Eclipse Modeling Framework (EMF)

Objectif de ce TP :

— Manipulation avancée de Eclipse Modelling Framework (EMF)
— Création et manipulation d'éditeurs arborescents des modèles avec EMF
— Manipulation des métamodèles et/ou modèles en Java dans EMF
— Transformation modèle à modèle (M2M : model to model transformation) avec Java

La version d'Eclipse à utiliser est la suivante (Eclipse EMF version 2020-09 (4.17.0), intsallée sur les machines de l'N7) :

```
/mnt/n7fs/ens/tp_ouederni/eclipse-idm/eclipse
```

1 Comprendre les plug-ins Eclipse

Exercice 1 : Étendre Eclipse grâce aux greffons (*plug-ins*)

L'architecture d'Eclipse repose sur un système de greffons (*plug-ins* en anglais) qui permet d'étendre les fonctionnalités de l'IDE. L'objectif de cet exercice n'est pas d'apprendre comment développer des greffons, ni de comprendre Equinox, le framework OSGI utilisé par Eclipse pour gérer ces greffons mais tout simplement de comprendre la notion de greffon d'Eclipse, le mécanisme de déploiement et comment tester un greffon en cours de développement sans avoir à le déployer. Nous nous appuyons sur les exemples de greffons présents dans Eclipse.

1.1 *Création d'un greffon.* Commençons par créer un projet *Plug-in Project*. Nous pouvons l'appeler `fr.n7.eclipse.plugin.exemple`. Après avoir fait *Next*, vérifier que l'option « *This plug-in will make contributions to the UI* » est cochée sur le deuxième écran de l'assistant. Sur le troisième écran, sélectionner « *Hello, World Command* ». Comme l'indiquent les explications sur la partie droite, l'assistant va créer la structure d'un projet de plug-in qui ajoute un nouveau menu appelé « *Sample Menu* » à la barre de menu d'Eclipse et un bouton dans la barre l'outil. Les deux appellent la même action « *Sample Action* » qui affiche un message dans une boîte de dialogue. L'écran suivant permet en particulier de préciser le texte à afficher. Sélectionner *Finish* pour créer le greffon.

Eclipse propose de basculer sur la perspective *Plug-in Development* qui est adaptée à ce type de projet. Il faut donc accepter la proposition.

Le projet, donc le greffon, est créé !

1.2 *Tester le greffon.* Pour tester le greffon, Eclipse permet de lancer un deuxième Eclipse, dit Eclipse de déploiement, depuis le premier Eclipse.

Pour lancer l'Eclipse de déploiement, sélectionner le projet de greffon [1], cliquer à droite pour

1. Ou le fichier META-INF/MANISTEST.MF du projet de greffon.

sélectionner *Run as... / Eclipse Application*. Ce deuxième Eclipse a accès aux projets du premier Eclipse, en particulier à notre greffon.

Constater que dans l'Eclipse de déploiement, la barre de menus propose bien le menu *Sample Menu*. En sélectionnant l'action *Sample Command*, la boîte de dialog apparaît.

Constater que l'un des boutons de la barre d'outil (celui qui a le symbole d'Eclipse) donne aussi accès à la boîte de dialogue.

L'Eclipse de déploiement peut maintenant être quitté.

1.3 *Déploiement du greffon dans le premier Eclipse.* Au lieu de lancer un deuxième Eclipse pour avoir accès aux greffons de l'Eclipse de travail, on peut directement déployer un greffon dans l'Eclipse actuel.

1.3.1 *Déployer un greffon.*

1. Sélectionner à la souris l'un des projets à intégrer dans la plateforme Eclipse.
2. Faire un clic droit à la souris et sélectionner "Export..."
3. Dans "Plug-in Development", sélectionner "Deployable plug-ins and fragments". Il s'agit de projets qui peuvent être intégrés dans la plateforme Eclipse : des greffons ou des profils (*features* en anglais) intégrant plusieurs greffons.
4. Dans "Available Plug-ins and Fragments", sélectionner tous les projets à intégrer.
5. Dans l'onglet "Destination", sélectionner "Install into host. Repository :". Il s'agit d'installer dans la plateforme actuelle. Il n'est pas utile de modifier le répertoire de destination.
6. Dans l'onglet "Options", sélectionner "Use class files compiled in the workspace".
7. Cliquer sur "Finish". Accepter d'installer des éléments qui ne sont pas sécurisés. Accepter de redémarrer Eclipse.
8. Lors du redémarrage, constater que le menu "Sample Menu" et le bouton avec le logo Eclipse sont appraus dans l'interface.

1.3.2 *Consulter les greffons et profils installés.*

1. Sélectionner l'entrée "About Eclipse" dans le menu *Help* [2] d'Eclipse. Cliquer sur le bouton "*Installation Details*".
2. Vérifier en utilisant le filtre (zone de texte en dessous des onglets) que les greffons ont bien été installés (préfixe fr.n7).

1.3.3 *Supprimer un greffon ou profil installé.*

1. Dans le dialogue "*Eclipse Installation Details*", dans l'onglet "*Installed Software*", sélectionner le greffon à supprimer (fr.n7.plugins.exemple) et cliquer sur le bouton "*Uninstall*".
2. Suivre les indications jusqu'au redémarrage d'Eclipse et consulter les greffons installés pour vérifier que celui-ci n'est plus présent. Noter également que le menu "*Sample Menu*" et le bouton avec le logo Eclipse ont disparu de l'interface.

2. Sa position dépend de l'environnement utilisé, sous Linux elle se situe dans le menu *Help* et sous MacOSX dans le menu *Eclipse*.

2 Creation et manipulation d'éditeurs avec EMF

Nous allons voir comment un métamodèle Ecore peut être utilisé pour engendrer de l'outillage pour manipuler des modèles conformes à ce métamodèle.

Exercice 2 : Engendrer le code Java et un éditeur arborescent
Commençons par générer les classes Java nécessaires à la création de notre éditeur.

2.1 *Configurer la génération du code Java.* Dans *EMF*, la génération de code Java est configurée à l'aide d'un modèle appelé *genmodel*. Pour créer ce modèle, cliquer droit sur notre fichier *SimplePDL.ecore*, puis faire *New / Other...* et rechercher *EMF Generator Model*. Cliquer sur *Next*, choisir un nom pour le fichier genmodel (SimplePDL.genmodel fera l'affaire) et le placer dans le même dossier que le métamodèle (déjà sélectionné). Faire *Next*. Sélectionner *ecore model* et faire *Next*. Cliquer sur *Load* pour charger le fichier, faire *Next* puis *Finish*. L'assistant se termine et le fichier *SimplePDL.genmodel* doit s'ouvrir automatiquement.

2.2 *Configurer la génération.* Ouvrir le fichier *genmodel* (*SimmplePDL.genmodel*). Un éditeur arborescent s'ouvre. Sélectionner sa racine. Dans le vue *Properties*[3] il est possible de modifier les paramètres de génération des classes Java. Par exemple, on peut décider si la valeur d'un attribut sera éditable ou non. Nous nous contenterons ici de garder les paramètres par défaut.

2.3 *Engendrer le code Java.* Cliquer à droite sur la racine de *SimplePDL.genmodel* (dans l'éditeur arborescent) et déclencher l'action *Generate Model Code*. Les classes Java correspondant au métamodèle SimplePDL ont été engendrées dans le dossier `src`. Le vérifier. En particulier regarder le contenu des paquetages `simplepdl` et `simplepdl.impl`. Le premier contient des interfaces, le second des réalisations de ces interfaces. Regarder aussi le contenu de `SimplepdlFactory`.

2.4 *Engendrer le code pour l'éditeur arborescent.* De la même manière que précédement effectuer les actions de génération : *Generate Edit Code* et *Generate Editor Code*. Ces actions nous permettent de générer un éditeur arborescent pour les modèles conformes au métamodèle SimplePDL. Cet éditeur est généré dans les nouveaux projets fr.n7.simplepdl.edit et fr.n7.simplepdl.editor.

2.5 Les plus attentifs auront remarqué une quatrième option nommée *Generate Test Code*. Elle permet de générer automatiquement une suite de Tests pour notre architecture. Une fois générée, cette suite de tests ne demanderait qu'à être complétée...

Exercice 3 : Utiliser l'éditeur arborescent
Pour utiliser l'éditeur arborescent que l'on vient d'engendrer, il faut déployer les greffons.

3.1 *Déployer les greffons.* Déployer les trois greffons `fr.n7.simplepdl`, `fr.n7.simplepdl.edit` et `fr.n7.simplepdl.editor` en suivant les indications de la question 1.3.1.

3.2 *Créer un projet.* Nons commençons, comme toujours, par créer un projet (*File / New / Project*, puis *General / Project*) que l'on peut appeler *fr.n7.simplepdl.exemple*.

3.3 *Lancer l'éditeur arborescent.* Dans le projet que l'on vient de créer, on peut lancer l'éditeur arborescent en faisant *New / Other...* puis dans *Example EMF Model Creation Wizards*, on

3. Il faut éventuellement passer en perspective Ecore si ce n'est pas le cas.

sélectionne *SimplePDL Model*. C'est bien le notre ! On peut ensuite conserver le nom par défaut proposé pour le modèle (*my.simplepdl*). Sur l'écran suivant, il faut choisir le *Model Object*, l'élément racine de notre modèle. On prend *Process*. On peut enfin faire *Finish*.

3.4 *Saisir un modèle de processus.* Le fichier `My.SimplePDL` est dans la fenêtre principale. Il contient l'élément *Process*. On peut cliquer à droite pour créer des activités (*WorkDefinition*) ou des dépendances (*WorkSequence*). Cet éditeur s'appuie sur la propriété *containment* pour savoir ce qui peut être créé (en utilisant *New Child* du menu contextuel). Par exemple, en se plaçant sur un élément *WorkDefinition*, on ne peut pas créer de *WorkSequence* puisque les références *linkToPredecessors* ou *linkToSuccessors* sont des références avec *containment* positionné à faux.

Pour avoir accès aux propriétés, il est conseillé de repasser dans la perspective *Modeling* (ou *Ecore*) et d'utiliser la vue « Properties » (si elle ne s'est pas affichée lors du changement de perspective, faire *Windows* > *Show View* > *Properties*).

Créer un modèle de processus appelé `exemple` avec deux activités `a1` et `a2` et une dépendance entre de type *startToFinish* entre `a1` et `a2`.

On peut utiliser l'action *Validate* du menu contextuel sur chacun des éléments du modèle. Ceci vérifie que cet élément et ses sous-éléments sont conformes au métamodèle SimplePDL.

Exercice 4 : Améliorer l'affichage de l'éditeur

On constate que seul le type des WorkSequence est affiché. Ceci rend difficile leur identification. On se propose de modifier le code engendré pour afficher l'activité précédente et la suivante.

4.1 Dans le projet `fr.n7.simplepdl.edit`, modifier le code de la méthode `getText(Object)` de la classe `WorkSequenceItemProvider` tel que suit après avoir mis en commentaire le code actuel :

```
WorkSequence ws = (WorkSequence) object;
WorkSequenceType labelValue = ws.getLinkType();
String label = "--" + (labelValue == null ? "?" : labelValue.toString()) + "-->";
String previous = ws.getPredecessor() == null ? "?" : ws.getPredecessor().getName();
String next = ws.getSuccessor() == null ? "?" : ws.getSuccessor().getName();
return label == null || label.length() == 0 ?
      getString("_UI_WorkSequence_type") :
      getString("_UI_WorkSequence_type") + "␣" + previous + "␣" + label + "␣" + next;
```

Ajouter `NOT` après @generated dans le commentaire de documentation pour éviter que la prochaine génération à partir du *genmodel* n'écrase nos modifications.

4.2 Redéployer les greffons. Constater que la *WorkSequence* fait bien apparaître le nom de l'activité précédente et la suivante.

Créer une nouvelle *WorkSequence*. Comment s'affiche-t-elle ?

Initialiser son activité précédente avec `a1`. Comment s'affiche la *WorkSequence* ? Pourquoi les « `?` » sont-ils toujours là ?

4.3 Regarder le code de la méthode `notifyChanged(Notification)` de la classe `WorkSequenceItemProvider`.

Y ajouter le code suivant :

```
case SimplepdlPackage.WORK_SEQUENCE__PREDECESSOR:
case SimplepdlPackage.WORK_SEQUENCE__SUCCESSOR:
```

Redéployer les greffons.

4.4 Définir l'activité suivante de la deuxième *WorkSequence* et constater que son nom est bien mis à jour.

3 Utilisation du code Java/EMF pour manipuler des modèles

Exercice 5 : Manipuler des modèles en Java
Dans cet exercice, nous allons manipuler des modèles EMF à partir de code Java.

5.1 *Chargement de l'exemple de code.* Nous fournissons avec ce TP deux classes Java nommées *SimplePDLCreator.java* et *SimplePDLManipulator.java*. Dans le projet contenant les sources générées à l'aide du fichier *.genmodel*, ouvrir le dossier contenant les sources Java (dossier *src* à la racine du projet). Créer un *Package* Java nommé par exemple *simplepdl.manip*. Importer les fichiers Java précédemment cités dans ce nouveau *Package*.

5.2 *Exemple de code pour la création de modèles.* Comprendre le contenu du fichier *SimplePDLCreator.java* puis l'exécuter. Constater qu'un nouveau dossier *models* a été créé dans le projet (faire *Refresh*, F5, sur le projet si le dossier n'apparaît pas). Ouvrir le fichier *SimplePDLCreator_Created_Process.xmi* qu'il contient et vérifier son contenu.
Remarque : Pour exécuter une classe Java contenant une méthode *main*, cliquer droit sur le fichier source à exécuter puis *Run As / Java Application*. Le résultat de l'exécution doit s'afficher dans la console d'Eclipse.

5.3 *Exemple de code pour la manipulation de modèles.* Comprendre le contenu du fichier *SimplePDLManipulator.java* puis l'exécuter. Vérifier les affichages produits dans la console.

Remarque : L'exercice suivant fait partie du mini-projet.

Exercice 6 : Transformer des modèles de processus en réseaux de Petri
Écrire un code Java qui transforme un modèle de processus en un modèle de réseau de Pétri et la tester sur plusieurs modèles de processus.

Il est conseillé d'avancer progressivement. En particulier d'exécuter régulièrement le programme pour constater les effets sur le modèle produit.