



UNIVERSITE SIDI MOHAMED BEN ABDELLAH
FACULTÉ DES SCIENCES DHAR EL MAHRAZ
-FES-

Département : Informatique

Filière : Qualité du Logiciel

Étude sur

Le standard XMI (XML Meta Interchange)

Le modèle MOF (Meta Object Facility)

EMF (Eclipse Modeling Framework)

Réalisé par : Ouchta Ismail

Année universitaire : 2022/2023

Table des matières

Définition XMI:	3
Motivation :	3
Historique – versions XMI et versions MOF :	3
Qu’y a-t-il dans XMI 1.1:	4
Et cela signifie pour UML :	4
Les règles de production:	4
Définition MOF:	7
Couches de modélisation :	7
Différences entre MOF et UML :	8
Définition :	8
Création du modèle EMF d’un carnet d’adresses :	8

Le standard XMI (XML Meta Interchange):

Définition XMI:

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. XML Metadata Interchange (XMI) est une norme OMG (Object Management Group) pour l'échange d'informations de métadonnées via Extensible Markup Language (XML).

Il peut être utilisé pour toutes les métadonnées dont le métamodèle peut être exprimé dans Meta-Object Facility (MOF).

L'utilisation la plus courante de XMI est comme format d'échange pour les modèles UML.

Motivation :

La motivation pour l'introduction de XMI était la nécessité de fournir un moyen standard par lequel les outils UML pourraient échanger des modèles UML. Le XMI produit par un outil peut généralement être importé par un autre outil, ce qui permet l'échange de modèles entre outils de différents fournisseurs, ou l'échange de modèles avec d'autres types d'outils en amont ou en aval de la chaîne d'outils. Comme indiqué ci-dessus, XMI ne se limite pas au mappage UML vers XML, mais il fournit des règles pour générer des DTD ou des schémas XML et des documents XML à partir de n'importe quel MOF compatible.

Historique – versions XMI et versions MOF :

- XMI 1.1 correspond à MOF 1.3
- XMI 1.2 correspond à MOF 1.4
- XMI 1.3 (added Schema support) correspond à MOF 1.4
- XMI 2.0 correspond à MOF 1.4

(adds Schema support and changes document format)

- XMI 2.1 correspond à MOF 2.0

Qu'y a-t-il dans XMI 1.1:

- Principes de conception.
- Un ensemble de règles de production XML Document Type Definition (DTD) pour transformer les métamodèles basés sur MOF en DTD XML.
- Un ensemble de règles de production de documents XML pour l'encodage et le décodage des métadonnées basées sur MOF.
- (DTD concrètes pour UML et MOF.)

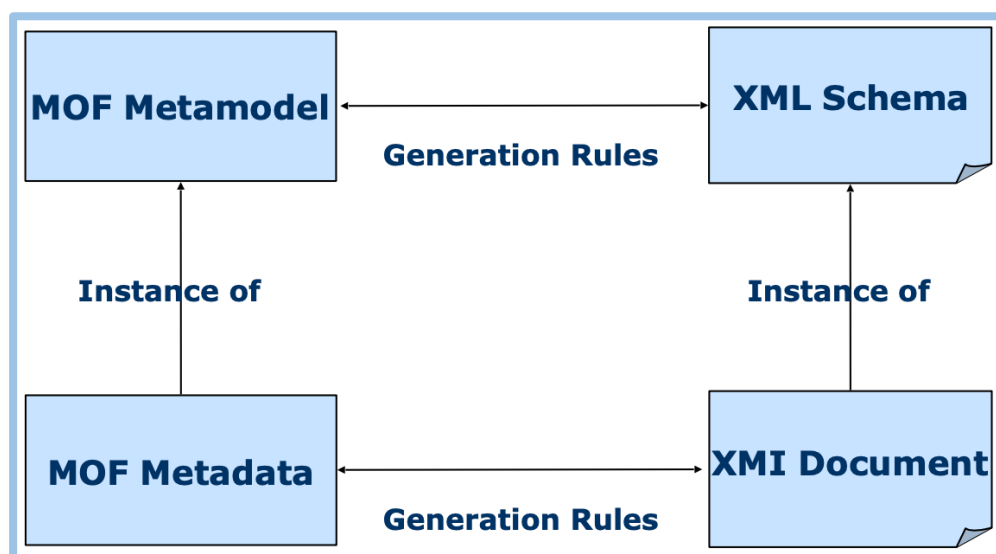
Et cela signifie pour UML :

Parce que vous pouvez rechercher vers le haut ou vers le bas l'architecture du métamodèle, UML peut être considéré comme :

- un document XML conforme à une DTD décrivant MOF
- une DTD XML à laquelle les modèles UML doivent se conformer.

Les règles de production:

Les règles de production spécifient aussi formellement que possible la manière dont les modèles basés sur MOF doivent être transformés à la fois en schémas XML et en documents.



Ensemble de règles qui montrent quelles déclarations doivent figurer dans tout document bien formé Schéma XML (contenu) et comment l'information est structurée (structure).

Mappage entre tout type d'élément de modèle MOF et une déclaration de schéma.

```
<xsd:schema xmlns:xsd="http://www.w3c.org/2001/XMLSchema"
  xmlns:xmi="http://www.omg.org/XMI"
  targetNamespace="http://www.sintef.org/CDs"
  xmlns:cds="http://www.sintef.org/CDs">

  <xsd:complexType name="CD">
    <xsd:choice minOccurs="0" maxOccurs="unbounded">
      <xsd:element name="title" type="xsd:string"/>
      <xsd:element name="artist" type="xsd:string"/>
      <xsd:element name="num_tracs" type="xsd:integer"/>
      <xsd:element ref="xmi:Extension"/>
    </xsd:choice>
    <xsd:attribute ref="xmi:id"/>

    <xsd:attribute name="title" type="xsd:string"/>
    <xsd:attribute name="artist" type="xsd:string"/>
    <xsd:attribute name="num_tracs" type="xsd:integer"/>
  </xsd:complexType>

  <xsd:element name="CD" type="cds:CD"/>
</xsd:schema>
```

CD

- title : String
- artist : String
- num_tracs : Integer

Figure 1 Exemple de production de schéma

Exemple de production de documents basé sur le schéma généré

Born to Run
Bruce Springsteen
8 tracks

```
<?xml version="1.0" encoding="UTF-8"?>
<cds:CD xmlns:cds="http://www.sintef.org/CDs"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xmi="http://www.omg.org/XMI"
  xsi:schemaLocation="http://www.sintef.org/CDs"

  artist="Bruce Springsteen" title="Born To Run" num_tracs="8"
  xmi:id="_1">
</cgs:CD>
```

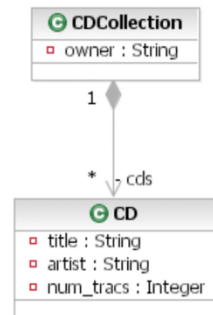
Figure 2 Exemple de production de documents

- Association entre les classes
 - Un `xsd:element` est créé avec le nom défini sur le nom de la référence et le type défini sur le nom de type de la classe référencée

```

<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:cdlib="http://www.sintef.org"
  xmlns:xmi="http://www.omg.org/XMI"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://www.sintef.org">
  <xsd:import namespace="http://www.omg.org/XMI" schemaLocation="XMI.xsd" />
  <xsd:complexType name="CD">
    <xsd:choice maxOccurs="unbounded" minOccurs="0">
      <xsd:element ref="xmi:Extension" />
    </xsd:choice>
    <xsd:attribute ref="xmi:id" />
    <xsd:attributeGroup ref="xmi:ObjectAttribs" />
    <xsd:attribute name="title" type="xsd:string" use="required" />
    <xsd:attribute name="artist" type="xsd:string" />
    <xsd:attribute name="num_tracs" type="xsd:int" />
  </xsd:complexType>
  <xsd:element name="CD" type="cdlib:CD" />
  <xsd:complexType name="CDLibrary">
    <xsd:choice maxOccurs="unbounded" minOccurs="0">
      <xsd:element name="cds" type="cdlib:CD" />
      <xsd:element ref="xmi:Extension" />
    </xsd:choice>
    <xsd:attribute ref="xmi:id" />
    <xsd:attributeGroup ref="xmi:ObjectAttribs" />
    <xsd:attribute name="owner" type="xsd:string" />
  </xsd:complexType>
  <xsd:element name="CDLibrary" type="cdlib:CDLibrary" />
</xsd:schema>

```



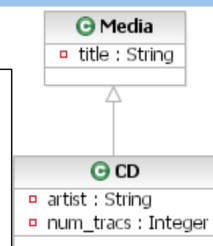
- Héritage

Les schémas XML n'autorisent qu'un seul héritage, mais MOF autorisent l'héritage multiple. La solution : XMI utilise une stratégie de copie pour implémenter l'héritage

```

<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xmi="http://www.omg.org/XMI"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns="ml">
  <xsd:import namespace="http://www.omg.org/XMI"
    schemaLocation="XMI.xsd"/>
  <xsd:complexType name="Media">
    <xsd:choice maxOccurs="unbounded" minOccurs="0">
      <xsd:element ref="xmi:Extension" />
    </xsd:choice>
    <xsd:attribute ref="xmi:id"/>
    <xsd:attributeGroup ref="xmi:ObjectAttribs"/>
    <xsd:attribute name="title" type="xsd:string"/>
  </xsd:complexType>
  <xsd:element name="Media" type="ml:Media"/>
  <xsd:complexType name="CD">
    <xsd:attribute name="title" type="xsd:string"/>
    <xsd:attribute name="artist" type="xsd:string"/>
    <xsd:attribute name="num_tracs" type="xsd:int"/>
  </xsd:complexType>
  <xsd:element name="CD" type="ml:CD"/>
</xsd:schema>

```



En raison des similitudes entre le modèle MOF M3 et les modèles de structure UML, les métamodèles MOF sont généralement modélisés sous forme de diagrammes de classes UML. Une norme de support de MOF est XMI, qui définit un format d'échange basé sur XML pour les modèles sur la couche M3, M2 ou M1.

Différences entre MOF et UML :

- MOF
 - Fournit les services de métadonnées
 - Définit le langage de méta-métamodélisation pour définir d'autres métamodèles comme UML (Niveau M3 : doit être plus simple qu'UML)
 - Définit une norme d'échange de modèles (XMI)
- UML
 - Fournit la notation de modélisation (et de métamodélisation) (Niveaux M2 et M1 : les éléments du modèle ont des annotations ajoutées)
 - Langage de modélisation à usage général

EMF (Eclipse Modeling Framework):

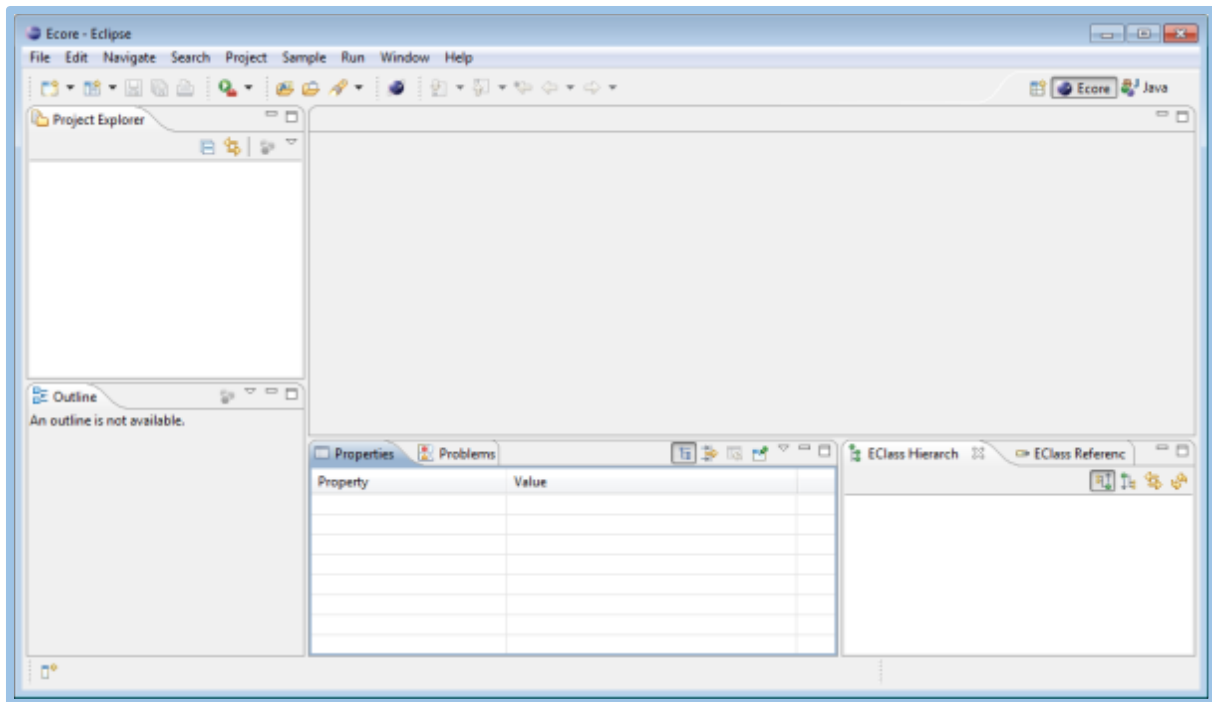
Définition :

Eclipse Modeling Framework (EMF) est un cadre de modélisation basé sur Eclipse et une fonction de génération de code pour la création d'outils et d'autres applications basées sur un modèle de données structuré.

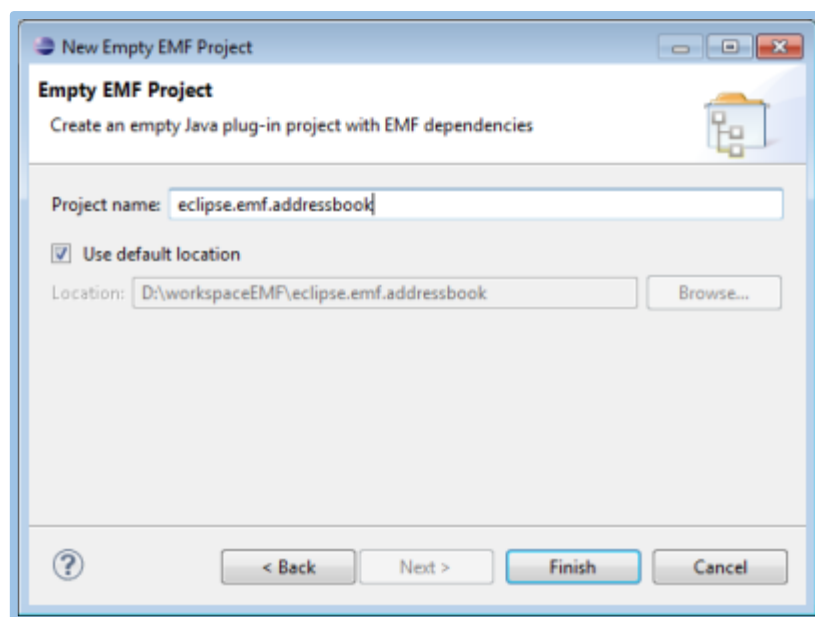
À partir d'une spécification de modèle décrite dans XML Metadata Interchange (XMI), EMF fournit des outils et un support d'exécution pour produire un ensemble de classes Java pour le modèle, un ensemble de classes d'adaptateur qui permettent l'affichage et l'édition basée sur des commandes du modèle, et une base éditeur. Les modèles peuvent être spécifiés à l'aide de documents Java, UML, XML annotés ou d'outils de modélisation, puis importés dans EMF. Plus important encore, EMF fournit la base de l'interopérabilité avec d'autres outils et applications basés sur EMF.

Création du modèle EMF d'un carnet d'adresses :

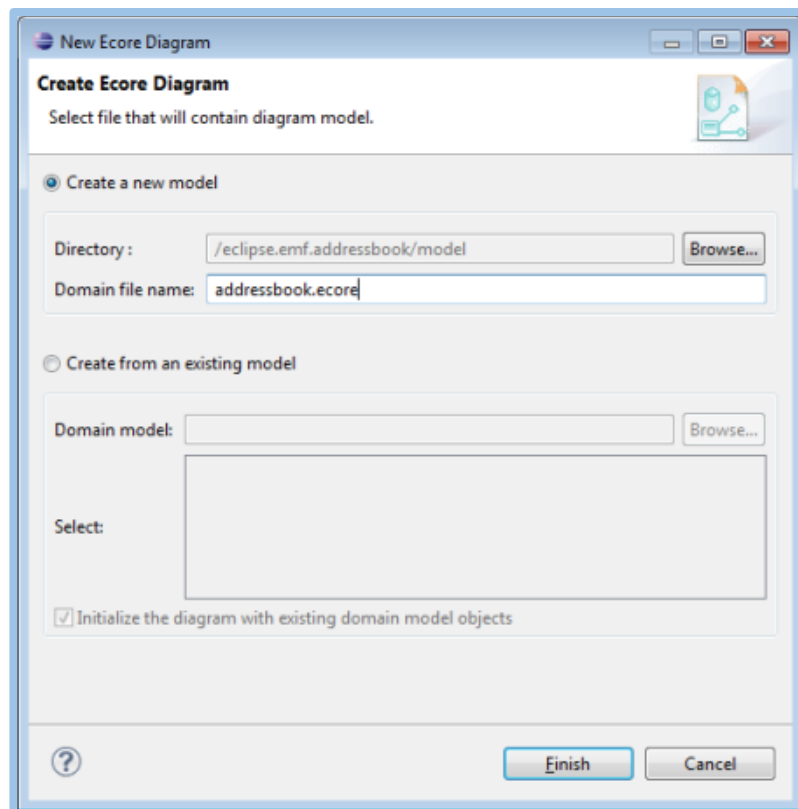
- Démarrer l'environnement de développement Eclipse contenant les plugins de modélisation puis créer un nouveau Workspace (workspaceEMF) afin de disposer d'un répertoire spécifique à la modélisation.
- Pour afficher les vues Eclipse spécifiques à la modélisation EMF, ouvrir la perspective Ecore.



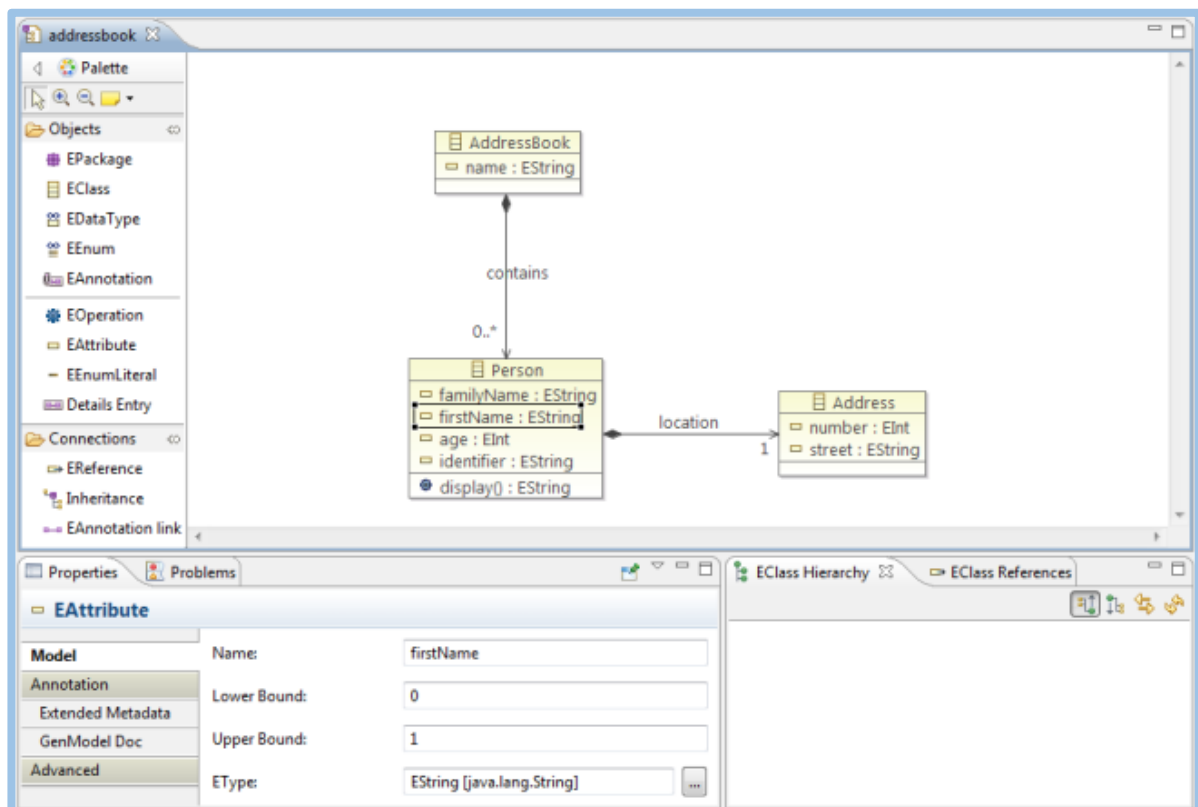
- Créer un nouveau projet EMF vide (File -> New -> Project ... -> Eclipse Modeling Framework -> Empty EMF Project) et nommer le projet eclipse.emf.addressbook.



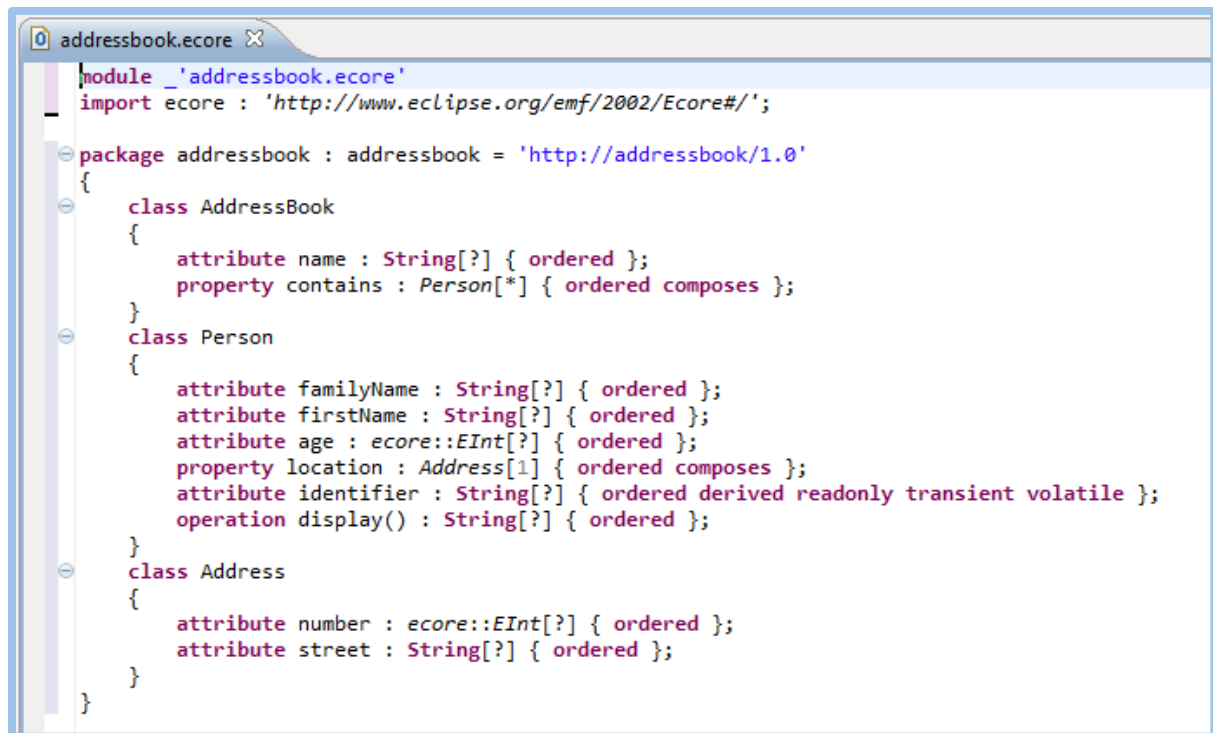
- Sélectionner le répertoire model et créer un diagramme Ecore (Ecore Diagram). Nommer le fichier ecore addressbook.ecore.



- 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.

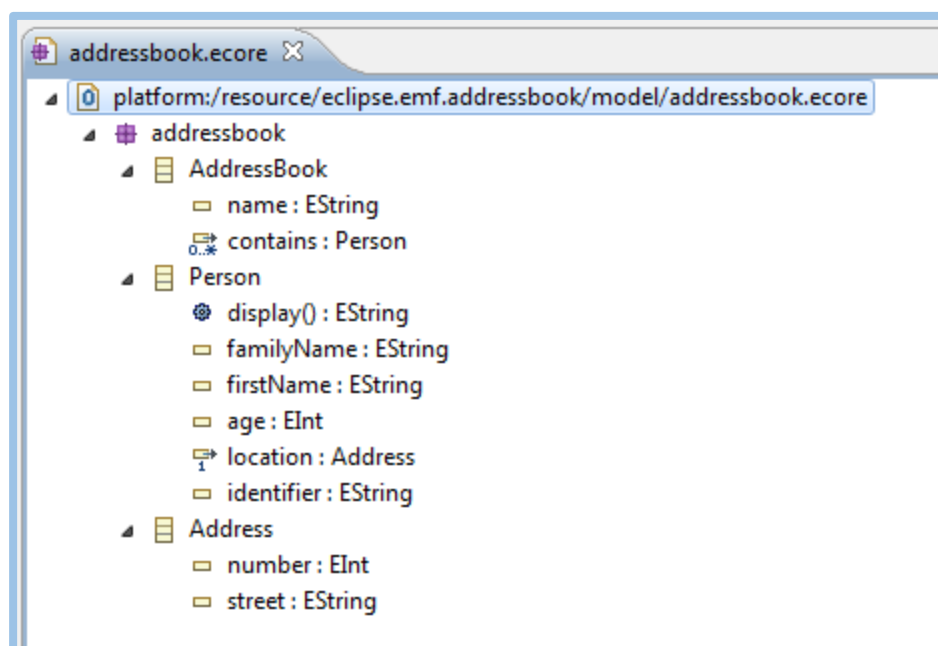


Visualiser le modèle sous différentes représentations via l'utilisation d'éditeurs adaptés : OCLinEcore (Ecore) Editor et Sample Ecore Model Editor.



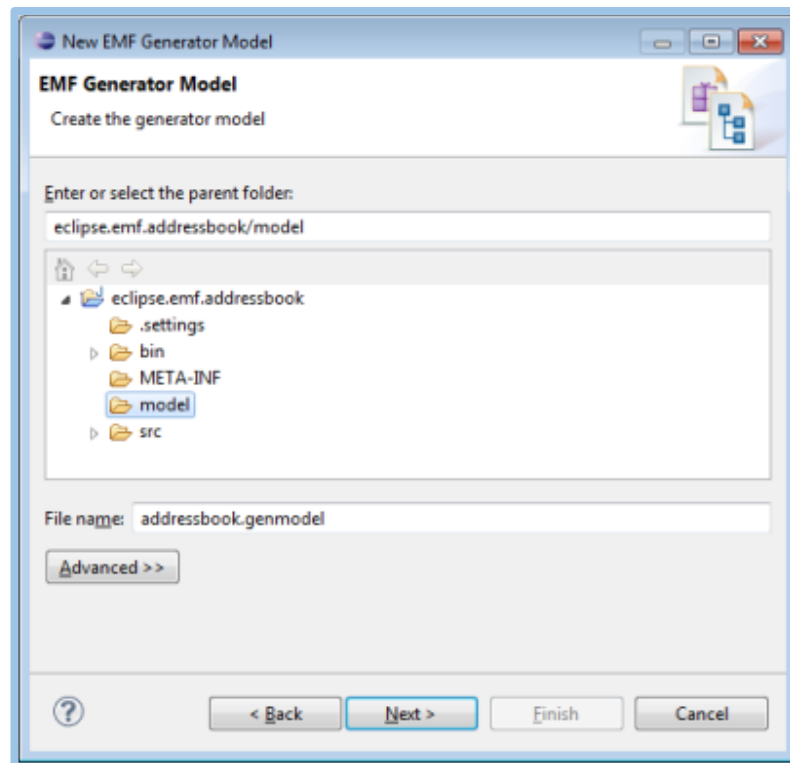
```
module _'addressbook.ecore'
import ecore : 'http://www.eclipse.org/emf/2002/Ecore#/' ;

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 };
    }
}
```

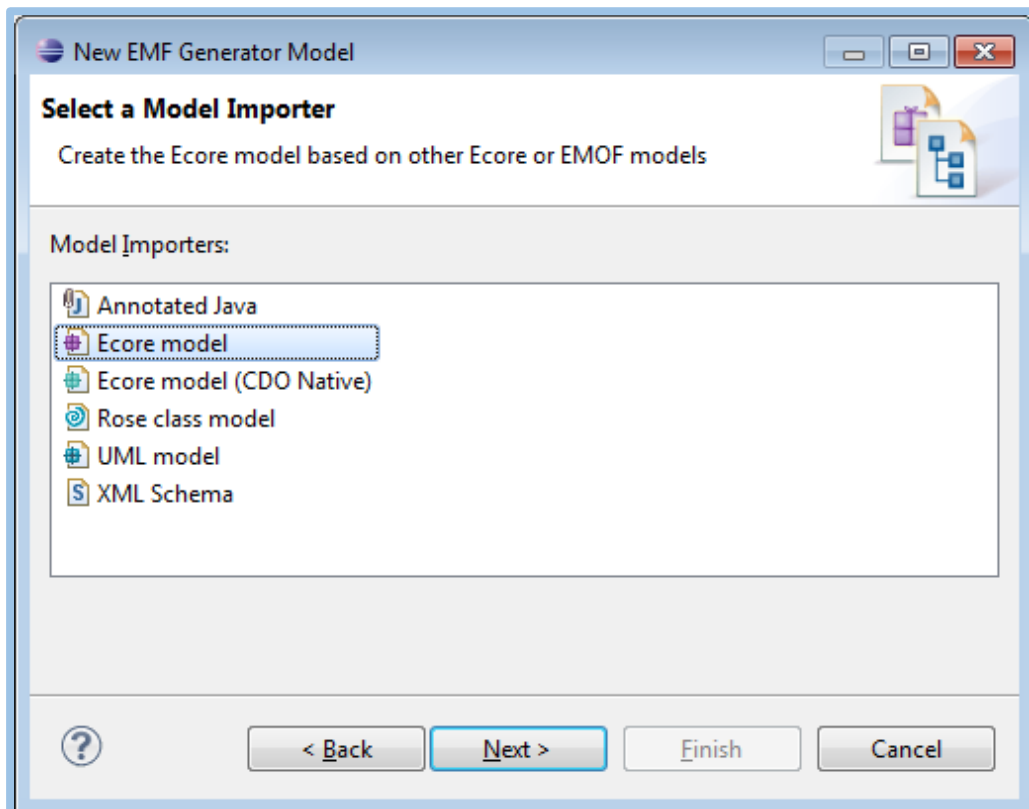


- Visualiser finalement votre modèle au format XML, vous remarquerez qu'il s'agit d'un fichier XMI dont les données correspondent à des instances du métamodèle Ecore. Nous reviendrons sur cette notion dans les prochaines sections.
- Créer un modèle de génération (New -> Other... -> Eclipse Modeling Framework -> EMF Generator Model), sélectionner ensuite le répertoire

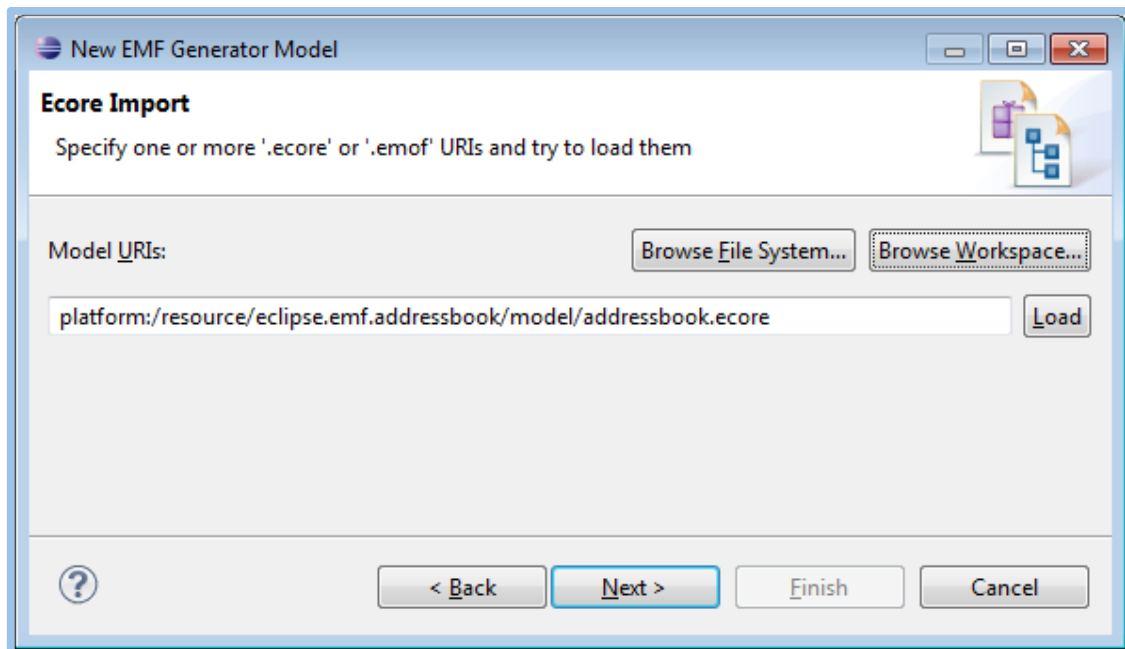
model du projet eclipse.emf.addressbook puis nommer le fichier addressbook.genmodel.



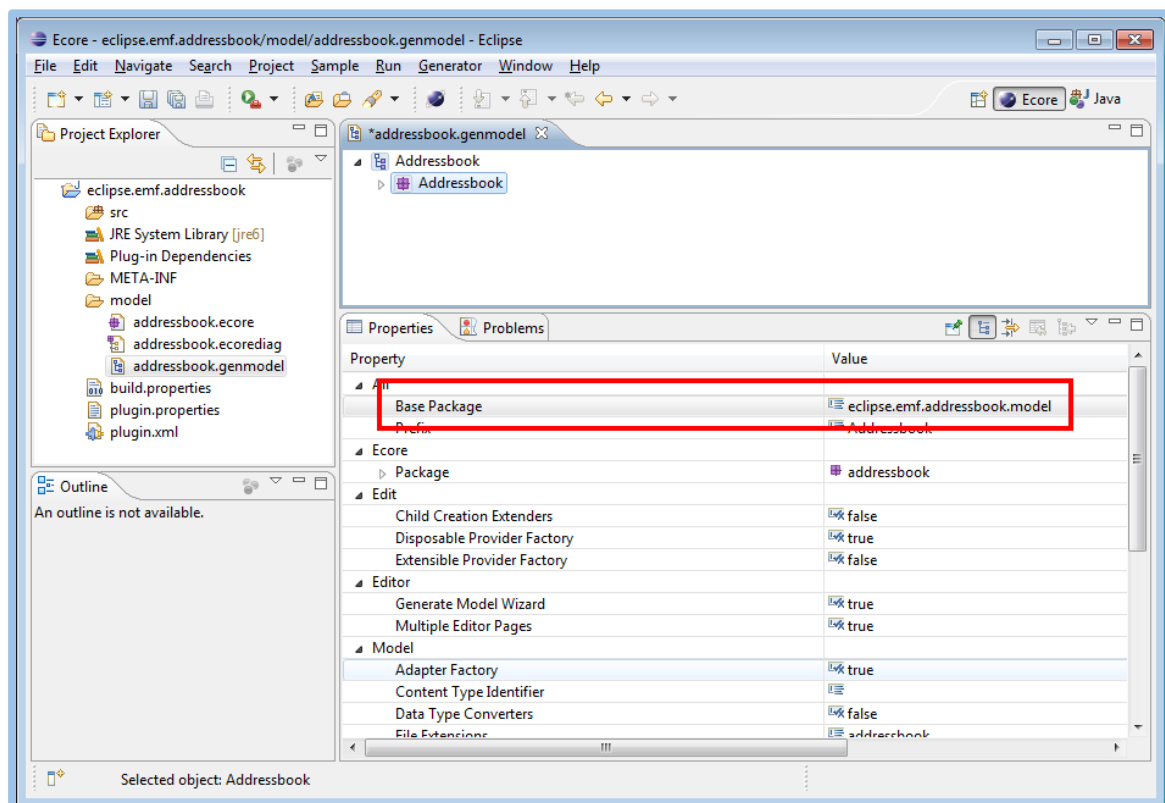
- Sélectionner ensuite Ecore model comme type de modèle utilisé pour créer ce modèle de génération.



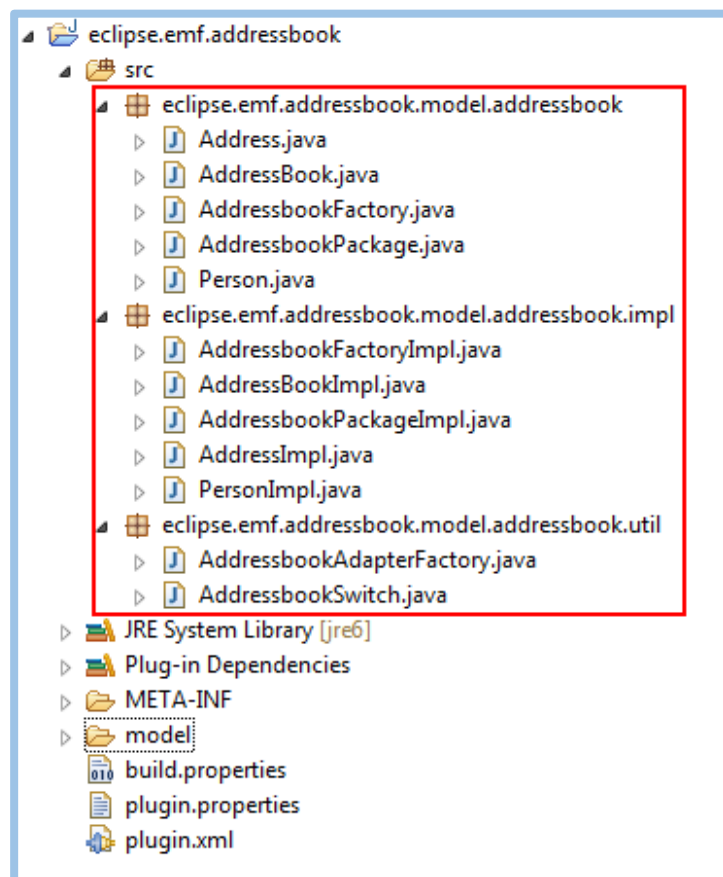
- Sélectionner enfin le fichier addressbook.ecore (à partir de la navigation du Workspace courant Browse Workspace...) puis terminer.



- 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.

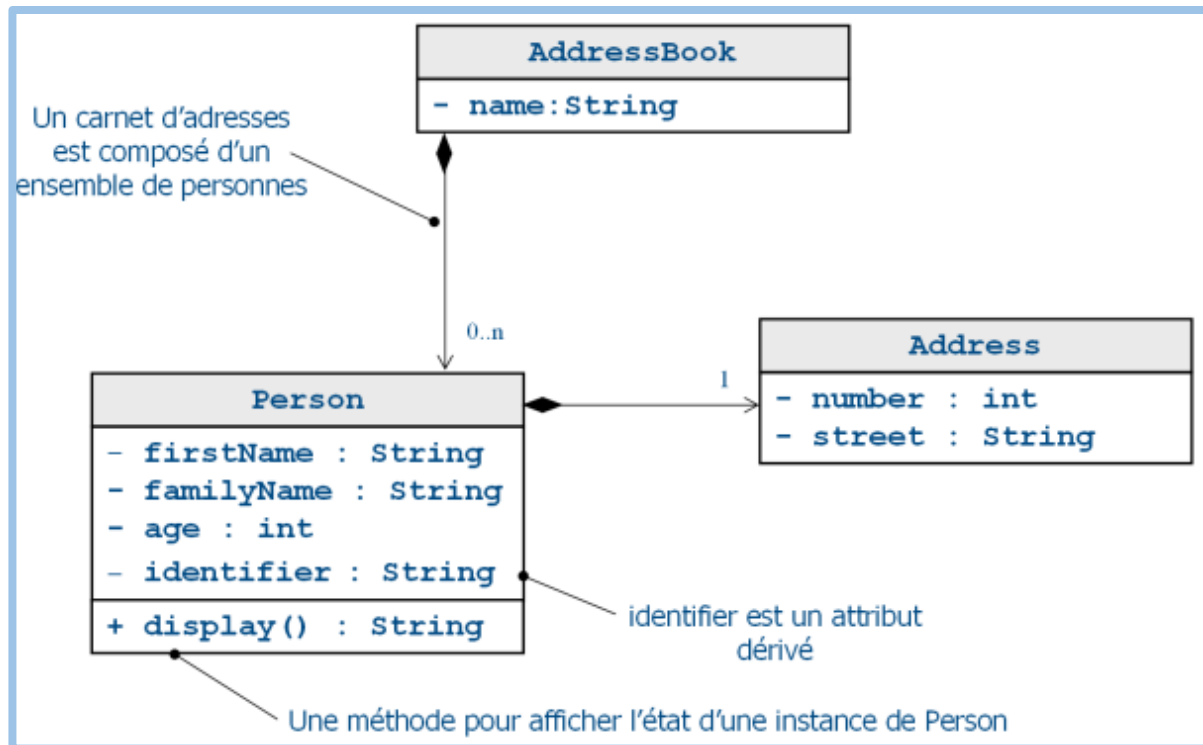


- Sélectionner depuis le fichier genmodel le package racine Addressbook et générer le code Java correspondant au modèle (Generate Model Code). Un ensemble de classes Java doivent être générées dans le package eclipse.emf.addressbook.model.addressbook.
- Examiner les classes générées et remarquer le découpage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



- 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.

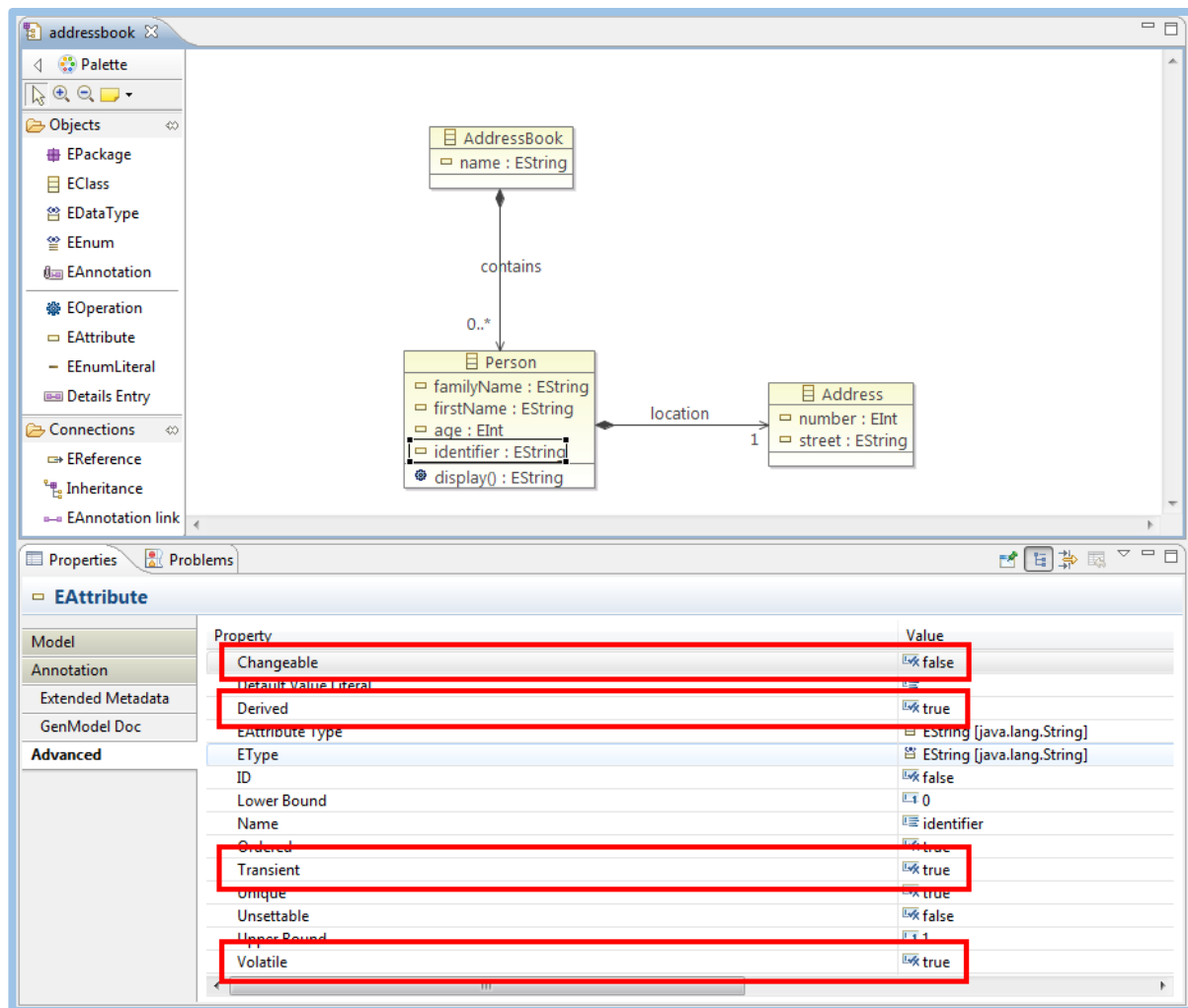
Le schéma ci-dessous représente graphiquement la modélisation attendue par cette modification.



- 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 pas 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.