

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 greffon¹, 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.

L'Eclipse de déploiement peut maintenant être quitté.

Exercice 2 : Définir et utiliser un « update site »

Plutôt que de lancer l'Eclipse de déploiement, on peut déployer des greffons sur un site de mise à jour (*update site*) après les avoir regroupés dans des caractéristiques (*feature*).

Après chaque modification des greffons, il faut reconstruire les caractéristiques, redéployer les caractéristiques, et mettre à jour les ateliers Eclipse exploitant ces caractéristiques. Eclipse exploite des étiquettes temporelles pour identifier les mises à jour en plus des numéros de version des greffons et des caractéristiques.

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

2.1. Construire une caractéristique Eclipse.

1. Créer un projet *Feature Project* appelé `fr.n7.eclipse.feature.exemple` dont la caractéristique s'appellera *N7 Eclipse Exemple Feature (Feature Name)*. Ne pas cliquer sur le bouton *Finish* mais passer à l'étape suivante (bouton *Next*) pour ajouter les greffons qui composent la caractéristique.
2. Sélectionner le greffon développé précédemment (`fr.n7.eclipse.plugin.exemple`) en tapant « n7 » dans la zone de saisie pour filtrer les greffons.
3. Faire *Finish*.

2.2. Construite un site de déploiement Eclipse.

1. Créer un projet *Update Site Project* appelé `fr.n7.eclipse.update_site.exemple`.
2. Créer une catégorie (*New category*) dont le nom est *N7 exemple (ID et Name)*.
3. Y ajouter (*Add feature*) la caractéristique créée précédemment `fr.n7.eclipse.feature.exemple`.
4. Synchroniser (*Synchronize*) les caractéristiques (à faire à chaque modification des greffons ou de leurs dépendances).
5. Construire l'intégralité du site (*Build All*).
6. Observer dans le projet l'apparition de plusieurs fichiers archives JAR qui contiennent les greffons et caractéristiques contenues dans le site de déploiement.

2.3. Déployer le site localement et installer la caractéristique.

1. Sélectionner le projet contenant le site et lancer la procédure d'export en tant que Caractéristique Déployable (*Deployable Features*).
2. Dans le formulaire suivant, sélectionner la caractéristique à déployer.
3. Choisir un dossier, par exemple `$HOME/nosave/update-site-gls`, et terminer le déploiement.
4. Aller dans le menu Aide d'Eclipse, choisir l'entrée *Installation de Nouveau Logiciel...*
5. Ajouter le dossier local comme source de logiciel. Décocher le groupement des éléments par catégorie (*Group items by category*). La caractéristique *N7 Eclipse Exemple Feature* doit apparaître, là sélectionner et l'installer (plusieurs *Next*, accepter la licence, *Finish*, accepter d'installer un greffon non signé).
6. L'assistant proposer de redémarrer Eclipse, l'accepter et constater que le menu *Sample Menu* est de nouveau disponible.

2.4. Mettre à jour une caractéristique.

1. Modifier le greffon en changeant le message affiché par le menu.
2. Régénérer la caractéristique
3. Redéployer le site
4. Dans le menu d'aide d'Eclipse, faire une recherche de mise à jour, puis faire la mise à jour et observer le changement de message.

2.5. Supprimer la caractéristique déployée.

1. Consulter la liste des caractéristiques déployées : choisir « *About Eclipse IDE* » dans le menu *Help*²
2. Sélectionner la caractéristique *N7 Eclipse Example Feature* (utiliser le sélecteur en tapant N7 ou Exemple).
3. La désinstaller.
4. Constater que le menu a disparu après le redémarrage d'Eclipse.

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 3 : Engendrer le code Java et un éditeur arborescent

Commençons par générer les classes Java nécessaires à la création de notre éditeur.

3.1. 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.

3.2. Configurer la génération. Ouvrir le fichier *genmodel* (*SimplePDL.genmodel*). Un éditeur arborescent s'ouvre. Sélectionner sa racine. Dans la 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.

3.3. 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 paquets *simplepdl* et *simplepdl.impl*. Le premier contient des interfaces, le second des réalisations de ces interfaces. Regarder aussi le contenu de *SimplepdlFactory*.

3.4. 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*.

3.5. 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 4 : Utiliser l'éditeur arborescent

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

2. Ou menu *Eclipse* sous MacOSX.

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

4.1. Déployer les greffons. Déployer les trois greffons `fr.n7.simplepdl`, `fr.n7.simplepdl.edit` et `fr.n7.simplepdl.editor` en suivant les indications de l'exercice 2. On utilisera une caractéristique appelée *N7 GLS Feature*.

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

4.3. 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 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*.

4.4. 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 5 : 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.

5.1. 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.

5.2. Redéployer les greffons. Il s'agit de mettre à jour la caractéristique *N7 GLS Feature*, l'exporter et mettre à jour Eclipse. Constater que la *WorkSequence* fait bien apparaître le nom des activité précédente et 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?

5.3. Regarder le code de la méthode `notifyChanged(Notification)` de `WorkSequenceItemProvider`. Y ajouter les **case** suivants. Redéployer les greffons.

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

5.4. 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 6 : Manipuler des modèles en Java

Dans cet exercice, nous allons manipuler des modèles EMF à partir de code Java.

6.1. *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*.

6.2. *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.

6.3. *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 7 : 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.