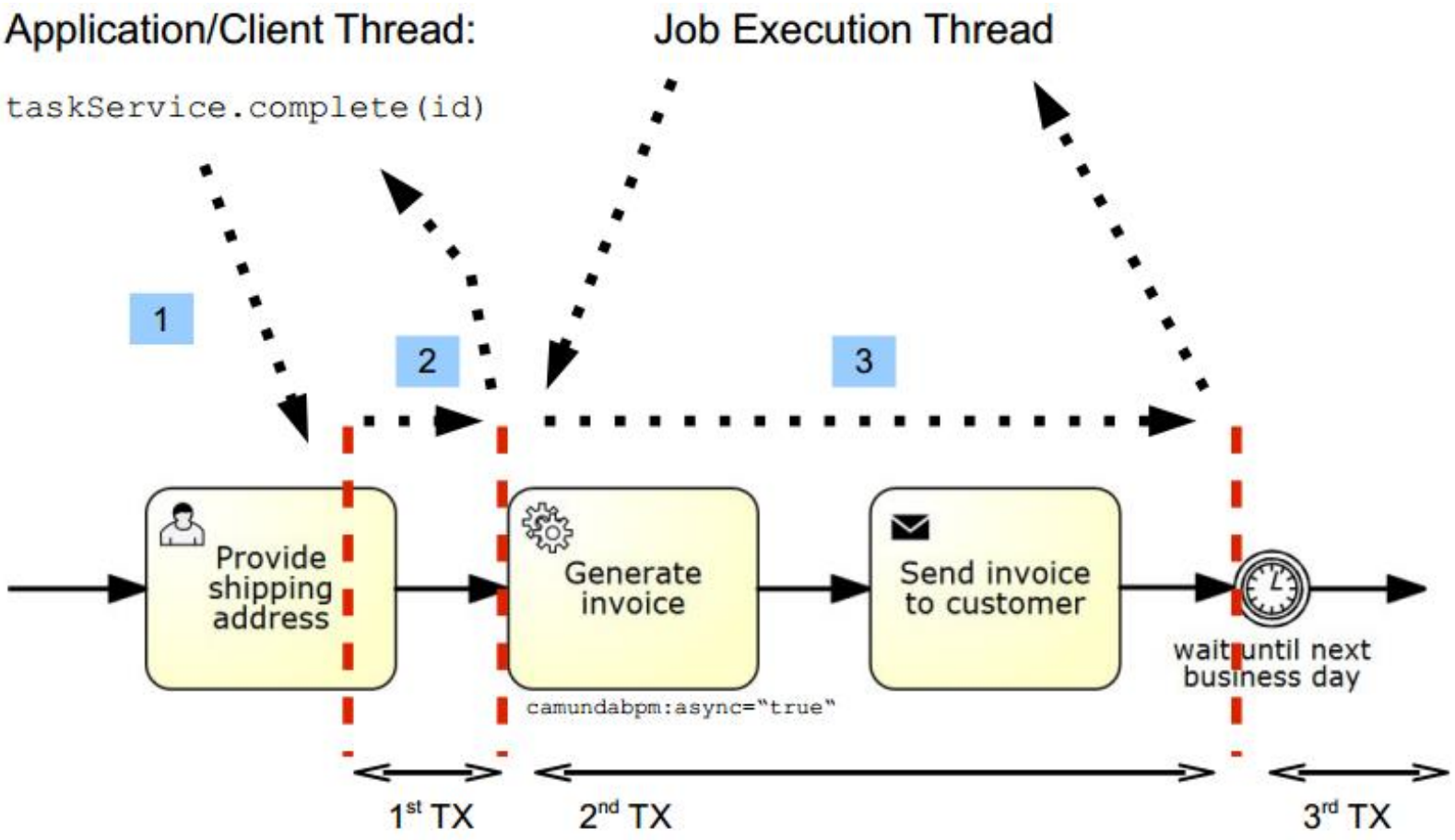
The Event Based Gateway:



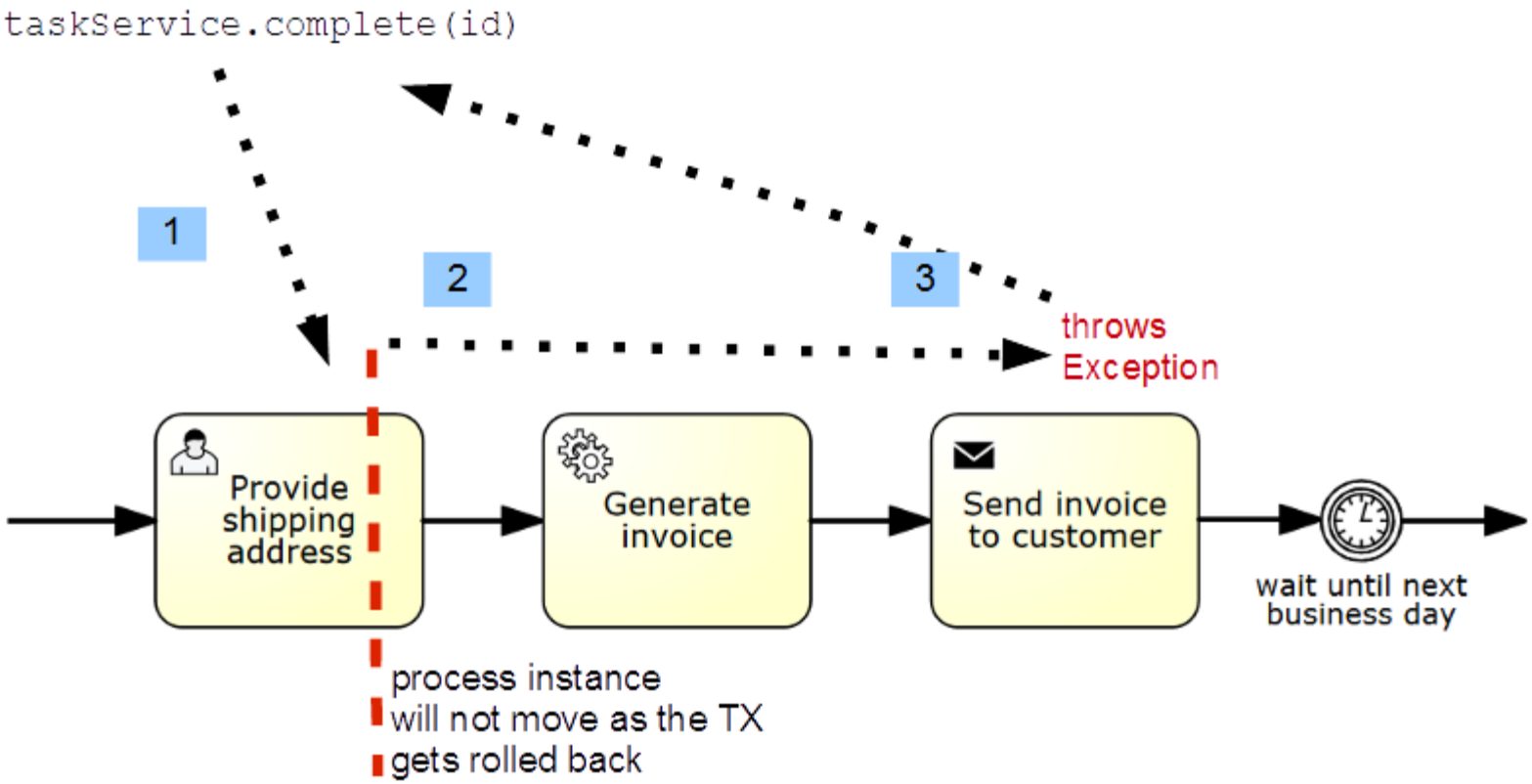
Transaction Boundaries



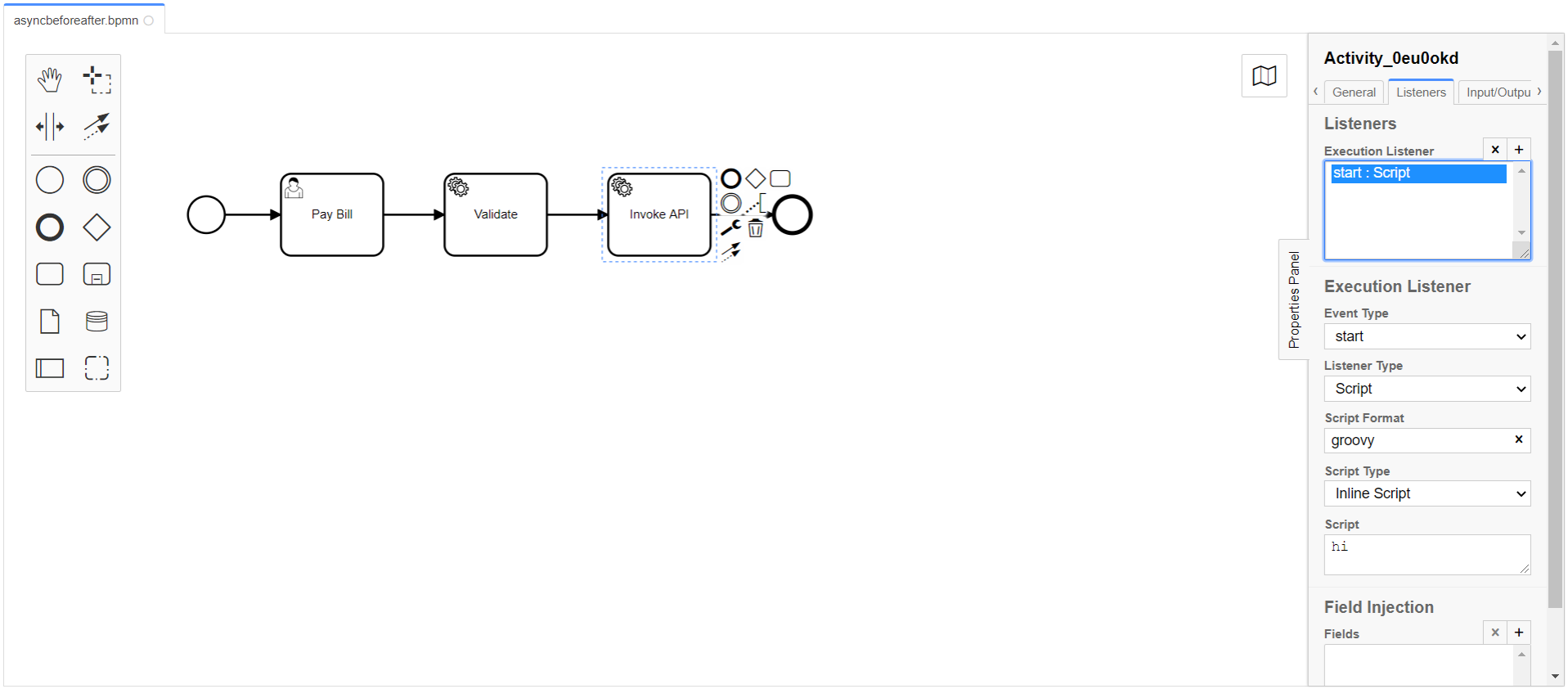
Why Asynchronous Continuations?



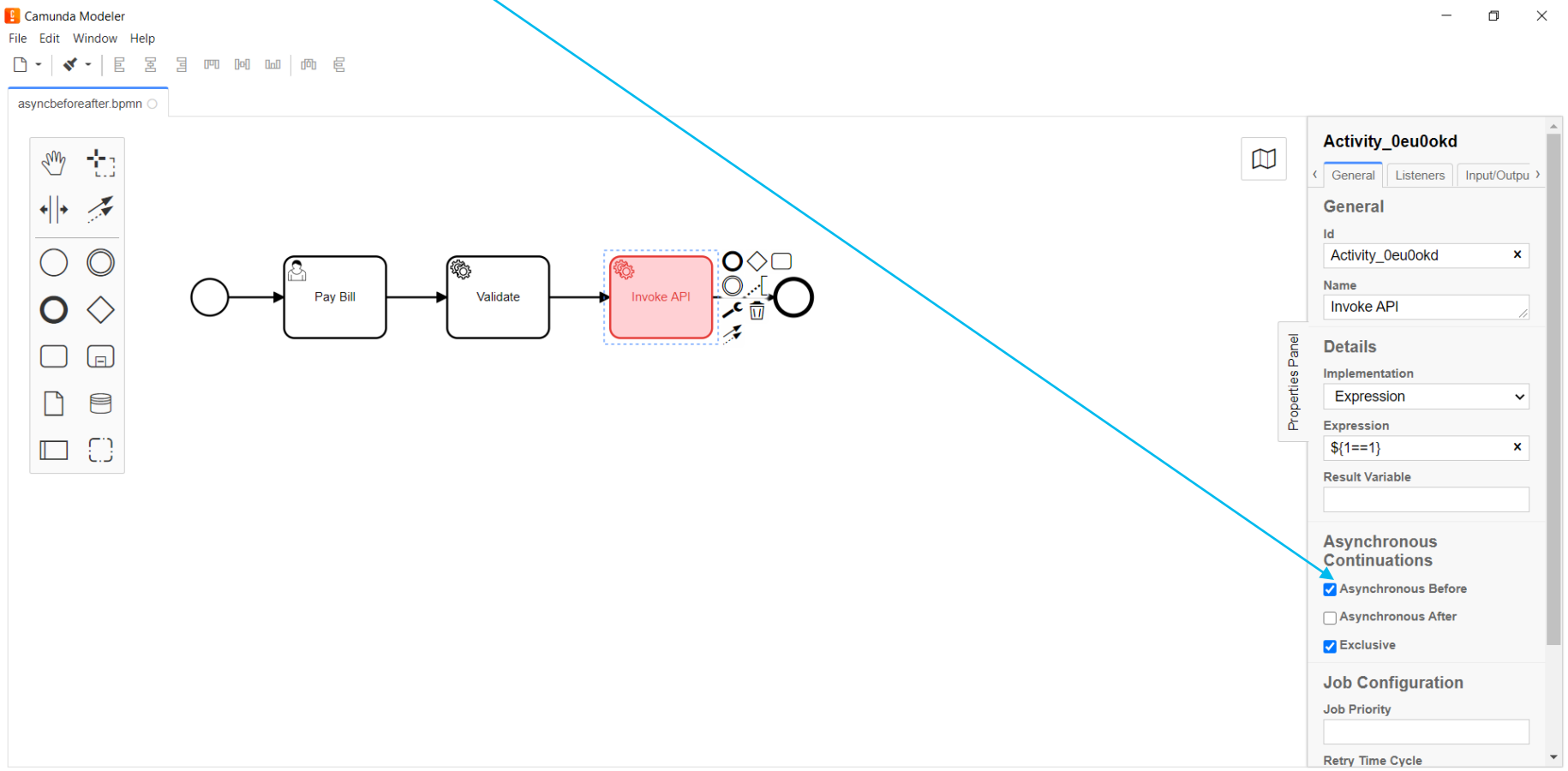
Rollback on Exception



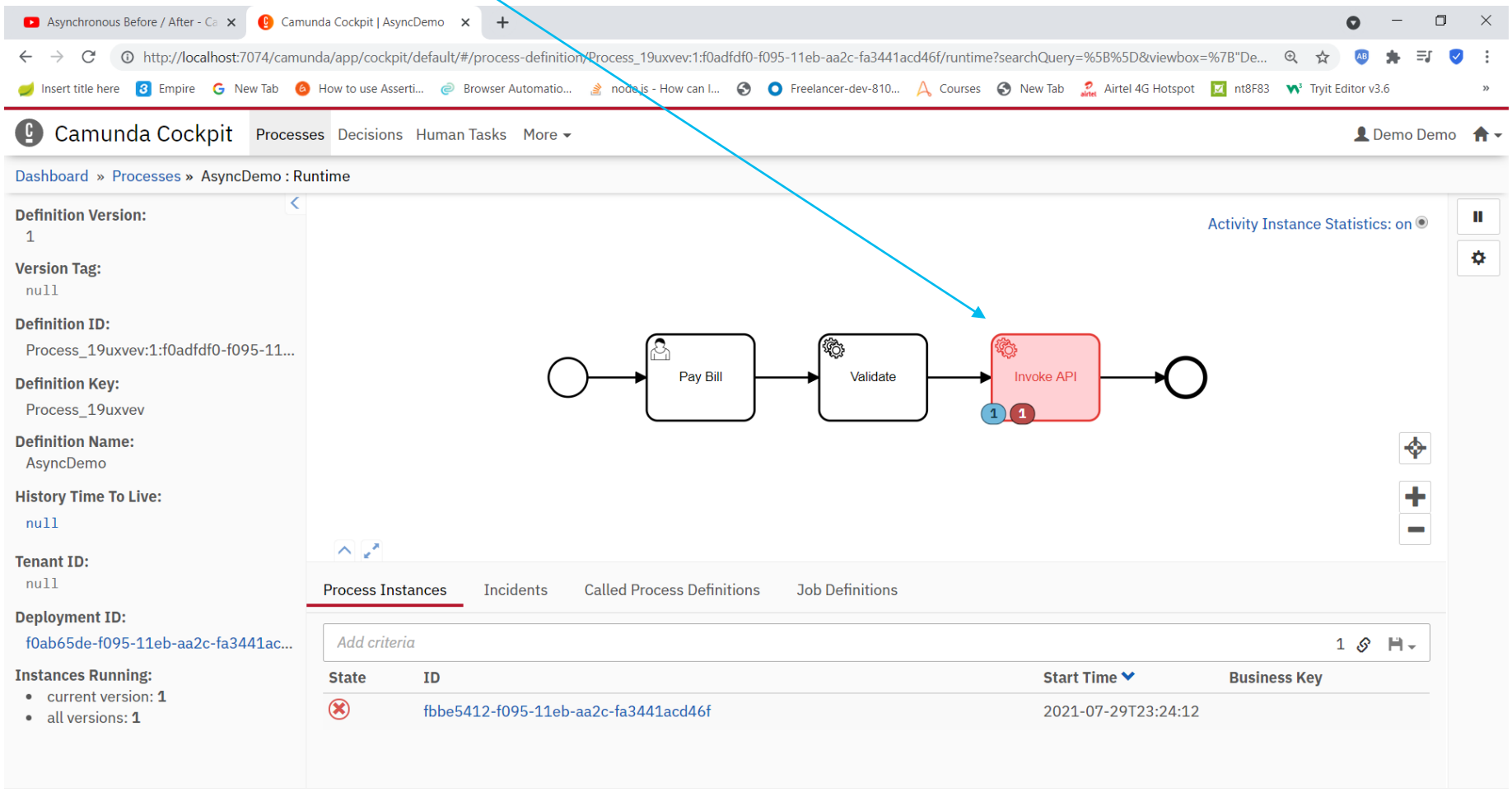
Async Demo


asyncbeforeafter.bpmn
Pay Bill
Validate
Invoke API
Activity_0eu0okd
General
Listeners
Input/Outpu
Listeners
Execution Listener
start : Script
Properties Panel
Execution Listener
Event Type
start
Listener Type
Script
Script Format
groovy
Script Type
Inline Script
Script
hi
Field Injection
Fields

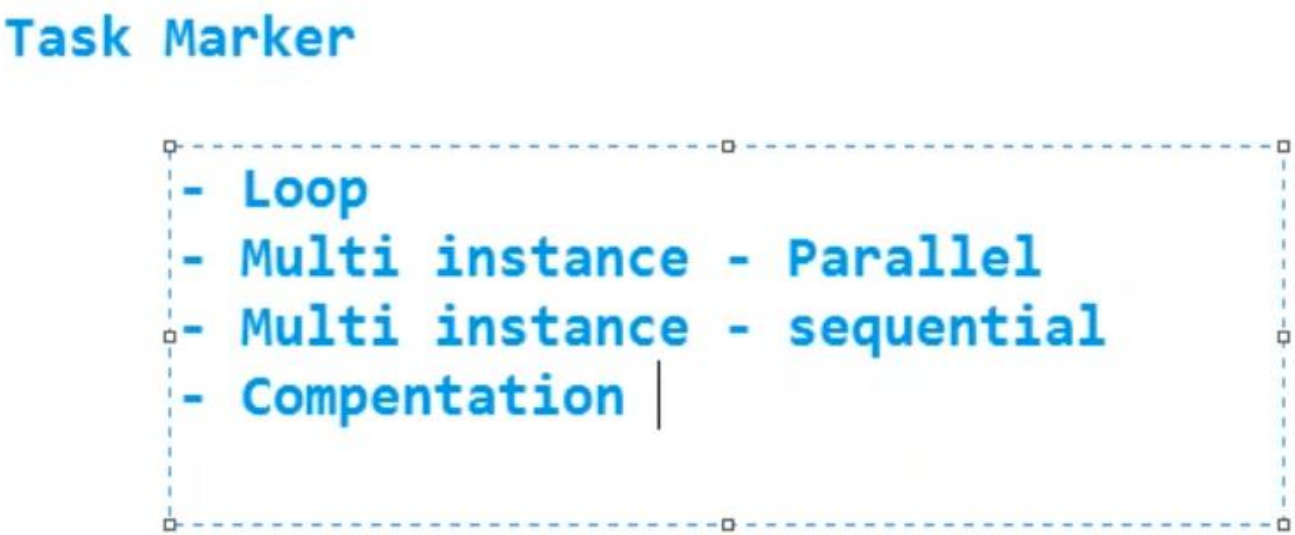
Async Demo


Camunda Modeler
File Edit Window Help
asyncbeforeafter.bpmn
Pay Bill
Validate
Invoke API
Activity_0eu0okd
General
Listeners
Input/Outpu
General
Id
Activity_0eu0okd
Name
Invoke API
Details
Implementation
Expression
Expression
${1==1}
Result Variable
Asynchronous Continuations
Asynchronous Before
Asynchronous After
Exclusive
Job Configuration
Job Priority
Retry Time Cycle
Properties Panel

Async Demo


Asynchronous Before / After - Ca
Camunda Cockpit | AsyncDemo
http://localhost:7074/camunda/app/cockpit/default/#/process-definition/Process_19uxvev:1:f0adfdf0-f095-11eb-aa2c-fa3441acd46f/runtime?searchQuery=%5B%5D&viewbox=%7B"De...
Insert title here
Empire
New Tab
How to use Asserti...
Browser Automatio...
node.js - How can I...
Freelancer-dev-810...
Courses
New Tab
Airtel 4G Hotspot
nt8F83
Tryit Editor v3.6
Camunda Cockpit
Processes
Decisions
Human Tasks
More
Demo Demo
Dashboard » Processes » AsyncDemo : Runtime
Definition Version:
1
Version Tag:
null
Definition ID:
Process_19uxvev:1:f0adfdf0-f095-11...
Definition Key:
Process_19uxvev
Definition Name:
AsyncDemo
History Time To Live:
null
Tenant ID:
null
Deployment ID:
f0ab65de-f095-11eb-aa2c-fa3441ac...
Instances Running:
current version: 1
all versions: 1
Activity Instance Statistics: on
Pay Bill
Validate
Invoke API
1
1
Process Instances
Incidents
Called Process Definitions
Job Definitions
Add criteria
1
State
ID
Start Time
Business Key
fbbe5412-f095-11eb-aa2c-fa3441acd46f
2021-07-29T23:24:12
Date and Time displayed in local timezone: Asia/Calcutta
Powered by Camunda Platform / v7.15.0

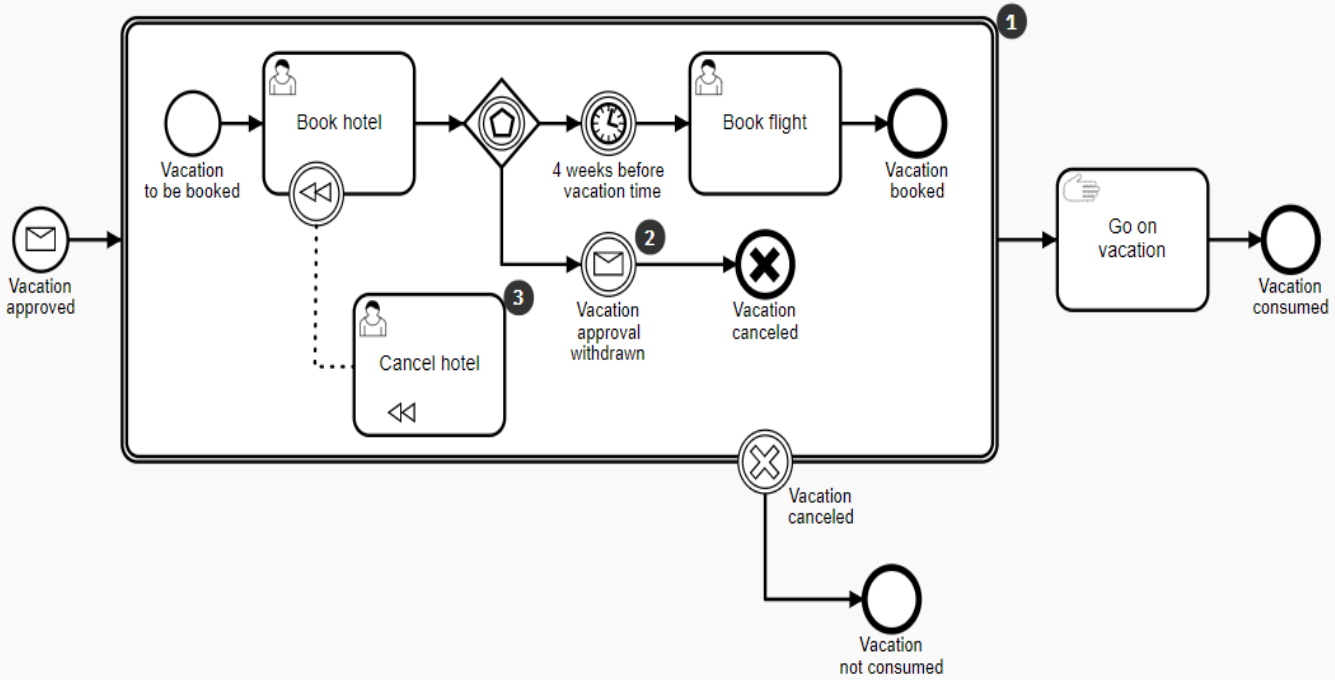
Task Marker


Task Marker
- Loop
- Multi instance - Parallel
- Multi instance - sequential
- Compentation

Dealing With Problems and Exceptions

Business Transactions

Sometimes when we refer to "transactions" in processes we refer to a very different concept, which must be clearly distinguished from "technical" database transactions. A **business transaction** marks a section in a process for which 'all or nothing' semantics similar to a technical transaction should apply, but from a pure business perspective.

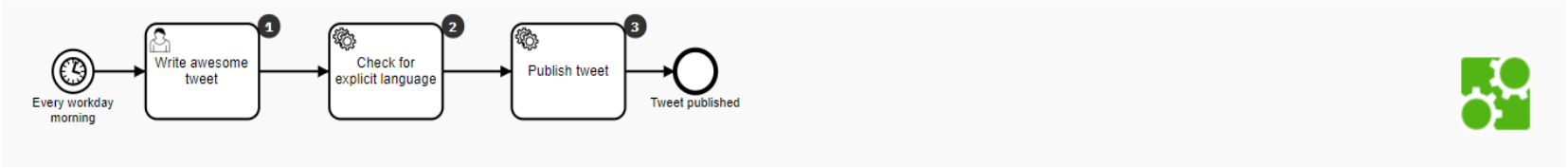


Dealing With Problems and Exceptions

Demarcating Custom **Transaction Borders**

Using Additional **Save Points**

You have fine grained control over transaction borders by introducing **additional optional "save points"** on top of the obligatory "wait states". Use the `asyncBefore='true'` and `asyncAfter='true'` attributes in your process definition BPMN XML. The process state will then be persisted at these points and a background job executor will make sure that it is continued asynchronously.



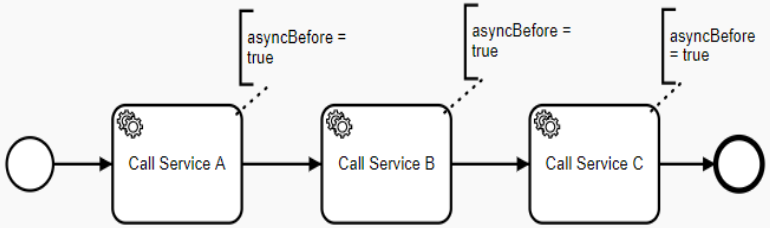
1. A user task is an **obligatory wait state** for the process engine. After the creation of the user task, the process state will be persisted and committed to the database. The engine will wait for user interaction.
2. This service task is executed **"synchronously"** (by default), in other words within the same thread and the same database transaction with which a user attempts to complete the "Write tweet" user task. When we assume that this service fails in cases in which the language used is deemed to be too explicit, the database transaction rolls back and the user task will therefore remain uncompleted. The user must re-attempt, e.g. by correcting the tweet.
3. This service task is executed **"asynchronously"**. By setting the `asyncBefore='true'` attribute we introduce an additional save point at which the process state will be persisted and committed to the database. A separate job executor thread will continue the process asynchronously by using a separate database transaction. In case this transaction fails the service task will be retried and eventually marked as failed - in order to be dealt with by a human operator.

Dealing With Problems and Exceptions

Marking Every Service Task as **Asynchronous**

A typical **rule of thumb**, especially when doing a lot of service orchestration, is to **mark every service task** being **asynchronous**.

If you want to know more about the underlying reasons for marking service tasks as asynchronous - or not - we strongly recommend to read the next section about Knowing Typical **Do's and Don'ts** for Save Points.



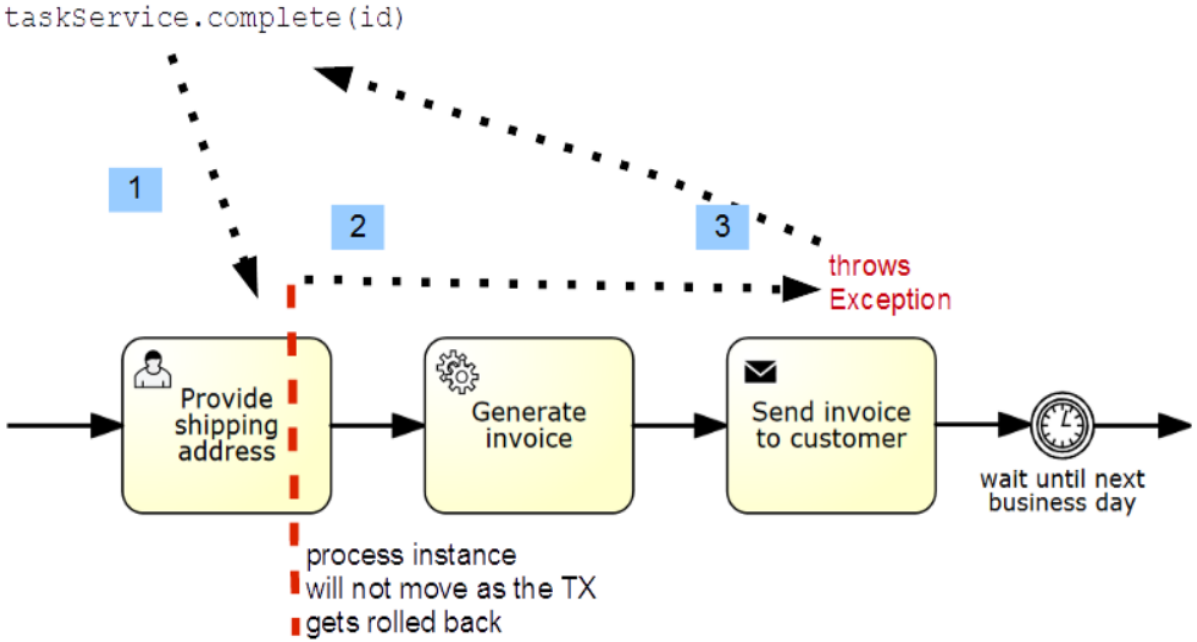
The downside is that the jobs slightly increase the overall resource consumption. But this is often worth it, as it has a couple of advantages for operations:

- The process stops at the service task causing the specific error.
- You can configure a meaningful retry strategy for every service task.
- You can leverage the suspension features for service tasks.

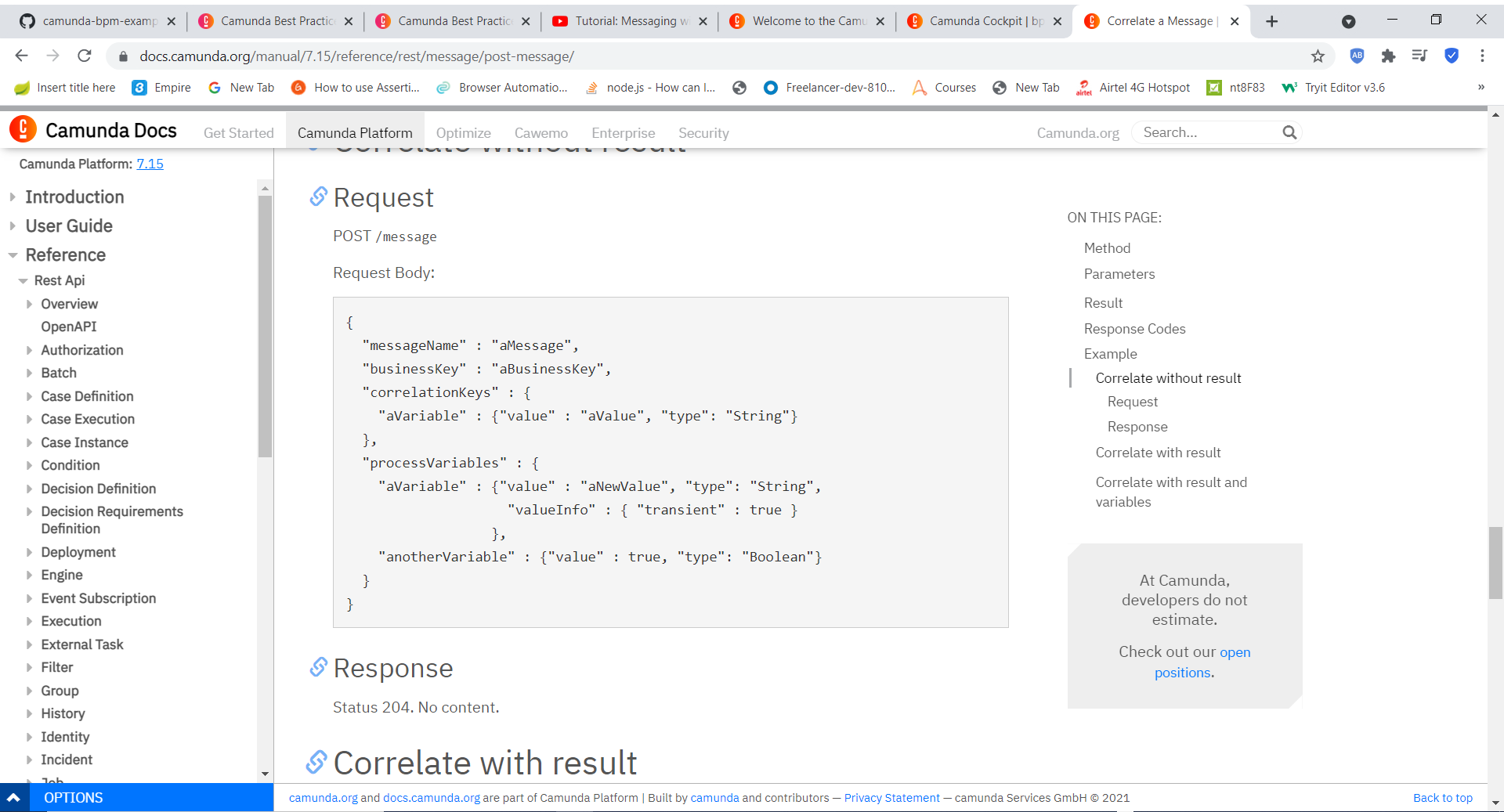
Dealing With Problems and Exceptions

Rolling Back a Transaction

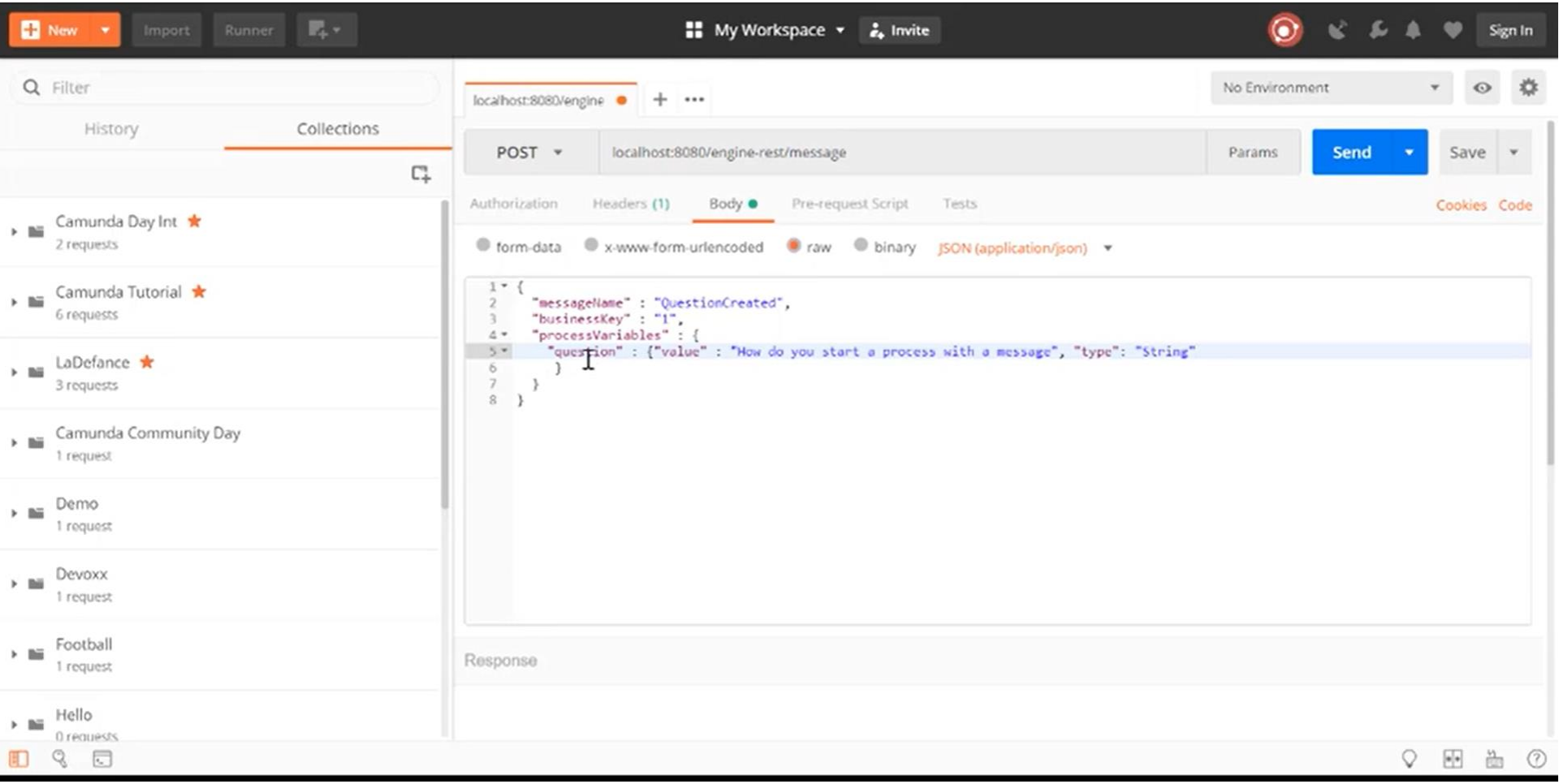
It is important to understand that every **non-handled, propagated exception** happening during process execution rolls back the current technical transaction. Therefore the process instance will find its last known **wait state** (or save point). The following image visualizes that default behavior.



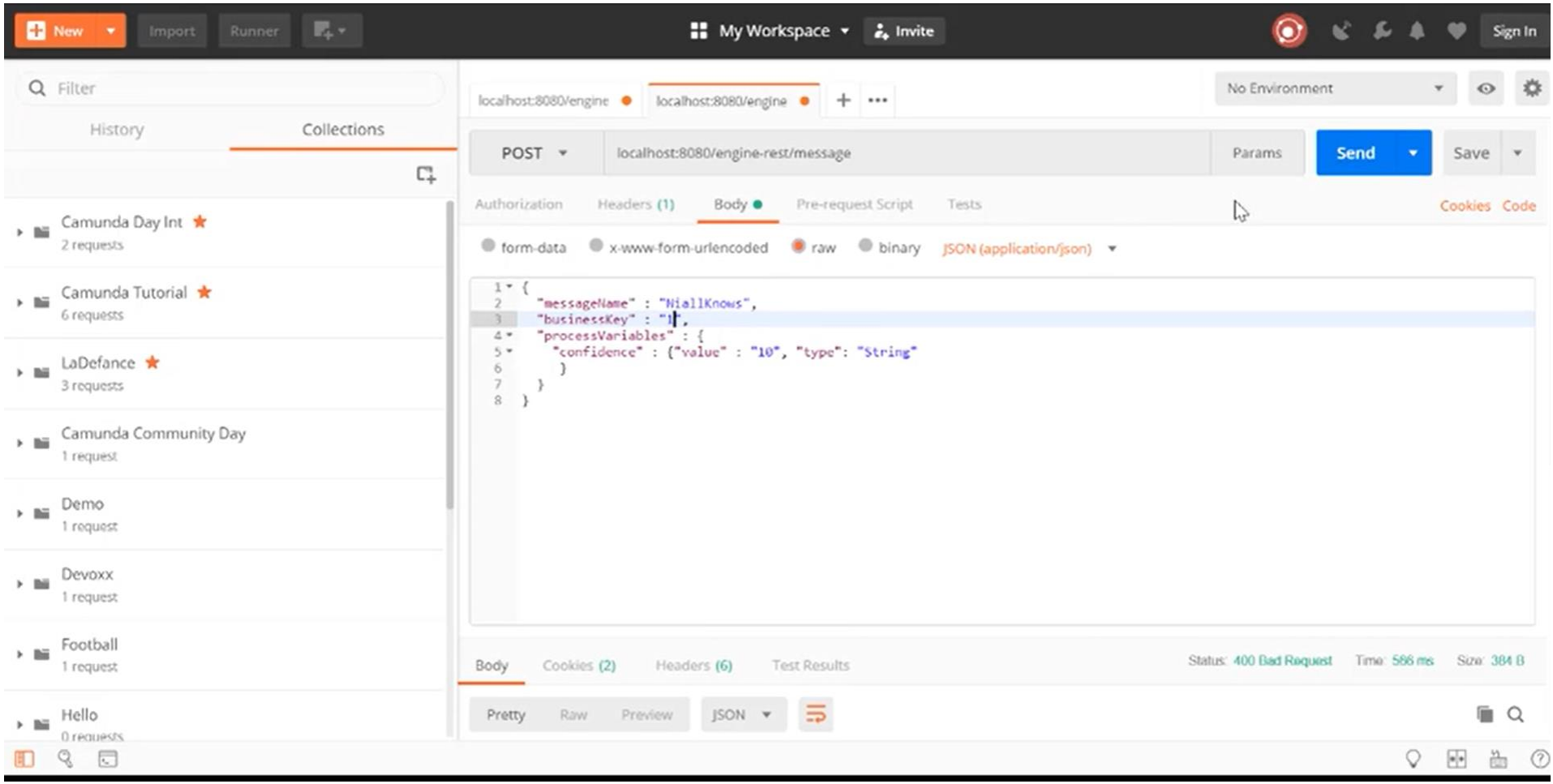
Camunda Rest Message



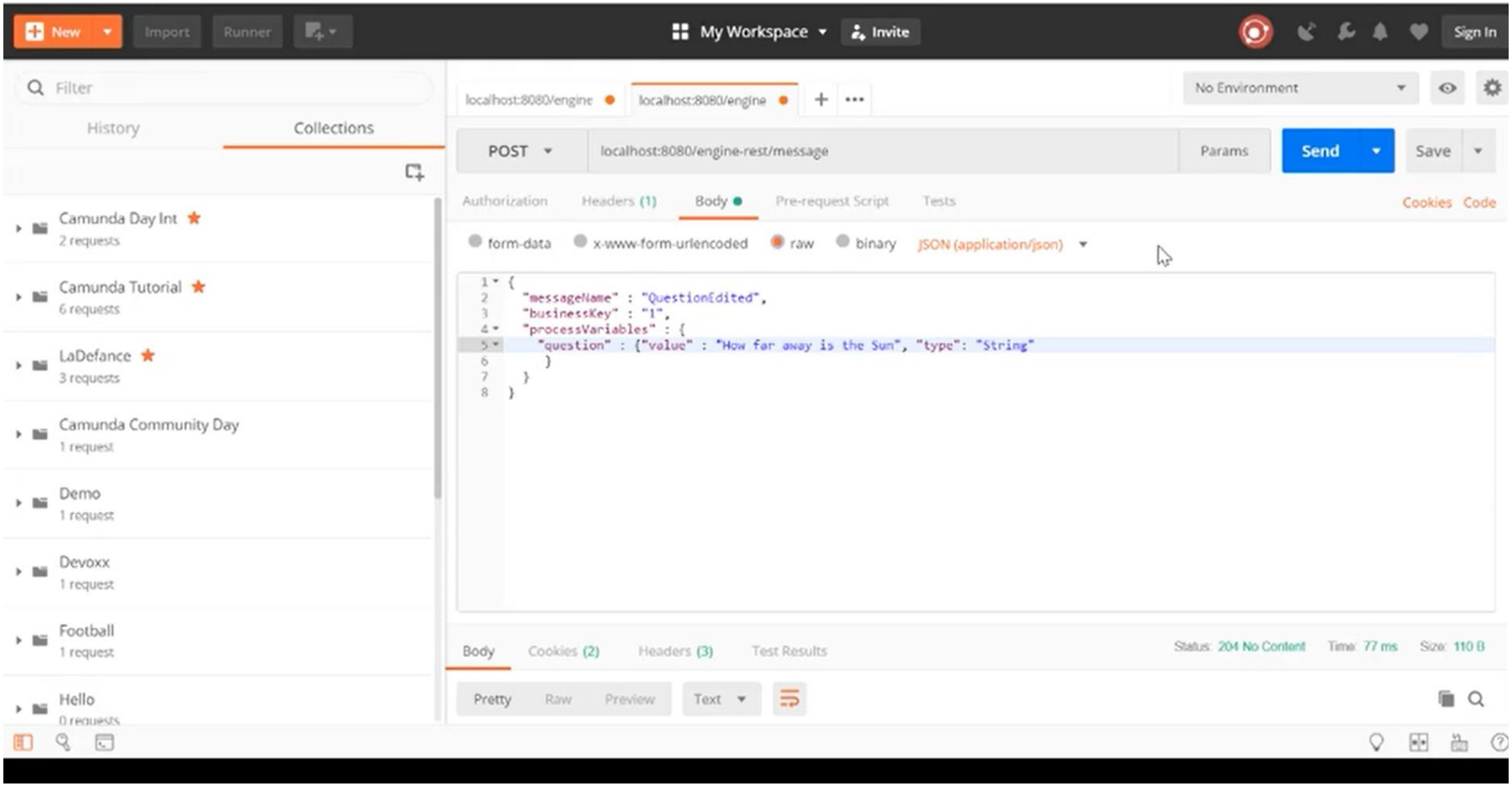
Send Message



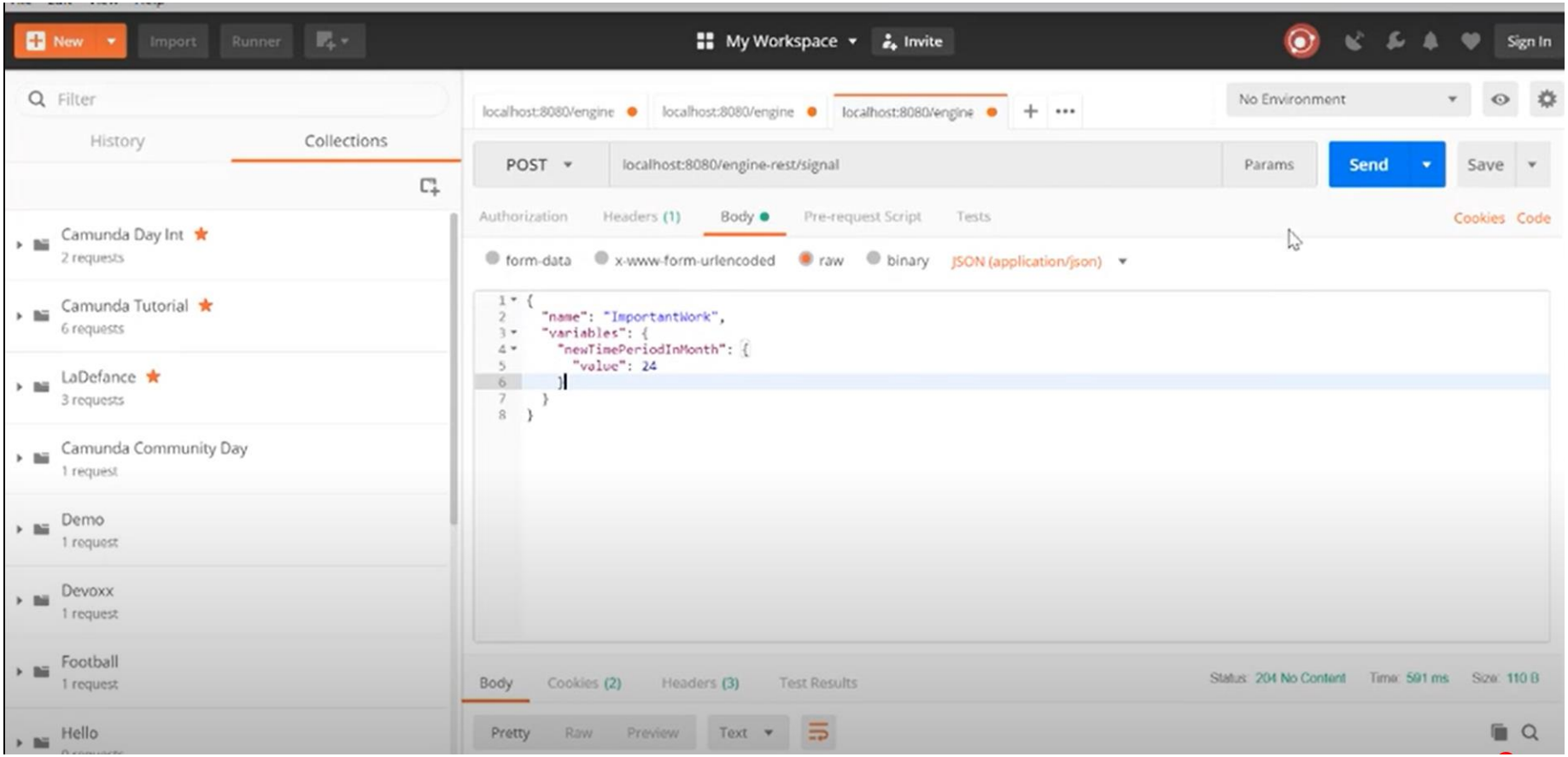
Send Message



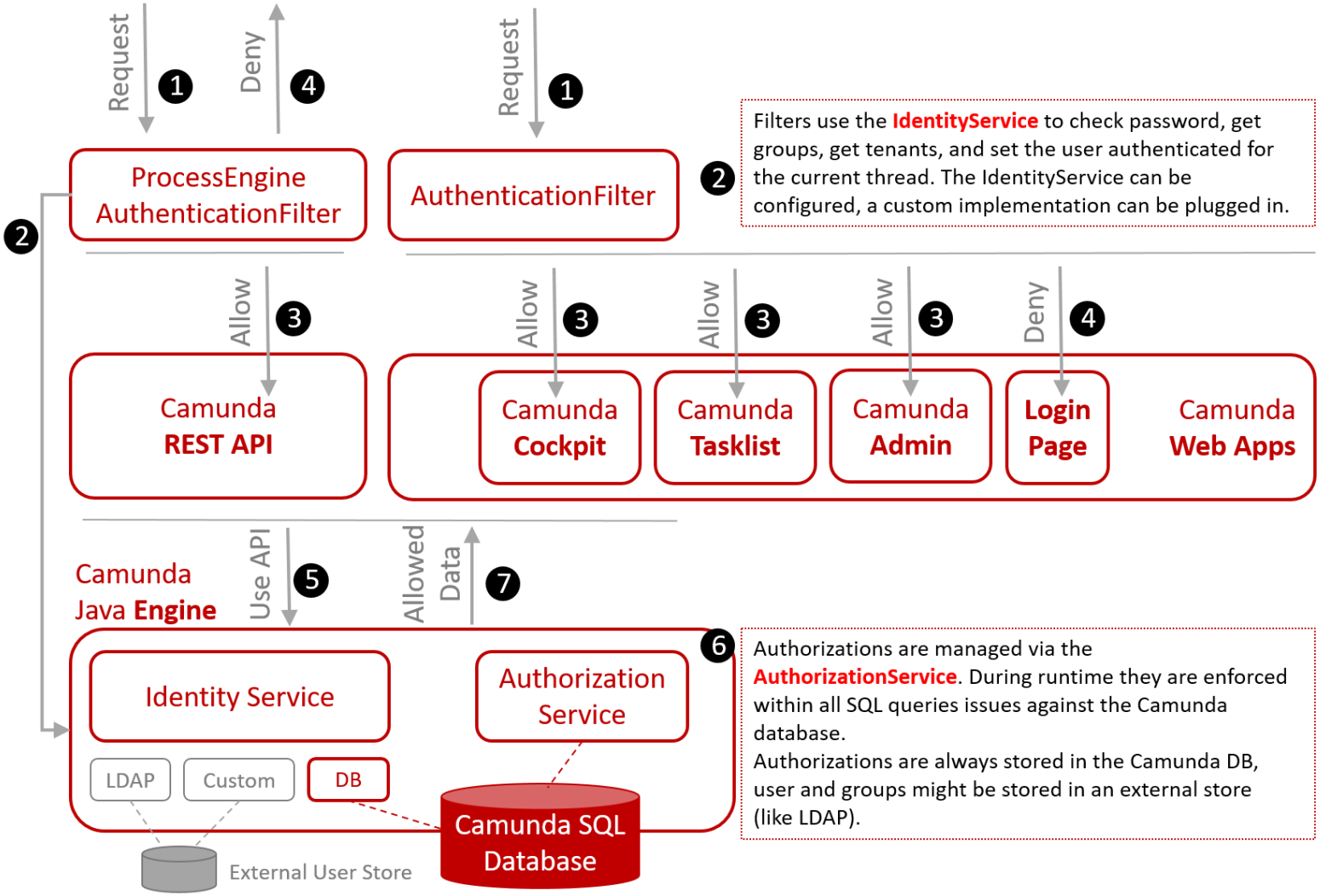
Send Message



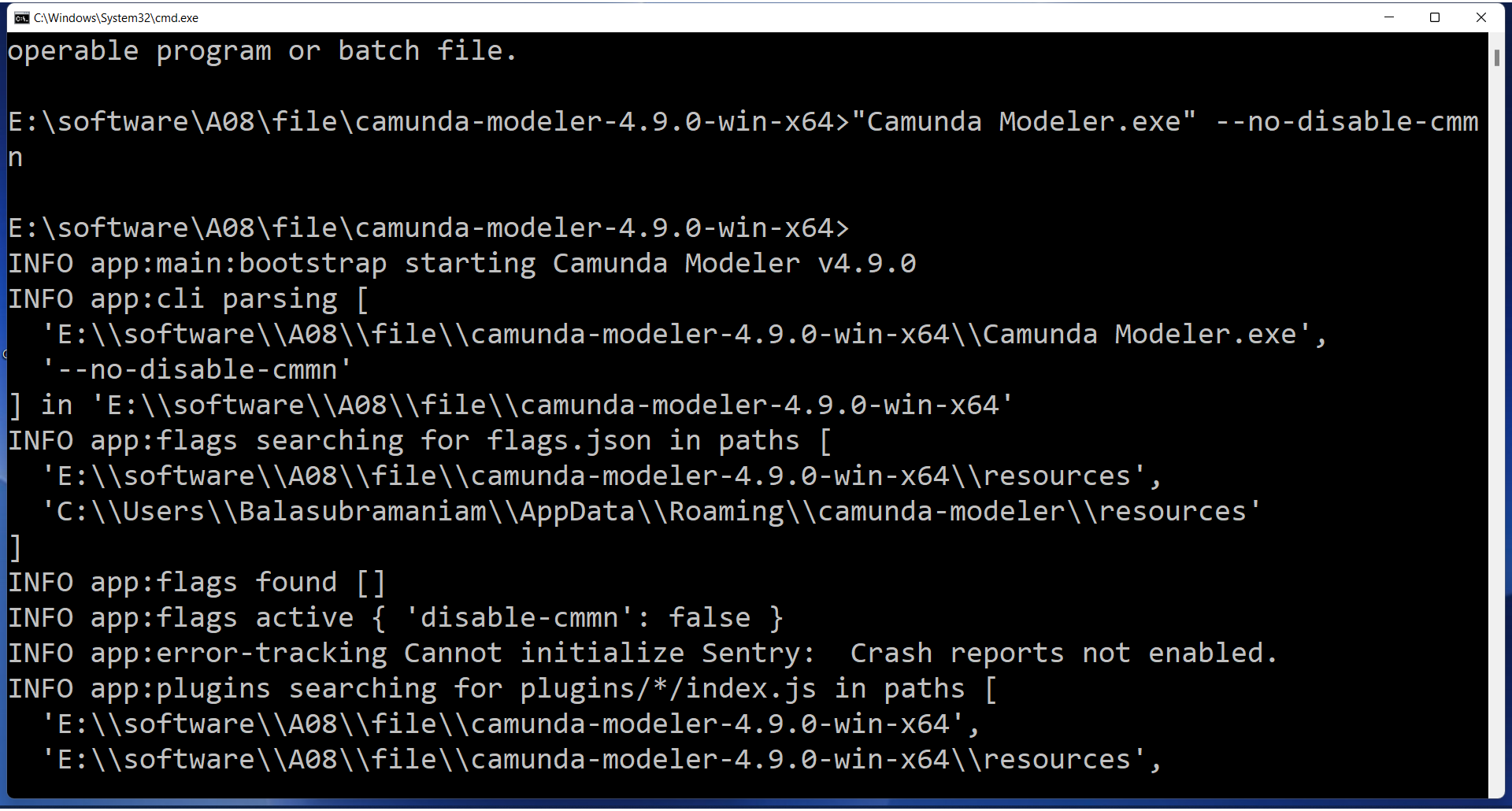
Send Message



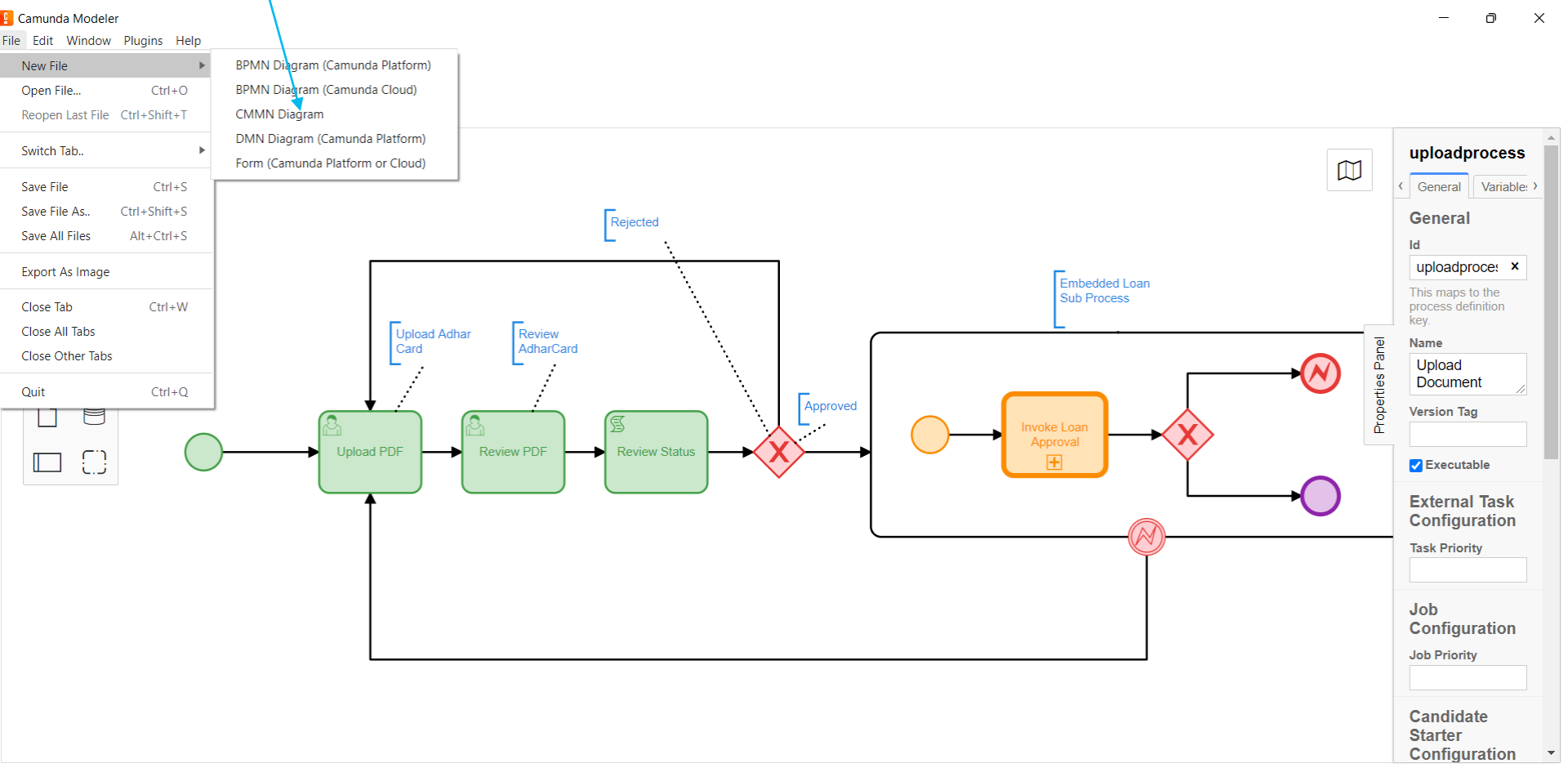
Security


Request
1
Deny
4
ProcessEngine AuthenticationFilter
Request
1
AuthenticationFilter
2
Filters use the IdentityService to check password, get groups, get tenants, and set the user authenticated for the current thread. The IdentityService can be configured, a custom implementation can be plugged in.
2
Allow
3
Camunda REST API
Allow
3
Camunda Cockpit
Allow
3
Camunda Tasklist
Allow
3
Camunda Admin
Deny
4
Login Page
Camunda Web Apps
Camunda Java Engine
Use API
5
Allowed Data
7
Identity Service
Authorization Service
LDAP
Custom
DB
6
Authorizations are managed via the AuthorizationService. During runtime they are enforced within all SQL queries issues against the Camunda database.
Authorizations are always stored in the Camunda DB, user and groups might be stored in an external store (like LDAP).
Camunda SQL Database
External User Store

CMMN


C:\Windows\System32\cmd.exe
operable program or batch file.
E:\software\A08\file\camunda-modeler-4.9.0-win-x64>"Camunda Modeler.exe" --no-disable-cmm
n
E:\software\A08\file\camunda-modeler-4.9.0-win-x64>
INFO app:main:bootstrap starting Camunda Modeler v4.9.0
INFO app:cli parsing [
'E:\\software\\A08\\file\\camunda-modeler-4.9.0-win-x64\\Camunda Modeler.exe',
'--no-disable-cmmn'
] in 'E:\\software\\A08\\file\\camunda-modeler-4.9.0-win-x64'
INFO app:flags searching for flags.json in paths [
'E:\\software\\A08\\file\\camunda-modeler-4.9.0-win-x64\\resources',
'C:\\Users\\Balasubramaniam\\AppData\\Roaming\\camunda-modeler\\resources'
]
INFO app:flags found []
INFO app:flags active { 'disable-cmmn': false }
INFO app:error-tracking Cannot initialize Sentry: Crash reports not enabled.
INFO app:plugins searching for plugins/*/index.js in paths [
'E:\\software\\A08\\file\\camunda-modeler-4.9.0-win-x64',
'E:\\software\\A08\\file\\camunda-modeler-4.9.0-win-x64\\resources',

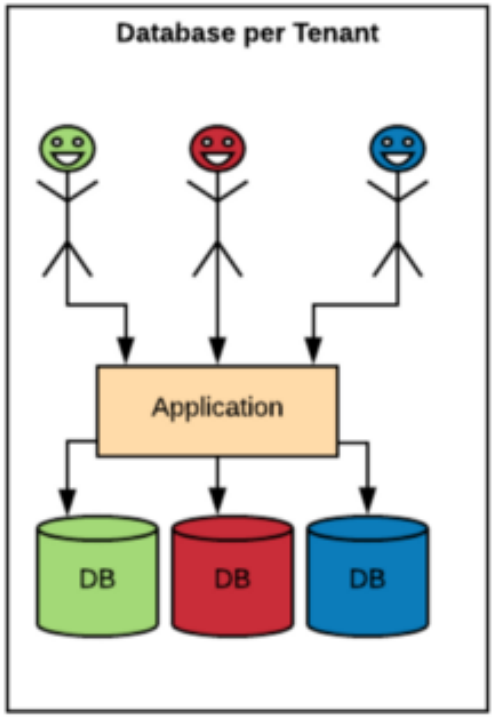
CMMN

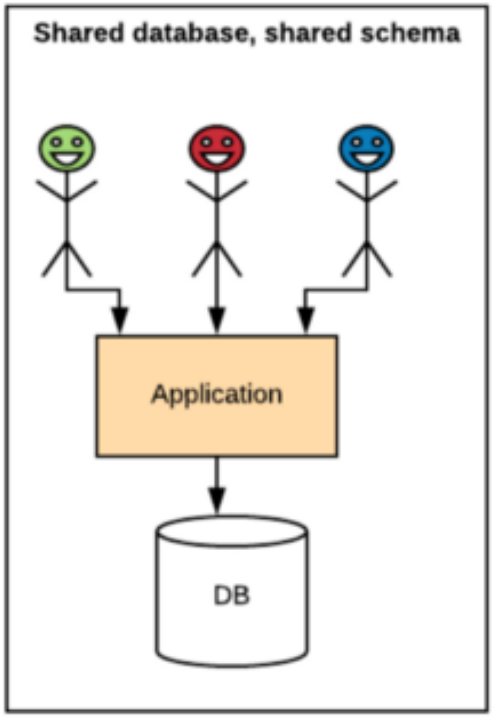

Camunda Modeler
File Edit Window Plugins Help
New File
Open File... Ctrl+O
Reopen Last File Ctrl+Shift+T
Switch Tab..
Save File Ctrl+S
Save File As.. Ctrl+Shift+S
Save All Files Alt+Ctrl+S
Export As Image
Close Tab Ctrl+W
Close All Tabs
Close Other Tabs
Quit Ctrl+Q
BPMN Diagram (Camunda Platform)
BPMN Diagram (Camunda Cloud)
CMMN Diagram
DMN Diagram (Camunda Platform)
Form (Camunda Platform or Cloud)
Rejected
Upload Adhar Card
Review AdharCard
Embedded Loan Sub Process
Approved
Upload PDF
Review PDF
Review Status
Invoke Loan Approval
uploadprocess
General
Variable
Id
uploadproces
This maps to the process definition key.
Name
Upload Document
Version Tag
Executable
External Task Configuration
Task Priority
Job Configuration
Job Priority
Candidate Starter Configuration
Properties Panel
XML
Camunda Platform
Log
4.9.0

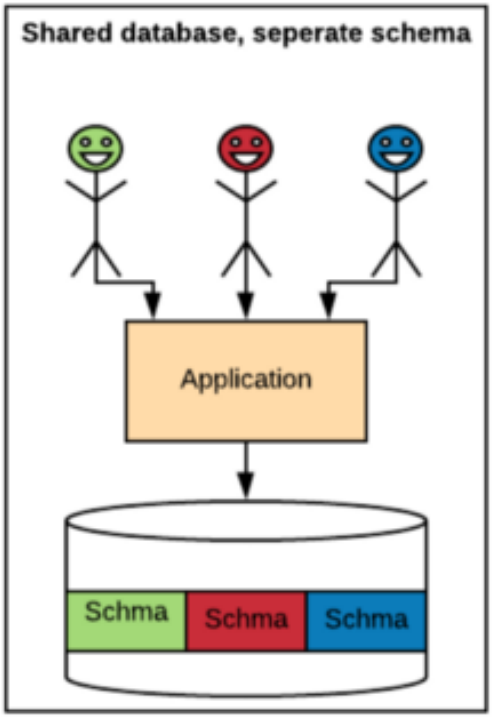
Tenant

- Multi-Tenancy is where the same application is run as multiple incarnations or tenants on the same server.
- That server may be catering for differently defined instances of that application.
- Camunda engine is driving business processes for a SaaS application.
- In this case each tenant is operated on by a completely different client.
- When a client logs into their tenant they'll of course only be expecting to see their own data and functions.

Multi Tenancy







Tenant

- With multi-tenancy, granting a new client access to a tenant has the lowest cost and quickest implementation time.
- The alternative is to deploy and maintain a brand new server, database and software installation each time a new client joins.

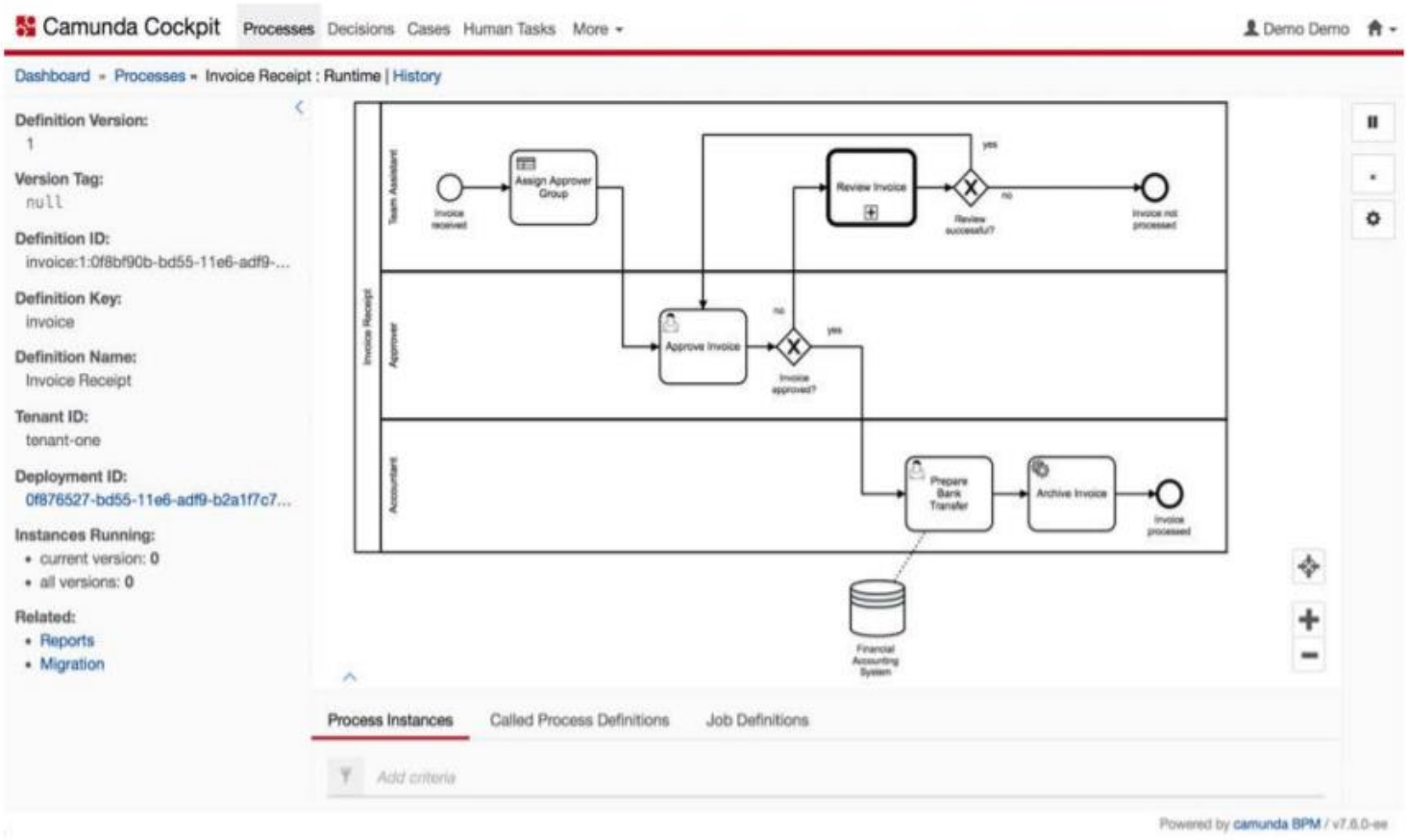
Tenant

- **Data Isolation vs. Resource Consumption**
- Each type of multi-tenancy implementation walks a line between data isolation and resource consumption.
- To have the maximum possible data isolation (i.e. a single database per tenant) requires the use of a lot of resources and is generally quite wasteful of those resources – and so this is often avoided.
- On the other end of the spectrum is of course perfectly optimized resource consumption (i.e. all tenants sharing the same database and tables) but in this case all it takes is a lazily written database query to completely break down the data isolation between tenants.

Tenant Identifiers

- Starting on the scale of least data isolation is Tenant Identifiers.
- This means that all tenants run separately but write data to exactly the same tables.
- Each running tenant requires a unique marker in order to gain access to their tenant specific data.
- This means that all queries and tables must have this tenant identifier added or data isolation will be lost.
- Camunda introduced out-of-the-box support for this approach with version 7.5 of the product by adding the tenant ID to all relevant database tables as well as to internal queries.

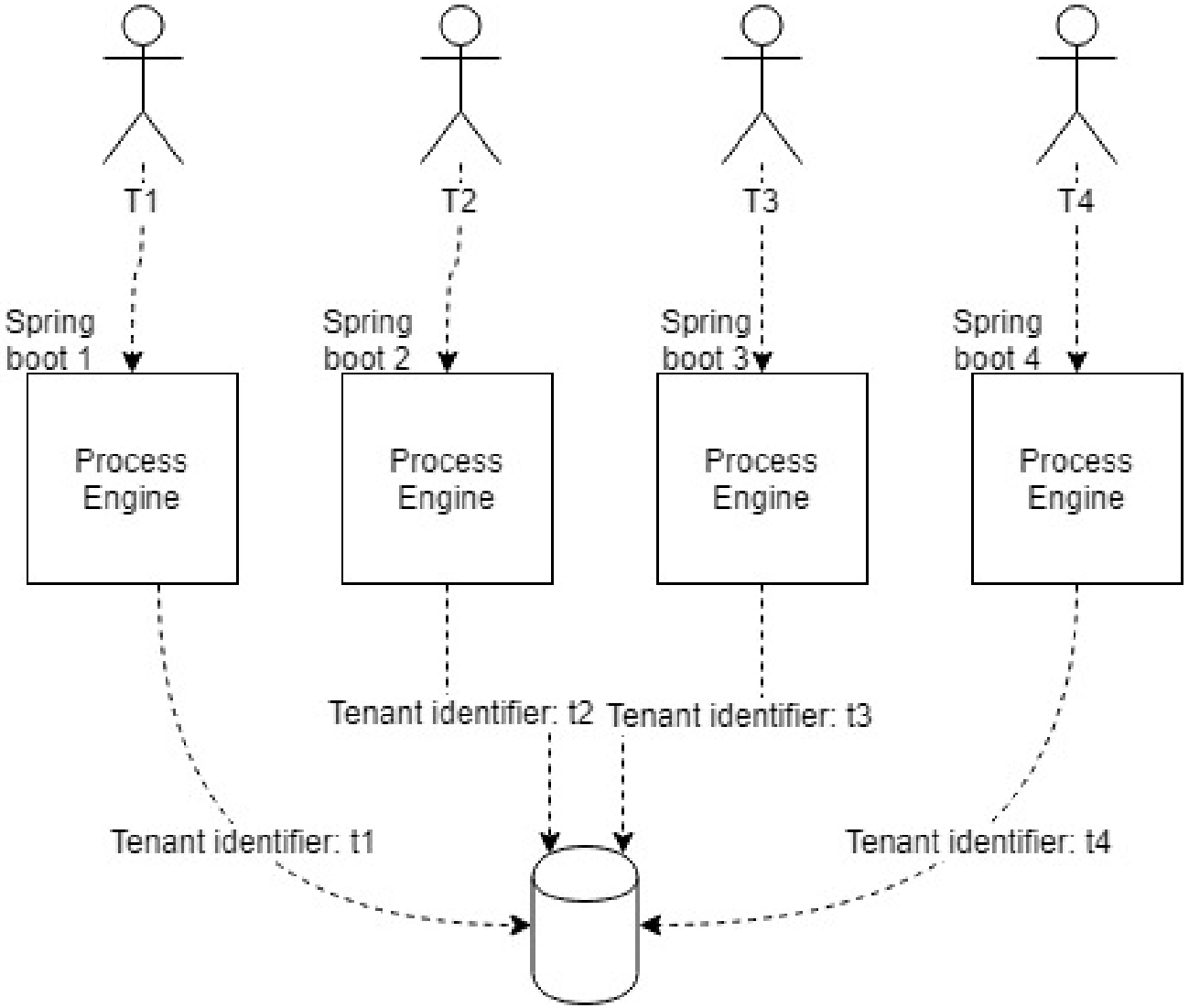
Tenant Management in Cockpit



Why Multitenancy

#	Data Isolation	Resource Consumption	Multi-Tenant Option
1	Very Important	Unimportant	Server per Tenant
2	Unimportant	Very Important	Tenant Markers
3	Important	Important	Schema per Tenant

Multi Tenancy in Spring boot



Specify the Tenant Identifier via Java API

- repositoryService
- .createDeployment()
- .tenantId("tenant1")
- .addZipInputStream(inputStream)
- .deploy()

Deployment Descriptor


```
<process-application
  xmlns="http://www.camunda.org/schema/1.0/ProcessApplication"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <process-archive tenantId="tenant1">
    <process-engine>default</process-engine>
    <properties>
      <property name="isDeleteUponUndeploy">false</property>
      <property name="isScanForProcessDefinitions">true</property>
    </properties>
  </process-archive>
</process-application>
```

Using Spring

- <bean id="processEngineConfiguration" class="org.camunda.bpm.engine.spring.SpringProcessEngineConfiguration">
- <property name="deploymentResources">
- <array>
- <value>classpath*:/org/camunda/bpm/engine/spring/test/autodeployment/autodeploy.*.cmmn</value>
- <value>classpath*:/org/camunda/bpm/engine/spring/test/autodeployment/autodeploy.*.bpmn20.xml</value>
- </array>
- </property>
- <property name="deploymentTenantId" value="tenant1" />
- </bean>

Query Tenant Id


```
List<Deployment> deployments = repositoryService
  .createDeploymentQuery()
  .tenantIdIn("tenant1", "tenant2")
  .orderByTenantId()
  .asc()
  .list();
```

Understanding Users, Groups and Tenants

- A user refers to a human individual and a group is any custom defined "bundle" of users sharing some usage relevant attributes (like e.g. working on specific business functions).
- Setup groups corresponding to your workflow roles or create new logical roles for that purpose.
- Both groups and users can be added to one or more tenants in order to ensure a certain degree data isolation between different logical entities

Understanding Users, Groups and Tenants

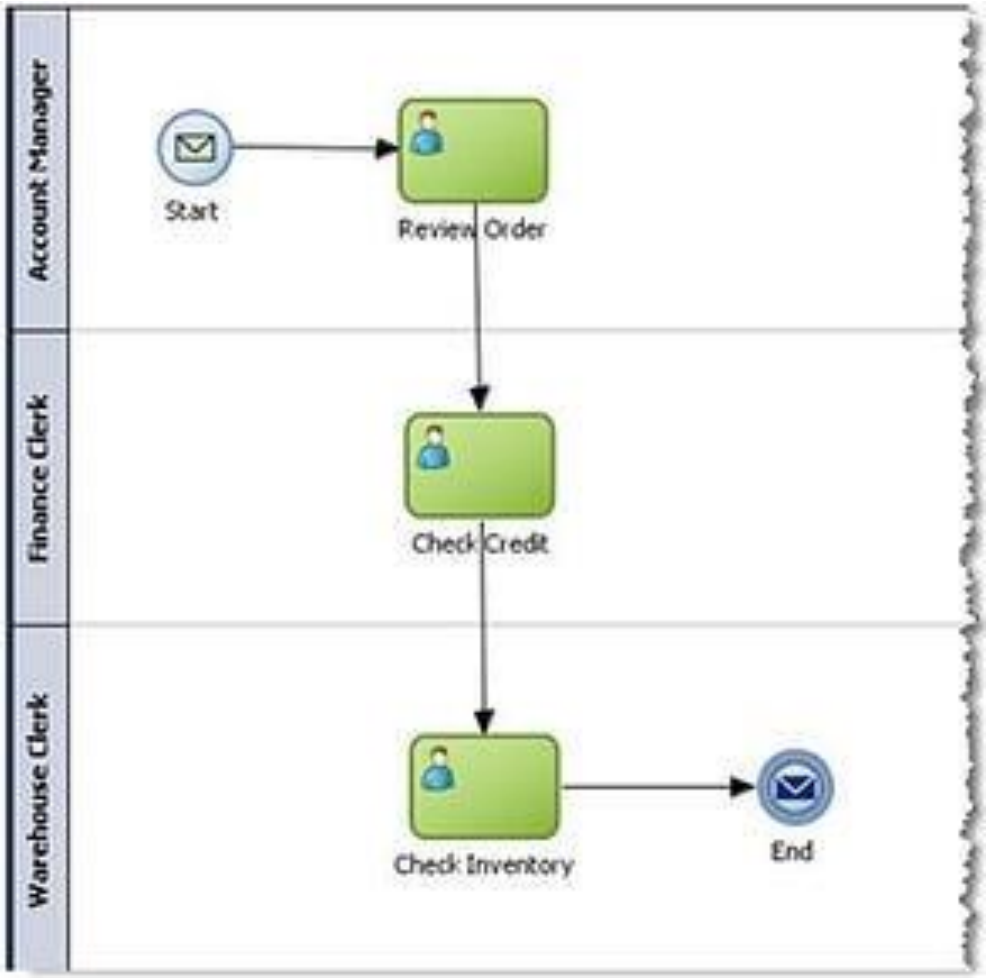
- <bpmn:userTask id="Invoice customer" camunda:candidateGroups="finance"/>
- taskService.claim(taskId, "fozzie");

Camunda Patterns

- Procedural Patterns
- Advanced Branching and Synchronization Patterns
- Structural Patterns
- Multiple Instance Patterns
- State Based Patterns
- Cancellation Patterns

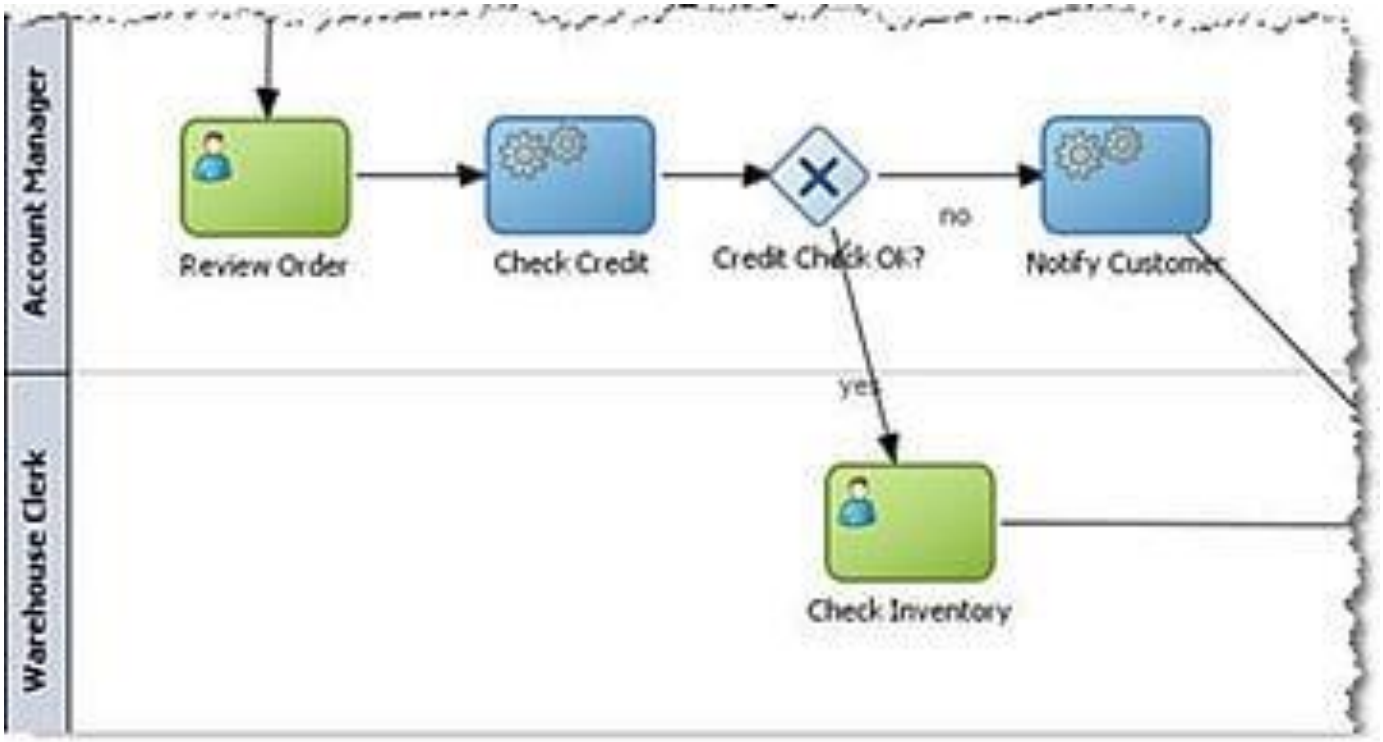
Procedural Patterns

Sequence Pattern



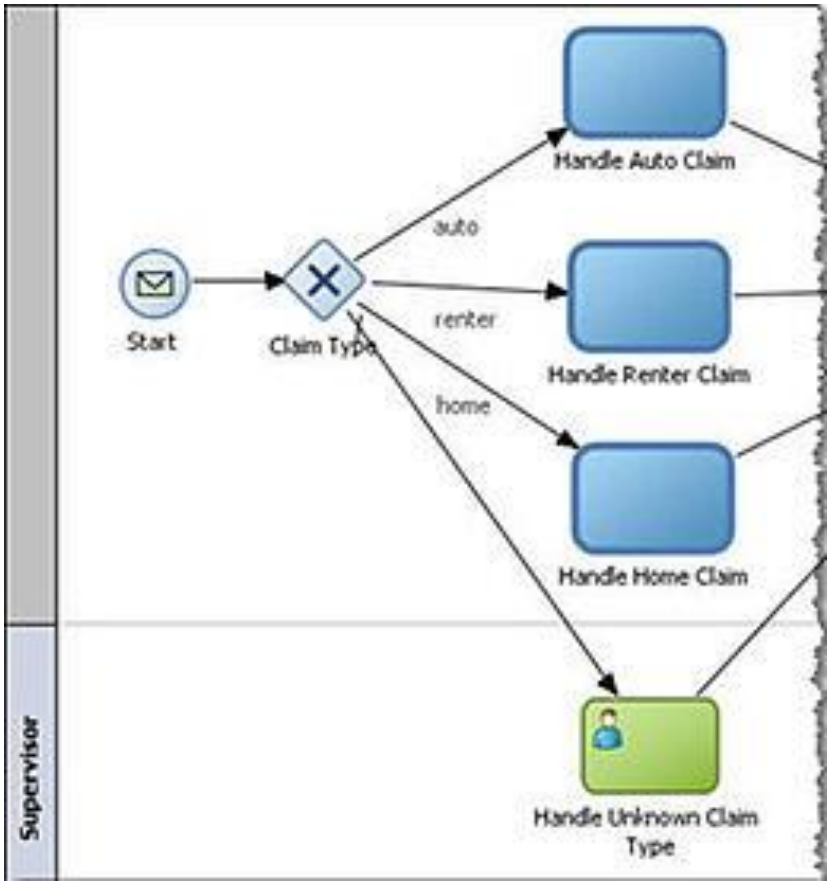
Procedural Patterns

Exclusive Choice Pattern



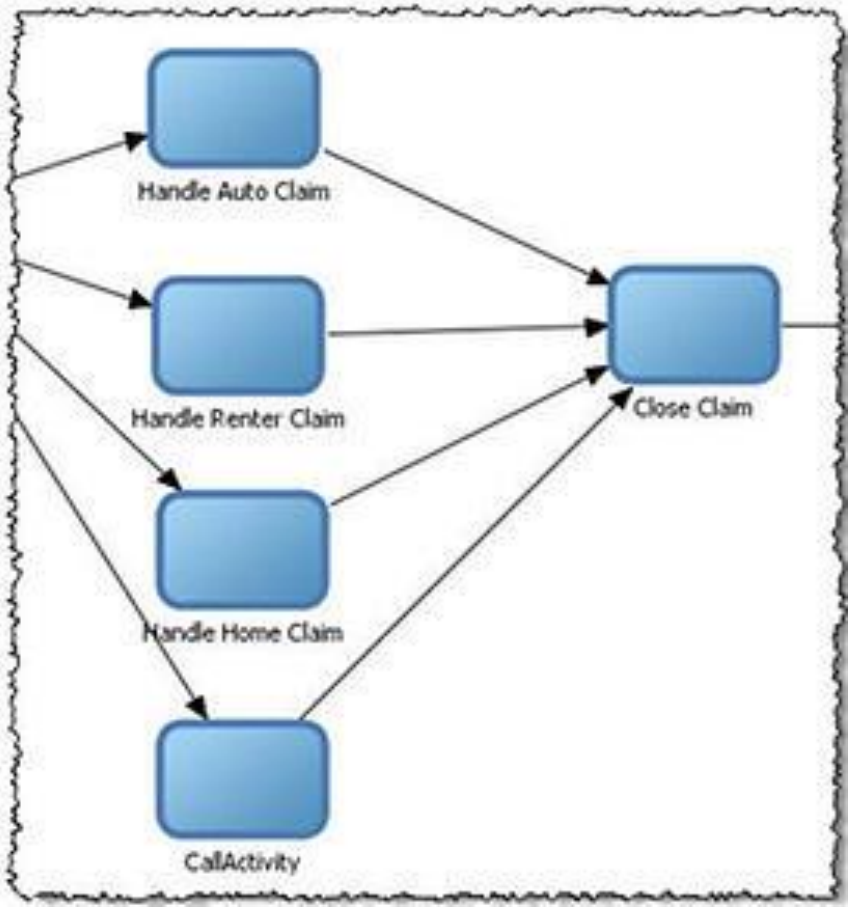
Procedural Patterns

Exclusive Choice Pattern



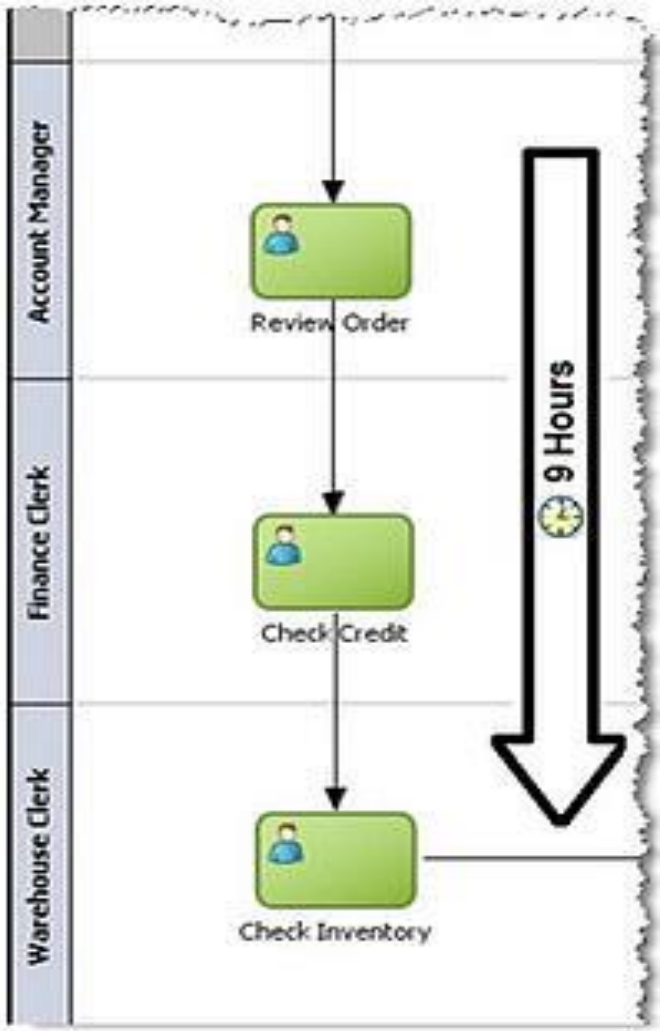
Procedural Patterns

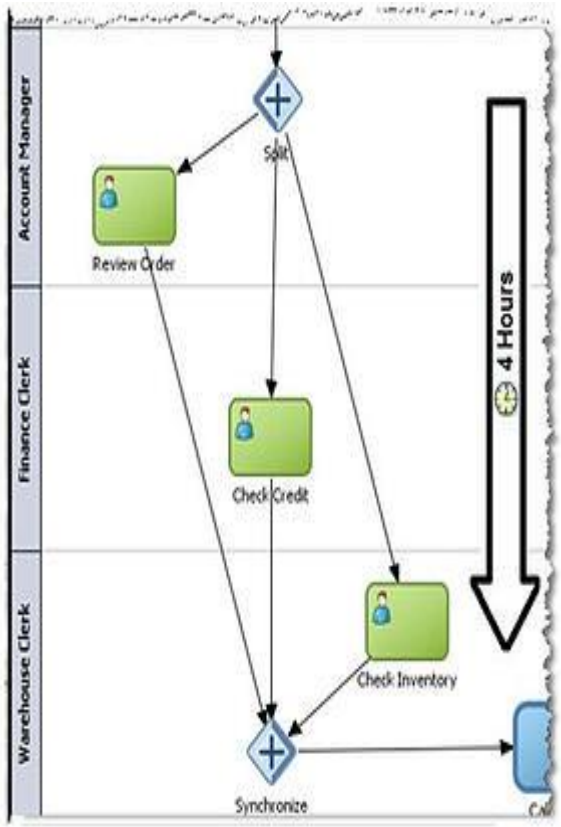
Simple Mega Pattern



Procedural Patterns

Parallel Split and Synchronization





Parallel Split and Synchronization Pattern

- In this example, an Exclusive Gateway activity was used to simultaneously ensure:
- an order can be reviewed (taking 2 hours), while
- the customer's credit is being checked (taking 3 hours), while
- the inventory for the items ordered is being checked (taking 4 hours).
- Instead of taking 9 hours, the order now takes the 4 hours to complete the three activities (the longest of the three activities in the parallel paths).
- The Parallel Split and Synchronization pattern speeds up the process by having the instance travel all the parallel paths through it simultaneously.
- The order the activities execute is not important in this pattern.

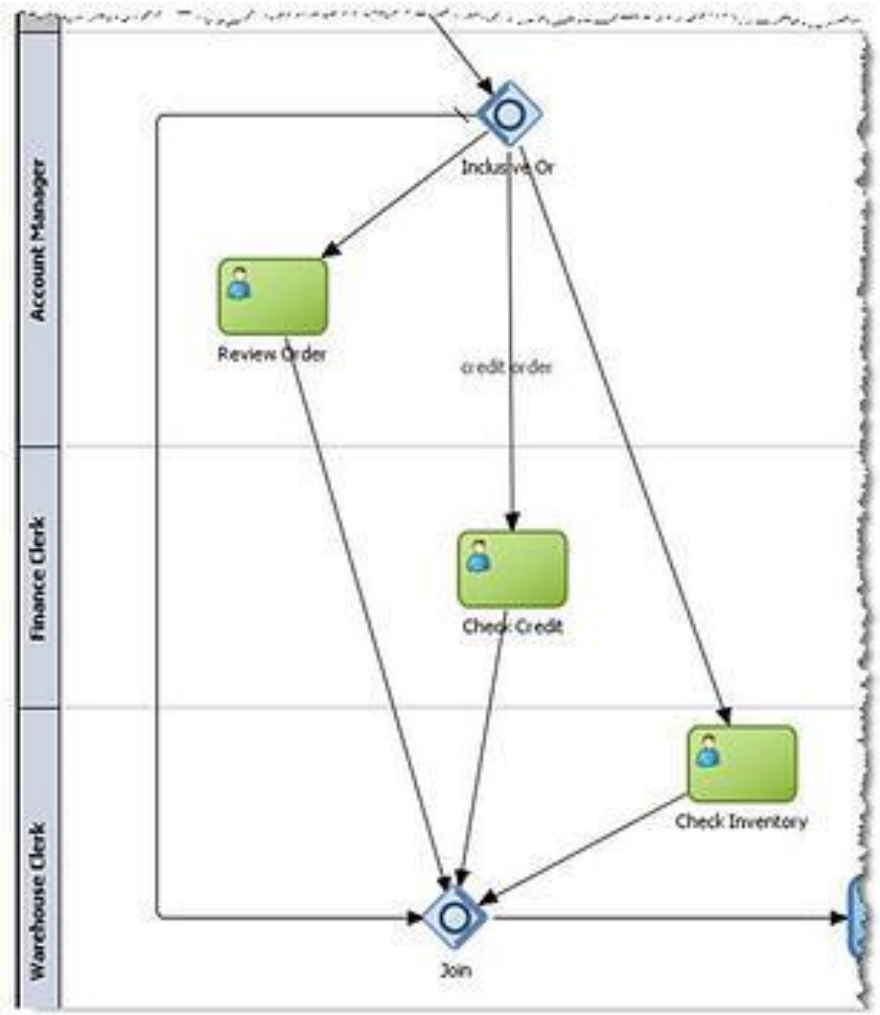
Procedural Patterns

Parallel Split and Synchronization Pattern

- Using the above example, an instance might execute the activities in the process in any one of these sequences:
- (a) Check Inventory
- (b) Check Credit
- (c) Review Order
- (a) Check Inventory
- (b) Review Order
- (c) Check Credit
- (a) Check Credit
- (b) Review Order
- (c) Check Inventory
- (a) Check Credit
- (b) Check Inventory
- (c) Review Order
- (a) Review Order
- (b) Check Credit
- (c) Check Inventory
- (a) Review Order
- (b) Check Inventory
- (c) Check Credit

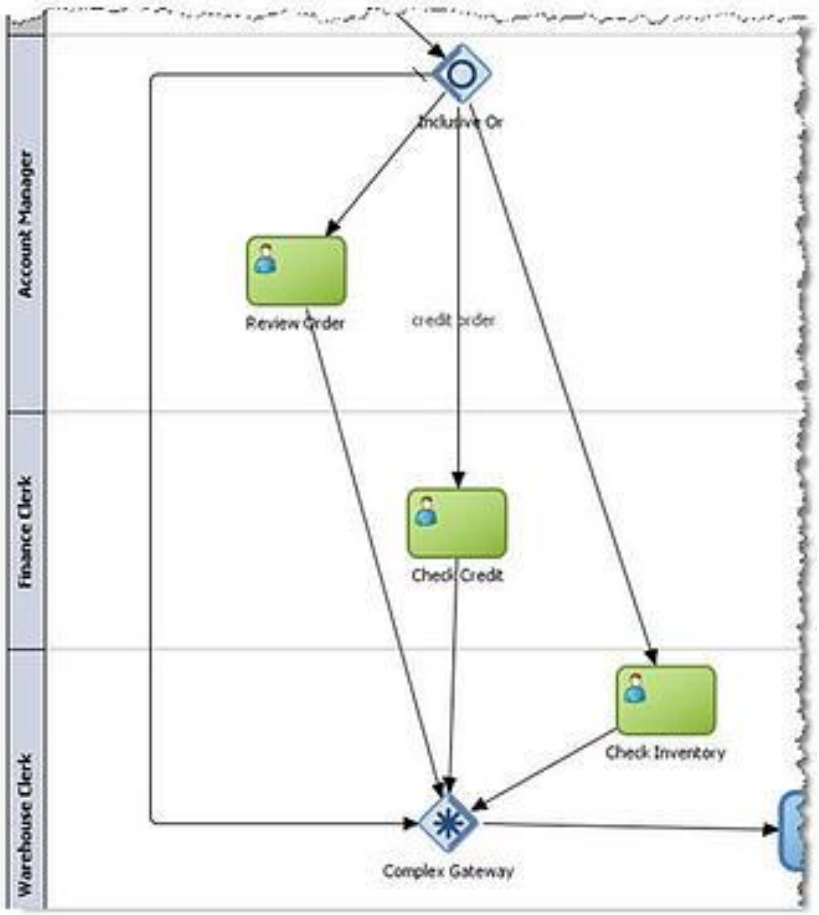
Advanced Branching and Synchronization Patterns

- **Multiple Choice and Synchronizing Merge Patterns**



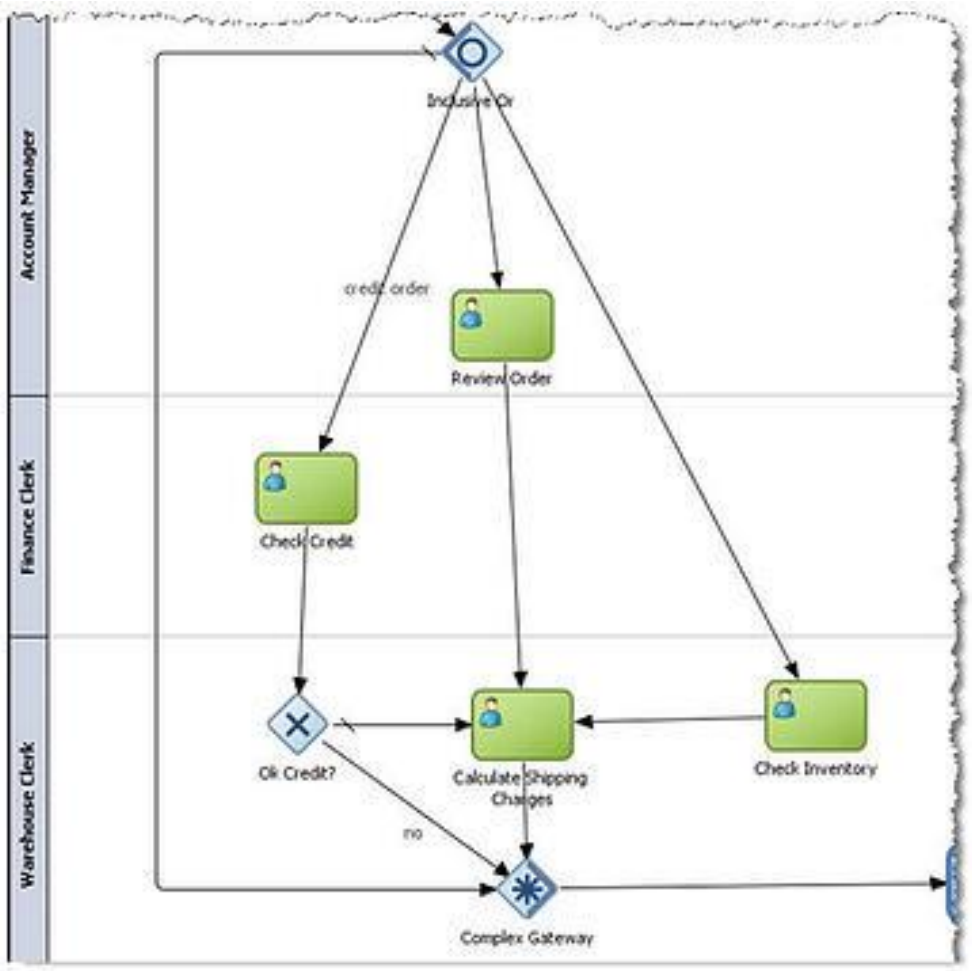
Advanced Branching and Synchronization Patterns

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



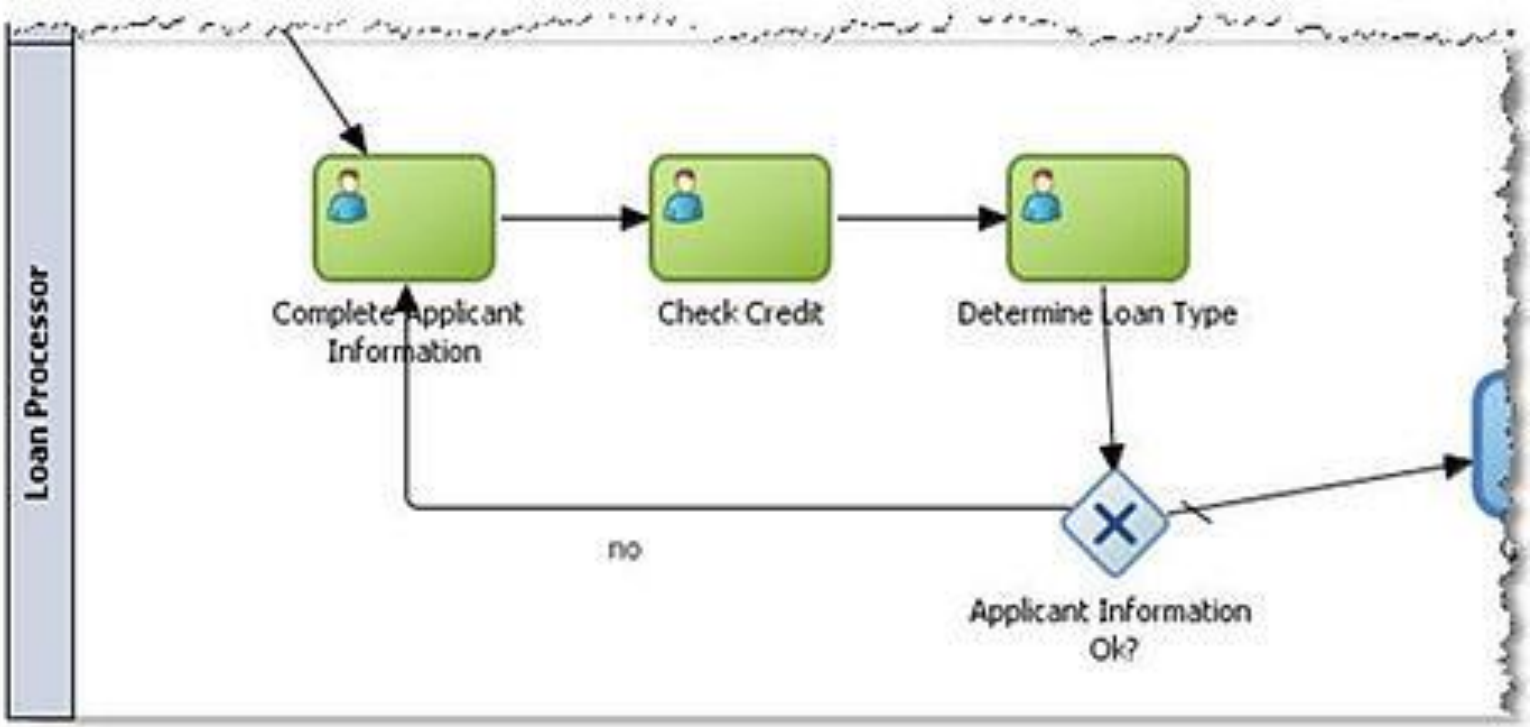
Advanced Branching and Synchronization Patterns

- **Multiple Merge Patterns**



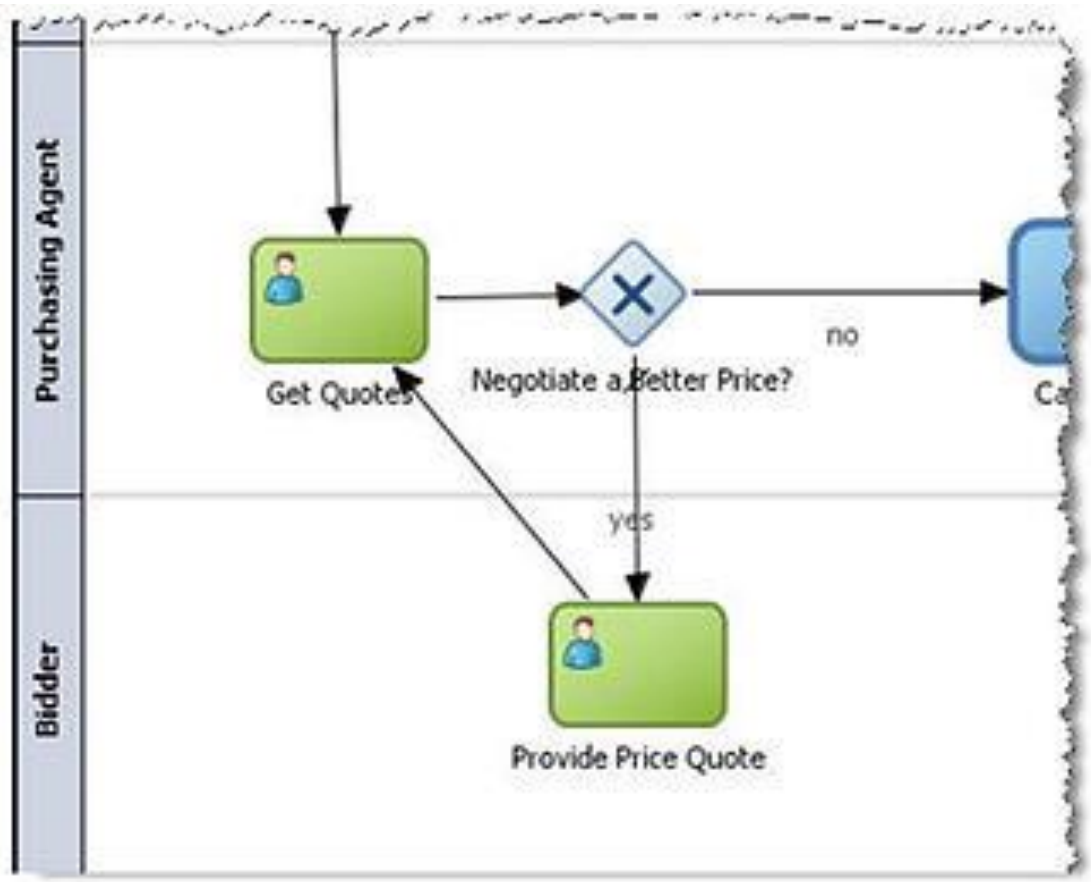
Structural Patterns

- **Arbitrary Cycles Pattern**



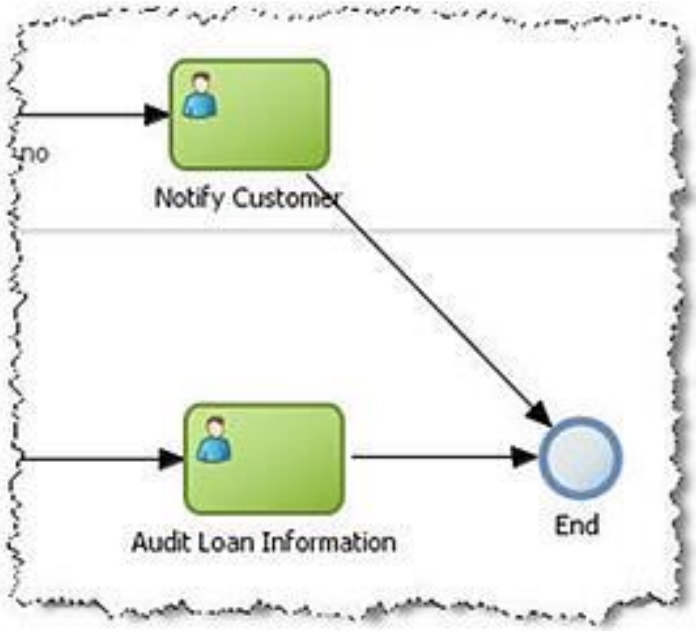
Structural Patterns

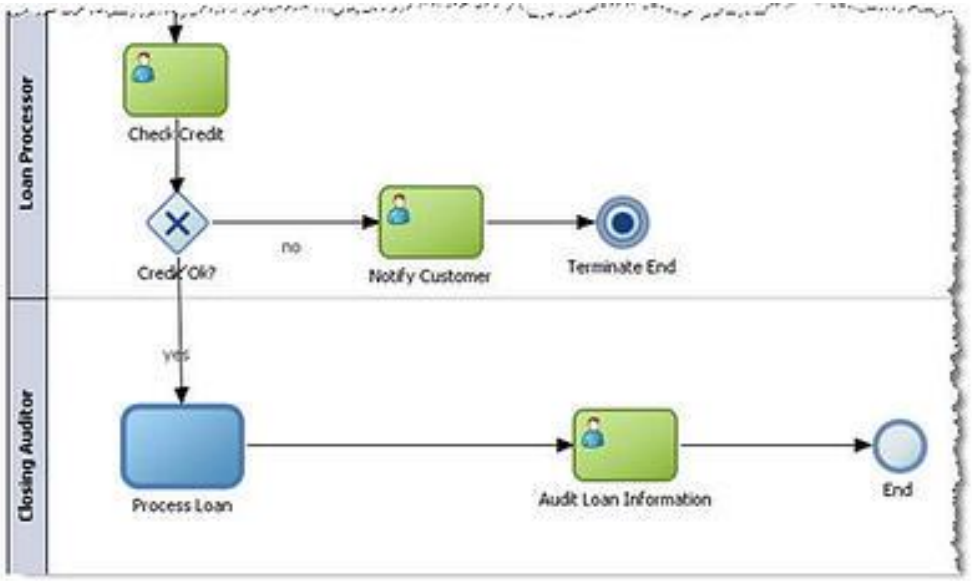
- Collaboration Pattern



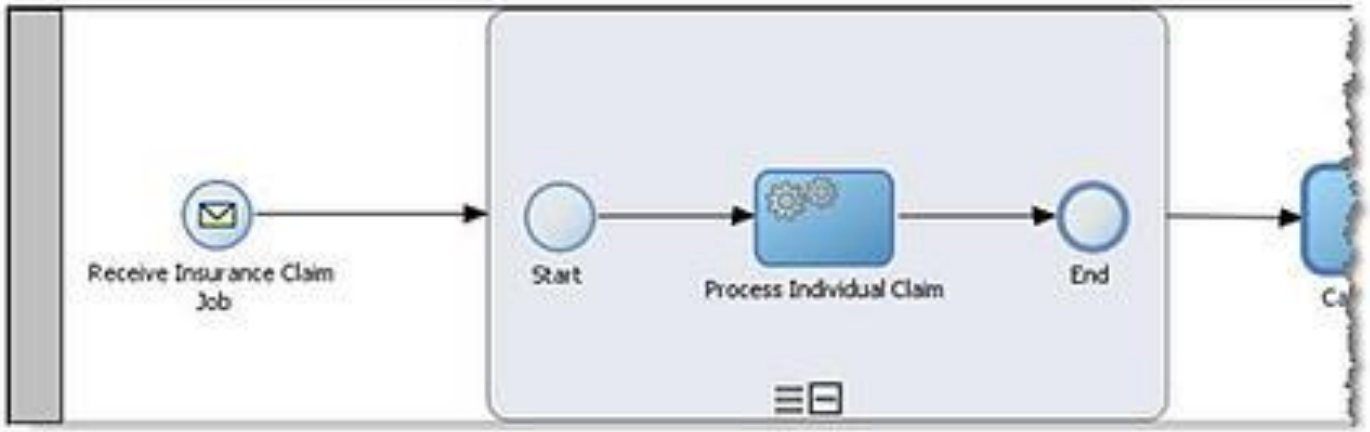
Structural Patterns

- Implicit Termination Pattern





Multiple Instance Patterns



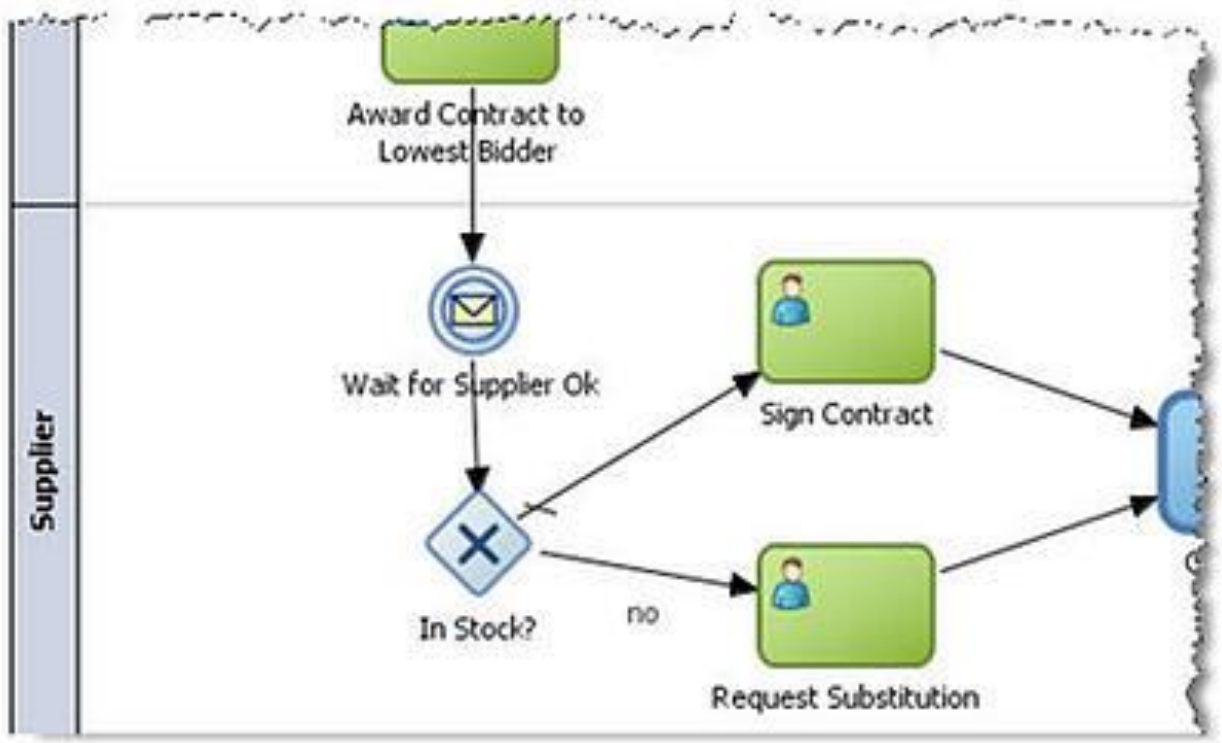
Here, depending on the number of claims in a job, the Subprocess activity is set to handle multiple instances. It simultaneously parses each claim and each claim is then individually sent without synchronization (asynchronously) to a web service activity. While and individual claim is sent to the web service, the context in the parent process the original batch job. For every claim in the batch job, a new instance is created in the Subprocess activity.

Multiple Instance Patterns

Here, depending on the number of claims in a job, the Subprocess activity is set to handle multiple instances. It simultaneously parses each claim and each claim is then individually sent without synchronization (asynchronously) to a web service activity. While and individual claim is sent to the web service, the context in the parent process the original batch job. For every claim in the batch job, a new instance is created in the Subprocess activity.

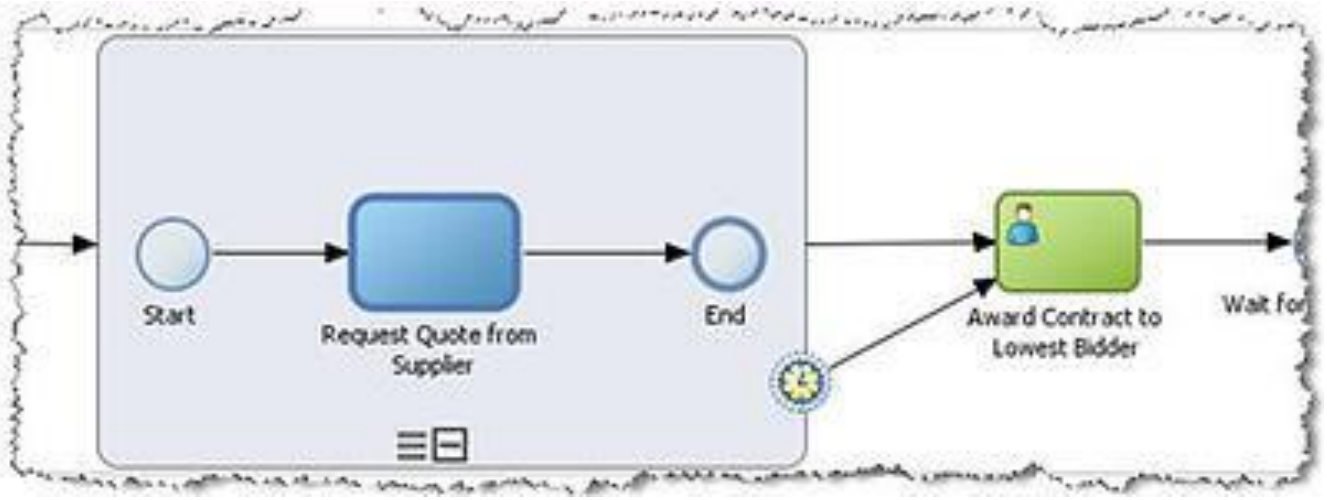
State Based Patterns

Deferred Choice Pattern



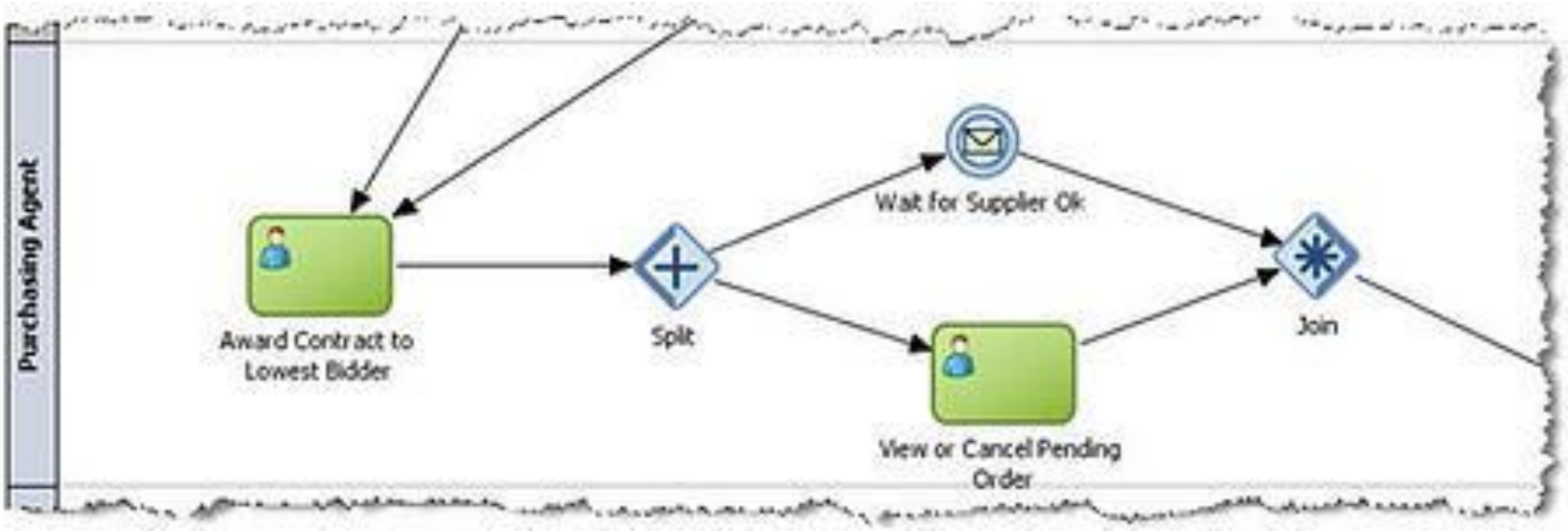
State Based Patterns

Milestone Pattern



Cancellation Patterns

Cancel Activity Pattern

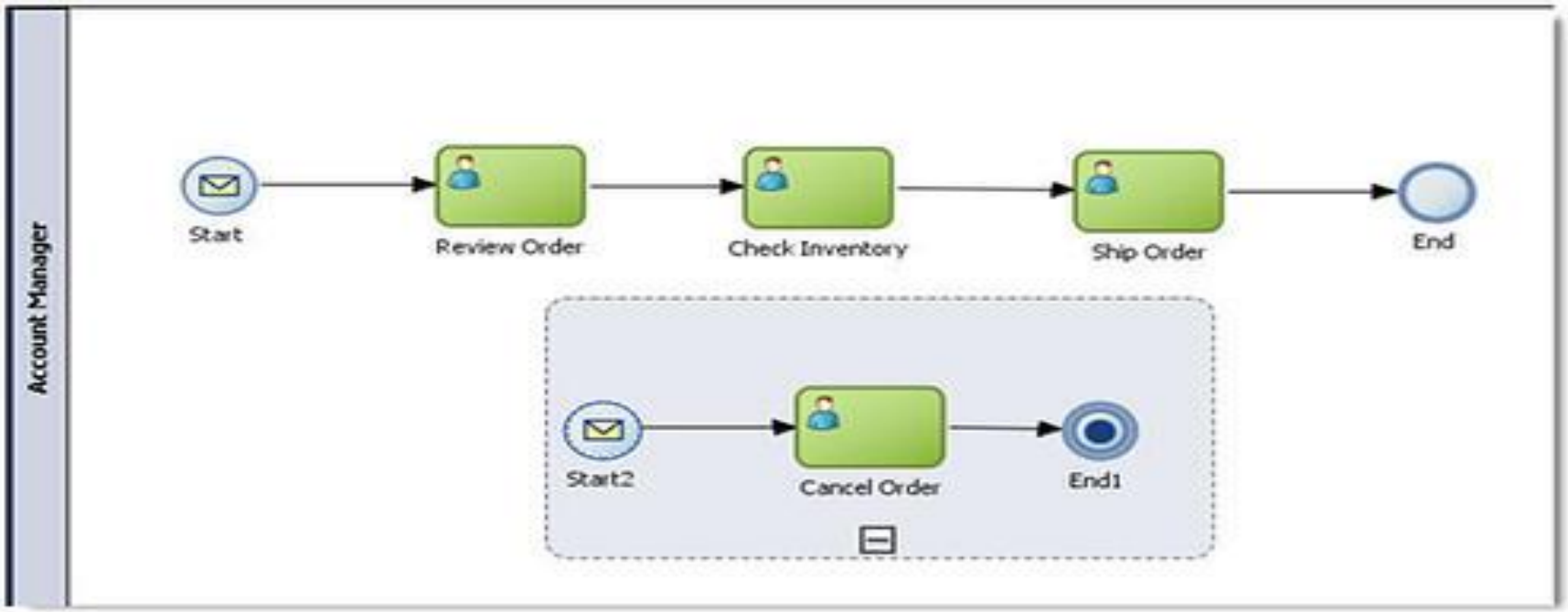


Give the Purchasing Agent the chance to view the bids that are pending in the upper leg and Allow the Purchasing Agent the chance to cancel any orders that are pending in the upper leg.

Once the clerk cancels an instance, it reaches the Join activity. The instance is removed from the Wait for Supplier Ok activity and it continues on through the rest of the process.

Cancellation Patterns

Cancel Case Pattern



Cancellation Patterns

Cancel Case Pattern

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

Cancellation Patterns

Cancel Case Pattern

- Instead, consider this pattern.
- Here the Event Subprocess appears to not be part of the process flow.
- Once a message is received that initiates the subprocess, it interrupts the instance in any activity of the process and direct it immediately to the Cancel Order activity.
- Notifications cause instances sitting anywhere in the process (Review Order, Check Inventory or Ship Order) to immediately and automatically move to the Customer Cancellation activity. Sequence flows do not need to be drawn to this type of activity.

Questions