Eclipse Modeling Framework (EMF)

Attention : Version d'Eclipse à utiliser : `/mnt/n7fs/ens/tp_cregut/eclipse-gls/eclipse`

1 Comprendre les greffons (*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. *Créer un greffon.* Créer un projet *Plug-in Project* appelé `fr.n7.eclipse.plugin.exemple`. Sur le deuxième écran de l'assistant (après *Next*), vérifier que l'option « *This plug-in will make contributions to the UI* » est cochée. Sur le troisième écran, cocher « *Create a plug-in using one of the templates* » et sélectionner « *Hello, World Command* ». L'assistant créera un projet de greffon qui ajoute un menu appelé « *Sample Menu* » à la barre de menu d'Eclipse et un bouton dans la barre d'outils ; les deux déclenchent la même action « *Sample Action* » qui affiche un message dans une boîte de dialogue. Le dernier écran permet de définir ce message. Faire *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.

1.2. *Tester le greffon.* Pour tester le greffon, Eclipse permet de lancer un deuxième Eclipse, dit Eclipse de déploiement, depuis le premier Eclipse : sélectionner le projet de greffon [1], cliquer à droite et sélectionner *Run as... / Eclipse Application*.

L'Eclipse de déploiement se lance. Il a accès aux projets du premier Eclipse, donc à notre greffon : *Sample Menu* apparaît dans la barre de menus ainsi qu'un bouton *Say hello world* (icône d'Eclipse) dans la barre d'outils. Les utiliser : le message apparaît dans une boîte de dialogue.

Laisser l'eclipse de déploiement ouvert.

1.3. *Modifier le greffon.* Pour modifier le greffon, il faut modifier son projet dans le premier Eclipse, puis redémarrer l'eclipse de déploiement pour qu'il prenne en compte ces modifications.

1.3.1. Dans le premier eclipse, modifier le message "Hello, Eclipse world" qui est dans la classe `SampleHandler.java` pour qu'il devienne : "Hey, Eclipse folks.". Sauver le fichier.

1.3.2. Sur l'eclipse de déploiement, cliquer sur l'icone ou activer le Sample Menu. C'est toujours l'ancien message qui s'affiche.

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

Sur l'eclipse de déploiement faire *File / Restart*. L'eclipse se relance. Après redémarrage, il a accès à la nouvelle version des greffons. On peut le constater en activant la Sample Menu : le nouveau message apparaît.
1.4. L'Eclipse de déploiement peut maintenant être quitté.

2 Création 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*[2] 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 droit sur la racine de *SimplePDL.genmodel* (dans l'éditeur arborescent) et déclencher l'action *Generate Model Code*. Les classes Java correspondant au métamodèle SimplePDL sont 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.* Comme précédemment, effectuer les actions de génération : *Generate Edit Code* puis *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 lancer un nouvel Eclipse, dit Eclipse de déploiement car il est lancé à partir du premier Eclipse et a accès aux greffons des projets de ce premier Eclipse.
3.1. *Lancer un Eclipse de déploiement.* Sélectionner le projet *fr.n7.simplePDL* (ou tout autre projet de type greffon), faire un clic droit, sélectionner *Run As / Run...*, puis *Eclipse Application*.

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

Une nouvelle instance d'Eclipse se lance. Si l'Eclipse de déploiement tournait déjà, il faut certainement faire un *File > Restart* pour qu'ils prennent en compte les nouveaux greffons liés aux projets EMF.

3.2. *Créer un projet.* Dans le premier Eclipse, nous commençons, comme toujours, par créer un projet (*File / New / Project*, puis *General / Project*) que l'on peut appeler *fr.n7.simplepdl.exemples*. Nous créons ce projet dans le premier Eclipse pour que tous les projets soient au même endroit. Cependant, ce projet nécessite les greffons associés à SimplePDL. Nous allons donc l'importer dans l'Eclipse de déploiement : dans le *Project Explorer*, faire clic droit *Import / General / Existing Projects into Workspace*, faire *Browse* pour sélectionner le workspace du premier eclipse (son nom s'affiche dans le bandeau de la fenêtre). Tous les projets sont marqués à être importer. Les délectionner (Deselect All) et ne conserver que `fr.n7.simplepdl.exemples`.

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

Les modifications sur l'éditeur devront être faites dans le premier Eclipse. Pour utiliser l'éditeur modifié, il faudra passer par l'Eclipse de déploiement qu'il faudra redémarrer (*File > Restart*) pour qu'il prenne en compte les dernières modifications.

4.1. Dans le projet `fr.n7.simplepdl.edit`, remplacer le code de la méthode `getText(Object)` de la classe `WorkSequenceItemProvider` (dans le dossier `src`) par :

```
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. Constater que la *WorkSequence* fait bien apparaître le nom de l'activité précédente et le nom de l'activité 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 `WorkSequenceItemProvider`. Y ajouter les **`case`** suivants.

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

4.4. Définir l'activité suivante de la deuxième *WorkSequence* : 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.

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.

Pour cette transformation, on dépend des deux métamodèles (processus et réseau de Petri). Quand on utilisera le métamodèle des réseaux de Petri des erreurs vont apparaître. Si on choisit *Fix project setup* dans les actions correctives proposées par Eclipse, les dépendances du projet devraient être correctement mises à jour.