Introduction à MDA par la pratique

par Pierre Parrend

Date de publication : 08/12/2006

Dernière mise à jour :

Introduction au développement orienté modèle (MDA - Model Driven Architecture) par la pratique. Le langage utilisé est Java. Vous pouvez vous référez également à la présentation théorique de MDA.

I - Introduction

I-A - A qui s'adresse ce tutoriel ?

Ce tutoriel a pour vocation d'être une introduction à MDA (Model Driven Architecture) par la pratique. Il vise à donner une bonne connaissance des outils utilisés dans un processus MDA, afin de permettre de mieux comprendre les problèmatiques de conception.

Toute personne intéressée par MDA trouvera donc un point d'entrée dans cette méthode nouvelle de conception de logiciels : développeurs, étudiants, mais aussi concepteurs qui ont besoin de savoir quels sont les outils disponibles en appui des concepts.

Ce tutoriel n'aborde pas en détail les principes de MDA. Vous trouverez ici une présentation de MDA.

Ce tutoriel n'aborde pas ces problèmatiques de conception. Ni les aspects, nombreux, spécifiques à tel ou tel domaine d'application. Et par conséquent passe sous silence des étapes telles que la conception des modèles, la création de langage spécifique à un domaine.

Il s'agit de montrer ce qui est possible de faire avec MDA, et certains des outils existants.

Le développeur qui a besoin de ces outils pourra travailler en exploitant les différentes transformations proposées.

Le concepteur qui voudra mettre en place un processus MDA pour des applications de taille conséquente pourra utiliser les informations présentes comme point de départ, qui sera utilement complété par l'approfondissement du processus de conception, tel qu'il est présenté par exemple sur le site web du 'Model-Driven Software Development' :

http://www.mdsd.info

I-B - Organisation du tutoriel

Le tutoriel est organisé comme suit :

Comment passer d'un modèle UML à un code exécutable ?

1. Génération de code par la plate-forme Eclipse. Le modèle est alors exprimé sous forme de Java annoté. Le code peut être complété parallèlement au modèle, à l'intérieur de Eclipse : la regénération ne provoque pas la perte des modifications manuelles.
2. Génération de code par la plate-forme Poséidon. Le modèle UML est représenté sous forme graphique. Par contre, la génération de code écrase les versions préexistantes.

I-C - Remerciements

Merci à loka pour sa relecture et ses remarques.

II - Du modèle au code

La première partie du tutoriel est une introduction à la génération automatique de code à partir d'un modèle. Nous nous plaçons dans le cadre de la plate-forme Eclipse.

Les étapes de la manipulation sont :

1 La première étape consiste à créer un modèle UML.
2 Ce diagramme UML doit être traduit en Java annoté pour être compris par Eclipse. Nous verrons comment faire la traduction.
3 Ensuite, un modèle de type MOF est généré, à partir de Java Annoté. Le format eCore est l'implémentation Eclipse de la spécification MOF.
4 Le modèle existant peut être enrichi.
5 Le code java est généré à partir des objets eCore enrichis.
6 Une fois les classes générées, il est souvent indispensable d'implémenter les fonctionnalités de ces classes. Les méthodes et variables peuvent être complétées, mais le modèle lui-même ne peut pas être modifié (afin de permettre la regénération de code sans perte de données).
7 Si l'on a besoin de regénérer le code - souvent après avoir enrichi le modèle - les ajouts manuels sont conservés.

Cette méthode permet donc un développement incrémental.

II-A - Installer Eclipse

Installation

Téléchargez la plate-forme de développement intégré Eclipse depuis l'adresse suivante : http://www.eclipse.org/downloads/index.php

Installez-la.

Prise en main

Vous pouvez créer un programme 'Hello World' pour vous aclimater à Eclipse.

1 Créez un nouveau projet Java (Fichier > Nouveau > Autres; Sélectionnez Projet Java; suivez le wizard),
2 Créez une classe Java, avec une méthode main() (Fichier > Nouveau > Classe Java; suivez le Wizard),
3 Modifier le code pour afficher la phrase voulue,
4 Exécutez votre programme (Exécutez > Exécutez en tant que > Application Java).

II-B - Installer EMF

Téléchargez et installez le framework EMF (Eclipse Modeling Framework) depuis l'adresse suivante : http://download.eclipse.org/tools/emf/scripts/downloads.php

Une façon plus simple de l'installer est de passer par la fonctionnalité d'update d'Eclipse : Help | Software | Update | Find and install ... | search for new features to install | New Remote Site, et de renseigner avec http://download.eclipse.org/tools/emf/updates.

EMF doit être installé avec SDO (Service Data Objects) et XSD (XML Schema Infoset Model).

SDO permet d'unifier l'accès à des données hétérogènes, par exemple stockés sous forme XML, en Base de Données Relationnelle, dans des Services Web, par les applications.

XSD est une librairie qui permet de manipuler, créer et modifier des schémas XML.

II-C - Mon premier programme avec EMF

II-C-1 - Comment ça marche ?

Pour créer un modèle EMF (au format Ecore), plusieurs formats de fichiers sont supportés :

1 Java Annoté
2 XML Schéma
3 Format Rational Rose

Java annoté permet de mettre un place une solution légère, qui ne nécessite pas d'outil extérieur, et qui ne restreint pas les possibilités de modélisation, contrairement à XML schéma. C'est donc la solution choisie.

Les étapes de la création d'un programme avec EMF sont les suivantes. Vous trouverez le détail des manipulations dans l'exemple du paragraph suivant :

1 Création d'un projet EMF : Fichier > Nouveau > Autres > Eclipse Modeling Framework > EMF Project,
2 Création du modèle, sous forme de Java Annoté, dans le nouveau projet (voir exemple),
3 Création d'un modèle Ecore dans votre projet Eclipse: Fichier > Nouveau > Autres > Eclipse Modeling Framework > EMF Model,
4 Utilisez l'option 'créer un modèle à partir de Java Annoté', et veillez à la création du fichier .ecore, qui contient le modèle (plus exactement le métamodèle) de l'application,
5 Si besoin, vous pouvez enrichir le modèle (fichier .ecore),
6 Dans le fichier de génération (.genmodel), vous pouvez générer le code de votre programme, ainsi qu'un éditeur de modèle (Generate Edit Code, Generate Editor Code),
7 Exécution de l'éditeur de modèle (extension du modèle créé - création du modèle de l'application à partir du méta-modèle Ecore).

Pour créer vos propres programmes, vous pouvez vous référer au document suivant, qui établit la correspondance entre UML et Java Annoté :

Manuel de traduction UML vers Java Annoté

II-C-2 - L'exemple

Un exemple intéressant d'introduction à EMF se trouve à l'adresse suivante : http://www.eclipse.org/emf/docs/1.x/tutorials/clibmod/clibmod_emf1.1.html

Complément : Enrichissement du modèle

Vous pouvez compléter votre modèle en créant, dans l'éditeur de modèle, les classes suivantes :

1 Journaliste, héritant de Auteur, avec un attribut journal (le nom du journal pour lequel il travaille), une méthode ecrireArticle() (pour simuler l'écriture d'un article quotidien),
2 Romancier, héritant de Auteur, avec des méthodes commencerRoman(), et finirRoman(), ainsi qu'un attribut

booléen romanEnCours.

L'enrichissement du modèle permet de réaliser la transition entre le méta-modèle de l'application (réalisé ici par le biais de Java annoté) et le modèle de notre application, qui prend en compte des contraintes spécifiques.

Création d'un programme exécutable

Vous pouvez ensuite générer le code correspondant, puis compléter les méthodes par un code adéquat. Pour les besoins du tutoriel, il peut être pratique d'effectuer des affichages à l'aide de System.out.println(), afin de suivre le déroulement du programme.

L'étape ultime consiste à créer une nouvelle classe maSimulation, qui permet de jouer des scénarios sans nécessiter la mise en place d'une interface graphique. Elle a pour rôle de simuler les manipulations d'un utilisateur, par exemple dans le cadre d'une application web.

1 Création d'un bibliothèque,
2 Création de 3 ouvrages de votre choix, ainsi que de leurs auteurs (un des auteurs doit être un Journaliste, un autre un Romancier),
3 Introduction de ces ouvrages dans la bibliothèque,
4 Ecriture d'un roman : un romancier commence un roman, le finit (par appel des méthodes correspondantes). Dans la mehode finirRoman(), le Roman doit être introduit dans la bibliothèque.

Nous avons vu comment il est possible de créer des modèles, de les enrichir, et de créer des programmes exécutables à partir de ces modèles, en utilisant la plate-forme Eclipse, et son framework EMF (Eclipse Modeling Framework). Un avantage de cette approche est que la génération de code est faite en respectant les compléments ajoutés manuellement.

Il est donc possible d'utiliser EMF pour un développement incrémental, et donc pour réaliser des applications de taille conséquente.

III - Une autre approche de la création de modèles : les diagrammes UML

L'usage de la plate-forme Eclipse permet une manipulation aisée, ainsi que des transformations comme nous allons le voir dans la partie suivante. Toutefois, la création de modèle par le biais de Java annoté n'est pas la plus intuitive.

Il est possible de créer des modèles EMF pour Eclipse à partir de l'outil Rational Rose, encore faut-il en disposer. Pour palier à ce manque, nous allons utiliser Poséidon, qui est un outil puissant et disponible - ce qui n'est pas la moindre de ses qualités - gratuitement dans une version Communauté. Il permet de mettre à profit l'un des aspects qui a fait le succès de UML : les représentations graphiques des différents diagrammes.

III-A - Installer Poseidon

L'usage de l'outil Poseidon est assez intuitif. Suivez les étapes pour créer votre premier exemple.

1 Téléchargez Poseidon, Community Edition, depuis http://www.gentleware.com/,
2 Créez un modèle UML,
3 Ajoutez les commentaires Javadoc dans le modèle UML,
4 Sauvez votre modèle,
5 Exportez en tant que Classe Java (seul les diagrammes de classes sont exportés, les autres existent uniquement à fin de documentation),
6 Exportez sous format XMI,
7 Exportez en tant que graphique (si besoin),
8 Lancez la plate-forme Eclipse, et importez le fichier XMI créé.

III-B - Mon premier programme avec Poseidon

Nous allons voir comment il est possible de créer, de manière simple et intruitive, un programme à l'aide de la version Communauté de la plate-forme Poséidon.

La modélisation UML

Créez les exemple de modèles UML suivants, et générez le code correspondant :

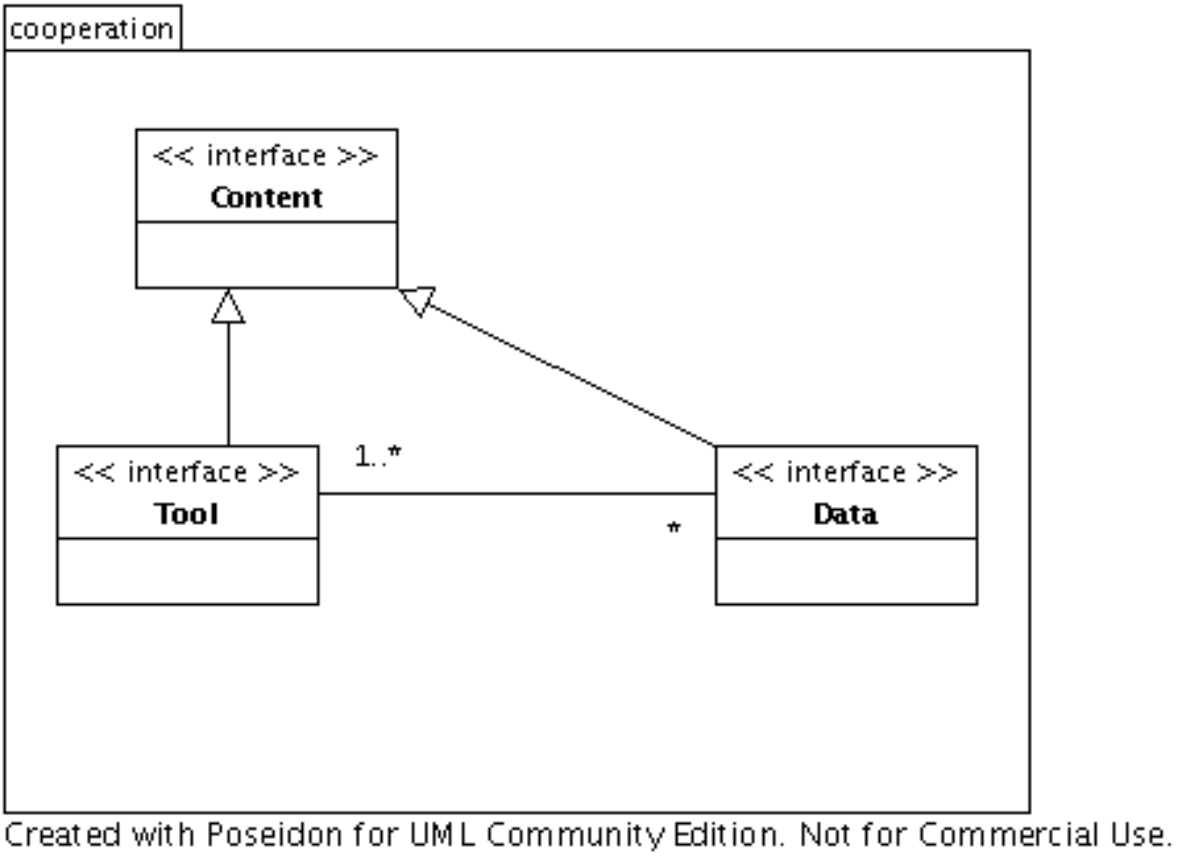
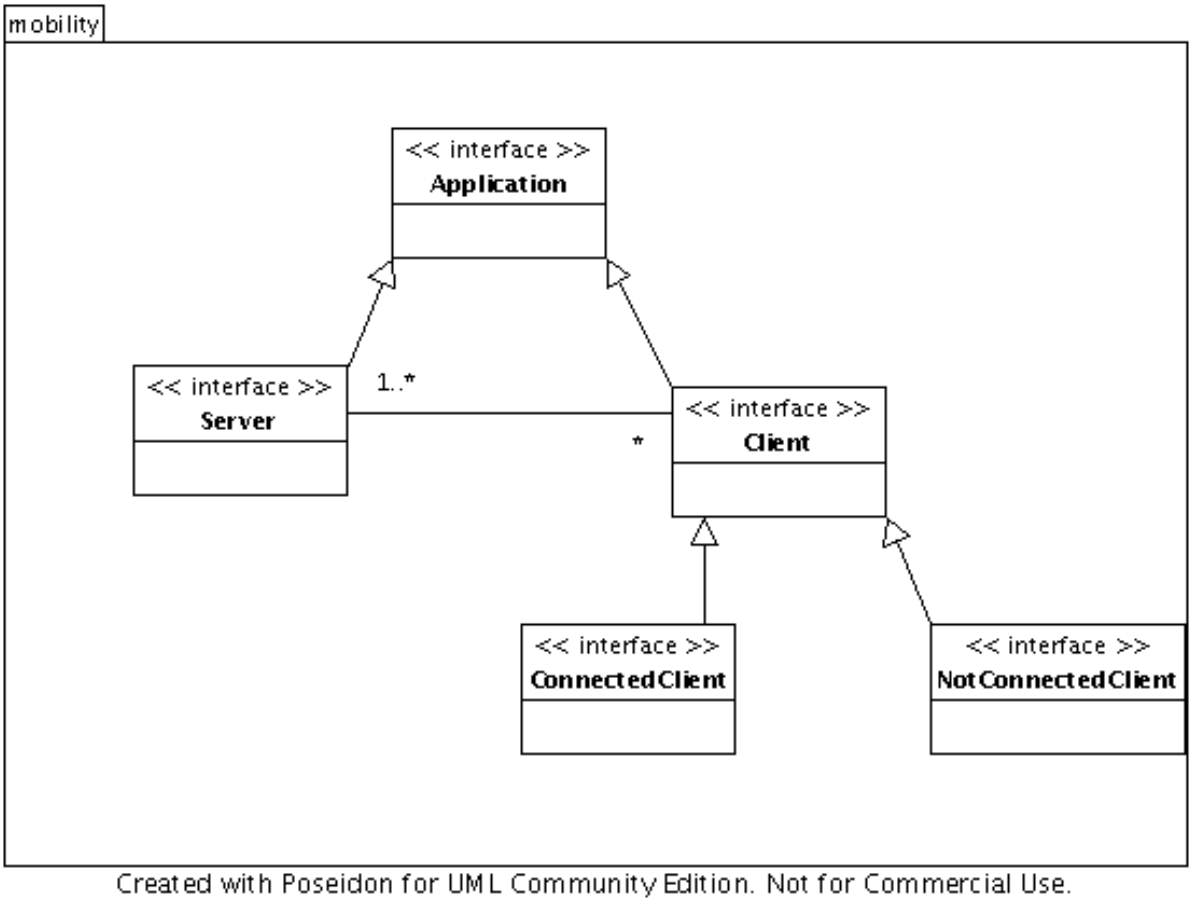


Diagramme de Classe pour support de coopération



Modèle de Classe pour support de mobilité

Réalisation d'un modèle complet

Réaliser les diagrammes UML qui vous permettront de créer un programme fonctionnel.

Il s'agit de simuler deux clients, un connecté (de manière permanente), et un non connecté (ou connecté de manière intermitente), qui réalise les actions suivantes :

1 Ils accèdent par une méthode seConnecter() un serveur comportant des outils (tool) et des données (data),
2 Ils accèdent par une méthode travail() à un outil du serveur qui effectue un manipulation (quelconque) sur les données.
3 L'outil peut être implémenté comme un singleton, de telle sorte que les deux clients travaillent sur les mêmes outils et donc les mêmes données. Ceci permet la mise en place de travail coopératif.

1 Diagramme de cas d'usage
2 Diagramme de séquence, qui permettent de mettre en évidence l'enchaînement des appels de méthodes
3 Complétion du diagramme de classe précédement réalisé

Réalisation d'un programme exécutable

Pour aboutir à un programme exécutable à partir des modèles réalisés dans la partie précédente, il reste à faire :

1 Effectuez la génération de code pour votre dernier diagramme.
2 Il faut ensuite écrire le code fonctionnel des méthodes présentes.
3 Pour valider le bon fonctionnement de votre programme, écrivez une classe maSimulation, qui crée deux clients (un connecté et un non connecté), se connecte, et effectue un travail().

Votre modélisation ayant été faite de manière précise, la structure des classes (méthodes, attributs) n'a pas besoin d'être modifié, il suffit d'implémenter les méthodes.

Bilan

Nous avons vu comment il est possible de créer des modèles, et de créer des programmes exécutables à partir de ces modèles, en utilisant la plate-forme Poséidon. L'interface graphique permet d'efectuer la modélisation de manière intuitive et communicable.

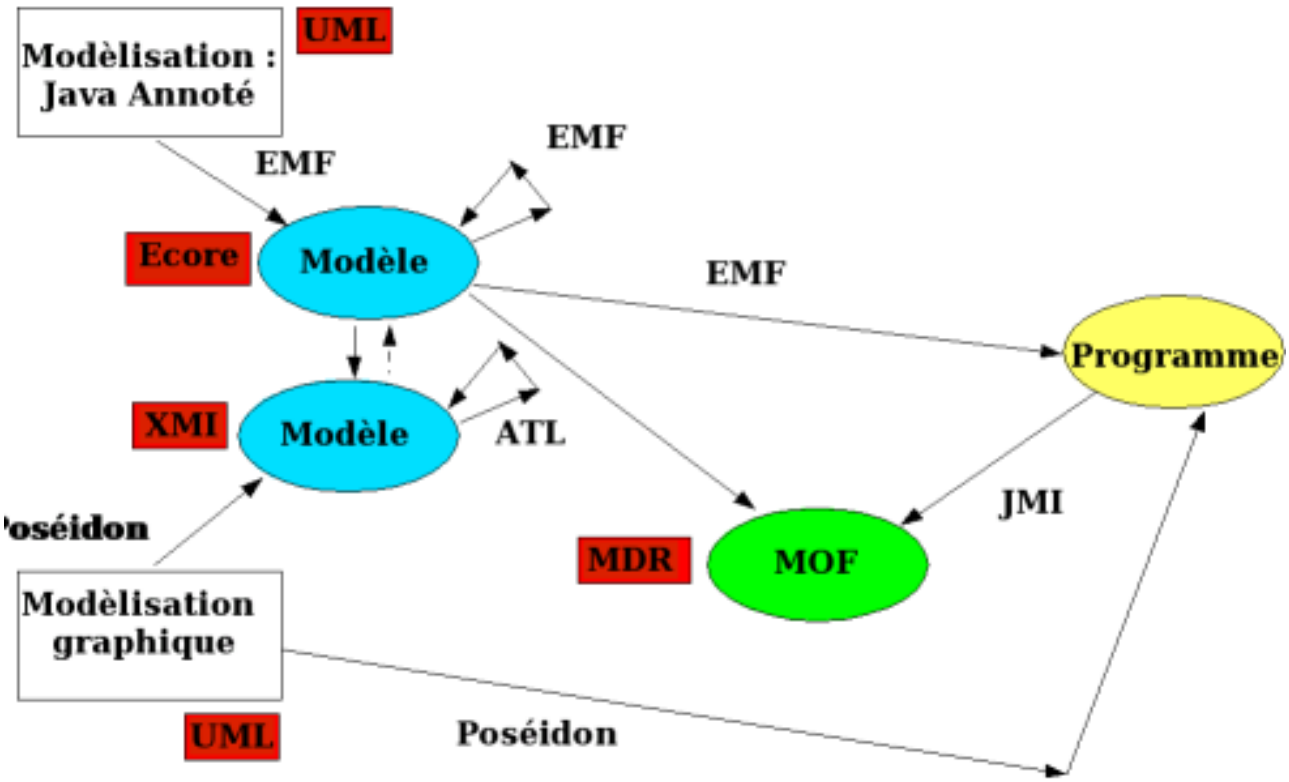
L'inconvénient de cette approche est que la génération de code écrase le code préexistant, le développement incrémental est donc plus difficile.

Il existe une version commerciale de Poséidon disponible sous forme de plugin Eclipse. L'intégration entre les deux outils est donc possible. En utilisant la version Communauté, il est nécessaire pour passer de l'un à l'autre d'utiliser le format XMI, moins aisément manipulable.

IV - Bilan

IV-A - Les transformations réalisées

Le schéma suivant présente un récapitulatif des transformations réalisées au cours de ce tutoriel.



IV-B - Les formats

Les formats de données sont indiqués sur fond rouge sur le schéma. Ce sont :

1. **UML**, représentation des modèles de manière graphique, avec exploitation directe (Poséidon), ou après traduction en Java annoté (EMF),
2. **XMI**, format standard défini par l'OMG, tous les modèles sont disponibles en format XMI, mais la représentation sous forme de fichier texte fait que, pour la manipulation, les formats graphiques UML et les modèles Ecore d'EMF sont préférés,
3. **Ecore**, c'est l'implémentation de MOF par Eclipse. La création de modèle est possible à partir de nombreux formats (Java Annoté, XML schema, Rational Rose, etc.), l'enrichissement est possible de manière simple par la représentation hiérarchique des modèles. Un langage de transformation de modèle peut également être défini.
4. **MDR**, Meta Data Repository, c'est l'implémentation d'une base de données pour modèles compatibles MOF

de Sun. Elle fait partie de l'outil NetBeans.

IV-C - Les outils

Les outils utilisés sont :

1. **EMF** (Eclipse Modeling Framework), l'environnement de la plate-forme Eclipse dédié au MDA,
2. **ATL** (Atlas Transformation Language), disponible sous forme de plugg-in pour Eclipse, qui permet de réaliser des transformations quelconques sur des données au format XMI,
3. **Poséidon**, qui existe également sous forme de plugin Eclipse (en version payante), mais qui est utilisé ici dans sa version Communauté, en libre accès.
4. **JMI**, Java Metadata Interface, permet d'accéder à une base de données de métamodèle et de les manipuler

- **Eclipse** : http://www.eclipse.org/downloads/index.php
- **EMF** : http://download.eclipse.org/tools/emf/scripts/downloads.php
- **Poseidon** : http://www.gentleware.com/products/download.php4