



Drools

David THIBAU – 2024

david.thibau@gmail.com



Agenda

- Introduction

- BRMS
- Présentation KIE
- Projets KIE
- Kogito

- DMN

- Démarrage
- Compléments



Introduction

BRMS

Présentation KIE

Projet KIE

Kogito

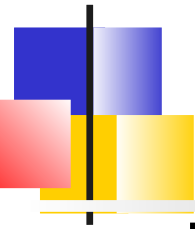


Problématique

Aujourd'hui, le principal challenge des applications métier est l'agilité

Les applications doivent pouvoir s'adapter rapidement et réagir :

- A des demandes d'évolutions fonctionnelles
- A des changement de règles métier
- A des changements d'organisation
- A l'intégration de nouveaux systèmes



Système de gestion de règles

Un **système de gestion de règles (BRMS¹)**

identifie la notion de règle métier comme une ressource pouvant être gérée indépendamment du code applicatif

- Les règles peuvent donc être gérées, versionnées, monitorées
- Elles peuvent être mises à jour par un technicien ou un expert métier sans redéploiement de l'application
- Elles peuvent être testées indépendamment
- Elles peuvent être documentées et auditées



Moteur de règle

Les BRMS intègrent un **moteur de règle**

C'est un composant qui évaluent des instructions IF-THEN basées sur le modèle métier : les règles

Lorsque les conditions des règles sont satisfaites (*IF*), il exécute les actions associées (*THEN*) qui peuvent modifier le modèle

=> La programmation devient alors déclarative

=> La logique métier n'est plus dispersée dans le code mais centralisée dans un repository de règles



Scénario d'utilisation

Service de règle :

- Parse, compile stocke l'ensemble des règles.
- Expose une API (Rest ou autre)

Application Client :

- Obtient une référence sur le service de règle
- Insère des faits (Objets du domaine) dans la mémoire de travail
- Demande au service de déclencher les règles
- Récupère les objets du domaine modifiés par le service de règles



Mise en place

Adopter une solution basée sur les règles nécessite :

- L'identification des règles métier : **Expert métier**
- Leur organisation et leur expression dans un langage de règles : **Expert métier / technique**
- L'intégration du service de règles à l'application :
Technique : Embarqué ou distant ?
- L'utilisation d'une UI de gestion des règles (BRMS) :
Principalement **Métier**
- La mise en place de procédure de mise à jour
(Automatisation des tests/Déploiement) :
Technique



Considérations

- Un moteur de règles s'appuie sur des technologies relativement complexes
- Il peut être déroutant de se baser sur une boîte noire
- Comprendre/Debugger le déclenchement n'est pas très intuitif
- L'extraction des règles n'est pas toujours simple



Avantages

- **Programmation déclarative** : Les moteurs de règles permettent de spécifier « Quoi faire » et non pas « Comment le faire ». Ils sont capables de résoudre des problèmes difficiles et en plus de fournir des explications !
- **Séparation de la logique et des données** : Les données sont dans les objets du domaine, la logique centralisée dans un fichier de règles. La logique n'est plus dispersée dans du code.
- **Centralisation de la connaissance** : Tout est centralisé dans la base de connaissance, relativement lisible et pouvant servir de documentation
- **Règles compréhensibles** : Les règles peuvent être exprimées dans des formalisme accessibles à des profils non technique.



Quand utiliser un moteur de règles ?

Le problème est trop complexe pour le code classique (problèmes d'optimisation par exemple, système expert)

Le problème n'est pas complexe mais aucune solution robuste s'impose.

La logique change souvent, dans ce cas les règles peuvent être changées rapidement sans trop de risques.

Les experts métiers existent mais ne sont pas techniques. Les règles permettent alors d'exprimer la logique métier dans leur propre termes.



Domaines et cas d'utilisation

Domaines

- Finance et Assurance : Aide à la décision
- Réglementation, règles gouvernementales
- E-commerce, campagne de promotion
- Gestion des risques

Exemple

- Autorisation basée sur de nombreux critères (Rôle, propriété de l'entité, organisation, Localité, ...)
- Personnalisation d'application (ex : gestion d'une page d'accueil personnalisée d'un site de e-commerce)
- Diagnostique
- Validation complexe
- Problèmes d'optimisation (routage, planning, ...)



Introduction

BRMS
Présentation KIE
Projet KIE
Kogito



Projets KIE

KIE (Knowledge Is Everything) regroupe plusieurs projets qui partagent la même API et les mêmes techniques de construction et de déploiement

- **Drools** : Moteur de règles et traitement complexe d'événements (Complex Event Processing)
- **JBPM** : Moteur de workflow
- **OptaPlanner** permet d'optimiser des problèmes complexes soumis à des contraintes (Planification, itinéraire des véhicules, ...)



Base de connaissances et session

Les projets KIE sont constitués de bases de connaissances dans lesquelles des règles et processus métier ont été compilés.

Les applications clientes interagissent avec les bases de connaissance par l'intermédiaire de session qui permettent d'évaluer des règles, démarrer ou faire progresser des processus.

L'interaction peut se faire :

- Via une API
- Via une API Rest, si la base de connaissance est déployée dans un service



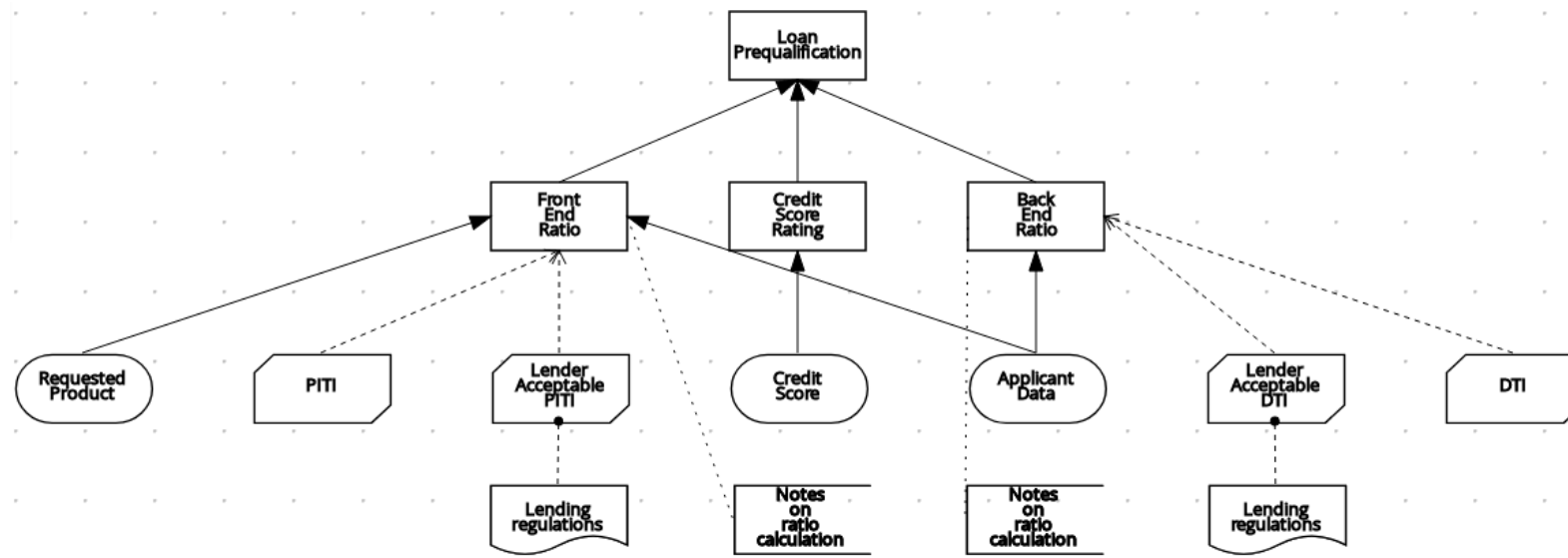
Syntaxe Drools

Drools supporte différentes syntaxes pour exprimer les règles :

- **Decision Model and Notation (DMN)** :
Standard OMG : Diagramme de décision basé sur XML
- **Tables de décision** : Tableur Excel avec un format prédéfini
- **DRL** : À l'origine de Drools permet l'inférence
- **Predictive Model Markup Language (PMML)** :
Modèles d'analyse de données prédictives en XML (~Machine learning)

Exemple DMN

Les DMN permettent une interaction stateless, i.e on fournit des données d'entrées et on obtient un résultat en sortie.





Tables de décision

Les tables de décision peuvent être vues comme un sous-ensemble simplifié de DMN. L'interaction est également stateless

Exemple

	B	C	D	E
7				
8				
9		RuleSet	Some business rules	
10		Import	org.drools.decisiontable.Cheese, org.drools.dec	
11		Sequential	true	
12				
13		RuleTable Cheese fans		
14		CONDITION	CONDITION	ACTION
15		Person	Cheese	list
16	(descriptions)	age	type	add(* \$param*)
17	Case	Persons age	Cheese type	Log
18	Old guy	42	stilton	Old man stilton
19	Young guy	21	cheddar	Young man cheddar
20				
21		Variables	java.util.List list	



DRL

Drools Rules Language est un langage spécifique qui permet des interactions stateless ou stateful.

Dans une interactions stateful :

- On insère des faits
- On déclenche le moteur
- Une règle est sélectionnée, elle peut modifier les faits d'entrée
- La modification des faits peut alors déclencher d'autres règles
- L'interaction se termine lorsqu'il n'y a plus de règle à appliquer



Exemple – Règle

```
rule "When there is a fire turn on the sprinkler"
```

```
when
```

```
    Fire($room : room)
```

```
    $sprinkler : Sprinkler( room == $room, on ==  
false )
```

```
then
```

```
    modify( $sprinkler ) { setOn( true ) };
```

```
    System.out.println( "Turn on the sprinkler for room  
" + $room.getName() );
```

```
end
```



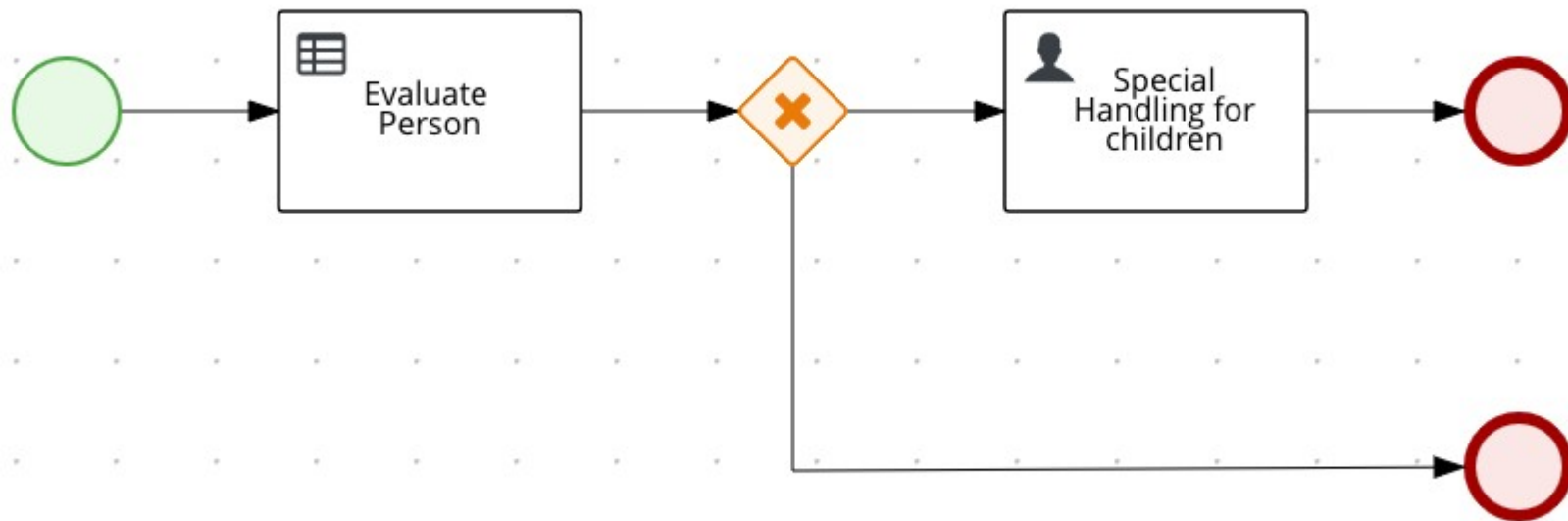
JBPM et BPMN2.0

JBPM permet de modéliser des processus métier pouvant inclure des tâches humaines.

L'interaction avec jBPMN est également stateful :

- Un évènement démarre un processus métier. Certains nœuds sont exécutés, l'état de processus est persisté
- Un autre évènement comme la fin d'une tâche utilisateur, un message technique fait progresser le processus puis stocke son état
- Le processus à un évènement de fin

Exemple BPMN2





Drools et jBPMN

Drools et jBPM se complètent, permettant aux utilisateurs finaux de décrire les connaissances métier à l'aide de différents paradigmes: les règles et les processus

Les projets partagent les concepts de KIE :

- La même API (même classe Java)
- Les mêmes patterns d'intégration avec les applications clientes
- Les mêmes mécanismes de construction et de déploiement



Accéder aux processus à partir des règles

En utilisant DRL, le déclenchement d'une règle peut interagir avec les processus métier :

- créer, annuler et signaler des processus
- Accédez au gestionnaire de tâches utilisateurs pour finaliser, annuler une tâche utilisateur



Les instances de processus comme faits

Toujours avec DRL, L'insertion d'instances de processus en tant que faits dans le moteur de règles permet d'écrire des règles sur nos processus ou groupes de processus

```
rule "Too many orders for just our Managers"
when
    List($managersCount:size > 0) from collect(Manager())
    List(size > ($managersCount * 3)) from
        collect(WorkflowProcessInstance(processId == "process-
order"))
then
    //There are more than 3 Process Order Flows per manager.
    // Please hire more people :)
end
```

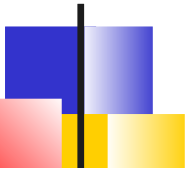


BPMN2 Business Rule Tasks

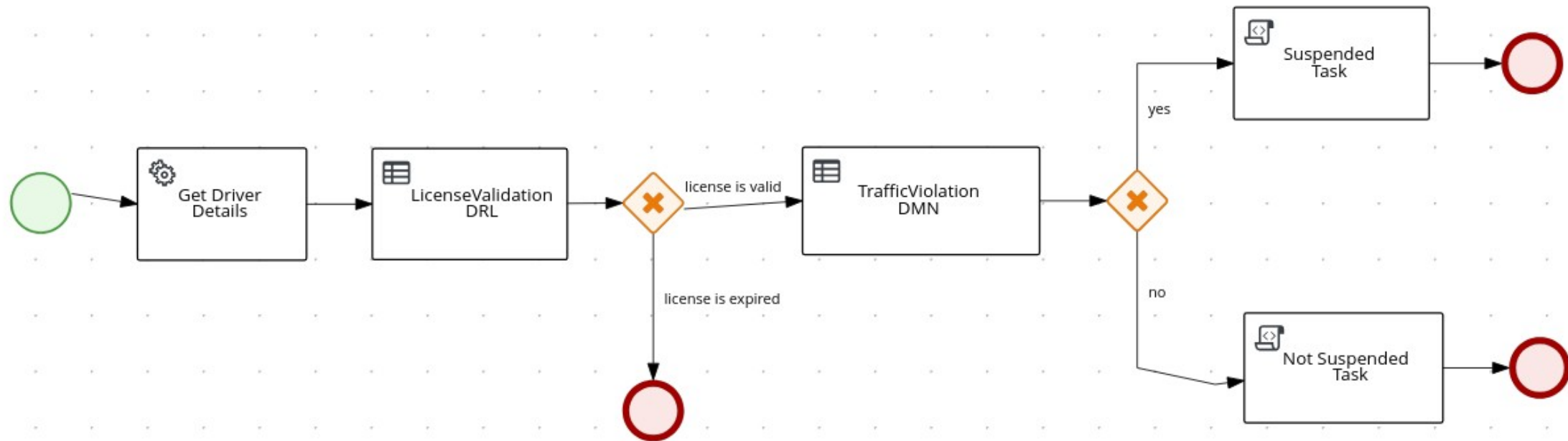
La spécification BPMN2 propose un type de nœud spécifique appelé ***Business Rule Task***


Lorsque le processus atteint le nœud, les règles (dans les différentes syntaxes) sont évaluées.

En fonction du retour du moteur de règle, le processus évolue différemment.



Exemple Business Rule Task

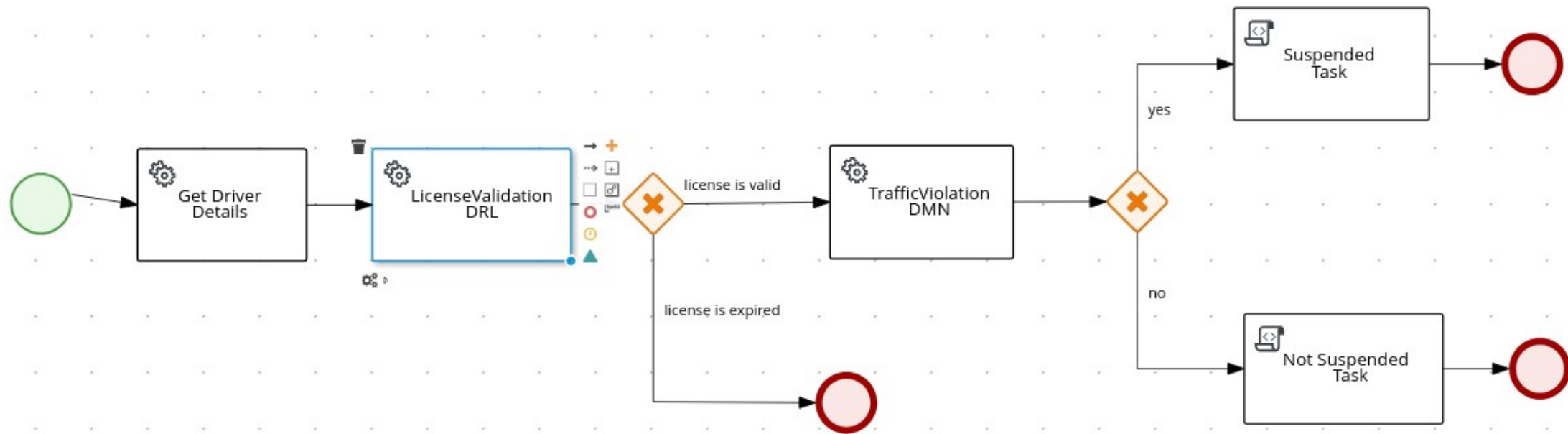




Appel du service de règle dans un nœud du processus

Lors d'un nœud de BPMN, un appel REST est effectué vers une API de règle locale ou distante, la réponse est mappée sur des variables de processus

Example





Introduction

BRMS
Présentation KIE
Projet KIE
Kogito



Maven

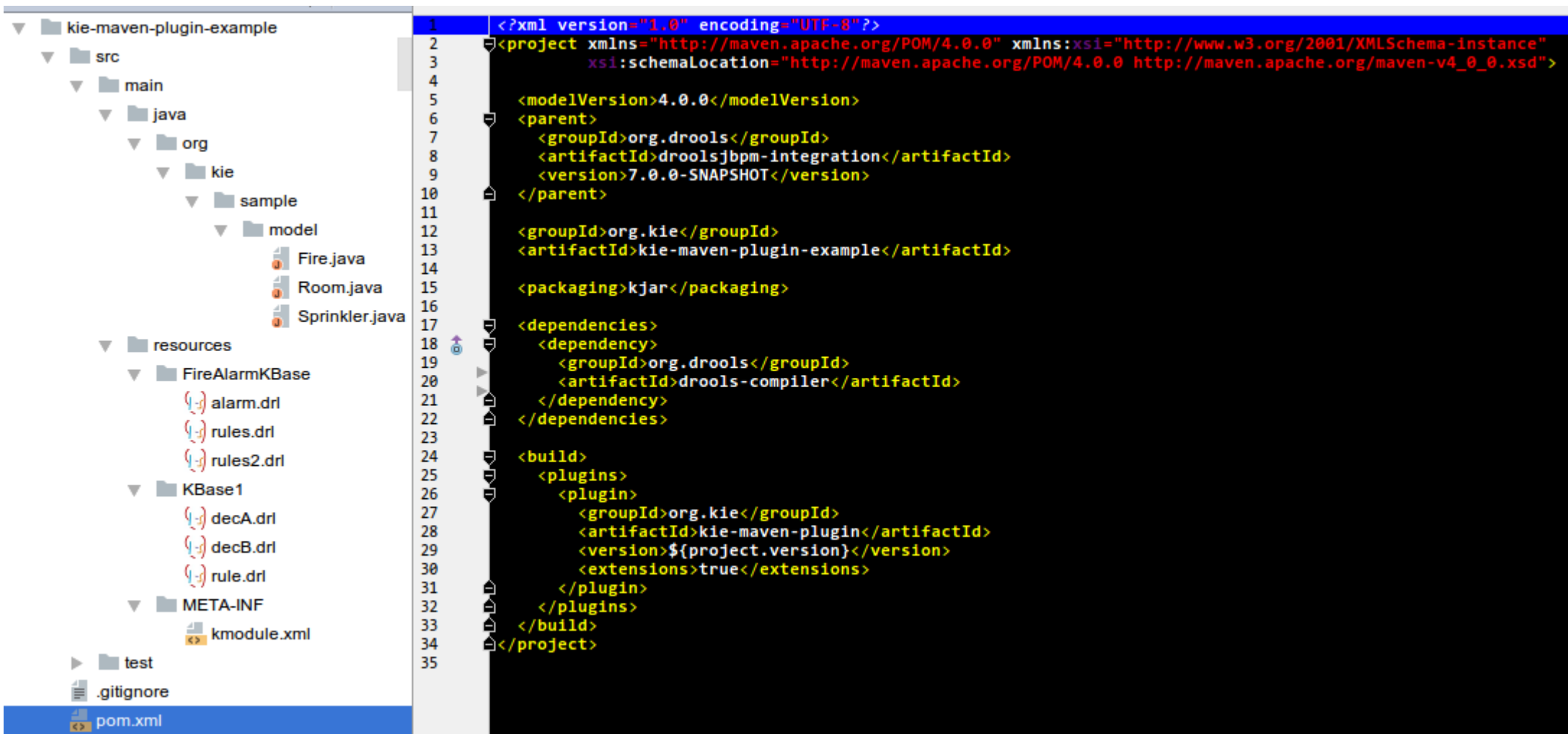
Les projets KIE sont des projets Java qui s'appuient sur Maven :

- ***src/main/java*** : Peut contenir du code Java. Modélisation des données, API Rest par exemple
- ***src/main/ressources*** : Contient les fichiers de règles et de processus
- ***src/main/resources/META-INF/kmodule.xml*** : Un fichier configurant la base de connaissances, les types de session possibles et d'autres aspects techniques
- ***src/test*** : Contient les ressources ou code Java permettant de valider les règles et processus

Kie propose également :

- Des archétypes Maven facilitant la création de projet
- Des plugins Maven validant la syntaxe des règles

Exemple : Projet DRL



The screenshot displays an IDE interface with a project structure on the left and the `pom.xml` file on the right.

Project Structure (Left Panel):

- kie-maven-plugin-example
 - src
 - main
 - java
 - org
 - kie
 - sample
 - model
 - Fire.java
 - Room.java
 - Sprinkler.java
 - resources
 - FireAlarmKBase
 - alarm.drl
 - rules.drl
 - rules2.drl
 - KBase1
 - decA.drl
 - decB.drl
 - rule.drl
 - META-INF
 - kmodule.xml
 - test
 - .gitignore
 - pom.xml

pom.xml Content (Right Panel):

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3     xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">
4
5     <modelVersion>4.0.0</modelVersion>
6     <parent>
7         <groupId>org.drools</groupId>
8         <artifactId>droolsjbpm-integration</artifactId>
9         <version>7.0.0-SNAPSHOT</version>
10    </parent>
11
12    <groupId>org.kie</groupId>
13    <artifactId>kie-maven-plugin-example</artifactId>
14
15    <packaging>kjar</packaging>
16
17    <dependencies>
18        <dependency>
19            <groupId>org.drools</groupId>
20            <artifactId>drools-compiler</artifactId>
21        </dependency>
22    </dependencies>
23
24    <build>
25        <plugins>
26            <plugin>
27                <groupId>org.kie</groupId>
28                <artifactId>kie-maven-plugin</artifactId>
29                <version>${project.version}</version>
30                <extensions>true</extensions>
31            </plugin>
32        </plugins>
33    </build>
34 </project>
35
```



kmodule.xml

Le descripteur *META-INF/kmodule.xml*

- Configure une ou plusieurs bases de connaissance en spécifiant les ressources (fichiers de règles ou processus)
- Pour chaque base de connaissance, configure une ou plusieurs types de sessions pouvant être créées.

Un descripteur vide applique une configuration par défaut :

- tous les fichiers ressources trouvés dans le *classpath* sont ajoutés à la même base de connaissance.
- 2 types de sessions (stateless et stateful) sont associées à l'unique base de connaissances



Exemple *kmodule.xml*

```
<kmodule xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://www.drools.org/xsd/kmodule">

  <kbase name="KBase1" default="true" eventProcessingMode="cloud" equalsBehavior="equality"
  declarativeAgenda="enabled" packages="org.domain.pkg1">
    <ksession name="KSession2_1" type="stateful" default="true"/>
    <ksession name="KSession2_2" type="stateless" default="false" beliefSystem="jtms"/>
  </kbase>
  <kbase name="KBase2" default="false" eventProcessingMode="stream" equalsBehavior="equality"
  declarativeAgenda="enabled" packages="org.domain.pkg2, org.domain.pkg3" includes="KBase1">
    <ksession name="KSession3_1" type="stateful" default="false" clockType="realtime">
      <fileLogger file="drools.log" threaded="true" interval="10"/>
      <workItemHandlers>
        <workItemHandler name="name" type="org.domain.WorkItemHandler"/>
      </workItemHandlers>
      <calendars>
        <calendar name="monday" type="org.domain.Monday"/>
      </calendars>
      <listeners>
        <ruleRuntimeEventListener type="org.domain.RuleRuntimeListener"/>
        <agendaEventListener type="org.domain.FirstAgendaListener"/>
        <agendaEventListener type="org.domain.SecondAgendaListener"/>
        <processEventListener type="org.domain.ProcessListener"/>
      </listeners>
    </ksession>
  </kbase>
</kmodule>
```



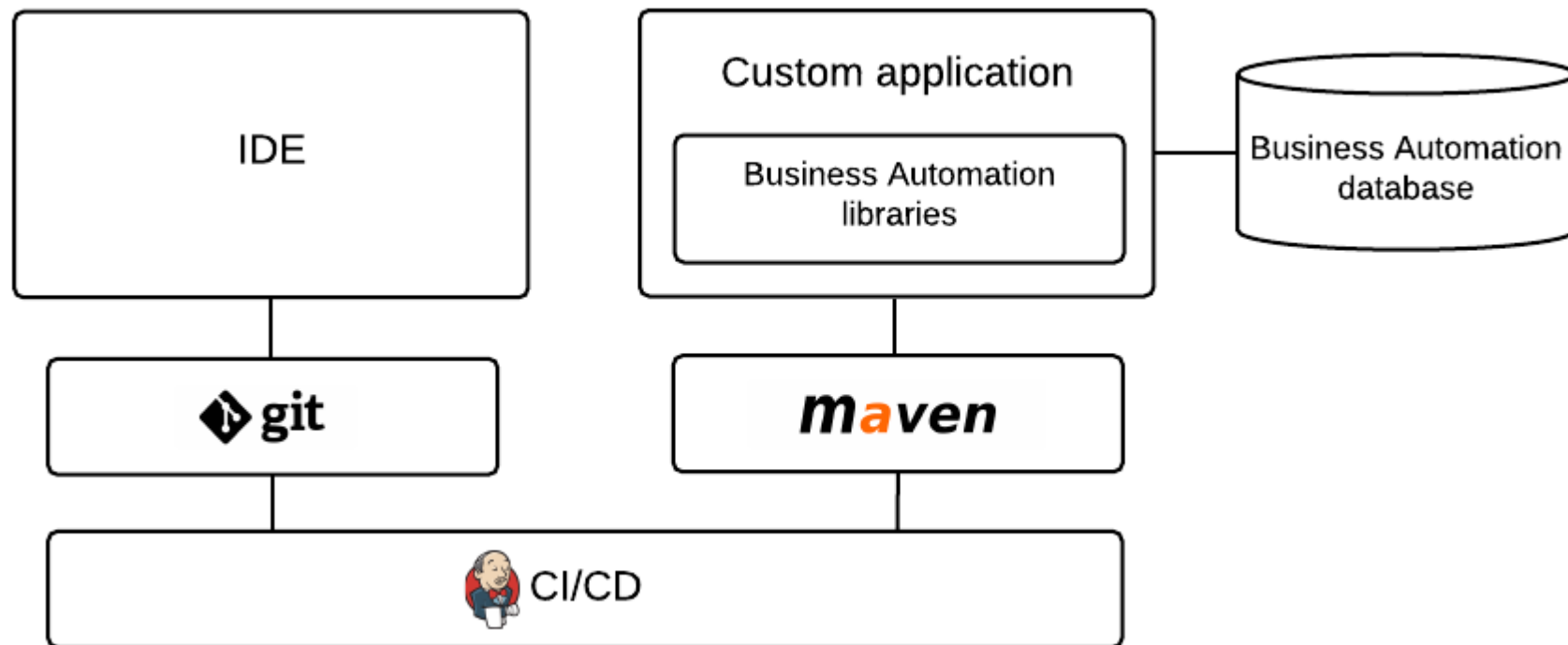
Scenarios d'intégration

2 scénarios d'intégration à une application voulant interagir avec la base de connaissance :

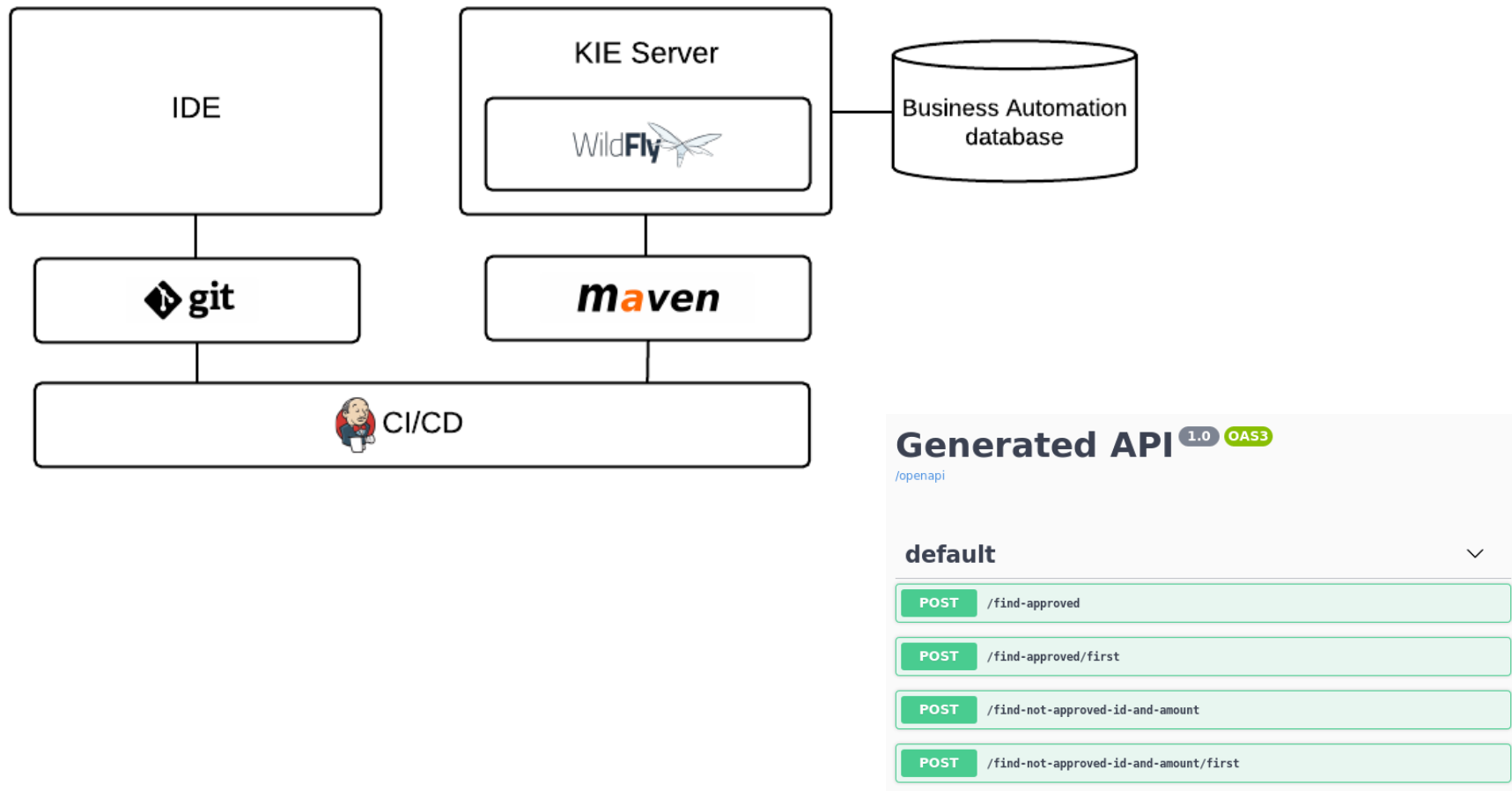
- Application Java embarquant les librairies Drools ou jBPM
- Déploiement de la base de connaissance comme service et interaction avec une API Rest



Architecture embarquée



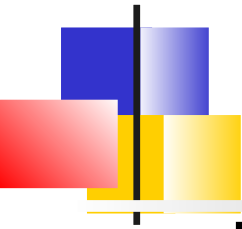
IDE + KieServer/Kogito





Introduction

BRMS
Projets KIE
Présentation KIE
Kogito



Kogito

L'objectif de Kogito est de permettre de déployer dans un cloud hybride un ensemble de processus et de décisions métier spécifiques à votre domaine. (Architecture micro-services)

Kogito inclut les composants de KIE : Drools, jBPM, and OptaPlanner

Il n'utilise pas de descripteur de déploiement *kmodule*, toutes les ressources du projet sont automatiquement chargées

Il offre des extensions à certains éditeurs pour éditer les règles et les processus



Le projet Kogito

La dernière version stable date de Septembre 2023

- Elle s'appuie sur quakus 2.x et JDK11
- La documentation en ligne est disponible pour cette version

Une version est en cours de développement 999-SNAPSHOT

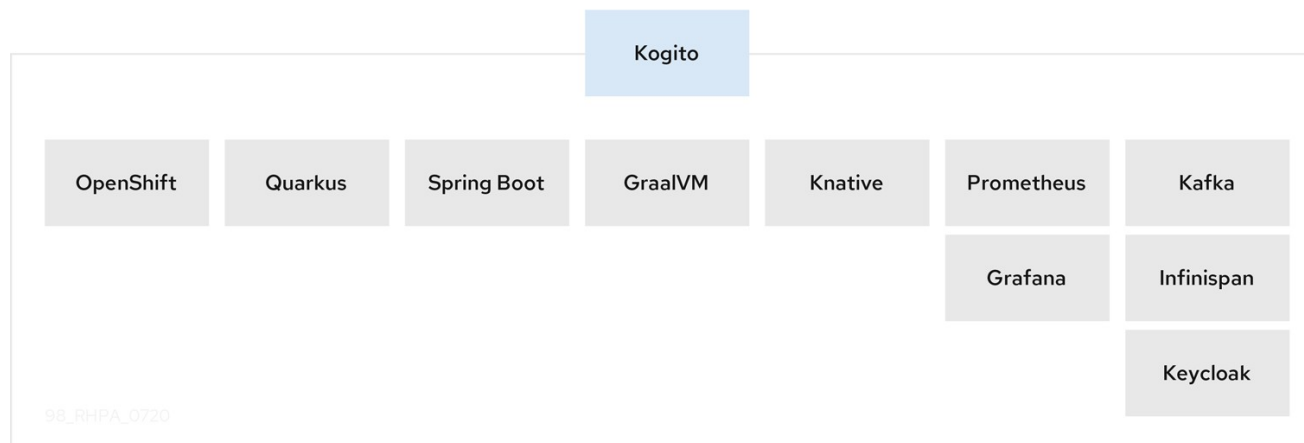
- Quarkus 3.x et JDK17
- Elle n'est pas documentée pour l'instant
- Le dépôt github est actif
- Elle propose de nombreux applications exemples



Cloud natif

Kogito est supposé de s'exécuter et de se scaler sur une infrastructure Cloud

Les dernières technologies basées sur le cloud, telles que *Quarkus*, *Knative* peuvent être utilisées pour augmenter les temps de démarrage et la scalabilité.





Flexibilité

Kogito s'adapte au domaine métier.

- => offrir des APIs REST qui manipule les les objets de votre domaine

Il s'intègre aux outils développeurs via les extensions

Il facilite les déploiements vers OpenShift avec Kogito Operator.

Quarkus ou SpringBoot : Frameworks sous-jacents aux services métier



Modeler

Kogito fournit des extensions permettant de modéliser des processus au format BPMN et des diagrammes de décision DMN :

- **VSCode (recommandé)** : Visualisation, Édition et test
- **Google Chrome** : Visualisation et édition
- **Éditeur en ligne** : Visualisation et édition
<https://sandbox.kie.org/>
- **Éditeur standalone** : technologie javascript et packages npm



Création de projet

Les projets Kogito s'appuient sur Maven.

Des archetypes sont fournis pour Quarkus
ou SpringBoot

Exemple Quarkus :

```
mvn io.quarkus:quarkus-maven-plugin:create \  
    -DprojectId=org.acme -DprojectArtifactId=sample-kogito \  
    -DprojectVersion=1.0.0-SNAPSHOT -Dextensions=kogito-quarkus
```



Fichiers ressources

Les fichiers ressources supportés par Kogito sont typiquement placés dans ***src/main/resources***

Dans le cas de DMN, ils incluent des définitions de données.

Par défaut, des endpoints REST sont automatiquement générés à partir du fichier DMN

Si l'on veut implémenter ses propres endpoints on peut désactiver la génération automatique avec la propriété :

```
kogito.generate.rest.decisions=false
```



Scénarios de Test

Kogito permet de mettre au point des scénarios de test validant les DMNs.

- On y définit alors les données d'entrée et les résultats attendus
- On utilise l'extension VSCode pour les définir

Dépendance *kogito-scenario-simulation*

Les scénarios sont placés dans le répertoire *src/test/java/testscenario* et une classe Java avec une extension Junit 5 déclenche les tests via *mvn test*



Test Drools

Avec l'API Kie et le support Drools, il est également possible des tests en Java

- Avant chaque test charger une KieBase, typiquement à partir du classpath et d'un fichier META-INF/kmodule.xml
- Extraire le modèle DMN du classpath
- Interagir avec le moteur embarqué



Exemple

```
/*
 * Récupération du modèle DMN à partir d'une KieBase, du nom de modèle et de son namespace
 */
@BeforeEach
public void setUp() {
    KieServices ks = KieServices.Factory.get();
    KieContainer kContainer = ks.getKieClasspathContainer();
    dmnRuntime = KieRuntimeFactory.of(kContainer.getKieBase()).get(DMNRuntime.class);

    dmnModel = dmnRuntime.getModel(NAMESPACE, MODEL);
    dmnContext = dmnRuntime.newContext();
}

@Test
public void testDMN(){
    InputData inputData = InputDataBuilder.....build() ;

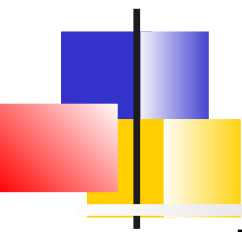
    dmnContext.set(INPUT_NODE, inputData);
    DMNResult dmnResult = dmnRuntime.evaluateAll(dmnModel, dmnContext);
    // Assertions sur dmnResult ;
}
```



Exécution du service

L'application Quarkus ou SpringBoot peut être démarré de différentes façons :

- Mode développement : Pour un test local. Permet un rechargement dynamique des processus et DMNs
- Mode Java : La cible Maven package génère un uber-jar qui peut être démarré en ligne de commande :
`java -jar target/your.jar`
- Mode Natif (Quarkus seulement) : Nécessite un GraalVM ou docker. Exécution de code natif (super rapide)



Interaction avec le service

L'interaction se fait via l'API Rest.

- Celle ci peut être générée automatiquement à partir des DMN ou développée de façon custom
- Quarkus ou SpringBoot permettent facilement de générer une interface swagger



Rest endpoints

A l'exécution, des endpoints REST sont automatiquement générés :

- **[POST] /{modelName}** : Permet d'évaluer l'intégralité du modèle DMN
- **[POST] /{modelName}/{decisionServiceName}** : Évaluation d'un composant de service de décision
- **[POST] /{modelName}/dmnresult** : Évalue l'intégralité du modèle et renvoi une réponse DMNResult encapsulant le contexte du domaine métier, les messages d'aide et les pointeurs de décision d'aide pour le modèle DMN
- **[POST] /{modelName}/{decisionServiceName}/dmnresult** : Idem pour un service
- **[GET] /{modelName}** : Renvoie le XML DMN sans logique de décision, généralement pour l'inspection du modèle DMN



Support pour OpenShift

Les services Kogito peuvent être déployés sur un Cloud ; du support pour OpenShift est fourni.

- **Kogitor Operator** : Permet d'automatiser les déploiements
Par exemple adopter du GitOps
- **Kogito CLI** : Permet d'interagir avec l'opérateur et de déployer des services à partir des sources ou des binaires
- **Probes** : Des endpoint de probes permet à OpenShift de surveiller le service



Kogito add-ons

Kogito propose de nombreux add-ons permettant d'intégrer d'autres outils. Citons quelques add-ons relatif à DMN :

- quarkus-messaging, quarkus-events-smallrye : Intégration à un système de messagerie tel que Kafka
- quarkus-events-decisions : Décisions par rapport à des évènements plutôt que des faits
- quarkus-knative-eventing : Application serverless connecté à une messagerie
- quarkus-monitoring-* : Métriques pourssés vers Prometheus, Elastic Search, ..
- quarkus-persistence-* : Support de Persistance (jdbc, infinispan, ...)



Console d'audit

La ***Kogito Audit Investigation Console*** est une interface utilisateur permettant de surveiller et d'enquêter sur les exécutions de modèles.

Elle nécessite 3 composants supplémentaires :

- Kogito Trusty Service : Permet à la console d'accéder aux événements liés au suivi des décisions stockés dans un support de persistance : Infinispan ou Apache Kafka
- Kogito Explainability Service : Génère des données stockées dans le service Kogito Trusty. Le service d'explicabilité Kogito nécessite la messagerie Apache Kafka.
- Kogito Tracing decision add-on : Permet au service Kogito de générer des événements et de les publier sur Apache Kafka. Il expose également le endpoint REST /predict pour le service Kogito Explainability.



Console jBPM

2 consoles pour jBPM sont également fournies :

- ***Kogito Management Console*** :
Permet de voir les services Kogito et de gérer les instances de processus
- ***Kogito Task Console*** : Permet de voir et interagir avec les tâches utilisateur d'un processus.



DMN

Démarrage
Compléments



Introduction

Decision Model Notation est un standard de l'OMG (comme BPMN2)

Support par Drools depuis la version 7.x

L'objectif est de fournir une notation standard compréhensible par :

- ***Experts métiers*** : Définir les spéc. Initiales gérer et surveiller les décisions
- ***Développeurs*** : Créer des diagrammes de décisions complexes et automatiser les décisions;

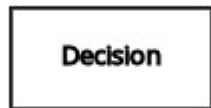


Support DMN dans Kogito

DMN version	DMN engine support	DMN modeler support	
	Execution	Open	Save
DMN 1.1	✓	✓	✗
DMN 1.2	✓	✓	✓
DMN 1.3	✓	✓	✗
DMN 1.4	✓	✗	✗

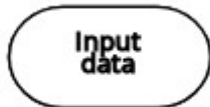


Éléments d'un diagramme



Decisions détermine une valeur de sortie en fonction de :

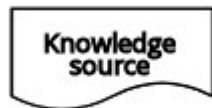
- Ses données d'entrée (Nœuds d'entrées ou valeur de sortie d'une autre décision)
- D'une logic de décisions exprimée via des expressions de différent types pouvant appelées des fonctions du BKM



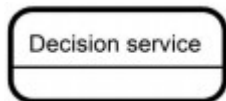
Input data : Information utilisé dans les nœuds de décisions. Peuvent être inclus dans un BKM, ils deviennent alors les paramètres du BKM.



BKMs encapsule la connaissance métier dans des fonctions réutilisables



Knowledge source désigne une autorité qui régule un BKM ou un nœud de décision.



Service de décision : Décision de niveau supérieur contenant un ensemble de décisions réutilisables publiées en tant que service pour invocation



Annotation : Note explicative associé à un nœud d'entrée



Connecteurs

Les connecteurs utilisés pour relier les différents éléments doivent également respecter la forme standard



Information

Connexion d'un nœud de données d'entrée ou d'un nœud de décision à un autre nœud de décision qui nécessite les informations



Knowledge

d'un BKM à un nœud de décision ou à un autre modèle de connaissances métier qui invoque la logique de décision.



Authority

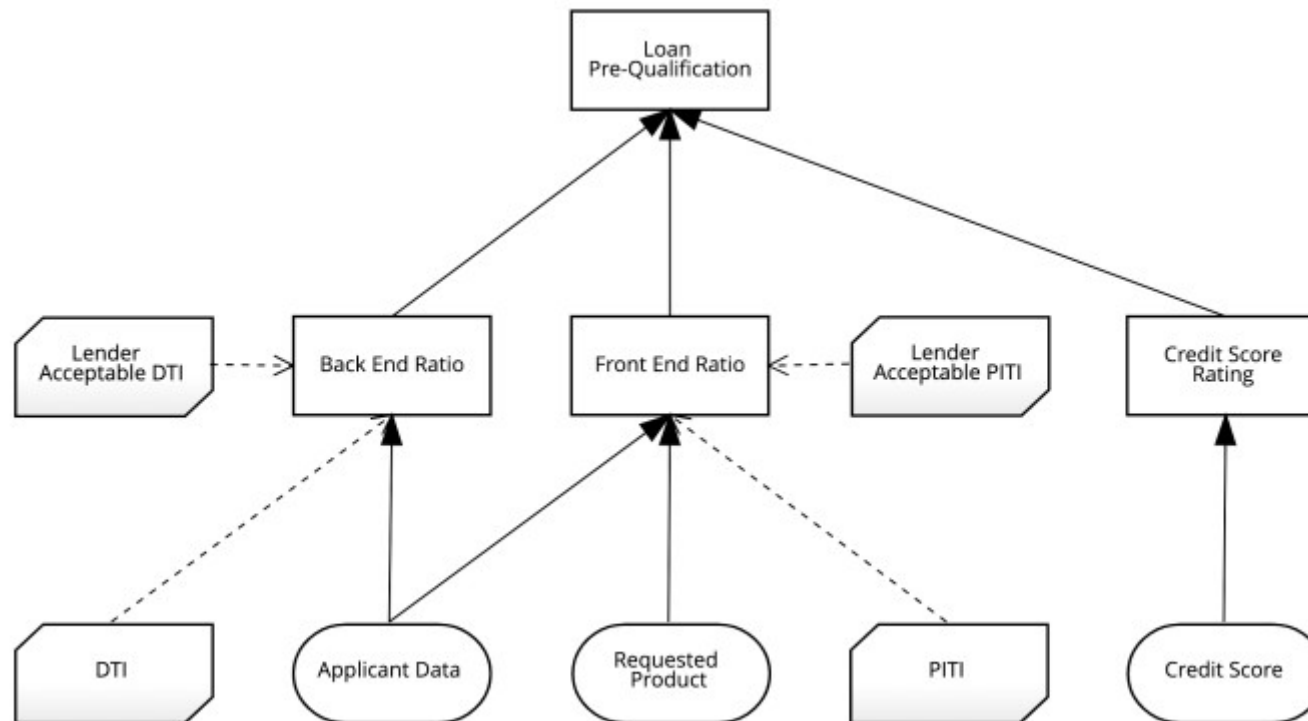
d'un nœud de données d'entrée ou d'un nœud de décision vers une source de connaissances dépendante ou d'une source de connaissances vers un nœud de décision, un modèle de connaissances métier ou une autre source de connaissances.



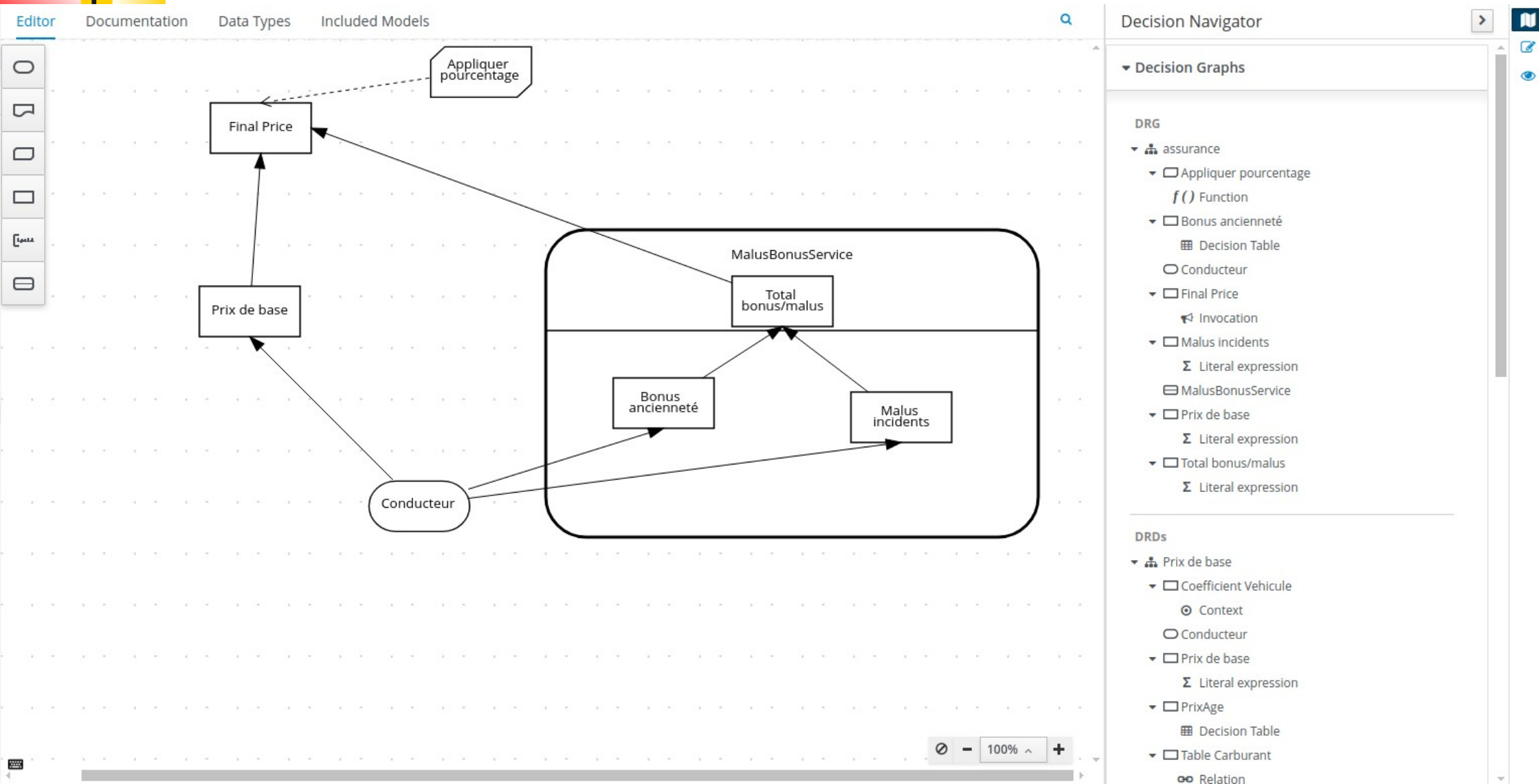
Association

Association d'un nœud à une annotation

Exemple



Le modeler VSCode





Nœud décision

Les nœuds décisions peuvent exprimer leur logique par une variété d'expressions encadrées. Toutes les expressions encadrées utilisent la syntaxe **FEEL**

- **Literal Expression** : expression pure FEEL qui produit une valeur de sortie
- **Contexts** représente une collection d'une ou plusieurs paires clé-valeur où la valeur est une expression FEEL et la clé est l'identifiant
- **Table de décision** : Table contenant des colonnes d'entrée, des colonnes de sortie et une stratégie de Hit
- **Relations** : Table de données donnant des informations sur les données utilisées dans le DMN
- **Functions** : Définit des opérations réutilisables généralement associé à un BKM
- **Invocation** : Permet l'invocation de nœud du BKM
- **List** : Une liste d'expression FEEL.



FEEL

FEEL (Friendly Enough Expression Language)

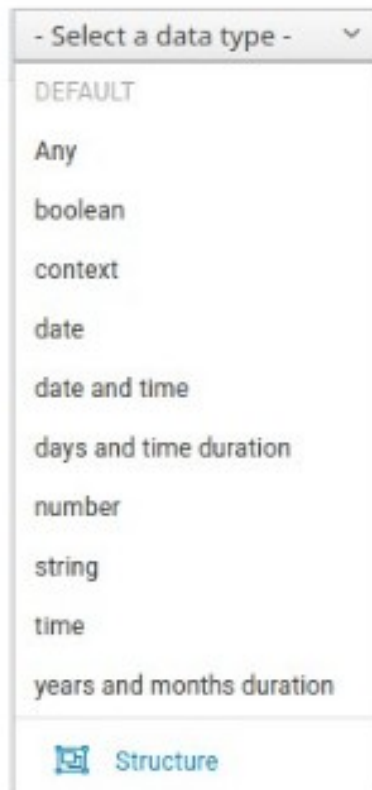
est destiné à servir de terrain d'entente entre les analystes commerciaux, les programmeurs, les experts du domaine et les parties prenantes.

Il fournit :

- Sans effet de bord
- Modèle de données simple avec nombres, dates, chaînes, listes et contextes(~Map)
- Syntaxe simple conçue pour un large public
- Logique à trois valeurs (vrai, faux, nul)



Types de données



Des types structurés
peuvent être définis à
partir des types de
base : ***Structure***

Le type de données
peut également être
une liste



Contraintes sur les données

Des contraintes peuvent être ajoutés sur un champ simple, structuré, ou un champ embarqué dans une structure.

3 types de contraintes peuvent être définies :

- **Enumération** : Liste de valeurs valides
- **Range** : Intervalle de valeur
- **Expression** : Expression FEEL renvoyant un booléen



Noms de variable et de fonctions

FEEL prend en charge les espaces et quelques caractères spéciaux dans les noms de variables et de fonctions.

Un nom *FEEL* doit commencer par une lettre, ? ou _.

Noms de variables valides :

- *Age*
- *Birth Date*
- *Flight 234 pre-check procedure*



Collections

FEEL propose plusieurs types collections :

- Les Listes utilisent [] dont les éléments sont accessibles par un indice
Ex : [1, 2, 3, 4, 5]
- Les Range définissant un intervalle
[1..10) // 1 inclus, 10 exclus.
- Les contexte sont similaire à des Map ou des structures JSON. Les valeurs sont accessibles par des clés
{"key1":1, "key2":100}
Ou
{
 Name: "John",
 Address: {
 City: "New York"
 }}
}}



Structure de contrôle

Dans la structure **if**, la clause else est obligatoire
`if 20 > 0 then "YES" else "NO" //→ "YES"`

for permet de fournir de nouvelles valeurs à partir d'une itération
`for i in [1, 2, 3] return i * i //→ [1, 4, 9]`

Les opérateurs **some** et **any** permettent de vérifier des conditions sur des listes
`some i in [1, 2, 3] satisfies i > 2 //→ true`
`every i in [1, 2, 3] satisfies i > 1 //→ false`

in teste si une valeur est dans un intervalle
`10 in [1..10) //→ false`

Logique à 3 valeurs (and, or)

`true and true //→ true`
`true and false and null //→ false`
`true and null and true //→ null`
`true or false or null //→ true`



Fonction String (1)

Concaténation : **+**

"some" + "string" = "somestring"

Conversion : ***string()***

string(1.1) = "1.1"

Sous-string : ***substring(string, start position, length?)***

substring("testing", 3, 3) = "sti"

Sous-string avec regexp

substring before("testing", "ing") = "test"

Longueur : ***length()***

string length("tes") = 3

Upper Case, Lower Case : ***upper case(), lower case()***

lower case("aBc4") = "abc4"

Remplacement : ***replace()***

replace("banana", "a", "o") = "bonono"



Fonction String (2)

Test si contient : ***contains()***

```
contains( "testing", "to" ) = false
```

Démarre ou finit par : ***starts with(), ends with()***

```
starts with( "testing", "te" ) = true
```

Vérification de gabarit via regexp : ***matches()***

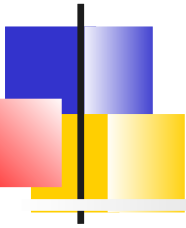
```
matches( "teeesting", "^te*sting" ) = true
```

Division en liste : ***split()***

```
split( "John Doe", "\\s" ) = ["John", "Doe"]
```

Regroupement à partir d'une liste : ***string join()***

```
string join( ["a", "b", "c"], "_and_")  
= "a_and_b_and_c"
```

Fonction sur les numériques

Arrondis : ***decimal(n, scale), floor(n), ceiling(n), round ...***

round down(1.121, 2) = 1.12

Math : ***abs, modulo, sqrt, log, exp***

decimal(exp(5), 2) = 148.41

Test partité : ***odd, even***

odd(5) = true



Fonctions sur les listes

Fonctions d'agrégation : ***count, min, max, sum, mean, stddev, mode, product***

```
min( [1,2,3] ) = 1
```

Manipulation : ***sublist, insert before, append, concatenate, remove, reverse, replace, distinct values, flatten, union***

```
append( [1], 2, 3 ) = [1,2,3]
```

```
flatten( [[1,2],[[3]], 4] ) = [1,2,3,4]
```

Test du contenu : ***contains, index of***

```
index of( [1,2,3,2], 2 ) = [2,4]
```

Tri : ***sort()***

```
sort( list: [3,1,4,5,2], precedes: function(x,y) x  
< y ) = [1,2,3,4,5]
```



Fonction sur les intervalles

Test si avant/après : ***before(), after()***

`before([1..10], 15) = true`

`after([11..20], [1..10]) = true`

Coïncide : ***meets(), met by, coincides()***

`meets([1..5], [5..10]) = true`

`met by([5..10], [1..5]) = true`

`coincides([1..5], [1..5]) = true`

Intersection : ***overlaps(), overlaps before(), overlaps after()***

`overlaps([1..5], [3..8]) = true`

`overlaps before([1..5], [3..8]) = true`

`overlaps after([1..5], [3..8]) = false`

Tests des bornes : ***finishes(), finished by(), starts(), started by()***

`finishes([5..10), [1..10)) = true`

`started by([1..10], 1) = true`

Inclusion : ***includes(), during()***

`includes([1..10], [4..6]) = true`

`during(5, [1..10]) = true`



Fonctions sur les contextes

Lecture : ***get value(), get entries()***

```
get value( {key1 : "value1"}, "key1" ) = "value1"
get entries( {key1 : "value1", key2 : "value2"} )
  = [ { key : "key1", value : "value1" }, {key : "key2", value
    : "value2"} ]
```

Ecriture : ***context put***

```
context put({x:1, y:0}, "y", 2) = {x:1, y:2}
context put({x:1, y: {a: 0} }, ["y", "a"], 2)
  = {x:1, y: {a: 2} }
```

Création à partir d'une liste clé/valeur :

```
context([ {key:"a", value:1}, {key:"b", value:2} ]) = {a:1, b:2}
```

Fusion :

```
context merge([ {x:1}, {y:2} ]) = {x:1, y:2}
```



Propriétés des types temporel

Les types *date and time*, *date*, *time*, *days and time duration*, *years and month duration* et *range* supporte les attributs suivants :

- **year**, **month** [1..12], **day** [1 ..31], **hour** [0..23], **minute** [0..59], **second** [0..60)
- **weekday** : jour de la semaine, [1..7]
- **time offset** : le décalage de durée correspondant au fuseau horaire
- **timezone** : fuseau horaire
- **years**, **months** : le composant années/mois d'un type *years and month duration*
- **days**, **hours**, **minutes**, **seconds** : les composant jours, heures, minutes secondes d'un type *days and time duration*



Fonctions temporelles

Date ou date time du jour

`now()`

`today()`

Jour de l'année, de la semaine

`day of year(date(2019, 9, 17)) = 260`

`day of week(date(2019, 9, 17)) = "Tuesday"`

Mois de l'année

`month of year(date(2019, 9, 17)) = "September"`

Semaine de l'année

`week of year(date(2019, 9, 17)) = 38`



Expressions encadrées

Les nœuds décisions peuvent utiliser plusieurs formes d'expression encadrée pour définir leur valeur de sortie.

Les plus utilisées sont :

- Literal Expression : Expression FEEL à partir des données du Contexte
- Table de décision : Expression FEEL en fonction de condition sur les données du contexte



Expression littérale

Expression littérale FEEL sous forme de texte dans une cellule de tableau

Lender Acceptable PITI (*Literal expression*)

Lender Acceptable PITI (<i>number</i>)
<code>decimal(acceptable rate, 2)</code>

Table de décision

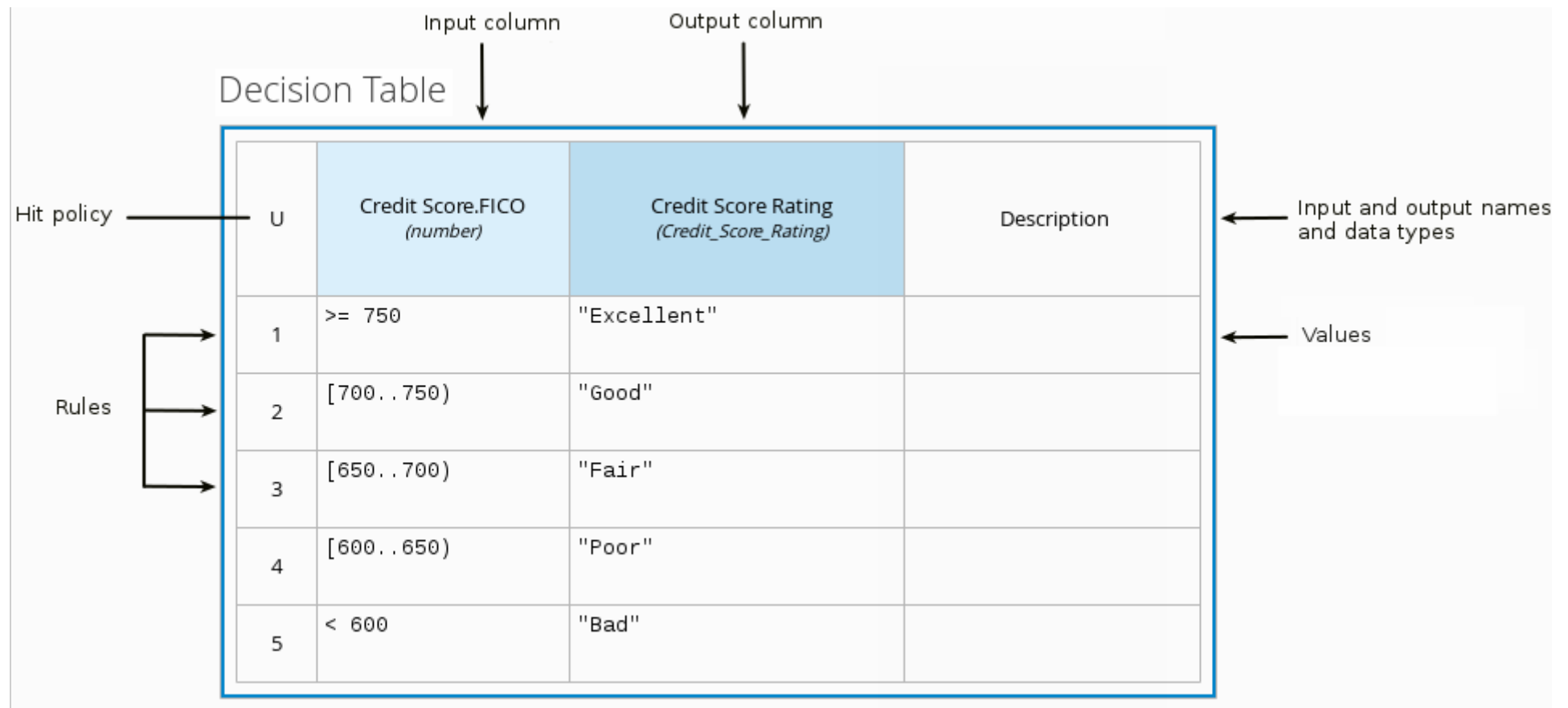




Table de décision

Chaque règle se compose d'une seule ligne dans le tableau et comprend des colonnes qui définissent les conditions (entrée) et le résultat (sortie) de cette ligne particulière.

Les valeurs d'entrée et de sortie peuvent être des expressions FEEL ou des valeurs de type de données définies.



Hit policies

Les politiques de hit déterminent ce qu'il faut faire lorsque plusieurs règles de la table de décision correspondent.

Plusieurs valeurs possibles :

- **Unique (U)**: Autorise une seule correspondance de règle. 2 correspondances ou + génèrent une erreur.
- **Any (A)**: Permet à plusieurs règles de correspondre, mais elles doivent toutes avoir la même sortie.
- **Priority (P)**: Permet de faire correspondre plusieurs règles avec des sorties différentes. Seule la règle avec la priorité la plus élevée est sélectionnée.
- **First (F)**: Utilise la première correspondance dans l'ordre des règles.
- **Collect** : Agrège les résultats de plusieurs règles en fonction d'une fonction d'agrégation.



DMN

Démarrage
Compléments



Context expression

Un ensemble de noms et de valeurs de variables avec une valeur de résultat

Prioritized Waiting List (*Context*)

#	Prioritized Waiting List (<i>tPassengerTable</i>)	
1	Cancelled Flights (<i>tFlightNumberList</i>)	Flight List[Status = "cancelled"].Flight Number
2	Waiting List (<i>tPassengerTable</i>)	Passenger List[list contains(Cancelled Flights, Flight Number)]
	<result>	sort(Waiting List, Passenger Priority)



Liste

Représente une liste d'éléments FEEL.

Utilisé pour définir des listes d'éléments pertinents pour un nœud particulier dans une décision.

Approved credit score agencies (*List*)

1	"Acme Agency, Inc."
2	"Top Scores, Inc."
3	"Global Scoring, Inc."



Relation expression

Tableau de données traditionnel avec des informations sur des entités données, répertoriées sous forme de lignes.

Utilisé pour définir des données de décision.

Employee Information (*Relation*)

#	Name (string)	Dept (string)	Salary (number)
1	"John"	"Sales"	100000
2	"Mary"	"Finances"	120000



Fonctions

Les fonctions peuvent être utilisées dans les expressions encadrées d'un nœud décision ou encapsulées dans un nœud BKM

Les fonctions prennent des paramètres d'entrée et produisent un résultat en sortie.

Les fonctions peuvent être exprimées en :

- **FEEL**. On peut alors utiliser toutes les expressions FEEL (Littérale, Table de décision, Context, ...)
- **Java**. Appel d'une méthode d'une classe accessible
- **PMML** (*Predictive Model Markup Language*) : Modèle de Machine Learning ou de statistiques. Sélection d'un document PMML puis du modèle

Édition d'une fonction

Evaluate Match (Function)

F	Evaluate Match (tCandidate)		
	(Lonely Soul, Candidate)		
	1	Profile1 (tProfile)	Lonely Soul
	2	Profile2 (tProfile)	Candidate
	3	Is Match (boolean)	Is Soul a Match(Lonely Soul, Candidate) Is Soul a Match(Candidate, Lonely Soul)
	4	Score (number)	Number of Matching Interests(Lonely Soul, Candidate) - absolute(Lonely Soul.Age - Candidate.Age)
		<result>	Select expression

Edit Parameters

Add parameter

Lonely Soul

tProfile ▾

Delete

Candidate

tProfile ▾

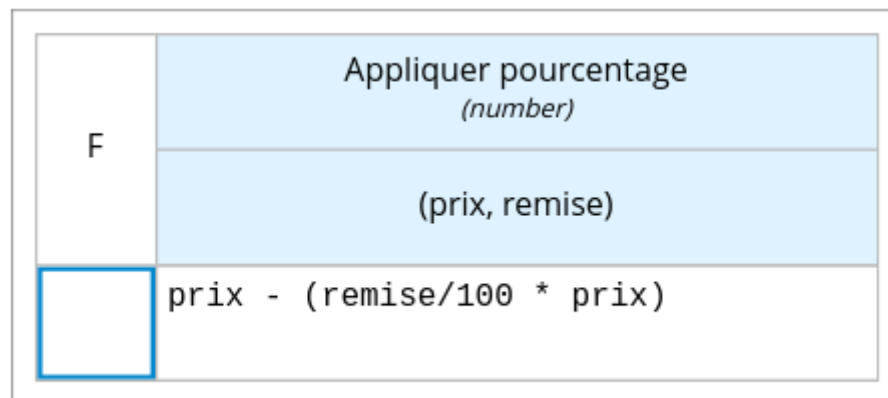
Delete



Nœuds Business Knowledge Model

Ce type de nœud permet de mettre à disposition une fonctions réutilisable dans le diagramme.

Les autres nœuds décisions peuvent alors invoquer la fonction





Invocation d'une fonction

Appelle un modèle de connaissances métier. Il contient :

- le nom des connaissances métier
- une liste de liaisons de paramètres
 - Nom du paramètre
 - Expression de liaison (valeur)

Rebooked Passengers (*Invocation*)

#	Rebooked Passengers (tPassengerTable)	
	Reassign Next Passenger	
1	Waiting List (tPassengerTable)	Prioritized Waiting List
2	Reassigned Passengers List (tPassengerTable)	[]
3	Flights (tFlightTable)	Flight List



Nœuds de Service de décision

Dans un nœud de service de décision,

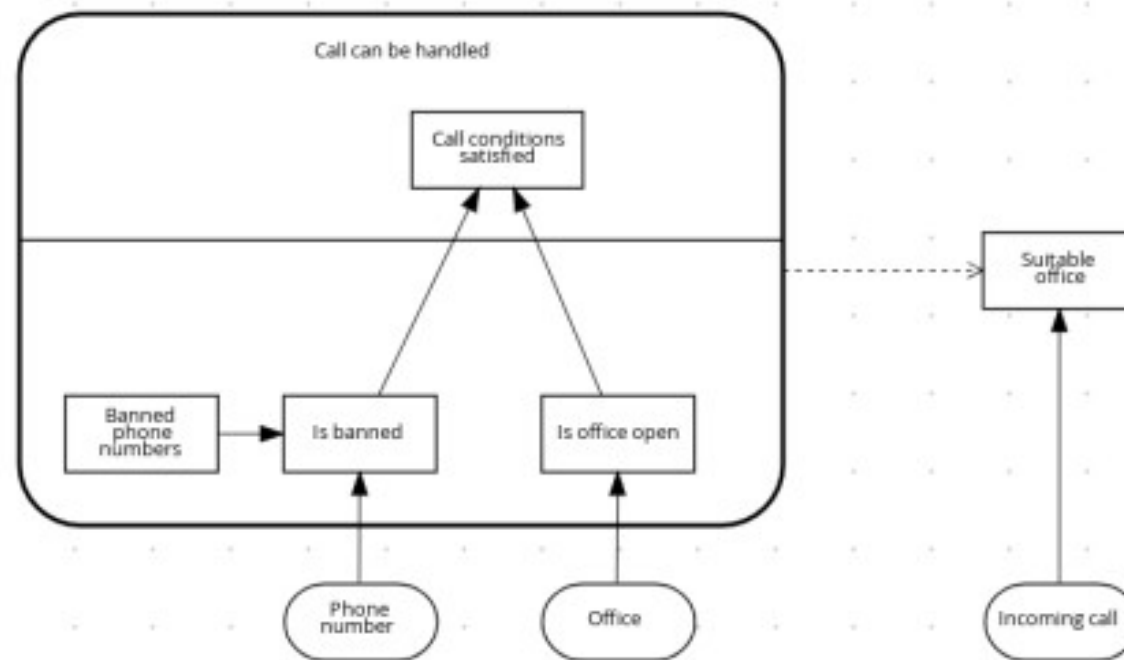
- les nœuds de décision du segment inférieur intègrent des données d'entrée provenant de l'extérieur du service
- La décision finale est reversé dans un nœud de décision du diagramme principale.

La décision finale est donc utilisable dans le diagramme principale.

Les services de décision DMN peuvent être réutilisés dans le diagramme pour appliquer la même logique de décision avec des données d'entrée différentes et des connexions sortantes différentes.

Le service de décision ajoute des endpoints dans l'API Rest permettant de l'invoquer indépendamment de l'API globale

Décision service





Inclusion de modèle

Il est possible d'inclure d'autres modèles DMN ou PMML dans un diagramme.

- Dans le cas d'un DMN, il est alors possible d'utiliser tous les nœuds et la logique des modèles importés
- Dans le cas de PMML, le modèle peut être appelée par une fonction d'un nœud décision ou d'un BK

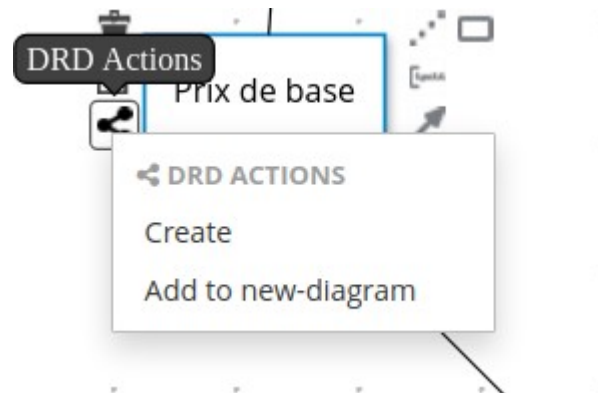
Les modèles doivent cependant être dans les mêmes répertoires.

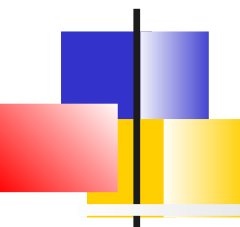
Le rafraîchissement automatique ne fonctionne pas dans le modèleur DMN.



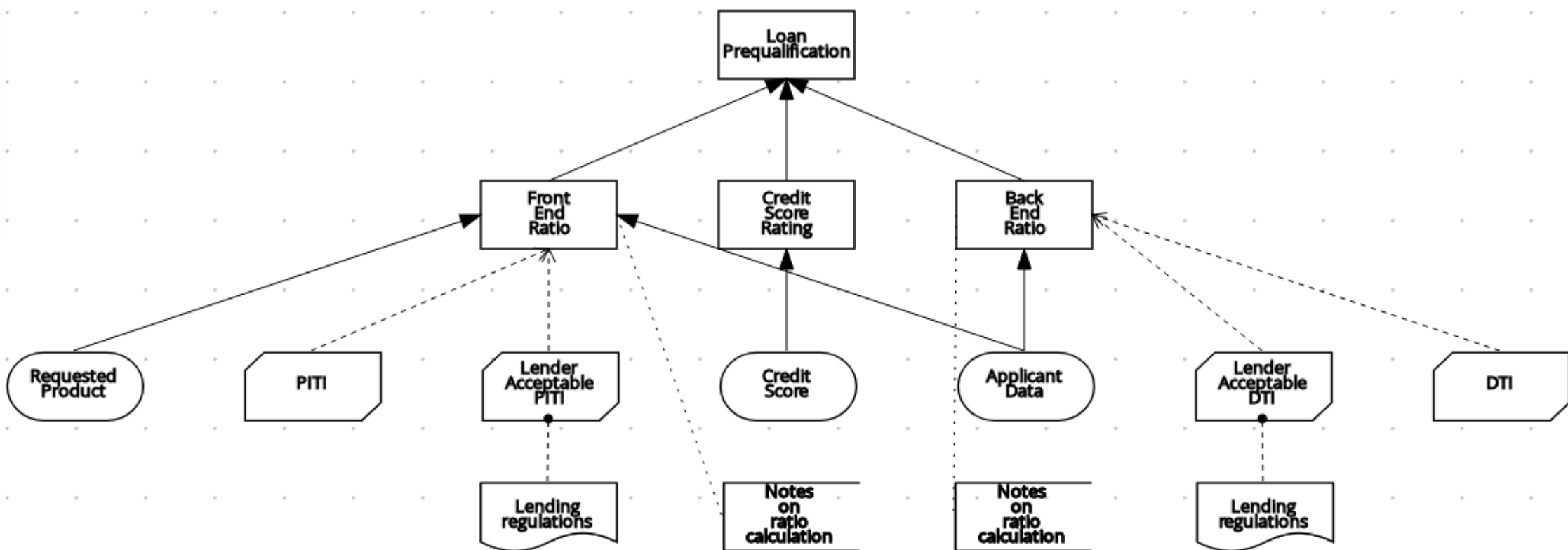
Modèle embarqué

Lorsque le modèle DMN devient complexe, il peut être intéressant de mettre en place des diagrammes d'exigences de décision DMN (DRD) qui représentent des parties du graphique global.





Avant





Après

