Cahier de TP
« DMN Kogito»

Pré-requis :

- Bonne connexion Internet
- JDK11
- Git
- VsCode avec extensions DMN Editor de RedHat ; IntellijIDEA
- Maven 3.6+
- Docker

Github solutions
https://github.com/dthibau/dmnkogito-solutions

Table des matières

Atelier 1 : Application Exemple

Les applications exemples proviennent de https://github.com/kiegroup/kogito-examples.git

Nous utilisons la version 1.44.1.Final

Se positionner dans le répertoire ***kogito-examples*** et effectuer :
```
git checkout 1.44.1.Final
```

1.1 Exemple simple DMN-quarkus

Ouvrir avec VSCode le projet
kogito-examples/kogito-quarkus-examples/dmn-quarkus-example

Visualiser les sources du projet en particulier le fichier DMN, le fichier scesim et les tests Java.

Commencer par exécuter les tests :
```
mvn test
```
Combien de tests ?

Exécuter ensuite l'application avec la commande :
```
mvn clean compile quarkus:dev
```

Accéder à l'interface Swagger :
http://localhost:8080/q/swagger-ui/

Utiliser le endpoint POST et poster les données suivantes :
```
{
    "Driver":{"Points":2},
    "Violation":{
        "Type":"speed",
        "Actual Speed":120,
        "Speed Limit":100
    }
}
```

Visualiser la réponse, en particulier le bloc fine et le bloc "Should the driver be suspended?"

Vous pouvez également essayer les autres packaging :

- Uber jar
  ```
  mvn clean package -DskipTests
  java -jar target/quarkus-app/quarkus-run.jar
  ```
- Si vos avez le courage construire un exécutable natif
  ```
  mvn package -Pnative -DskipTests -Dquarkus.native.container-build=true
  ./target/dmn-quarkus-example-runner
  ```

1.2 Example BPMN, Business Rule Task, DMN et DRL

Ouvrir avec VSCode (ou IntelliJIDEA) le projet
kogito-examples/kogito-quarkus-examples/process-decisions-rest-quarkus

Visualiser les sources de projet en particulier dans le répertoire ***src/main/resources :***

- Le processus BPMN : ***traffic-rules-dmn-service-tasks***
- Le diagramme de décision DMN : ***TrafficViolation.dmn***
- Les règles en DRL : ***LicenceValidationService.drl***

Démarrer l'application, accéder à l'interface Swagger. Utiliser le endpoint POST /traffic_service avec :

```
{
    "driverId": "12345",
    "violation":{
        "Type":"speed",
        "Speed Limit": 100,
        "Actual Speed":140
    }
}
```

Observer la réponse et regarder la console et les traces d'exécution du processus.

En utilisant le même endpoint poster les données suivantes :

```
{
  "driverId": "1234",
  "violation":{
    "Type":"speed",
    "Speed Limit": 100,
    "Actual Speed":110
  }
}
```

Atelier 2 : Nœuds décisions DMN

La problématique consiste à mettre au point un DMN permettant de calculer le prix d'une assurance auto.

Le prix est composé d'un prix de base, auquel on peut appliquer une remise en fonction de l'ancienneté du client et un malus en fonction des incidents qu'il a eu.

Le prix de base est calculé comme suit :

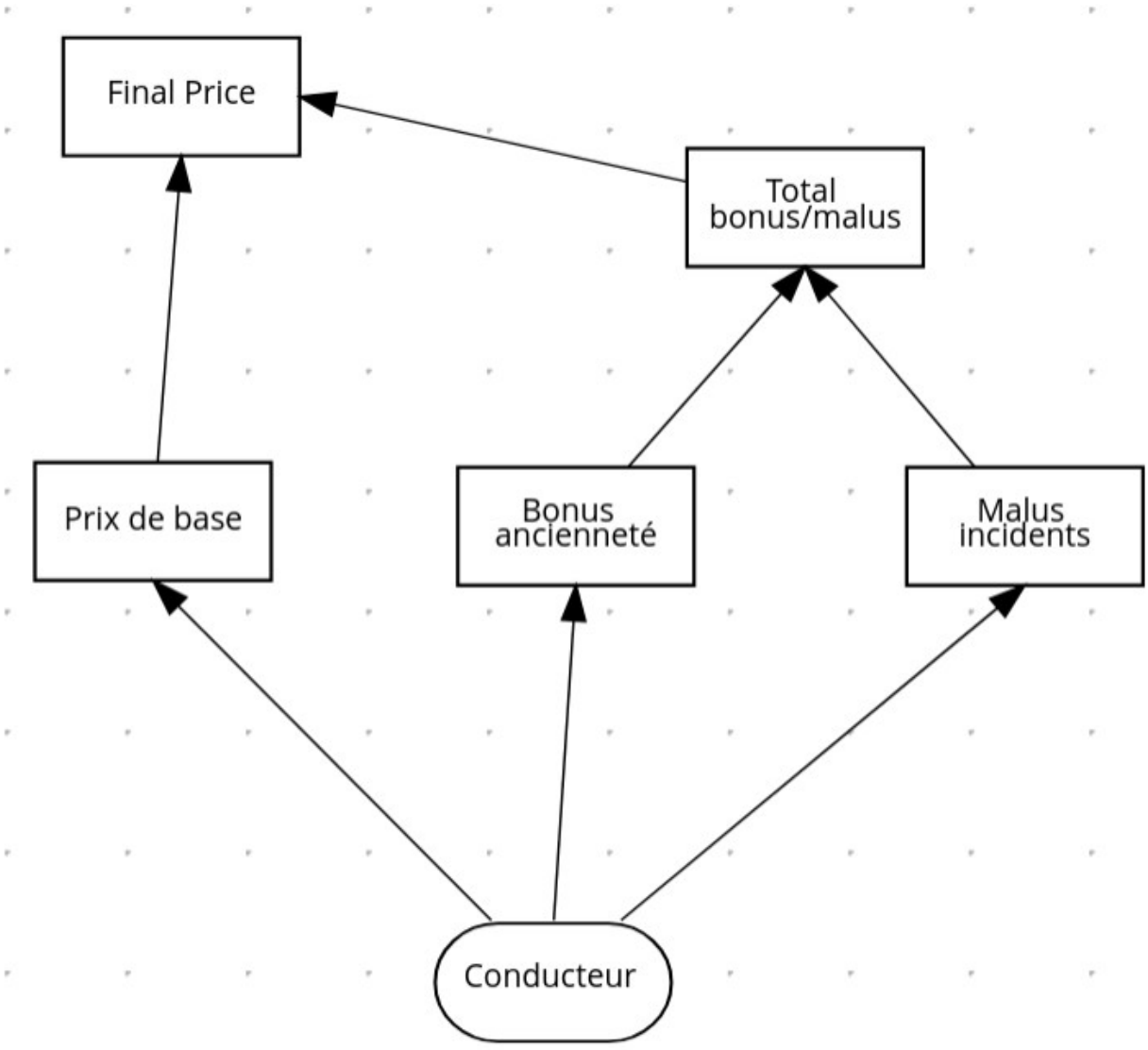
- Si le conducteur est un jeune conducteur (entre 18 et 25 ans), le prix de base est de 500€
- Si le conducteur est un conducteur confirmé (> 25ans), le prix de base est de 300€

Une remise est effectuée en fonction de l'ancienneté du client :

- Si le client a 5 ans d'ancienneté, on lui accorde une remise de 10%
- Si le client a 10 ans d'ancienneté, on lui accorde une remise de 20%

Enfin, un malus de 5% est appliqué par nombre d'incidents qu'a eu le conducteur.

Le DMN final ressemblera à ceci :

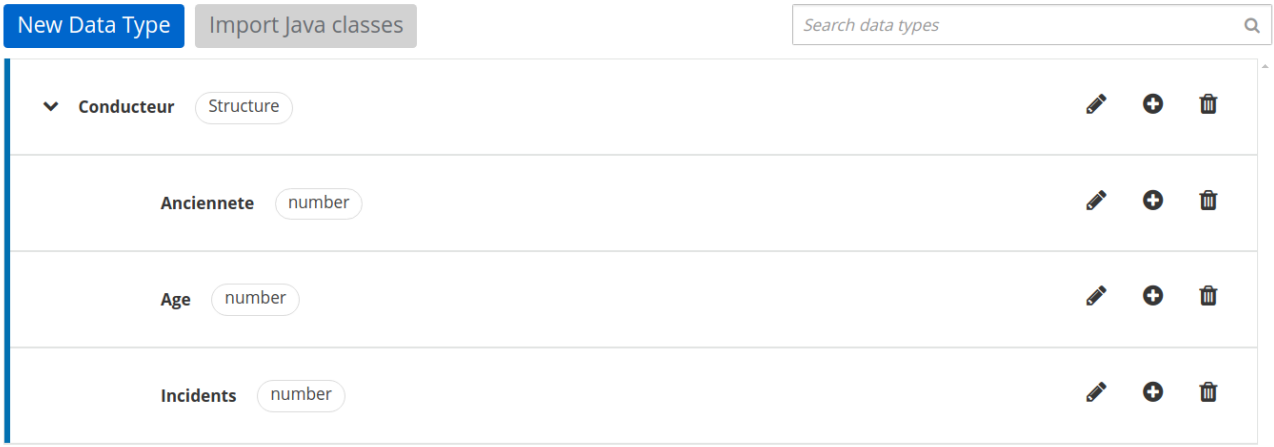


2.1 Implémentation

Ouvrir le projet fourni et créer un fichier ***src/main/resources/assurance.dmn***
Ouvrir le fichier avec l'éditeur DMN

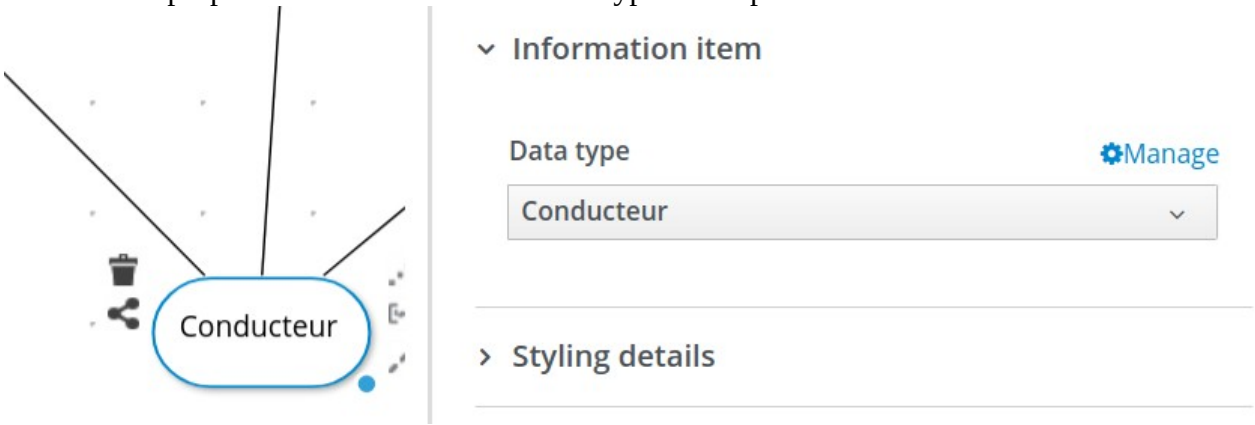
Commencer par créer un nouveau Data Type de type structure nommé ***Conducteur*** contenant les champs suivants :

- **Age**
- **Incidents**
- **Anciennete**



Revenir sur l'onglet Editor :

Insérer un nœud de type *Input*
Le nommer Conducteur
Afficher ses propriétés et sélectionner le Data Type défini précédemment



Créer ensuite un nœud Décision
Le nommé Prix de base
L'éditer et choisir une Table de décision comme expression :

Prix de base *(Decision Table)*

U	Conducteur.Age *(number)*	Prix de base *(number)*	annotation-1
1	[18..25]	500.0	25 compris
2	> 25	300.0	

Compléter de la même façon les autres nœuds de décisions.
Vous avez encore besoin de 3 nœuds décisions dont 2 utilisent de simple expressions FEEL et l'autre une table de décision.

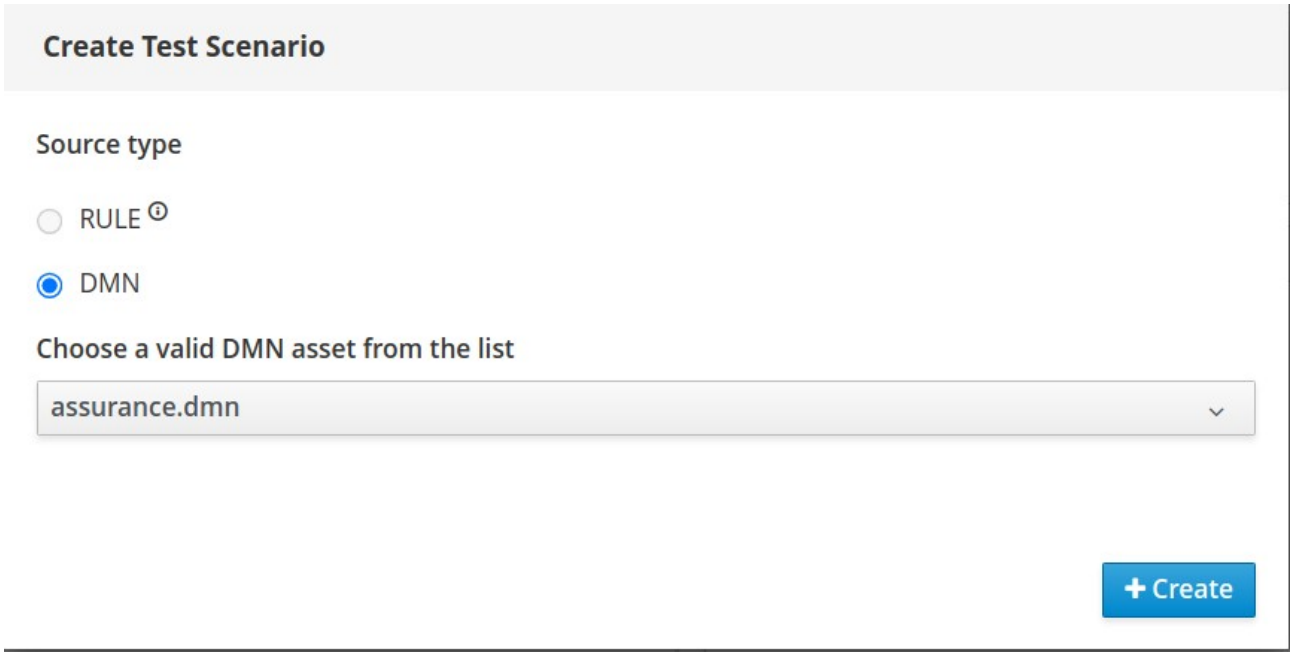
2.2 Test de l'implémentation via Drools

Exécuter la classe de test fourni ***src/test/java/org.formation/Assurance.test*** et vérifier que les tests passent

2.3 Écriture d'un scénario de test

Créer un fichier ***src/test/resources/assurance.scesim***

Éditer le avec VSCode et suivre l'assistant de création



Créer 2 scénario de test comme celui-ci :

#	Scenario description	GIVEN			EXPECT		
		Conducteur			Bonus ancienneté	Final Price	Malus incidents
		Age	Anciennete	Incidents	value	value	value
1	Nouveau Conducteur de 20 ans sans malus	20	0	0	0	500	0
2	cteur de 23 ans avec 5 ans d'ancienneté et 10 in	23	5	10	10	700	50

2.4 Exécution des tests

Dans une console, exécuter l'ensemble des tests via
`mvn test`

2.5 Démarrage en mode Dev

Exécuter l'application en mode développement
`mvn quarkus:dev`

Accèder à l'interface Swagger :
http://localhost:8080/q/swagger-ui/

Interagir avec l'API

Atelier 3 : Compléments DMN

On effectue de légères modification du DMN précédent pour illustrer les BKM, les modèles imbriqués et les nœuds services

3.1 Nœud BKM

Créer un nœud BKM qui encapsule une fonction FEEL Appliquer Pourcentage.
La fonction prend 2 paramètres :

- prix
- remise

Et fournit en résultat le prix remisé

Utiliser ce nœud dans le dernier nœud décision calculant le prix final.

Vérifier que les tests ne sont pas cassés

3.2 Nœud service

Encapsuler dans un nœud service ***BonusMalusService*** le calcul de la remise.

Le service prend 2 nœuds d'entrée :

- Bonus ancienneté
- Malus incidents

Et fournit un nœud de sortie :

- Total bonus/malus

Vérifier que les tests fonctionnent.

Exécuter la requête *curl* suivante :

```
curl -X POST http://localhost:8080/assurance/MalusBonusService -H 'content-type: application/json' -H 'accept: application/json' -d '{"Conducteur": {"Anciennete": 5,"Age": 5,"Incidents": 5}}'
```

3.3 Diagramme imbriqué

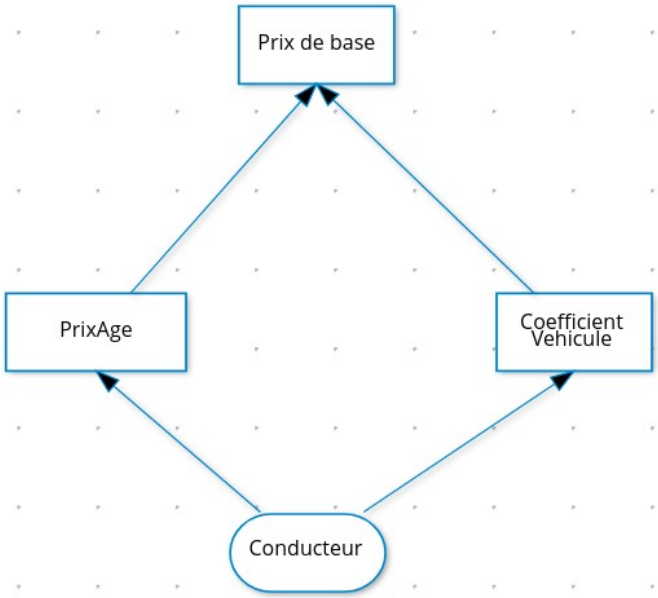
Nous voulons complexifier le calcul du prix de base, dorénavant le prix de base s'appuie sur les règles relatives à l'âge multiplié par un coefficient calculé à partir d'un certains nombre de caractéristiques du véhicule :

- Puissance (Chevaux fiscaux) : Entre 2 et 200CV
- Valeur initiale (Montant en euros)
- Age du véhicule (Nombre d'année) : Doit être supérieur à 2000
- Type de carburant (number Énumération : 1, *2, 3 ou 4)*

Dans un premier temps, inclure un champ ***Vehicule*** de type structure dans le type de données ***Conducteur***

Reprendre le nœud du calcul de prix et créer un nouveau DRD associé

Mettre au point le DRD Associé,



Vous pourrez implémenter le nœud de décision via un Contexte par exemple, cependant quelque soit les clés, renvoyer un coefficient 1 pour que les tests continuent à passer