FACULTÉ DES SCIENCES DHAR EL MAHRAZ
UNIVERSITÉ SIDI MOHAMED BEN ABDELLAH

MQL فاس

Ingénierie de l'objet Java

Etude sur :
Le standard xmi
Le modèle mof (meta object facility)
Emf (eclipse modeling framework)

Table des matières

XMI définit un format d’échange standard entre ateliers logiciels

Impossible de parler de XMI sans évoquer les deux autres standards qui lui sont intimement liés : UML et MOF. En quelques mots, UML (Unified Modeling Language) est un langage graphique utilisé pour la conception orientée objet, et MOF (Meta Object Facility) utilise un sous-ensemble de UML pour décrire les objets manipulés par les outils de conception. Enfin, XMI (XML Metadata Interchange) indique comment les modèles MOF peuvent être traduits en XML. Le but de ces standards est de permettre à des ateliers logiciels d’explorer et d’échanger les définitions des structures de données, leurs propriétés, les relations les unissant, etc.Avant de construire une voiture, par exemple, il faut en créer les plans détaillés. Le moindre élément du véhicule, carrosserie, moteur, châssis, roues et jusqu’au plus petit boulon, doit être décrit, ainsi que les relations entre chaque élément : c’est le plan de montage. Des règles précises normalisent la façon dont de tels plans sont représentés sur papier. Si l’on souhaite créer, non plus une voiture, mais un logiciel ou une base de données, UML est le langage graphique normalisé permettant de décrire les plans d’un logiciel, c’est-à-dire les structures des données et les relations qui les lient. On parle de modèles UML. Lesquels sont rarement dessinés à la main, mais plutôt à l’aide d’outils informatiques (comme Rational Rose). En poursuivant l’analogie industrielle, MOF (Meta Object Facility) serait un ensemble de règles de réalisation des plans de la voiture : épaisseur des traits, façon de placer les légendes, etc. On doit la création de cette architecture de métaobjets MOF à l’OMG (Object Management Group), soutenu par des organismes comme IBM, Oracle, Rational ou Unisys.

L'extension XMI (XML Métadata Interchange) [XMI2003] propose un format standard permettant d'exprimer les concepts de la modélisation objet. Pour transcrire les structures de graphe, très répandues dans ce domaine, on a recours à un mécanisme d'identifiants et de références à ces identifiants (car un document XML a les caractéristiques d'un arbre). Ainsi, il est possible d'encoder un modèle UML dans un fichier au format XML [OMG]. Les figures 1.7 et 1.8 présentent l'exemple d'un diagramme de classes élémentaire comportant deux classes et une association, puis sa transcription au format XMI.

Figure - Diagramme de classes UML.

Figure - Transcription au format XMI du diagramme de figure

Le format XMI

Si le format XMI constitue une norme de l'Object Management Group, chaque outil de modélisation UML en fait une interprétation personnelle. Ainsi, certains détails varient (le nom des balises, le choix de placer une information comme attribut ou comme contenu d'un noeud), rendant impossible à l'heure actuelle l'échange de fichiers XMI entre deux outils différents [XML].

Structure générale d'un document

La racine de tout document est une balise *XMI*, ayant comme attributs la version du standard XMI utilisé, ainsi que la date de création du fichier. À l'intérieur de cette balise XMI, on trouve une balise *XMI.header*, permettant d'identifier le modèle UML, et une balise *XMI.content*. Cette dernière contient principalement une balise *Diagram* renfermant toutes les informations liées à l'interface graphique (position des symboles, polices utilisées...), et une balise Model concernant le modèle UML en luimême. C'est au sein de cette balise Model que sont transcrits les différents diagrammes UML[XML].

De la conception objet à XML

XMI est à la frontière de deux mondes : d'une part la conception orientée objet, d'autre part les documents structurés et les langages de tag. Les deux présentent plusieurs points communs. Rien d'étonnant donc à ce que, dans la pratique, les fichiers XML conformes aux DTD XMI soient simples. On peut décrire la structure des métaobjets et leurs relations grâce à trois types seulement d'éléments de base : classes, attributs (champs des objets) et associations (différents types de liens : héritage, inclusions et références). Judicieusement employés dans une DTD, ces éléments traduisent n'importe quel métamodèle MOF, le fichier XML proprement dit contenant le modèle. La notion d'héritage, très importante en conception objet, pose un problème : elle n'existe pas en XML. Dans les DTD, on peut contourner le problème en indiquant explicitement les classes ancêtres (superclasses) et en utilisant le mécanisme standard de substitution de définition. Ce qui contraint toutefois à définir les entités dans un ordre précis dans la DTD. Dans les fichiers XML, les champs hérités doivent être répétés pour chaque objet. Il existe toutefois un format XMI différentiel où, plutôt qu'un modèle complet, le document transmis ne contient que les points différant d'un modèle de référence. La méthode permet des transmissions beaucoup plus compactes.

le modèle mof (meta object facility)

En génie logiciel, et plus particulièrement en architecture dirigée par les modèles, le Meta-Object Facility (MOF) est un standard de l'Object Management Group (OMG) s'intéressant à la représentation des métamodèles et leur manipulation. Le langage MOF est auto-descriptif, c'est-à-dire que la couche la plus abstraite est suffisamment expressive pour se représenter elle-même.

Couches de modélisation

La représentation des métamodèles avec MOF s'appuie le plus souvent sur quatre couches de modélisation1. Chaque couche est une abstraction de la couche inférieure :
M3 est un méta-métamodèle auto-descriptif de MOF ;
M2 est un métamodèle construit selon le standard MOF (par exemple, UML 2.0) ;
M1 est un modèle utilisateur ;
M0 est un objet de la réalité, sujet à une modélisation ;
L'architecture en quatre couches est considérée dans l'approche MDA comme étant suffisamment générale pour un processus de modélisation logicielle. Bien qu'étant la plus fréquente, cette architecture n'est cependant pas la seule. Le standard MOF requiert au minimum deux couches de modélisation et n'impose pas de nombre de couches maximum2. Ainsi, une spécification en deux couches (par exemple, classe (M1) – objet (M0)) est conforme à l'approche MDA bien qu'elle ne définisse pas de méta-métamodèle2.

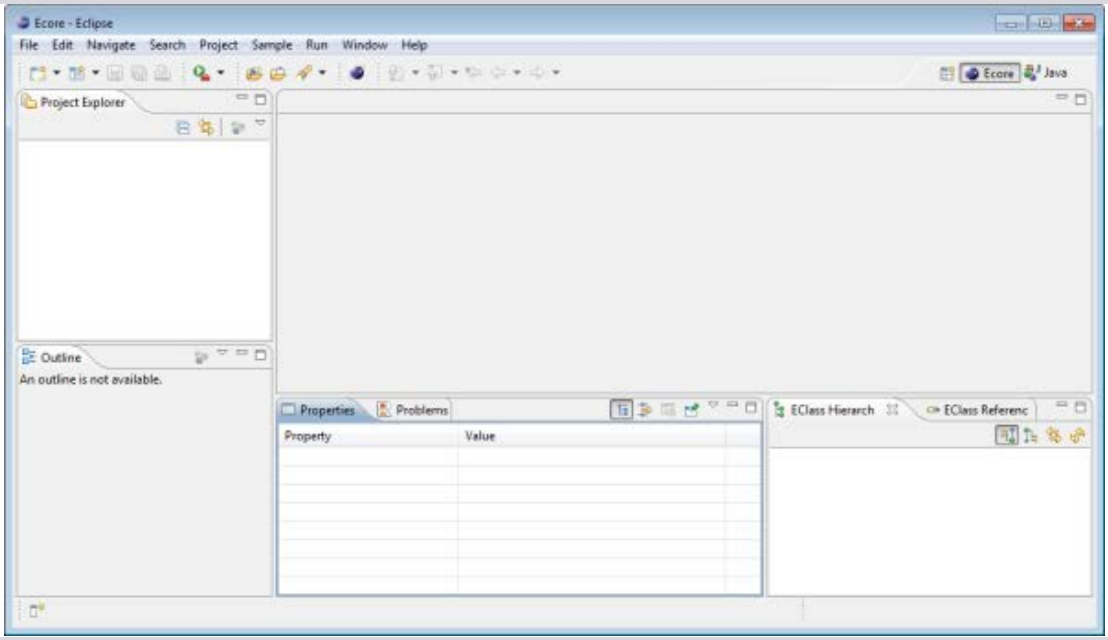
Métamodèles définis par le MOF

Le langage UML est décrit par un métamodèle conforme au MOF. Ainsi un modèle UML peut être sérialisé en XMI. Mais il y a également de nombreux autres métamodèles situés au même niveau que UML. On peut citer par exemple les métamodèles CWM, SPEM, SysML, etc. Le standard QVT, qui définit plusieurs langages de transformations de modèles, est également décrit à l'aide du méta-métamodèle MOF.

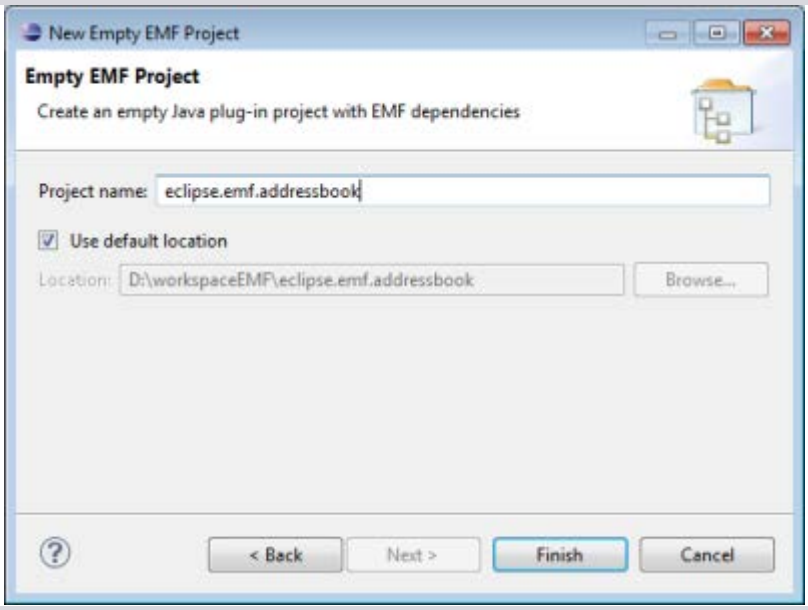
3- Emf

EMF est une structure Java et une fonction de génération de code permettant de construire des outils et d'autres applications basées sur un modèle structuré. Pour ceux d'entre vous qui ont acquis l'idée d'une modélisation orientée objet, EMF vous aide à faire évoluer rapidement vos modèles en un code Java efficace, correct et facilement personnalisable. Pour ceux d'entre vous qui ne seraient pas encore convaincus de la valeur des modèles formels, EMF a bien l'intention de vous offrir les mêmes avantages et un coût d'entrée vraiment très bas.
Relation EMF avec OMG MOF
Ceux d'entre vous qui connaissent le format MOF (Meta Object Facility) du groupe OMG (Object Management Group) doivent se demander quel est le rapport avec EMF. EMF a démarré comme une implémentation de la spécification de format MOF, qui a évolué depuis grâce à l'expérience acquise sur l'implémentation d'un grand nombre d'outils qui l'utilisent. EMF peut être vu comme une implémentation Java hautement efficace d'un sous-ensemble de base de l'API MOF. Toutefois, pour éviter toute confusion, nous appelons Ecore le métamodèle de base à l'aspect MOF.
Dans la proposition actuelle pour MOF 2.0, nous distinguons le sous-ensemble semblable au modèle MOF, appelé EMOF (Essential MOF). Il existe des petites différences essentiellement de désignation entre Ecore et EMOF ; sachez néanmoins qu'EMF peut de manière sûre lire et écrire des sérialisations d'EMOF.
Définition d'un modèle EMF

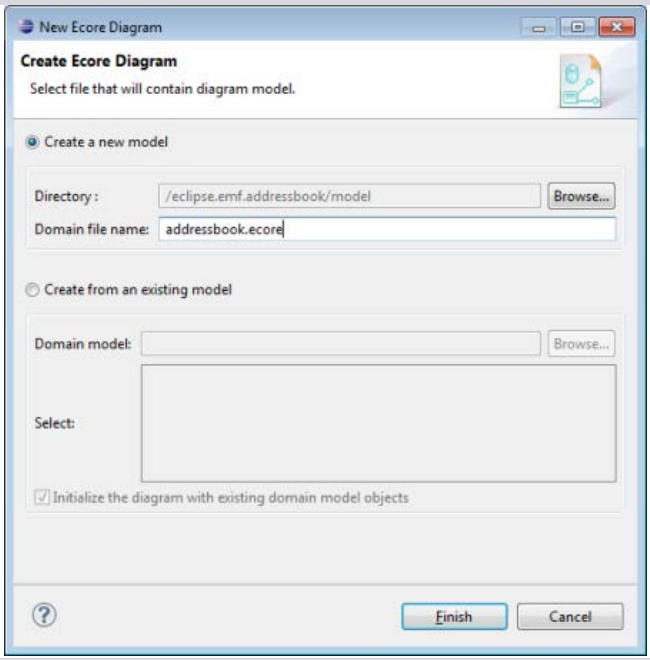
- Démarrer l'environnement de développement Eclipse contenant les plugins de modélisation puis créer un nouveau Workspace (workspaceEMF) afin de disposer d'un répertoire spécifique à la modélisation.
- Pour afficher les vues Eclipse spécifiques à la modélisation EMF, ouvrir la perspective Ecore.



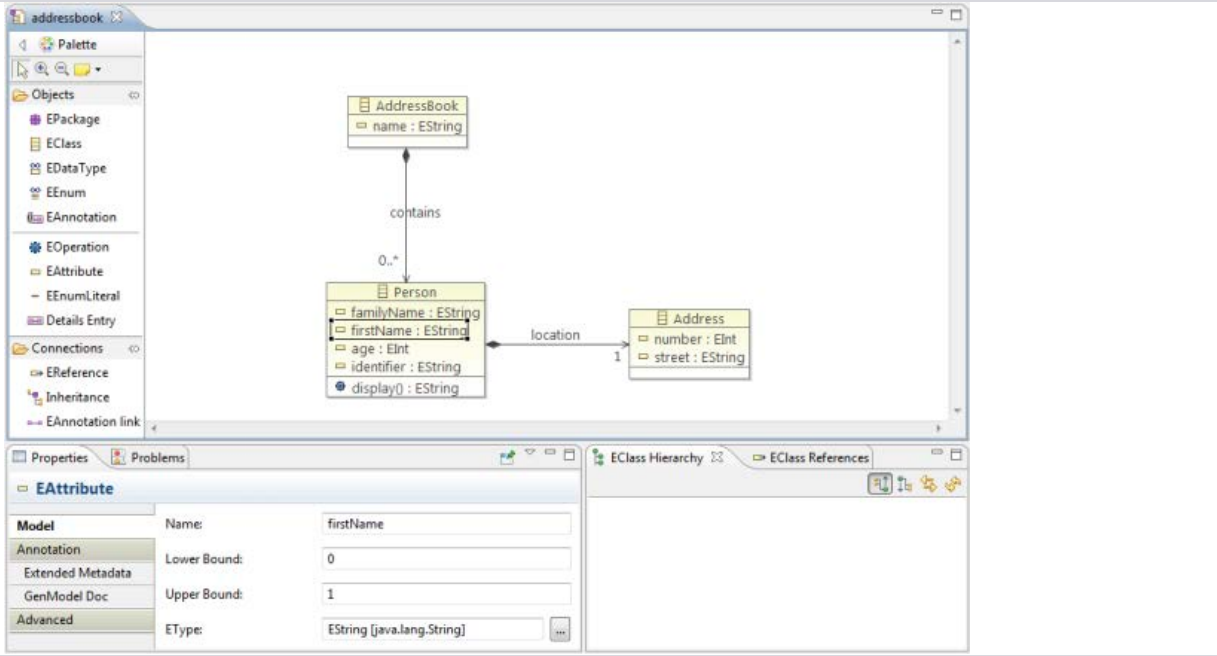
- Créer un nouveau projet EMF vide (File -> New -> Project … -> Eclipse Modeling Framework - > Empty EMF Project) et nommer le projet *eclipse.emf.addressbook*.



- Sélectionner le répertoire *model* et créer un diagramme Ecore (Ecore Diagram). Nommer le fichier ecore *addressbook.ecore*.



- Construire les trois classes, définir tous les attributs et créer les associations entre les classes. Veuillez respecter les contraintes de cardinalités exprimées sur le modèle UML précédent. Aidez-vous de la vue *Properties* afin de spécifier les cardinalités voulues.



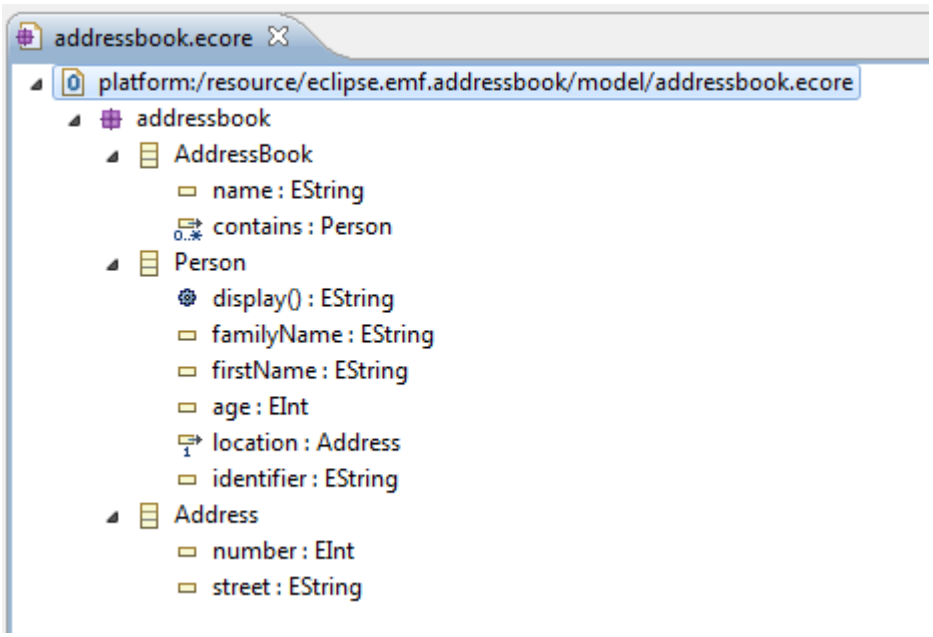
Vous pouvez visualiser votre modèle sous différentes représentations via l'utilisation d'éditeurs adaptés : *OCLinEcore (Ecore) Editor* et *Sample Ecore Model Editor*. Bien entendu, cette liste n'est pas exhaustive et d'autres éditeurs sont également disponibles. Il suffira juste d'installer les plugins adéquates si nécessaires.

addressbook.ecore

```
module _'addressbook.ecore'
import ecore : 'http://www.eclipse.org/emf/2002/Ecore#/';

package addressbook : addressbook = 'http://addressbook/1.0'
{
    class AddressBook
    {
        attribute name : String[?] { ordered };
        property contains : Person[*] { ordered composes };
    }
    class Person
    {
        attribute familyName : String[?] { ordered };
        attribute firstName : String[?] { ordered };
        attribute age : ecore::EInt[?] { ordered };
        property location : Address[1] { ordered composes };
        attribute identifier : String[?] { ordered derived readonly transient volatile };
        operation display() : String[?] { ordered };
    }
    class Address
    {
        attribute number : ecore::EInt[?] { ordered };
        attribute street : String[?] { ordered };
    }
}
```

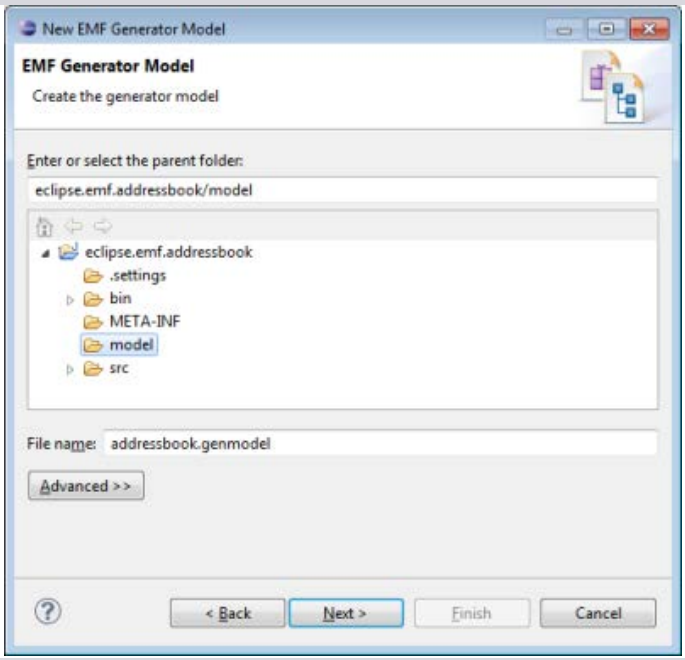
Visualisation du modèle EMF Carnet d'adresses via l'éditeur OCLinEcore



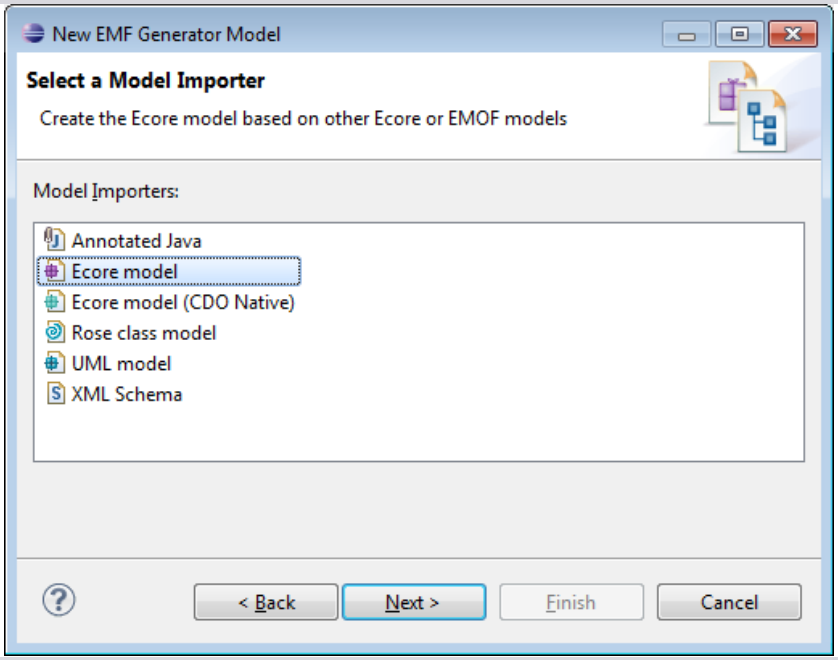
Visualisation du modèle EMF Carnet d'adresses via l'éditeur Sample Ecore Model Editor

- Visualiser finalement votre modèle au format XML, vous remarquerez qu'il s'agit d'un fichier XMI dont les données correspondent à des instances du métamodèle Ecore. Nous reviendrons sur cette notion dans les prochaines sections.

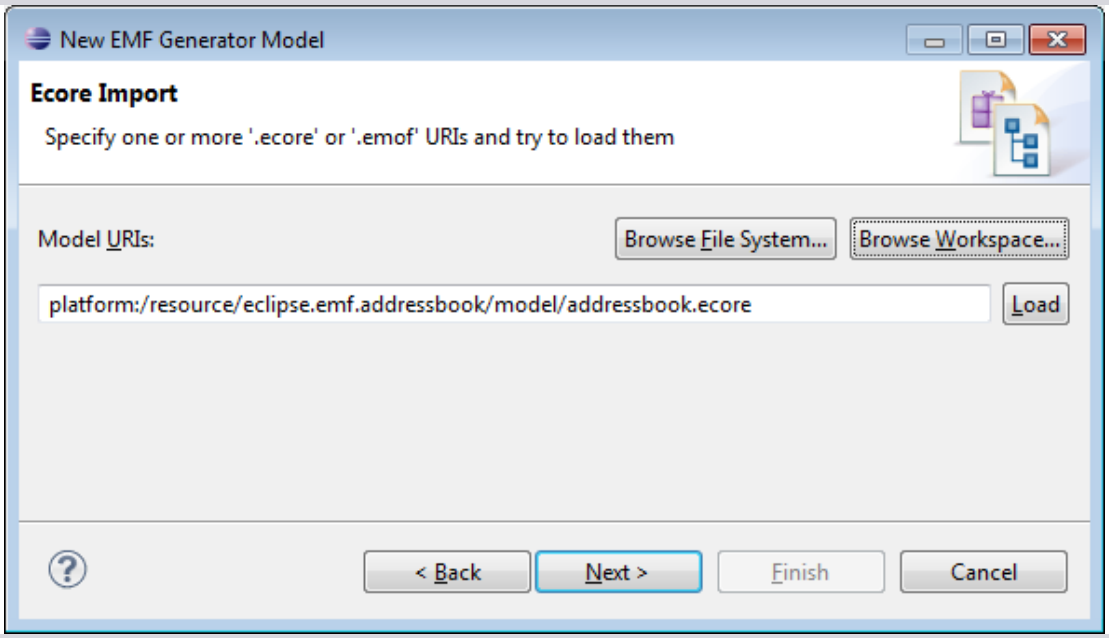
- Créer un modèle de génération (New -> Other… -> Eclipse Modeling Framework -> EMF Generator Model), sélectionner ensuite le répertoire *model* du projet *eclipse.emf.addressbook* puis nommer le fichier *addressbook.genmodel*.



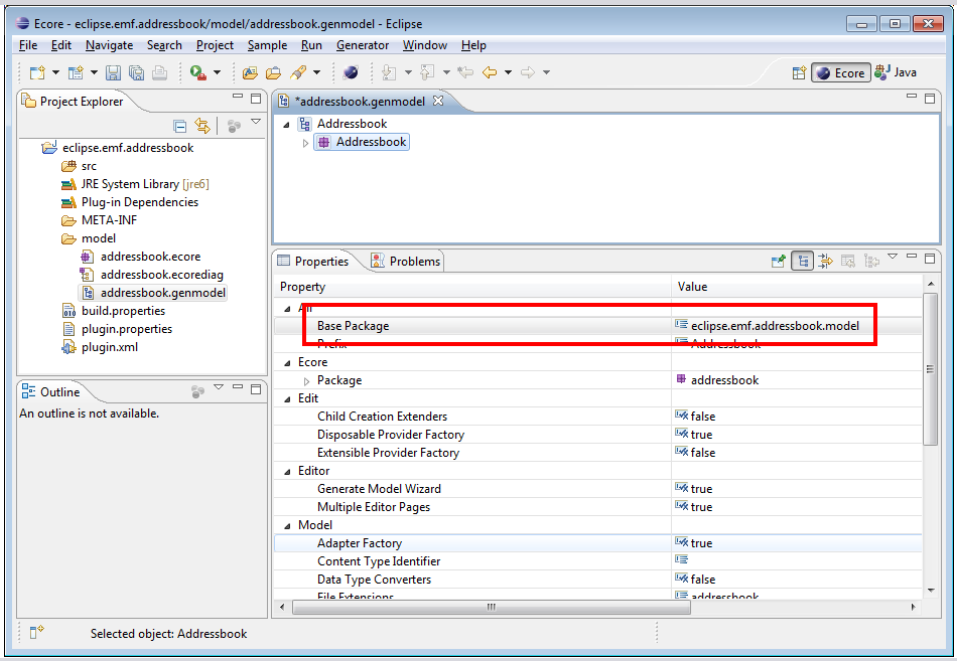
- Sélectionner ensuite *Ecore model* comme type de modèle utilisé pour créer ce modèle de génération.



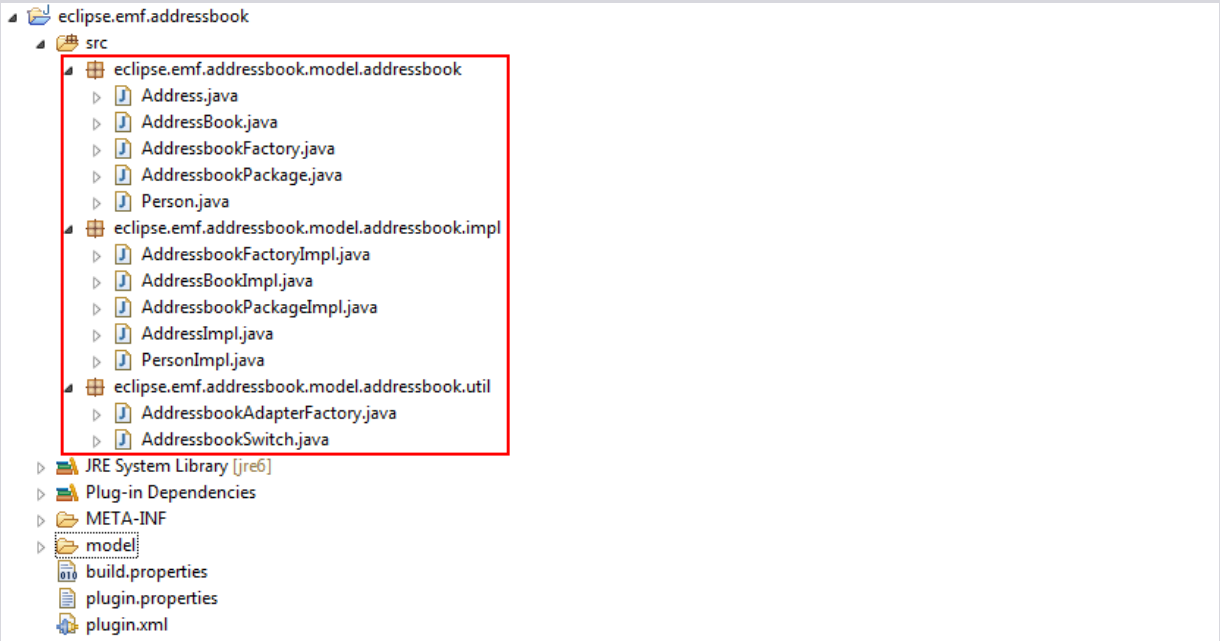
- Sélectionner enfin le fichier *addressbook.ecore* (à partir de la navigation du Workspace courant Browse Workspace…) puis terminer.



- Modifier le contenu du fichier genmodel pour que le package de génération soit *eclipse.emf.addressbook.model* (propriétés : Base Package). Utiliser pour cela la vue Properties en modifiant l'attribut *Base Package*.



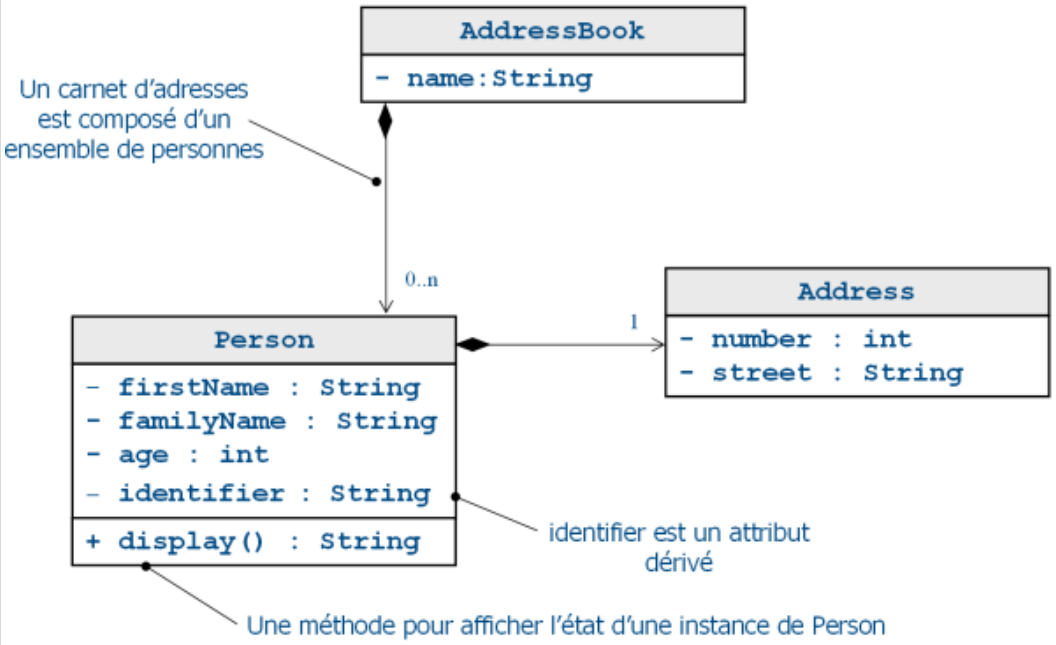
- Sélectionner depuis le fichier *genmodel* le package racine *Addressbook* et générer le code Java correspondant au modèle (Generate Model Code). Un ensemble de classes Java doivent être générées dans le package *eclipse.emf.addressbook.model.addressbook*.
- Examiner les classes générées et remarquer le découpage en trois catégories qui font apparaître une programmation par contrats : interfaces, implémentations et classes utilitaires.



Nous décidons par la suite de modifier le modèle de façon à :

- ajouter un attribut dérivé dans *Person* appelé *identifier* de type *String* qui retourne une chaîne de type (*firstName* + *familyName* + *age*),
- ajouter une opération *String display()* qui se chargera d'effectuer un affichage complet d'une instance de *Person*.

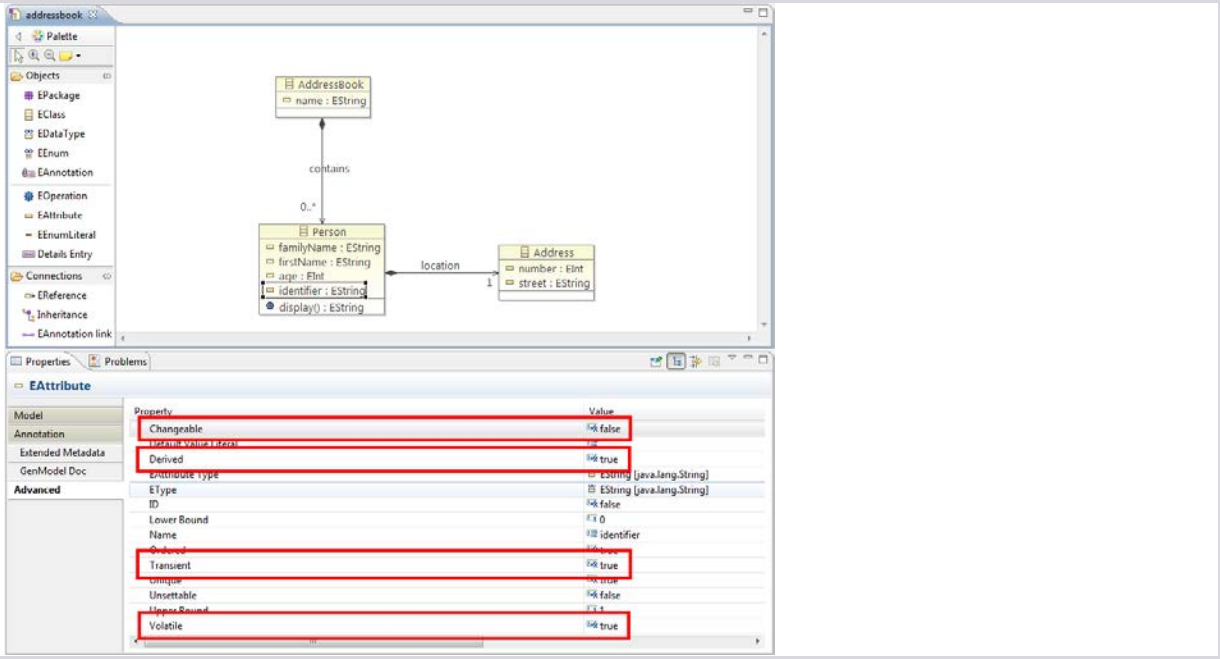
Le schéma ci-dessous représente graphiquement la modélisation attendue par cette modification.



- Compléter votre modèle EMF (via l'éditeur Ecore Diagram Editing par exemple) de façon à intégrer les modifications demandées. Pour l'attribut *identifier*, déclarer le *Derived*, *Volatile*, *Transient* et non *Changeable*.

Cela a pour effet pour l'attribut *identifier :*

- derived : calculé à partir d'autres attributs,
- volatile : ne génère par l'attribut pour stocker l'état, le corps de la méthode est également laissé à vide,
- transient : ne sera pas sauvegardé,
- changeable : valeur pouvant changer



- Le fichier *addressbook.ecore* est automatiquement impacté. Toutefois, le fichier *genmodel* doit être explicitement mis à jour. Sélectionner le fichier *addressbook.genmodel* puis cliquer sur Reload (via le menu contextuel). Sélectionner ensuite Ecore model et laisser les valeurs par défaut puis valider. Vous remarquerez que les nouveaux attributs ont été ajoutés et que les anciennes valeurs de configuration de génération (Base Package en l'occurrence) n'ont pas été supprimées.