Transformations

Modèles à modèles, génération de code...

Points abordés

- Différents types de transfos (endo/exo, graphes typés...etc)
- Chainage de transformations + ordres
- Chaine des compilations (parsing + modèles)
- Transformation bi-directionnelles

Transformation

Définition :

Une transformation est une opération qui

- Prend un (ou plusieurs) modèle source en entrée
- Fournit un (ou plusieurs) modèle cible en sortie

Or chaque modèle est conforme à un métamodèle

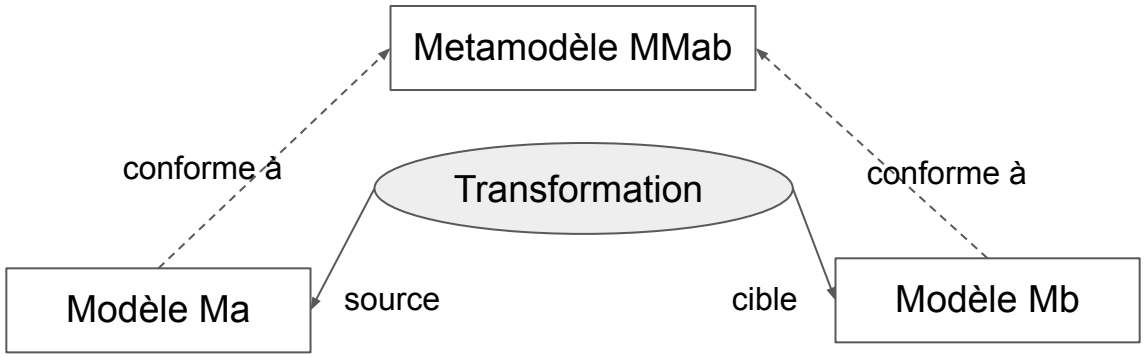
La transformation :

- est définie à partir des concepts des métamodèles source et cible
- suit des règles de transformation
- agit sur des modèles

Transfo endogène / exogène

Transformation endogène

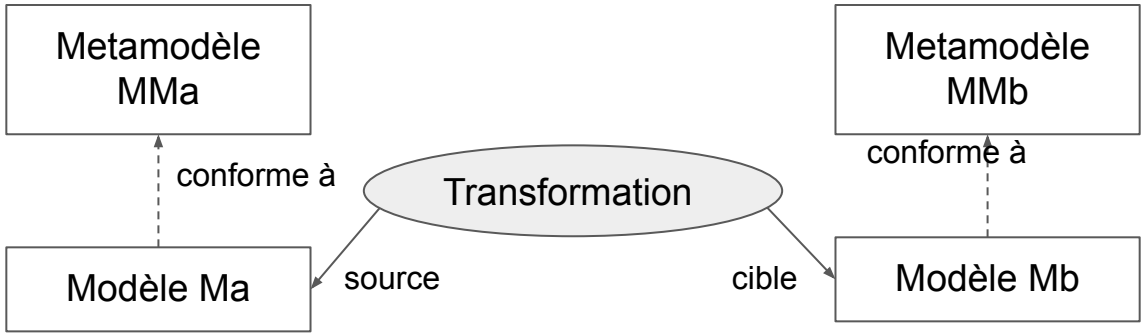
- Les modèles source et cible sont conformes au même métamodèle
- Les refactoring tombent dans cette catégorie
- Exemple : Transformation d'un modèle UML en un autre modèle UML



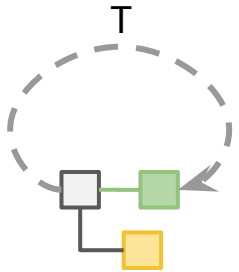
Transfo endogène / exogène

Transformation exogène

- Les modèles source et cible sont conformes à des métamodèles différents
- Le métamodèle source est généralement plus abstrait (mais pas une obligation)
- Exemples : Transformation d'un modèle UML en programme Java, Transformation d'un fichier XML en schéma de BDD

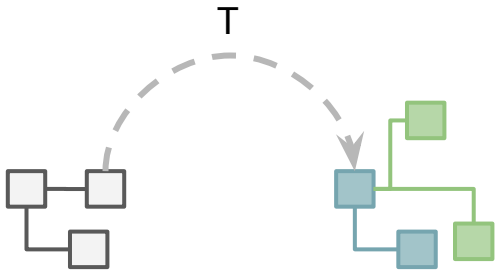


Transfo in-place / out-place



Transformation in-place :

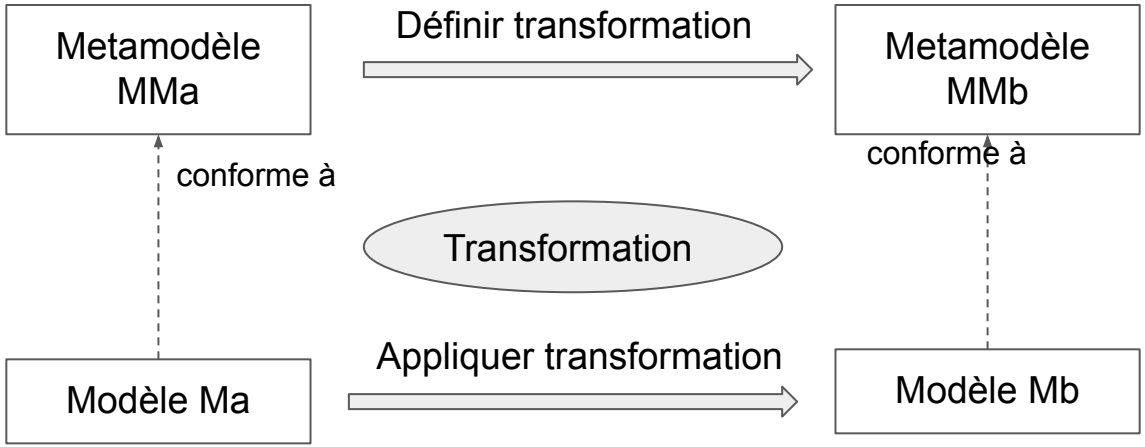
- le modèle source est aussi le modèle cible.
- le modèle source est modifié
- seuls les concepts qui changent sont manipulés
- Exemple : Refactoring



Transformation out-place :

- le modèle source et le modèle cible sont différents
- le modèle cible est créé à partir du modèle source
- tous les concepts même ceux qui restent inchangés doivent être créés
- Exemple : généralement les transformations de modèle

Architecture des transformations



Découpage en règles de transformation

Comme pour les recettes de cuisine, décomposition en étapes

Chaque étape / règle décrit comment transformer un (ou plusieurs) concept(s) source en un (ou plusieurs) concept(s) cible.

Ecrire une transformation c'est écrire l'ensemble des règles de transformation

Executer une transformation c'est identifier tous les éléments du modèle source qui satisfont éventuellement les conditions et créer les éléments du modèle cible correspondant

Approche déclarative

Focalisation sur *ce qui* est créé

- Fonctionnement :
  - Recherche de certains patrons (d'éléments et de leurs relations) dans le modèle source
  - Chaque patron trouvé est remplacé dans le modèle cible par une nouvelle structure d'élément
- Propriété :
  - écriture de la transformation « assez » simple
  - mais ne permet pas toujours d'exprimer toutes les transformations facilement
  - besoin de NAC (conditions non-applicatives) pour limiter l’action des règles

Approche déclarative

Parler des isomorphismes et homomorphismes de graphes pour la recherche de pattern et de la nécessité d’utiliser des NACs

parler de comment les transfos de refact sont gérées (si j’ajoute des éléments, comment je fais pour qu’ils ne soient pas ajouté à la liste des éléments à traiter)

Approche déclarative - en interne

- la recherche de “motifs” se fait par homomorphisme ou isomorphisme structurel (graphes)

Parler des isomorphismes et homomorphismes de graphes pour la recherche de pattern et de la nécessité d’utiliser des NACs

parler de comment les transfos de refact sont gérées (si j’ajoute des éléments, comment je fais pour qu’ils ne soient pas ajouté à la liste des éléments à traiter)

Approche impérative

Focalisation sur comment la règle est exécutée

- Fonctionnement
  - Proche des langages de programmation
  - On parcourt le modèle source dans un certain ordre et on génère le modèle cible lors de ce parcours
- Propriété
  - Ecriture transformation plus complexe…
  - mais permet de toutes les définir

Approche impérative

dérouler l’algo rapidement en fonction d’un point d’entrée (parcourt des éléments, puis recherche du main qui s’applique, puis appels des règles depuis les autres règles)

parler du cache des transformations, résolution d’élements

parler de comment les transfos de refact sont gérées (si j’ajoute des éléments, comment je fais pour qu’ils ne soient pas ajouté à la liste des éléments à traiter)

Approche hybride

à la fois déclarative et impérative

Approche qui est utilisée en pratique dans la plupart des outils

M2M vs M2T

Si la source et la cible de la transformations sont des modèles, on parle de transformation M2M (modèle vers modèle).

A l'inverse si la source de la transformation est un modèle et la cible du texte, on parle de transformation M2T (modèle vers texte).

Similarités :

- Découpage en règles
- Type d'approche

Différences :

- Génération de la cible (on ne génère pas un modèle comme du texte).

Chaîne de transformation

- Une chaîne de transformation est une série de transformations où les modèles cible servent de modèles source à la transformation suivante
- Développement d'un système :
  - Processus basé sur une série de transformations de modèles
- Exemple :
  1. Modèle de l'application au niveau abstrait, vers un modèle de composant abstrait
  2. Projection du modèle vers un modèle de composant EJB
  3. Raffinement de ce modèle pour ajouter des détails d'implémentation
  4. Génération du code de l'application modélisée vers la plateforme EJB

Réutilisation dans les chaînes

Problématiques :

- Comment décomposer les transformations en petites transformations faisant du sens et réutilisable ?
  - Une intention particulière,
  - Éventuellement plusieurs règles
- Comment réutiliser ces petites transformations d’une chaîne à une autre ?

Le concept de transformation localisée

Les bénéfices des transformations in-place :

- les règles de transformation ne concernent que les concepts qui changent.
- tout ce qui ne change pas reste en place

Les bénéfices des transformations out-place :

- possibilité d’avoir un métamodèle source et cible différent

De nouvelles contraintes de chaînage

Motra - M2M en Python

Motra est un DSL interne à Python de transformation de modèles. Il repose sur une sémantique proche de celle de QVTo (impérative) et permet d’écrire en pur Python des transformations de modèles.

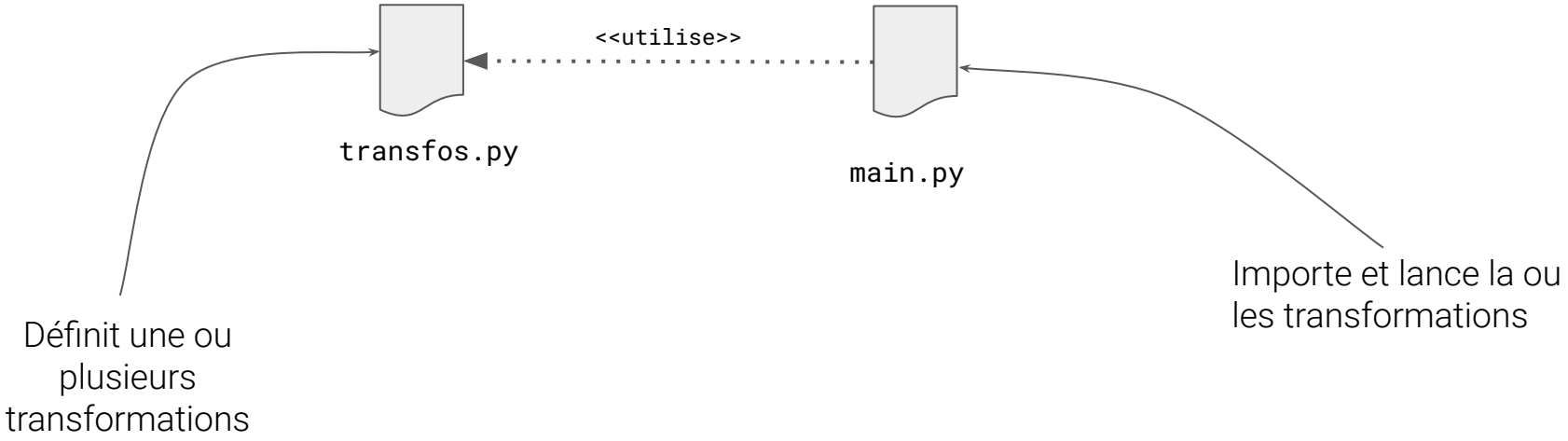
Pour des raisons pratiques et techniques, c’est ce langage que vous allez manipuler pour ce TP.

Motra - Vue d’ensemble

- DSL interne reposant sur la définition de **`mappings`** comme en QVTo
- Utilise beaucoup de méta-programmation pour rendre les choses magiques
- Définit 3 concepts principaux :
  - `@xxx.main :` le point d’entrée de la transformation
  - `@xxx.mapping :` comment transformer un élément de modèle
  - `@xxx.disjunct :` définit une séquence de règles à tester
- Se décline en deux saveurs : M2M et M2T
- Encore expérimental, mais gagne en utilisation (ex: Altran Pays-bas)

Flow de dev avec Motra

- Les transformations s’écrivent dans des modules et elles sont utilisables à partir d’un autre



Motra M2M - Définition des méta-datas d’une transfo

Avant d’écrire une transformation en elle-même, il faut définir ses méta-informations : son nom, ses paramètres d’entrée et de sortie. Cela se fait en définissant une variable globale qui porte le nom de la transformation

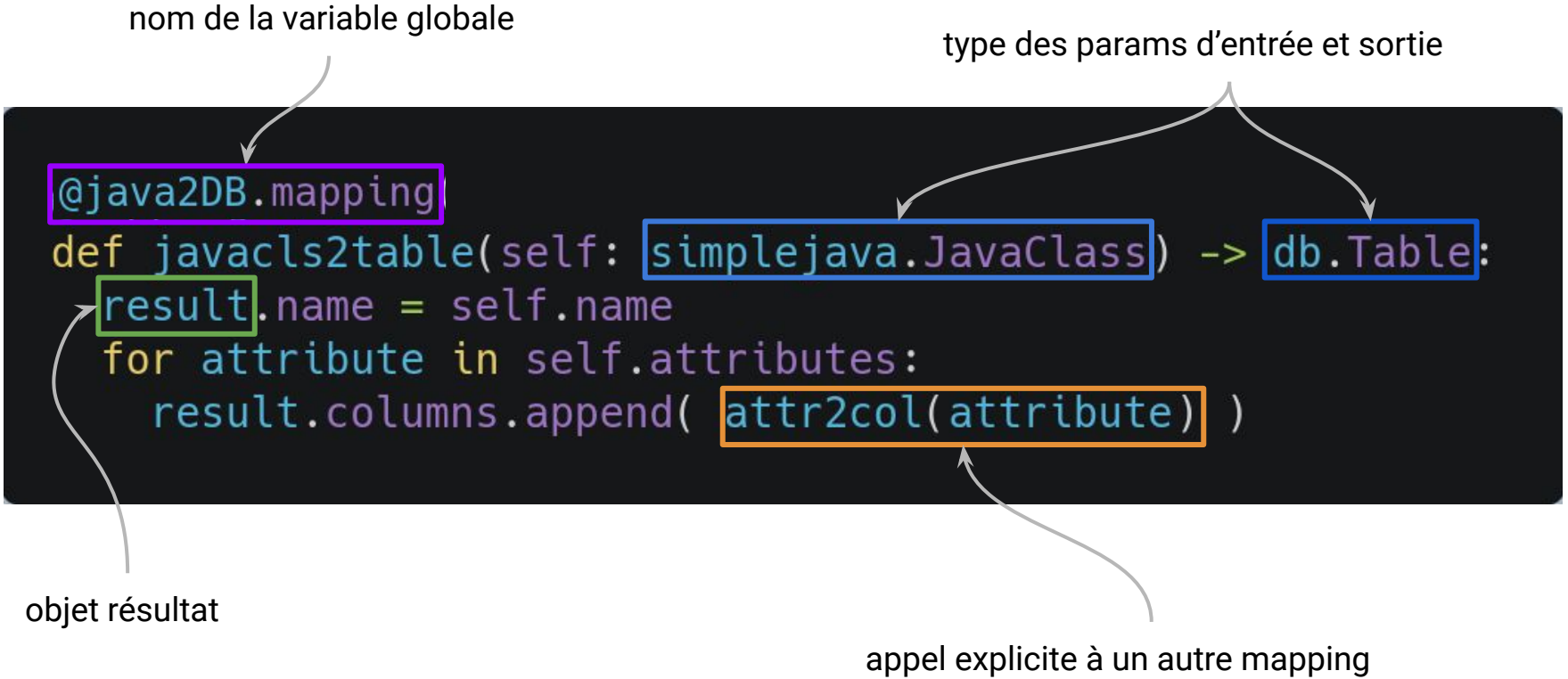
```
from motra import m2m

java2DB = m2m.Transformation('java2DB',
                             inputs=['java_model'],
                             outputs=['db_model'])
```

Motra M2M - Définition de mappings

La définition d'un mapping se fait avec le décorateur `@xxx.mapping` sur une fonction avec les spécificités suivantes :

- Il faut absolument utiliser les type-hint de Python
- Le premier paramètre doit forcément être nommé `self`
- Dans la fonction, l'élément créé est nommé `result` et est automatiquement créé/injecté dans le contexte courant
- Le résultat du mapping est mis en cache. En conséquence, deux appels successifs avec les mêmes paramètres d'entrée retourne le même élément

Motra M2M - Définition de mappings

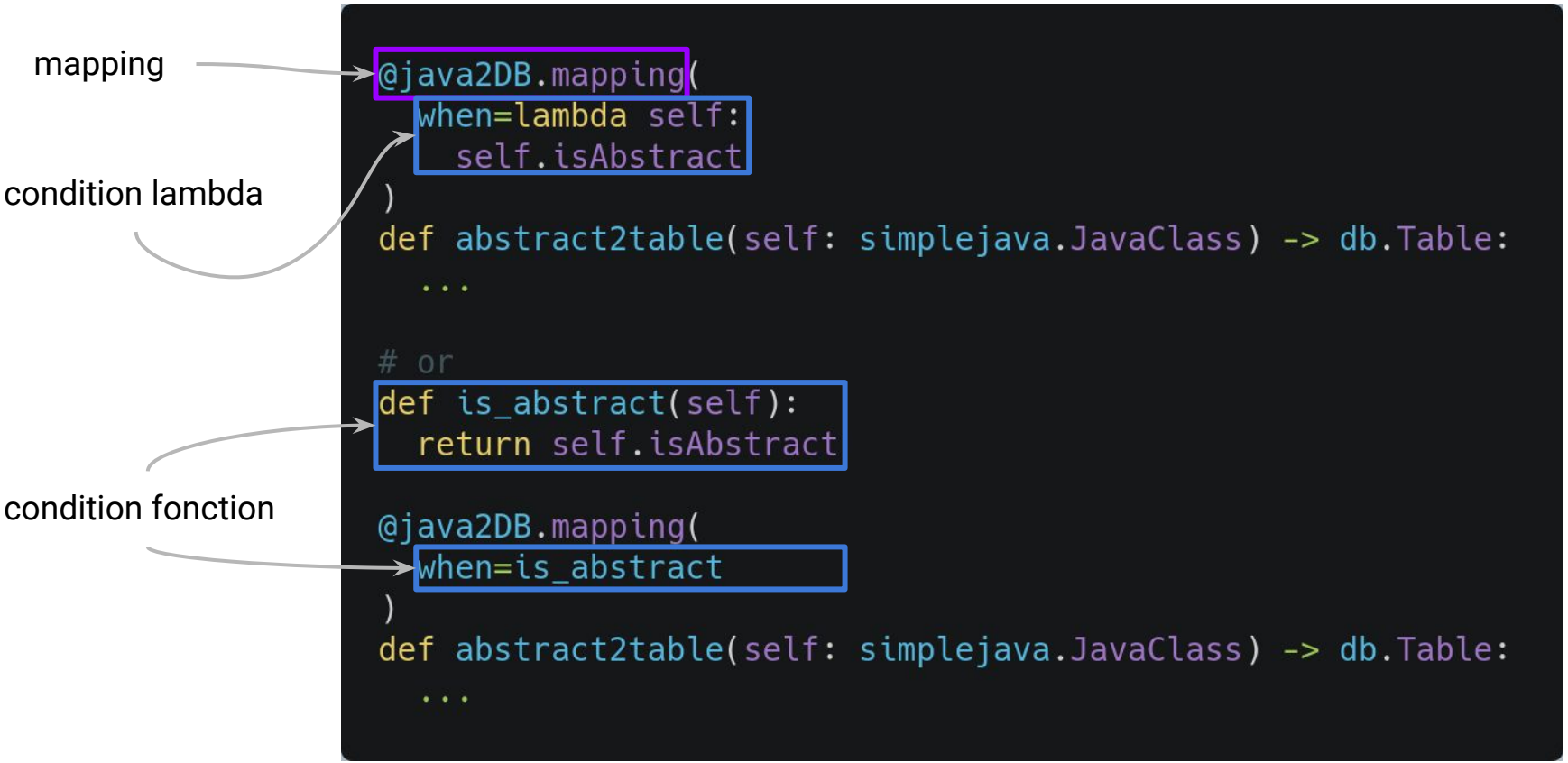


Motra M2M - Définition de mappings conditionnels

Un mapping conditionnel est un mapping qui ne s’active que lorsqu’une condition est remplie :

- cette condition est représentée par une fonction qui prend en paramètre les mêmes paramètres que le mapping sur lequel il s’applique
- si la condition ne s’applique pas, il ne se passe rien de spécial
- pour exprimer la condition, on utilise soit une fonction, soit une lambda expression

Motra M2M - Définition de mappings conditionnel



Motra M2M - Définition du point d’entrée

Les transformations Motra sont des transformations impératives. Il est nécessaire d’indiquer quel est le point d’entrée de la transformation à partir duquel la transformation commence et les règles font s’enchaîner. Cela se fait avec l’annotation @xxx.main :

- la fonction main doit avoir le même nombre de paramètres avec les mêmes noms que ceux définis dans les méta-données de la transformation
- une transformation ne peut avoir qu’un seul “main”
- les éléments passé en paramètres du main sont les Resources qui contiennent les modèles d’entrée et sortie

Motra M2M - Définition de point d’entrée

Définition du “main”

Paramètres tel que définit dans les méta-données de la transformation

```
@java2DB.main
def entry(java_model, db_model):
    root = java_model.contents[0]
    jmodel2DBModel(root)
```

Appel de la première règle (plusieurs peuvent être appelés à partir du main)

Motra M2M - Définition de disjuncts

Les disjuncts sont créés avec l'annotation `@xxx.disjunct` et permettent de créer une super règle qui est l'aggrégation de plusieurs mapping. Le disjunct tente d'exécuter chaque mapping les uns après les autres et s'arrête dès qu'un mapping est exécuté :

- tous les mappings ajoutés au disjuncts doivent avoir le même nombre de paramètres avec les mêmes noms
- l'ordre d'ajout des mappings au disjunct est important
- il peut-être utilisé pour implémenter un single dispatch

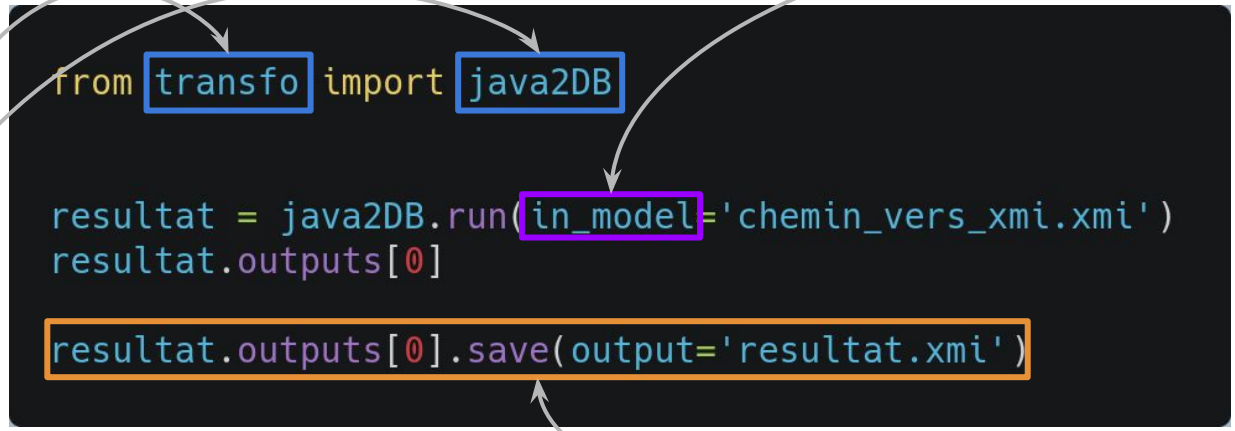
Motra M2M -Disjunct

Lancer une transformation et sauver le résultat

Une fois que la transformation est écrite, il faut la lancer à partir d'un autre module (ça peut-être le même, mais c'est mieux de découper la partie définition transformation de la partie utilisation).

- il faut importer la transformation pour l'utiliser
- il suffit de "run" la transformation pour avoir un contexte résultat
- c'est sur dans le contexte résultat que le modèle transformé peut-être récupéré

Lancer une transformation et sauver le résultat



utilise le nom des “inputs” définis pour la transformation

importe la transformation définie depuis le fichier de transfo

le modèle résultat est accessible et on le sauve dans un fichier spécifique

Ce qu’il faut retenir

- Les transformations de modèles sont un des mécanismes qui donnent du sens aux modèles
- Elles peuvent permettre de raffiner des modèles, de changer d’espace technologique ou de modifier automatiquement des modèles
- Il y a 2 grands types de transformations : déclaratives et impératives
- Il est possible de chaîner les transformations pour créer des chaines de transformations
- L’ordre des transformations dans une chaîne est très important
- Les transformations de modèles vers texte (M2T) sont un moyen de générer du code

Ce que vous devez savoir faire

- Réfléchir abstraitement sur les différentes règles nécessaire à l’établissement d’une transformation
- Créer une transformation en utilisant un framework de transformation de modèles types QVTo et ATL (vu l’année passée) ou en pur Python (vu en TP)
- Donner l’ensemble des règles nécessaires à l’établissement d’une transformation (à l’issue des TPs)
- Comprendre les interactions entre règles