

Thème :

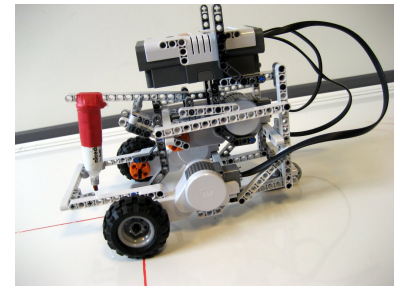
Utilisation de l'ingénierie dirigée par les modèles (IDM) pour construire des langages dédiés (DSL).

Mots clés :

Ingénierie dirigée par les modèles (IDM), génération de code, sémantique statique, sémantique structurelle, syntaxe abstraite, syntaxe concrète.

Objectifs :

- Montrer l'utilité de l'ingénierie dirigée par les modèles pour construire des langages dédiés à un domaine spécifique (DSL pour *Domain Specific Language* en anglais). Ces langages permettent d'avoir un modèle de spécification de haut niveau.
- Utiliser les techniques et les outils associés de l'IDM comme la méta-modélisation, la vérification structurelle, la génération de code et la transformation de modèles.



Introduction :

Dans ce TP, nous construisons un langage dédié (DSL) à manipuler le robot *NXT*. L'objectif est donc de construire un langage de haut niveau, plus proche de l'utilisateur et facile à manipuler qu'un langage de programmation classique (ex. Java, C++). Pour cela, il faut créer en plus du DSL, les outils nécessaires afin d'avoir une utilisation intuitive et ludique.

Ce TP se compose de dix étapes. Les outils, les fichiers et les codes sources nécessaires à chaque étape se trouvent dans un sous-répertoire portant le nom de l'étape.

1. Étape 1 : Préparation

- Afin de réaliser ce TP, vous devez impérativement utiliser **Eclipse Modeling Framework**. S'il n'existe pas déjà sur votre machine, vous pouvez le télécharger à partir de :
 - `//DataProfs/Informatique/yassine/A3/TP-IDM-NXT/etape1/eclipse-modeling-2018-09-linux-gtk-x86_64.tar.gz`
- Votre *Eclipse Modeling* doit contenir les plugins suivants :
 - **Ecore Tools** : pour la méta-modélisation et la modélisation.
 - **Sirius** : pour la création de la syntaxe concrète graphique ainsi que la création/personnalisation des diagrammes.
 - **OCL (Object Constraint Language)** : pour la vérification des modèles en utilisant la logique du premier ordre.
 - **Xtext** : pour la création de la grammaire et l'éditeur de texte afin d'avoir une syntaxe concrète textuelle du DSL
 - **Acceleo** : pour la transformation de type *model-to-text* qui servira pour générer le code *NXC (Not eXactly C)* à embarquer dans le robot ainsi que le code *Python* à exécuter sur la machine pour la simulation.
- Créez un nouveau « *Empty EMF project* » et appelez-le « *fr.ensma.idm.choreography* ».
- Importez le modèle « *choreography.ecore* » dans le répertoire « *model* » de votre projet.




Done

2. Étape 2 : Définition de la sémantique structurelle du DSL

Le modèle *choreography* représente la sémantique structurelle (ou la syntaxe abstraite) du langage à construire. Ce modèle définit les primitives principales d'un langage permettant de piloter le robot pour faire des chorégraphies (exemple : *GoForward*, *TurnLeft*, *TurnRight*), dessiner (exemple : *PenUp*), manipuler des objets (exemple : *Grab*), etc.

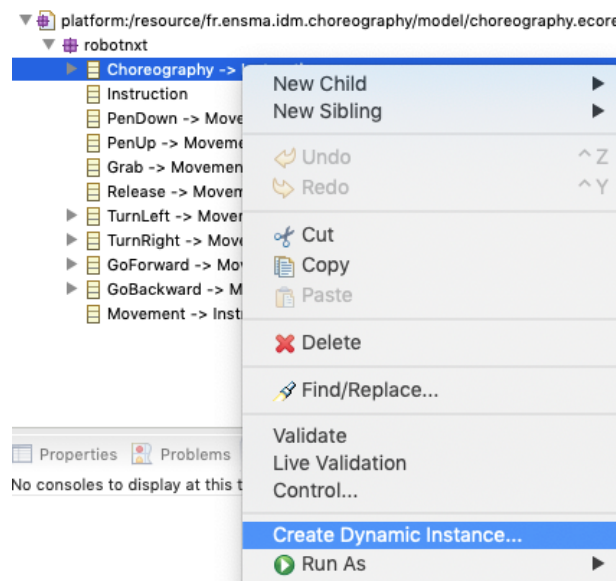
- Prenez le temps de comprendre la structure du méta-modèle fourni. Vous pouvez le visualiser sous différentes formes :

Done

- Avec l'éditeur d'arborescence *Ecore*  Sample Ecore Model Editor
- Ou l'éditeur textuel *OCLinEcore*  OCLinEcore (Ecore) Editor
- Ou l'éditeur textuel (format *XMI*)  Text Editor
- Ou à l'aide de l'éditeur de diagramme *Ecore* (clic droit sur le fichier *.ecore, puis choisir *Initialize Ecore Diagram ...* pour générer un diagramme sur la base du méta-modèle fourni)

3. Étape 3 : Création d'une instance dynamique

- Créez une instance dynamique dont la racine est la classe « *Choreography* »



Done

Figure 1

- Le contenu de votre instance doit correspondre au programme ci-dessous :

```
Choreography dessin {
  PenDown,
  GoForward (50),
  TurnRight (90),
  GoForward (50),
  TurnLeft (30),
  GoBackward (80),
  Choreography subdessin {
    TurnRight (190),
    PenUp,
    GoForward (50) }
  TurnRight (90) }
```

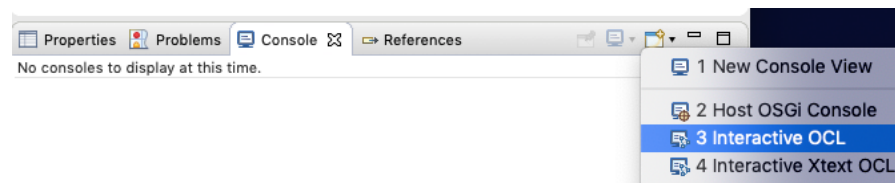
Done

4. Étape 4 : Définition de la sémantique statique du DSL

La sémantique statique est représentée par les contraintes définies sur les éléments du modèle (invariants, pré-conditions, post-conditions, et axiomes). Dans ce TP, nous utilisons le langage OCL (*Object Constraint Language*) pour définir les contraintes du DSL. OCL est un langage formel qui fonctionne parfaitement avec des langages de modélisation comme UML, Ecore, etc.

- Enregistrez et fermez toutes les autres fenêtres des éditeurs qui manipulent le fichier « *Choreography.ecore* »
- Ouvrez le modèle avec l'éditeur « *OCLinEcore* » et rajoutez la contrainte suivante :

- La valeur de l'attribut angle doit être positive et ne doit pas passer 180°. Done
- Pour comprendre l'impact de la définition de cette contraintes OCL sur le modèle « *choreography* », re-ouvrez le méta-modèle avec l'éditeur d'arborescence (après avoir fermé l'éditeur *OCLinEcore*). Remarquez les « EAnnotation » apparues dans le méta-modèle et les classes (contextes) concernées par les contraintes rajoutées. Done
- Vérifiez si votre instance respecte la contrainte que vous avez ajoutée au niveau méta-modèle (clic droit sur l'élément racine de votre instance « *.xmi* », puis choisir *Validate*).
- Vous pouvez également utiliser la console « Interactive OCL » et votre fichier instance « *.xmi* » pour vérifier la syntaxe de vos règles OCL et vérifier si les règles sont bien respectées par l'instance dynamique que vous avez créée à l'étape 3 (voir la figure 2 et 3).



Check done

Figure 2

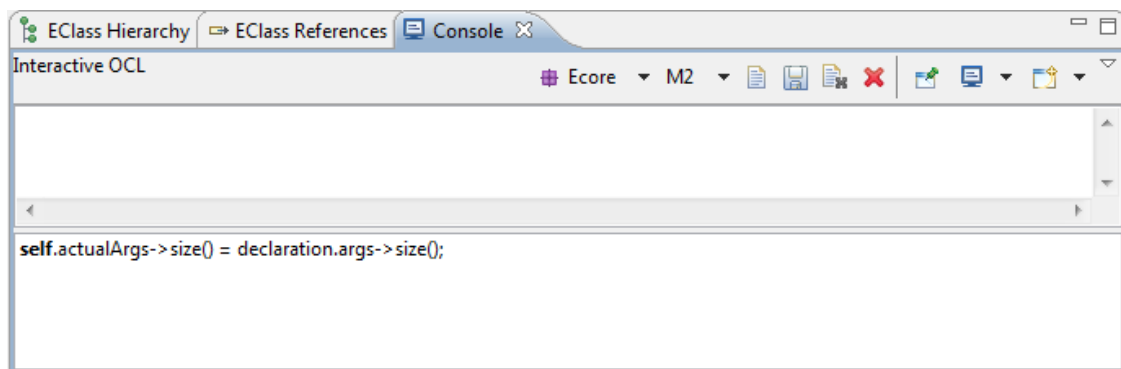


Figure 3

- Ouvrez une autre fois le modèle « *Choreography.ecore* » avec l'éditeur « *OCLinEcore* » et rajoutez les contraintes suivantes : Done
 - Les chorégraphies (instances de la classe *choreography*) ne doivent pas avoir le même nom.
 - Chaque chorégraphie doit contenir autant d'instances de *PenUp* que d'instances de *PenDown*.
 - La valeur de l'attribut *distance* doit être positive.

5. Étape 5 : Génération du plug-in pour le DSL

- Créez le générateur de plug-in pour le DSL (fichier *.genmodel*, voir la figure 4).

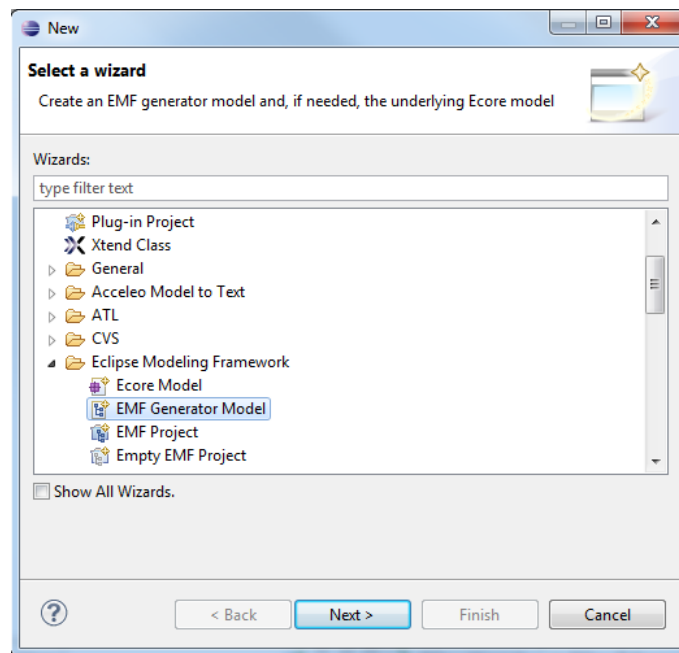


Figure 4

- Mettez à « *True* » la propriété « *Operation Reflection* » dans la catégorie « *Model* » afin que les contraintes OCL soient prises en considération (voir la figure 3).

Done

► All	
► Edit	
► Editor	
▼ Model	
Array Accessors	<input type="checkbox"/> false
Binary Compatible Reflective M...	<input type="checkbox"/> false
Class Name Pattern	<input type="checkbox"/>
Containment Proxies	<input type="checkbox"/> false
Feature Delegation	<input type="checkbox"/> None
Generate Schema	<input type="checkbox"/> false
Interface Name Pattern	<input type="checkbox"/>
Minimal Reflective Methods	<input type="checkbox"/> true
Model Directory	<input type="checkbox"/> /fr.ensma.idm.choreography/src
Model Plug-in Class	<input type="checkbox"/>
Model Plug-in ID	<input type="checkbox"/> fr.ensma.idm.choreography
Model Plug-in Variables	<input type="checkbox"/>
Operation Reflection	<input type="checkbox"/> true
Suppress Containment	<input type="checkbox"/> false
Suppress EMF Metadata	<input type="checkbox"/> false
Suppress EMF Model Tags	<input type="checkbox"/> false
Suppress GenModel Annotations	<input type="checkbox"/> true
Suppress Interfaces	<input type="checkbox"/> false
Suppress Notification	<input type="checkbox"/> false

Figure 5

- Générez le « *Model Code* », le « *Edit Code* », le « *Editor Code* ».

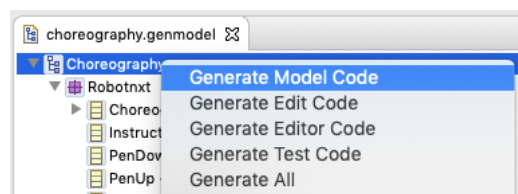


Figure 6

- Remplacez les icones (.gif) qui se trouvent dans le répertoire « *icons/full/obj16* » du plugin généré *fr.ensma.idm.choreography.edit* par les icônes fournies. Done
- Lancez maintenant une nouvelle instance d'Eclipse nommée « *RobotDSLConfig* » (voir la figure 5).

Optionnel : Pour éviter les problèmes d'espace mémoire, pensez à modifier les paramètres de configuration de l'exécution de la nouvelle instance d'Eclipse.

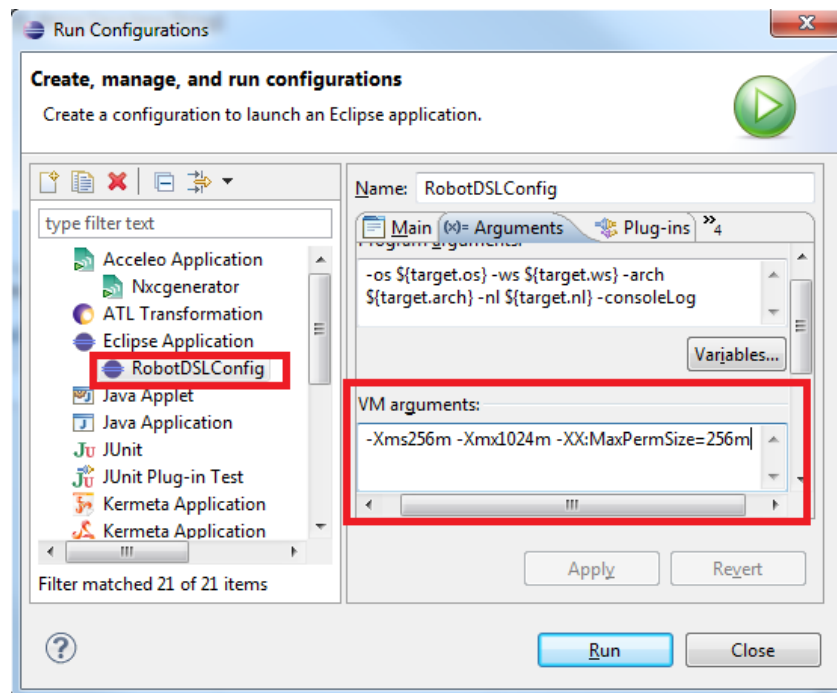


Figure 7

6. Étape 6 : Instanciation statique du DSL

- En utilisant l'instance d'Eclipse *RobotDSLConfig*
 - Créez un projet : General >>> Project
 - Créez une instance de votre DSL : new Other >>> Robotnxt Model (il se trouve dans la catégorie : *Example EMF Model Creation Wizards*)
- Faites une instance du DSL qui viole au moins l'une des contraintes OCL.
- Vérifiez la non-conformité de cette instance aux règles OCL.
- Faites une deuxième instance du DSL qui respecte toutes les contraintes. Cette instance devra exprimer le dessin d'un carré par le robot dessinateur.

Done

7. Étape 7 : Création d'une syntaxe concrète graphique du DSL

L'objectif de cette étape est de créer une syntaxe concrète graphique pour le DSL. Autrement dit, grâce à l'utilisation du plugin *Sirius*, vous devez pouvoir instancier les chorégraphies graphiquement à travers un diagramme doté d'une palette.

- Fermez la deuxième instance d'Eclipse
- A partir de la première instance d'Eclipse importer le projet « *fr.ensma.idm.choreography.design* »
- Visualisez le fichier « *choreography.odesign* » qui se trouve dans le répertoire « *description* »
 - L'objectif de ce fichier est la création des diagrammes, le mapping entre des aspects graphiques et les instances des éléments du méta-modèle, la création des palettes pour les diagrammes et la création des vues adaptées pour visualiser et modifier les propriétés des instances (un extrait du fichier .odesign est représenté par la figure 8).
- Ajoutez le nœud « *CD_TurnLeft* » (la représentation graphique des instances de classe *TurnLeft*) ainsi que sa représentation au niveau de la palette en s'inspirant du nœud « *CD_TurnRight* ».
- Vérifiez (éventuellement modifiez) si les représentations graphiques de tous les nœuds *CD_XXX* correspondent aux figures SVG adéquates.

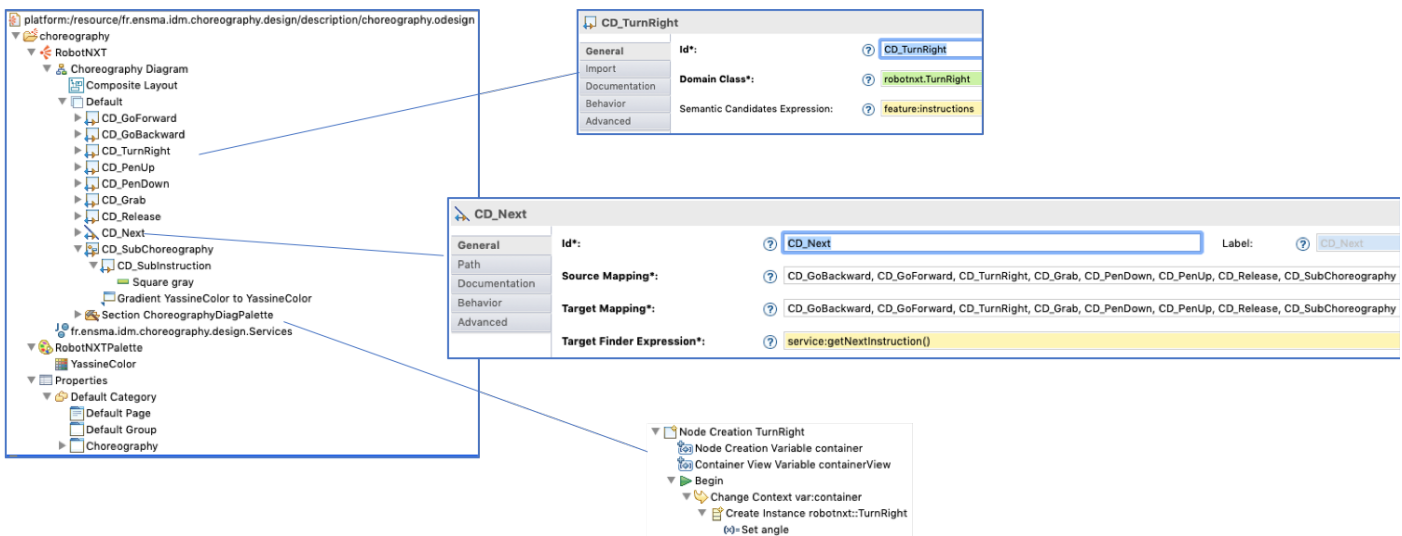


Figure 8

- En utilisant l'instance d'Eclipse *RobotDSLConfig*, vérifiez l'impact de vos modifications sur les fichiers du type «*robotnxt*» et générer/créer le(s) digramme(s) correspondant(s).

8. Étape 8 : Création d'une syntaxe concrète textuelle du DSL

L'objectif de cette étape est de créer une syntaxe concrète textuelle pour le DSL en utilisant le plugin Xtext. La création de la grammaire va se baser sur le fichier *.ecore* existant.

- Convertissez d'abord le projet «*fr.ensma.idm.choreography*» en Xtext Project Done

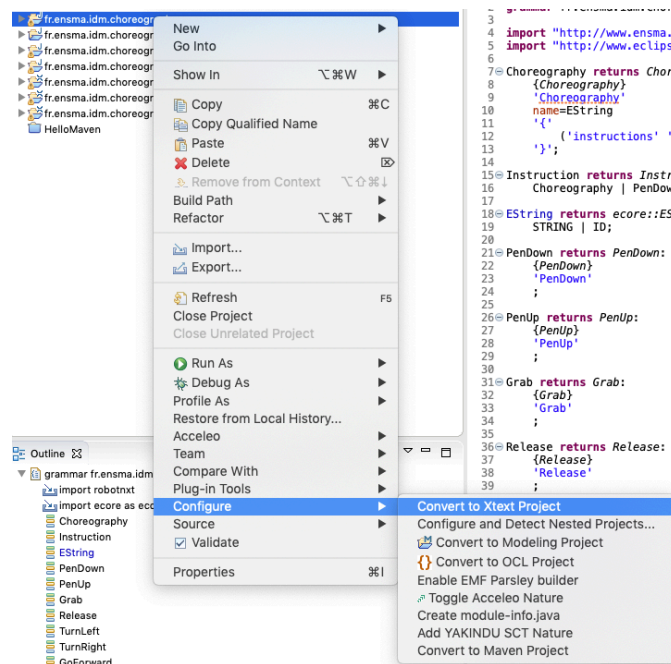
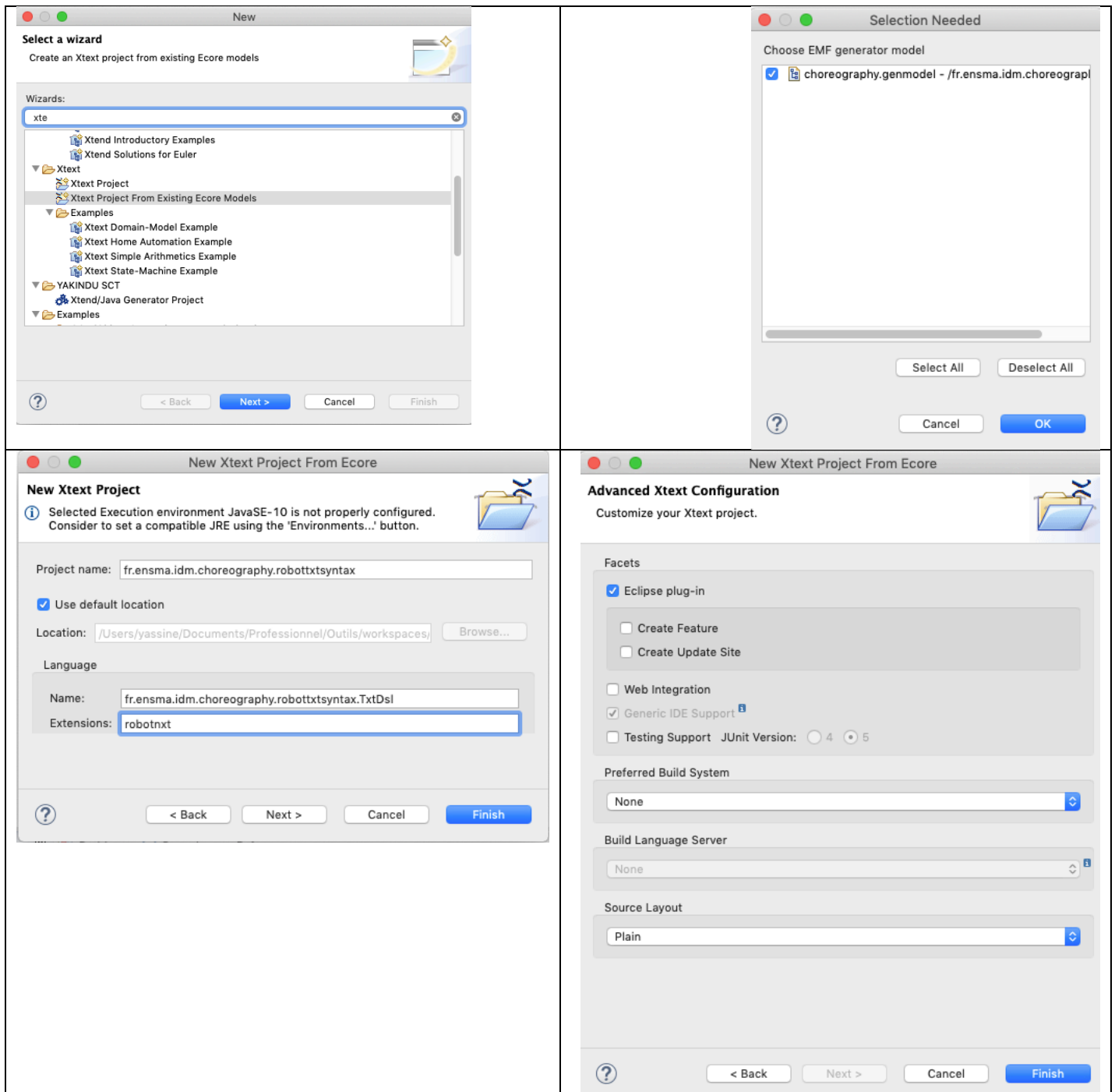


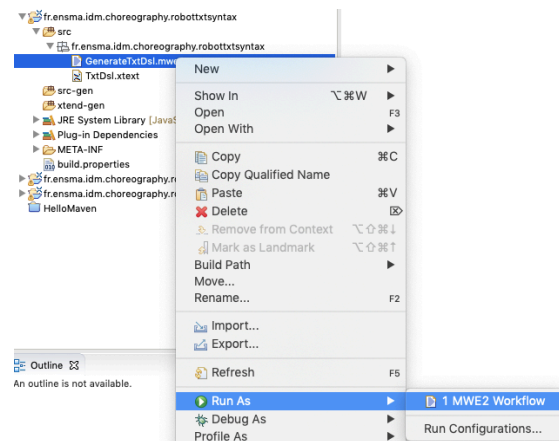
Figure 9

- Créez un projet Xtext en se basant sur le modèle EMF existant (Choreography.ecore). Le projet doit porter le nom suivant «*fr.ensma.idm.choreography.robottxtsyntax*». Suivez les étapes décrites dans les figures ci-après.



- Générer les plugins `fr.ensma.idm.choreography.robottxtsyntax.ide` et `fr.ensma.idm.choreography.robottxtsyntax.ui` à partir du fichier `.mwe2`

Done



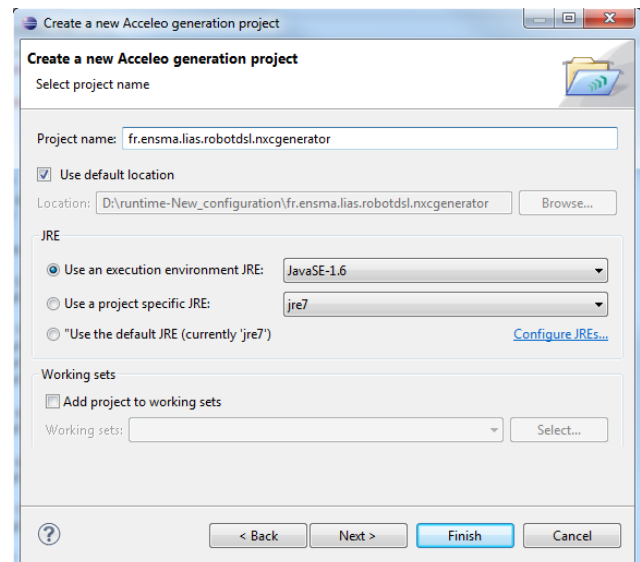
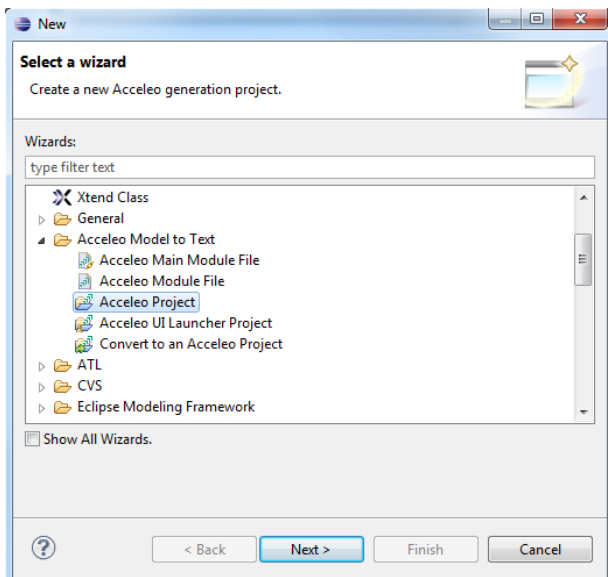
- En utilisant l'instance d'Eclipse *RobotDSLConfig*, vérifier l'impact de vos modifications sur les fichiers du type « *robotnxt* » et générer/créer d'autres fichiers en utilisant l'éditeur de texte généré.
- Modifier la grammaire générée afin d'obtenir une qui vous permet d'avoir la même syntaxe que celle de l'étape 3.
- Revérifiez votre modification apportée à la grammaire (il faut redémarrer l'instance d'Eclipse *RobotDSLConfig*).

9. Étape 9 : Génération du code NXC et du code Python

Nous utilisons l'outil « Acceleo » et son langage associé qui permet de générer du texte à partir de modèles.

Le but de cette étape est de générer deux programmes à la fois à partir d'une instance du DSL :

- Le code Python qui sera exécuté sur votre machine
 - Le code NXC qui sera embarqué dans le robot.
- En utilisant l'instance d'Eclipse *RobotDSLConfig*, créez un projet Acceleo.



- Cliquez sur *Next*.
- Ajoutez le méta-modèle du DSL dans la définition du module *Metamodel URIs*.
- Choisissez *Choreography* comme l'élément racine à partir duquel on fait la génération de code.
- Cochez *Generate file* et *Main template*.
- Copiez le bout de code fourni (le fichier *codemtl.txt*) dans votre fichier « *generate.mtl* »
- Enrichissez votre programme *Acceleo* de façon à pouvoir générer simultanément les codes (NXC et Python) qui correspondent à votre instance « *carre.robotnxt* ».

- Le tableau suivant présente les correspondances entre les classes du DSL et le langage Python.

DSL	Python	DSL	Python
PenDown	down()	GoBackward	backward(x)
PenUp	penup()	TurnLeft	left(y)
GoForward	forward(x)	TurnRight	right(x)

- Les deux fichiers « *exemple.py* » et « *exemple.nxc* » représentent des exemples des codes qui doivent être générés automatiquement.
- Lancez l'exécution du programme *Acceleo* afin de générer le code NXC de l'instance « *carre* ».
- Vous devez obtenir deux fichiers nommés *carre.nxc* et *carre.py*.
- Exécuter le programme python en tapant en ligne de commande : *python carre.py*
- Pour tester le fichier *carre.nxc*
 - Sous Windows : ouvrez le fichier avec l'éditeur Bricx, compilez et embarquez le programme sur le Robot
 - Sous Linux : regardez le fichier « *configurationLinux.txt* »

10. Étape 10 : Compte rendu

- Un fichier zip contenant les fichiers .ecore, .odesign, .xtext, .mtl, .py, .nxc à mettre sur le cloud.
- Envoyez le lien de téléchargement à yassine.ouhammou@ensma.fr