www.289eme.fr

Eclipse et ses plugins de modélisation

(EMF – GEF – GMF).

Entrée en matière

par

Jacques Barzic

MCours.com

Avertissement

Cette version incomplète du document (eclipse_emf_gef_gmf_alpha1.0.0, voir date édition ci-dessous) comprend :

- l'introdcution,
- la 1ére partie : préparation de la plate-forme d'expérimentation,
- la 2ème partie : Eclipse Modeling Framework (EMF),
- un début de bibliographie non finalisée.

PDF créé le 12 janvier 2008

Table des matières

Index des tableaux 5

Index des illustrations 7

Liste des abréviations 9

Introduction 11

1ère Partie : la plate-forme d'accueil 13
- **1. Eclipse : généralités** 13
- **2. Installation de Eclipse 3.2** 13
  - 2.1. Préalable 13
  - 2.2. Téléchargement - Installation 14
  - 2.3. Utilisation 14
- **3. Les versions utilisées** 14

2ème Partie : Eclipse Modeling Framework (EMF) 15
- **1. Introduction** 15
- **2. Objectif de EMF** 16
- **3. Les formats d'entrée standards** 16
  - 3.1. UML 16
  - 3.2. XMI 17
  - 3.3. Java Annoté 17
- **4. Le (méta ?)méta-modèle pivot : Ecore** 18
- **5. Génération du code** 19
  - 5.1. Organisation du code généré 19
  - 5.2. La re-génération et la fusion 20
  - 5.3. Le modèle générateur 20
- **6. Autres services proposés par EMF** 21
  - 6.1. La notification et les Adaptateurs 21
  - 6.2. La gestion de la persistance 21
  - 6.3. L'API réflective 21
  - 6.4. L'EMF dynamique 22
- **7. EMF et les standards OMG** 22
  - 7.1. Pour UML 22
  - 7.2. Pour MOF 22
  - 7.3. Pour XMI 23
  - 7.4. Pour MDA 23
- **8. Un peu d'exercice** 23
  - 8.1. Installer le plugin EMF 23
  - 8.2. Disposer d'un modèle d'entrée 24
  - 8.3. Générer le modèle ecore 25
  - 8.4. Descrire le modèles .genmodel 25
  - 8.5. Générer le code (vu ici comme un modèle) 26
  - 8.6. Autres générations possibles 27
- **9. Conclusion** 27

3ème Partie : Graphical Editing Framework (GEF) 29

4ème Partie : Graphical Modeling Framework (GMF) 29

5ème Partie : EMF, GEF et GMF en action 29

Conclusion 29

Annexe 1 : le méta-méta-modèle Ecore (simplifié) 31

Bibliographie 34

Index 36

Glossaire 37

Index des tableaux

Tableau I - Versions des logiciels utilisées..14
Tableau II - Les espaces techniques de modélisation à fédérer.. 18
Tableau III - L'espace technique de modélisation fédérateur.. 18

Index des illustrations

Illustration 1 - Organisation générale de EMF ([BUDINSKY et al. 2004])............................16
Illustration 2 - Le modèle de l'exemple...24
Illustration 3 - Arborescence du modèle Ecore de notre projet « Bibliothèque »....................26
Illustration 4 - Méta-méta-modèle Ecore : hiérarchie des classes..31
Illustration 5 - Méta-méta-modèle Ecore : éléments (1)..31
Illustration 6 - Méta-méta-modèle Ecore : éléments (2)..32
Illustration 7 - Méta-méta-modèle Ecore : éléments (3)..32
Illustration 8 - Méta-méta-modèle Ecore : éléments (4)..32

MCours.com

Liste des abréviations

API : *Application Programming Interface*.

BI : *Business Intelligence*.
CWM : *Commun Warehouse Metamodel.*

DTD : *Document Type Definition* (super ça marche aussi en français ! Définition de Type de Document).

EBNF : *Extended Backus-Naur Form.*

EMF : *Eclipse Modeling Framework*.

EPL : *Eclipse Public Licence.*

GEF : *Graphical Editing Framework*.

GMF : *Graphical Modeling Framework*.

MDA : *Model Driven Architecture.*

MOF : *Meta Object Facility.*

MVC : *Model-Vue-Controler.*

OMG : *Object Management Group.*

QVT : *Query – View – Transformation.*

UML : *Unified Modeling Language.*

W3C : *World Wide Web Consortium.*

XMI : *XML Metadata Interchange.*

XML : *eXtensible Markup Language.*

XSL : eX*tensible Stylesheet Language.*

Introduction

Lors d'un travail[1] sur la méthode MDA (*Model Driven Architecture*), j'ai été amené à rechercher des outils logiciels existants pour la mettre en oeuvre.
Très rapidement, une famille de solutions autour de la plate-forme *Eclipse* se dégage. Ces solutions font référence de manière récurrente aux *plugins* EMF (*Eclipse Modeling Framework*) , GEF (*Graphical Editing Framework*) et GMF (*Graphical Modeling Framework*).

Ayant, à ce moment là, utilisé *Eclipse* pour des développement d'application *Web* en *Java*, je décide d'étudier plus en avant les possibilités de cet outil pour expérimenter MDA. J'étais d'autant plus encouragé d'aller dans cette voie que les projets d'outils liés à MDA autour de *Eclipse* sont nombreux et prometteurs.

Un des freins à ce travail pour moi était ma méconnaissance de ces plugins, phénomène accentué par le manque de documentations synthétique (notamment en langue française[2]) sur le sujet.

Je me suis donc attelé à la rédaction du présent document, que je fournis à la communauté et aussi à sa critique. Il s'agit d'une introduction à ces technologies, rédigée par un non-spécialiste. Ce document n'a pas d'autre prétention que d'être une entrée en matière.

On trouvera donc dans ce document quatre parties.
La première décrira la plate-forme mise en place pour le reste de la démonstration (*Eclipse* et ses plugins) d'un point de vue pratique : installation, mise en oeuvre,...
Puis suivrons trois parties descriptives des trois plugins en question : objectifs, description, exemples pratiques.
La dernière et cinquième partie portera un regard sur l'usage que font de cette configuration les outils plus globaux se réclamant de l'Ingénierie Dirigée par les Modèles (IDM), dont MDA est une implémentation.

Ce document nécessite quelques pré-requis pour être lu avec profit :

- une connaissance de base de *Eclipse*,
- savoir ce que représentent les concepts de modèles, méta-modèles, méta-méta-modèles (mon document déjà cité sur MDA les reprend),
- avoir une pratique, sans être un expert, de UML (*Unified Modeling Language*) et de Java.

Le lecteur trouvera les sources créées lors des exercices sur le site *www.289eme.fr*.
Nom de l'archive : *eclipse_emf_gef_gmf_sources.zip*.
L'archive contient des répertoires dont le nom est codé de la manière suivante :

nom_x_y.z où :

nom = chemin du répertoire dans le projet (celui dans l'explorateur Windows),
x = numéro de la partie concernée du document,
y.x = numéro de la section dans la partie.

Les archives contiennent aussi les éléments des archives qui la précèdent dans le même projet.

1 *Voir document [BARZIC 2006].*
2 *Je n'ai rien contre l'anglais en soi, mais pour nous, francophones de naissance, le français est plus accessible !!*

1ère Partie : la plate-forme d'accueil

1. *Eclipse* : généralités

Le présent document n'a pas pour objet de décrire la plate-forme *Eclipse* en elle même. Les lecteurs qui désirent la découvrir pourront consulter le site francophone qui lui est consacré [ECLIPSE TOTALE]. Il résume bien ce qu'est *Eclipse* et d'où elle vient. Par ailleurs, ce site tient différentes rubriques et revoie vers les pages *Web* officielles des projets que fédère la *Fondation Eclipse*.

Un grand principe est à retenir quant à l'architecture de *Eclipse* : un noyau est diffusé sous ce nom qui offre les fonctionnalités de base. Ce noyau est conçu (et c'est un principe) pour pouvoir recevoir des extensions (les fameux *plugins*). Comme le noyau est diffusé en *Open Source* (sous licence *Eclipse Public Licence* – EPL), chacun peut développer de nouveaux plugins : il en existe sous licence libre et gratuite et il existe aussi des projet commerciaux et payants.

Depuis juin 2006, la version courante de *Eclipse* est la version 3.2 batisée *Callisto*.

La nouveauté organisationnelle, cette année, est que la livraison de nouvelles versions majeures, annuelle depuis 2004, est couplées à la livraison de neuf sous projets importants. Cela permet de se retrouver plus facilement dans les nombreux projets qui se basent sur *Eclipse*.

Callisto est le nom de code donné à la sortie simultanée des 10 projets suivants :

- Eclipse - Version 3.2,
- BIRT - *Business Intelligence and Reporting Tools* - Version 2.1,
- CDT - *C/C++ Development Tools* - Version 3.1,
- DTP - *Data Tools Project* - Version 0.9,
- EMF - *Eclipse Modeling Framework* - Version 2.2,
- GEF - *Graphical Editing Framework* - Version 3.2,
- GMF - *Graphical Modeling Framework* - Version 1.0,
- TPTP - *Test & Performance Tools Platform* - Version 4.2,
- VE - *Visual Editor* - Version 1.2,
- WTP - *Web Tools Project* - Version 1.5,

Source [ECLIPSE TOTALE].

Les quatre projets qui nous intéressent ici sont dans le lot.

2. Installation de *Eclipse* 3.2

2.1. Préalable

Eclipse étant développée en *Java*, une machine virtuelle doit être installée sur l'ordinateur pour pouvoir l'utiliser. Vous pouvez télécharger cette machine virtuelle sur [SUN JAVA a] et ensuite l'installer.

2.2. Téléchargement - Installation

Aller à la page *http://www.eclipse.org/downloads/* pour techarger *Eclipse*.
La version 3.2.1 est utilisée pour ce rapport.
Ensuite il suffit de décompresser l'archive dans le répertoire de votre choix.

Il peut être utile aussi de télécharger et d'installer le pack de traduction pour avoir la version française. Pour cela téléchargez le pack *Nlpack1-eclipse-SDK-3.2.1-win32.zip* sur la page *http://download.eclipse.org/eclipse/downloads/*.
Pour l'installer il suffit de le décompresser dans le répertoire d'installation de *Eclipse*.

Lancer *Eclipse* et procéder à une mise à jour :

1. menu *Aide -> Mise à jour de logiciels -> Rechercher et installer...*,
2. choisir dans le boite de dialogue : *Rechercher les mises à jour des dispositifs déjà installés,*
3. laisser l'opération se faire et répondre aux questions et/ou sélectionner les options à la demande.

2.3. Utilisation

Dans la suite nous considérons que le lecteur possède la connaissance des manipulations de base de *Eclips*e : création d'un espace de travail, création et manipulation d'un projet, affichage de vues et perspectives, mises à jour de la configuration, téléchargment/installation de plugins.

3. Les versions utilisées

Tableau I - Versions des logiciels utilisées

Logiciel	Version
Windows	2000 Pro
VM Java	J2SE Runtime Environment 5.0 Update 5
Eclipse	3.2.1 buildId=M20060921-0945
EMF	2.2.1 (septembre 2006)
GEF	
GMF	

2ème Partie : *Eclipse* Modeling Framework (EMF)

1. Introduction

> *Eclipse Modeling Framework est la partie Model du pattern MVC (Model-Vue-Controler) (à noter que le framework ne propose pas de visuel pour représenter le modèle). Le modèle peut être persisté sous différentes manières : XSL, fichiers java avec annotations, XMI, puis donne la possibilité de rajouter son système de persistance. A noter que EMF gère la persistance sous forme de plusieurs fichiers ressources reliées, et qu'en implémentant son propre système de persistance, vous ne perdez pas cet atout* [WIKIPEDIA-FR 2006].

Cette définition « à la » *Wikipédia*, si elle souffre de quelques manques, permet une entrée en matière :

- EMF est un « framework » qui traite des modèles : cela peut s'entendre ici sous le sens que EMF offre à ces utilisateurs un cadre de travail pour la manipulation des modèles (sous entendu, à propos des applications informatiques, credo de *Eclipse*),
- EMF permet de stocker les modèles sous forme de fichier pour en assurer la persistance,
- EMF permet de traiter différents types de fichiers : conformes à des standards reconnus (XML, XMI) et aussi sous des formes spécifiques (code Java) ou tout simplement sur mesure (au bon gré du concepteur),
- EMF ne propose pas d'outil graphique (de dessin) pour la modélisation.

Mais cette définition ne dit rien sur ce dont est capable EMF, sur la manière dont il y arrive et sur sa mise en oeuvre à travers *Eclipse*. C'est ce que nous allons tenter d'éclairer dans ce document.

J'ai choisi l'option (certes discutable) dans cette partie de ne pas placer les exemples en même temps que les explications plus théoriques. Le lecteur se reportera utilement au *§ 8 Un peu d'exercice*.
Le paragraphe d'exercice peut être utilisé au fur et a mesure de la progression de la lecture des autres paragraphes. Cette présentation permet de diminuer le volume de la partie théorique et de bien séparer les objectifs, tout en jalonnant la progression

Sauf indication contraire, la ressource utilisée pour la rédaction de cette partie est [BUDINSKY et al. 2004].

2. Objectif de EMF

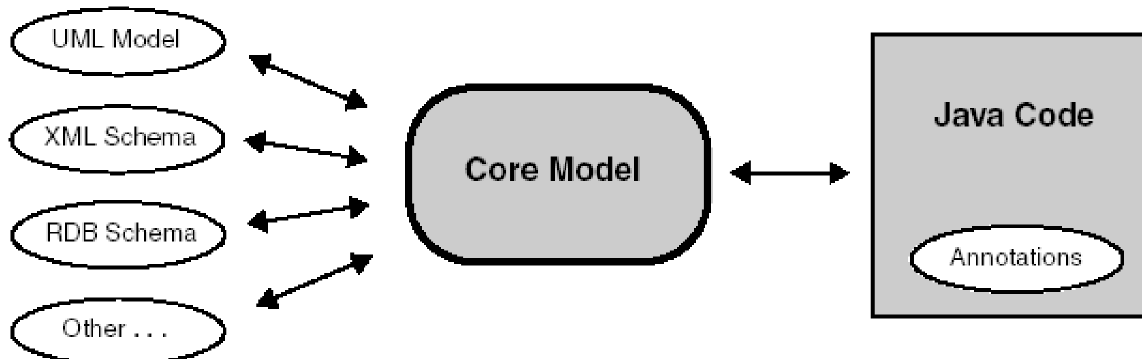

Illustration 1 - Organisation générale de EMF ([BUDINSKY et al. 2004])

Comme le montre l'*Illustration* ci-dessus, l'objectif général de EMF est de proposer un outillage qui permet de passer du modèle au code *Java* automatiquement. Pour cela le *framework* s'articule autour d'un modèle (le *Core Model*).
EMF va proposer plusieurs services :

1. la transformation des modèles d'entrées (à gauche sur l'*Illustration 1*), présentés sous diverse formes, en *Core Model*,
2. la gestion de la persistance du *Core Model*,
3. la transformation du *Core Model* en code Java.

Les doubles flèches symbolisent que les transformations inverses (ou les « imports/exports ») sont aussi gérées par EMF.

En cela, certains considèrent que EMF est à la fois :

- un outil de ré-unification de divers standards de modèles (XMI, UML, code Java - si du moins on considère ce dernier comme un modèle),
- un pont entre deux mondes du génie logiciel (celui des gourous de la modélisation et celui des partisans du code avant tout) en prenant une position médiane et en prenant le meilleur de chaque monde,...

... et tout cela pour un coût minimum. Le rêve en quelque sorte !

3. Les formats d'entrée standards

EMF peut de base gérer en entrée des modèles présentés sous trois formats : UML, XMI et en code Java Annoté.

3.1. UML

Pour cette option, il existe trois possibilités.

L'édition directe conformément au méta-modèle Ecore[3]

Il s'agit là, d'éditer des modèles graphiques UML conformément au méta-modèle *Ecore*. Cela sous-entent l'utilisation d'un outil de modélisation qui en soit capable[4]. Par voie de

3 *Ecore et le méta-modèle EMF pour le Core Modele (Voir § 4 : Le (méta ?)méta-modèle pivot : Ecore)*
4 *[BUDINSKY et al. 2004] cite en exemple Omondo(free) Eclipse UML.*

conséquence, la transformation d'entrée du modèle n'est plus : on dispose du *Core Model* sous le bon format.

L'importation de modèles UML

Il s'agit d'importer, à laide de la fonction ad hoc de EMF, un modèle dans son format natif (qui dépend de l'outil de modélisation utilisé). Seul le format IBM Rational (*.mdl*) permet de profiter de cet avantage. En effet, la gamme Rational et *Eclipse* sont des projets « frères » donc « génétiquement » compatibles.

L'exportation de modèles UML

C'est à peu près le même principe que l'option d'importation, sauf que la conversion du format natif en Ecore ne se fait pas avec EMF, mais avec l'outil de modélisation d'origine.

Discussion à propos des trois options

La première option ci-dessus présente l'avantage d'être simple et directe, de ne pas nécessiter d'opération d'importation ou d'exportation. Aucune synchronisation entre l'outil de modélisation natif et EMF n'est nécessaire.
Les deux autres options (équivalentes du point de vue de la conversion) offre l'avantage que l'outil de modélisation peut servir à autre chose que la « simple » modélisation. Par exemple, il peut proposer sa propre fonction de génération de code. Il donne aussi la possibilité de créer son propre *Core Model*, de le transformer en Ecore pour l'utiliser ainsi par la suite.

3.2. XMI

Ce format de fichier, standard de l'OMG (*Object Managment Group*), est utilisé conjointement à UML :

- UML se charge de décrire les contenus des modèles,
- XMI se charge de formater ces contenus pour permettre de leur assurer une persistance standardisée.

Malgrès quelques problèmes de maturité (en phase de résolution) qui demandent une attention particulière quant à l'association de différentes versions de UML avec les différentes versions de XMI [BLANC 2005], ce format tend à devenir le standard pour l'échange de données (et notamment des modèles) entre les différents outils du génie logiciel.

L'*Illustation 1* ne fait pas directement référence à ce format, mais il est permit de penser que ce choix est pertinent pour les quelques raisons suivantes :

- comme nous l'avons vu ci-dessus, il tend à devenir un standard, du moins pour le développement orienté objet (aussi le credo de Java),
- il est le standard utilisé par Ecore pour sa propre persistance,
- tout cela concoure à combler le fossé qui existe entre les modèles UML et les fichiers de code Java.

3.3. Java Annoté

Une des solutions tentantes pour modéliser les classes, qui vont être concrétisées par une application Java, est d'utiliser les interfaces Java :

- elles n'implémentent pas les méthodes : on s'abstrait donc de cette implémentation,
- les méthodes *get/set* peuvent être utilisées pour modéliser les attributs,

- une classe pourra implémenter plusieurs interfaces, ce qui est une manière détournée d'autoriser l'héritage multiple (impossible en Java de classe à classe et possible en UML),
- cela permet une évolution « douce » vers la modélisation des plus irréductibles codeurs.

Les annotations sont des *tags @model* placés dans la *javadoc* des interfaces. Ces *tags*, et leurs attributs éventuels, sont détectés par EMF qui considère ainsi les entités concernées (interfaces, méthodes) comme des éléments de modélisation.

4. Le (méta ?)méta-modèle pivot : Ecore

Nous avons vu que le *Core Model*, pivot des transformations possibles avec EMF, doit pouvoir modéliser les correspondances entre plusieurs types de modèles. On dirait, pour être puriste, que ces différents modèles sont conformes à autant de méta-modèles.
Le tableau ci-dessous synthétise les cas en présence pour EMF.

Tableau II - Les espaces techniques de modélisation à fédérer.

Modèles (M1)[5]	**Méta-modèles (M2)**	**Méta-méta-modèles (M3)**
Diagramme de classes (entrée)	UML	MOF
Fichier XML (entrée)	XMI	MOF
Ensemble d'interfaces Java (entrée)	Java Annoté	EBNF[6]
Programme Java (sortie)	Java	EBNF

Pour être fidèle à l'IDM, les transformations sont définies au niveau des méta-modèles. Cette stratégie permet de modéliser les règles de transformation (il faut là aussi un méta-modèle[7]), et ainsi de capitaliser les efforts faits pour leur définition.
La tableau ci-dessous représente l'espace technique standard de EMF.

Tableau III - L'espace technique de modélisation fédérateur.

Modèle (M1)	**Méta-modèle (M2)**	**Méta-méta-modèle (M3)**
MEcore (pivot)	MMEcore	Ecore

Il est hors de question de faire ici un exposé détaillé sur Ecore, mais voici quelques éléments de compréhension.
Ecore est un méta-méta-modèle très proche de MOF. Il est en fait un sous ensemble de MOF : il restreint celui-ci (l'*Annexe 1* en donne quelques éléments).
En effet, une des particularités de Ecore est qu'il accepte des méta-classes (dans le niveau *M2*)

5 *Les niveaux M1, M2, M3 sont les niveaux d'une pile de modélisation définis par l'OMG. Le niveau M0 est la réalité que l'on modélise : pour nous il s'agit de l'application informatique que l'on développe.*
6 *EBNF : Extended Backus-Naur Form. Langage de définition de langages de programmation.*
7 *L'OMG en stadardise un : QVT (Query – View – Transformation).*

sans associations. Pour associer deux classes, il faudra stéréotyper un attribut comme étant une association. Cette possibilité à été mise au point car en langage Java le concept d'association n'existe pas [BLANC 2005].
En Java, les associations d'un diagramme de classes UML s'implémentent par la création d'un attribut ayant pour type la classe partenaire. L'attribut doit être créé dans l'une, dans l'autre ou dans les deux classes partenaires (dans le cas d'une association binaire) selon la navigabilité de l'association.

Au niveau du méta-modèle l'on définira les caractéristiques du modèle de niveau M1. Par exemple, on définira les concepts d'un diagramme de classes UML si c'est cela que l'on veut traiter.
En l'occurrence, la seule entrée UML possible est le diagramme de classe, mais on peut imaginer de méta-modéliser d'autres diagrammes (cas d'utilisation, séquence,...) : cela permettrait d'utiliser EMF à d'autres stades du cycle de développement (transformation CIM vers PIM de MDA, par exemple – Voir [BARZIC 2006]).

Le caractère fédérateur de EMF et de son méta-méta-modèle Ecore tient dans le fait que les différents modèles d'entrée et de sortie (voir *tableau II*) ont leur méta-modèle Ecore (ceux cités sont fournis en standard) et les transformations de l'un à l'autre se feront en appuyant sur un (ou quelques) bouton.

5. Génération du code

Il est temps maintenant d'observer ce que va produire EMF : le code. C'est bien là son objectif premier.
Comme l'illustre les exemples pratiques proposés § 8, EMF répond bien a son objectif d'améliorer la productivité du développement d'application. Il y arrive en automatisant la génération du code à partir du modèle. Effectivement, une fois le modèle créé, « quelques clics » suffisent à cette génération

Nous n'allons pas ici entrer dans une analyse détaillée du code contenu dans les éléments générés : nous nous contenterons, dans cette introduction à EMF, d'une liste (non exhaustive) de ce qui est généré. Une analyse détaillée demanderais, au préalable, une étude plus approfondie de EMF et de Ecore[8].

5.1. Organisation du code généré

Un choix de conception a été fait par ses concepteurs et est imposé par EMF : la séparation interface/implémentation dans le code généré.
Cela va se concrétiser par la génération de deux ensembles (issus du modèle d'entrée, assimilable à un diagramme de classes UML) :

1. un ensemble d'interfaces Java,
2. un ensemble de classes implémentant ces interfaces.

Les raisons principales qui justifient ce choix sont : la correspondance avec un *pattern* utilisé par de nombreuses API, la nécessité de pouvoir disposer d'un héritage multiple (impossible en Java sans la notion d'interface).
En plus des classes correspondant au modèle d'entrée, EMF génère deux autres éléments

8 *À ce stade, l'auteur avoue ne pas avoir assez avancé dans l'étude du sujet pour cela.*

importants : une interface *Factory* et une interface *Package* ainsi que leurs classes d'implémentation.

La Factory

Cette interface comprend une méthode *create* pour chacune des classes du modèle d'entrée. Cela va permettre de créer des instances (des objets) des classes de l'application.
Le modèle de programmation EMF incite fortement à utiliser ces méthodes pour créer les objets lors de l'utilisation de l'application, en lieu et place de l'opérateur *new*.

Le Packetage

Cette classe apporte des facilités pour accéder aux méta-données Ecore du modèle. Il contient des accesseurs aux *EClasses*, EA*ttributes* et *EReferences*[9] implémentées dans le modèle, par exemple.

5.2. La re-génération et la fusion

Une des caractéristiques avantageuses de EMF est qu'il va permettre de compléter manuellement le code obtenu automatiquement, de pouvoir re-générer le code à partir du modèle modifié sans perdre les ajouts faits manuellement.

En effet, il est indispensable de pourvoir ajouter des méthodes et des attributs au code généré automatiquement. Celui-ci s'occupe uniquement (mais c'est déjà pas mal !) de générer le squelette des classes, les références aux autres classes (les associations) et les accesseurs aux attributs (les *get* et les *set*).

Les éléments générés automatiquement seront repérés par le *tag @generated* dans la *javadoc*. Lors d'une re-génération, suite à une modification du modèle, seuls ces éléments seront retouchés. En cas de conflit avec une modification manuelle, c'est cette dernière qui prime.

5.3. Le modèle générateur

En plus du modèle conforme au méta-méta-modèle Ecore, EMF utilise un modèle dit générateur (fichier d'extension *.genmodel*). Ce modèle, comme le modèle Ecore, est généré automatiquement (donc de manière transparente pour l'utilisateur) lors de la transformation du modèle d'entrée en modèle Ecore.

La plupart des informations nécessaires sont contenues dans le modèle « core » : le nom des classes les attributs, les références,...

Mais un certain nombre d'informations n'y sont pas telles que : les règles de préfixation du nom des classes, où mettre le code généré,...
Ces informations de paramétrage de la génération seront stockées dans le modèle générateur.

L'avantage de cette séparation (du générateur et du « core ») est que le méta-méta-modèle Ecore reste indépendant de toutes informations relevant de la stricte génération du code.
L'inconvénient est qu'il faut assurer une synchronisation des deux modèles en cas de modification (pour en garder la cohérence). EMF assure automatiquement cette synchronisation.

9 *Les élément Exxxx sont des éléments du méta-méta-modèle Ecore (voir annexe 1).*

MCours.com

6. Autres services proposés par EMF

En plus d'être un outil d'amélioration de la productivité, EMF a d'autres apports, tels que : la notification de modification du modèle, la gestion de la persistance par une sérialisation XMI, une API réflective pour la manipulation générique des objets EMF, l'EMF dynamique. La combinaison de tous ces services (et d'autres non cités) fait qu'EMF offre un socle pour l'inter-operabilité avec d'autres outils et applications basés sur EMF.
Nous ferons ici une simple évocation de ces possibilités. Pour en savoir plus, je renvoie le lecteur aux différentes documentations (voir bibliographie, et ailleurs aussi).

6.1. La notification et les *Adaptateurs*

Lors de l'observation du code des classes d'implémentation générées par EMF, on trouve dans les accesseurs de type *set* le code ci-dessous (extrait de la classe *AuteurImpl.java* de l'exemple du § 8) :

```
public void setNom(String newNom) {
	String oldNom = nom;
	nom = newNom;
	if (eNotificationRequired())
		eNotify(new ENotificationImpl(this, Notification.SET,
		BibliothequePackage.AUTEUR__NOM, oldNom, nom));
}
```

Le test conditionnel permet de vérifier si une notification de modification a été demandée sur l'élément (ici un attribut) et, le cas échéant, de l'opérer.

Cette possibilité de EMF est il intéressante : l'on va pouvoir transmettre automatiquement aux objets dépendants une modification faite sur un attribut ou une référence.

Dans EMF les observateurs de notification (les *listener* Java) son appelés *Adaptaters* (Adaptateurs). En effet, en plus de leur statut d'observateur, il peuvent modifier les comportements des objets auxquels ils sont attachés.

6.2. La gestion de la persistance

EMF fourni un mécanisme simple, et néanmoins puissant, pour la gestion de la persistance des objets.

Comme nous l'avons vu, le standard générique utilisé par EMF pour stocker les modèles est XMI. EMF propose aussi un mécanisme pour utiliser cette possibilité, déjà implémentée, pour sérialiser les objets des applications. Cela dégage l'utilisateur de la mise en place d'une sérialisation propre.

Le principe général consiste à créer des entités *Ressources* (classe *RessourceSetImpl*) et d'y placer les objets que l'on veut sérialiser. Il sont ainsi à disposition par la suite.

6.3. L'API réflective

Comme tous les éléments des modèles EMF héritent de la classe *EObject*[10] du méta-méta-modèle Ecore, l'utilisateur peut se servir de l'API réflective pour manipuler leurs instances.
Il pourra, par exemple, accéder directement à la valeur d'un attribut d'objet sans faire appel

10 *Voir Annexe 1.*

aux accesseurs (*get*, *set*) de sa classe. Des méthodes d'accès plus génériques existent.
Bien que cette technique d'accès est moins performante que l'accès direct par les accesseeurs des classes, elle permet une ouverture vers l'extérieur des modèles grâce à la généricité des méthodes de l'API réflective.

6.4. L'EMF dynamique

Jusqu'à présent nous avons utilisé EMF pour générer une implémentation de nos modèles. Dans de nombreux cas, nous ne désirerons pas implémenter le modèle Ecore de notre application. Il suffit pour une observation de l'architecture.

La particularité de l'API réflective (voir § 6.3) est qu'elle peut générer dynamiquement les classes nécessaires (et uniquement celles là) lors de l'utilisation d'une de ses méthodes.

En fait, le modèle Ecore est le seul nécessaire pour pouvoir utiliser l'application : seul un allongement du temps d'accès sera perçu.

7. EMF et les standards OMG

Des discussions ont cours à propos des relations qu'entretient EMF avec les différents standards de l'OMG que sont UML, XMI, MOF et MDA.
Ce paragraphe est de pure culture, en faire l'impasse ne nuira pas à la compréhension et l'expérimentation de EMF.

7.1. Pour UML

UML est un méta-modèle très utilisé pour modéliser les applications du monde objet. Les différents diagrammes conformes à UML sont prévus pour permettre autant de représentations d'une même application. Entre autres :

- la vue utilisateur : les cas d'utilisation,
- la vue dynamique : le diagramme de séquence,
- la vue architecturale : le diagramme de composants,
- la vue statique de conception : le diagramme de classes.

C'est sur cette dernière qu'intervient actuellement EMF : c'est le diagramme de classes qui sera utilisé pour générer le code (Java en l'occurrence).

7.2. Pour MOF

MOF (*Meta Object Facility)* est le méta-méta-modèle standard de l'OMG. Il est utilisé pour définir les méta-modèles promus par cette organisation. Citons les plus caractéristiques de l'IDM : UML, QVT, CWM (*Commun Warehouse Metamodel*)[11].

Ecore et MOF ont de nombreuses similitudes. Les différences se situent au niveau de la couverture des différents concepts tels que ceux de classes, de types de données, d'attributs, de relations entre paquetages et classes (voir § 4 de cette partie).

L'on peut aussi noter que le projet EMF, enfant du projet *Eclipse*, et son retour d'expérience ont une influence non négligeable sur les travaux de standardisation de MOF et/ou UML.

11 *CWM est utilisé pour définir des modélisations d'entrepôts de données et plus largement d'applications de BI (Business Intelligence).*

7.3. Pour XMI

XMI est un standard créé par l'OMG basé sur XML. Ce dernier est lui même un standard porté par le W3C (*World Wide Web Consortium*)[12].
Ce standard a été créé pour faciliter la sérialisation et les échanges de données, dans le cadre de la modélisation, entre les différents outils qui interviennent dans le cycle de vie d'une application informatique.
Il s'appuie sur les mécanisme de DTD (*Document Type Definition*) ou de schéma XML pour définir les structurations de balises nécessaires et suffisantes à la représentation des modèles MOF au format XML [BLANC 2005].

XMI peut être utilisé pour sérialiser toute sortes de modèles utilisés par EMF, il est aussi utilisé pour le méta-méta-modèle Ecore lui-même et comme forme canonique des fichiers « Ecore » (*.ecore*).

7.4. Pour MDA

Model Driven Architecture est une démarche d'ingénierie promue par l'OMG. Elle est basée sur la manipulation de différents modèles représentant l'application cible (indépendant de l'informatisation, indépendant de la plate-forme d'exécution, spécifique à cette plate-forme, de la plate-forme elle même, le code) et, par voie de conséquence, sur des transformations de modèles (voir [BARZIC 2006] et sa bibliographie).

EMF est bien dans la philosophie de cette démarche et peut s'intégrer dans l'outillage nécessaire comme générateur de code à partir de modèles (ce qui est l'objectif premier de EMF).

8. Un peu d'exercice

Ce paragraphe « pratique » est inspiré largement de [ECLIPSE-ORG-EMFa].
J'ai choisi, pour une question de longueur du document, de mettre ici le moins possible de copie d'écran, et plutôt d'utiliser les parcours dans les menus. Le lecteur pourra se référer au document de référence ci-dessus, en anglais, pour une version illustrée.
Le lecteur peut télécharger les sources sur le site www.289eme.fr (voir introduction).

8.1. Installer le plugin EMF

Avec *Eclipse*, il y a deux façons de procéder :

1. télécharger directement l'archive du plugin et la décompresser dans le répertoire d'installation de *Eclipse*,
2. utiliser la recherche et l'installation automatique du menu *Aide->Mise à jour de logiciels->Rechercher et installer...*

Bien que la deuxième solution apporte une productivité indéniable, nous utiliserons ici la première : on y fait tout « volontairement » et par étape, cela permet une meilleure pédagogie dans le cadre de ce type de document.

12 *W3C est un consortium fondé en octobre 1994 pour promouvoir la compatibilité des technologies du World Wide Web telles que HTML, XHTML, XML, RDF, CSS, PNG, SVG et SOAP. Le W3C n'émet pas des normes au sens européen, mais des recommandations à valeur de standards industriels [WIKIPEDIA-FR 2006].*

Étape 1 : télécharger les archives nécessaires.

Aller à l'adresse *http://www.eclipse.org/emf/downloads/*.
Choisir la version désirée : au moment de la rédaction c'est la version 2.2.x car c'est celle qui est cohérente avec le *Eclipse Calisto* (voir 1ère partie).

- J'ai installé la version 2.2.1 (du 21 septembre 2006).
- Archive téléchargée :
  - ✗ *emf-sdo-xsd-SDK-2.2.1.zip*,
  - ✗ *Nlpacks-emf-sdo-SDK-2.2.1.zip* pour les langues.

Étape 2 : décompression - installation

Décompresser les deux archives dans le répertoire d'installation de *Eclipse* et redémarrer éventuellement le logiciel.

8.2. Disposer d'un modèle d'entrée

Préliminaire

Comme nous l'avons vu EMF accepte différents formats de modèles en entrée (UML, XMI, Java Annoté). Pour simplifier cet exercice, et pour limiter le nombre d'outils en jeu, nous allons préparer un modèle en Java Annoté : on peut le créer directement dans *Eclipse*.

Le modèle réalisé

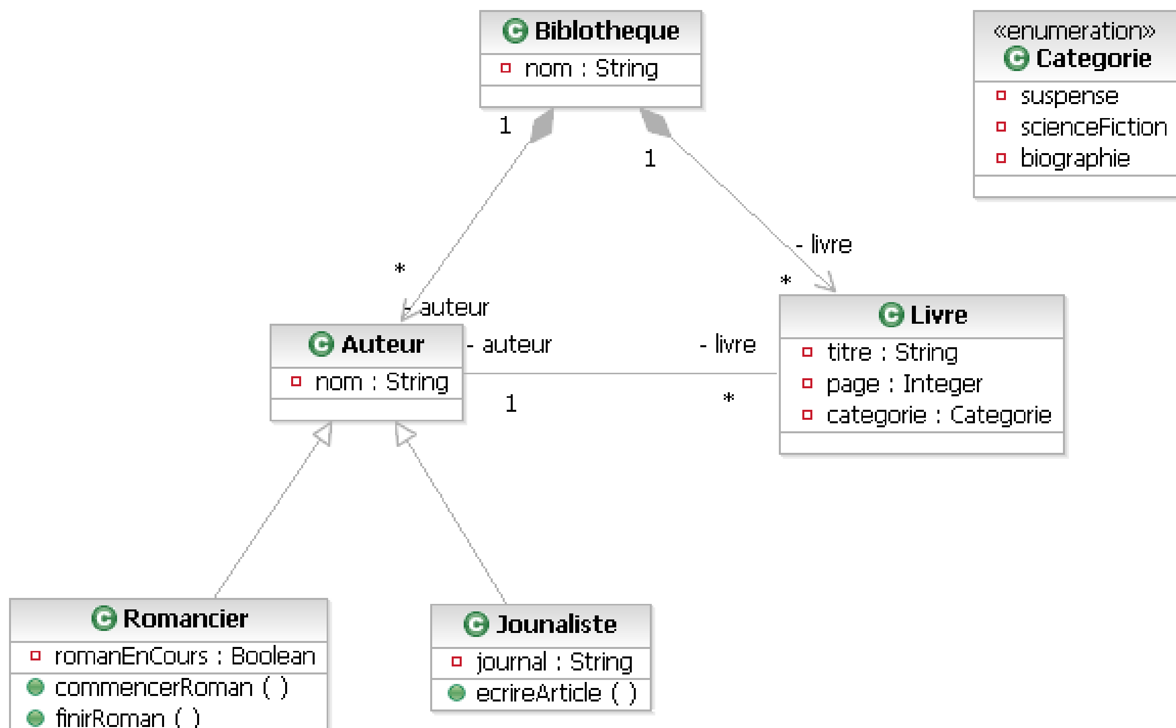

Illustration 2 - Le modèle de l'exemple

Les étapes

1. ***Créer un projet EMF vide***

- Menu : *Fichier->Nouveau->Projet...*
- Choisir : *Eclipse Modeling Framework (EMF)->Projet EMF vide* et *Suivant.*
- Nommer le projet et modifier éventuellement son emplacement et *Terminer*.

2. ***Créer les interfaces en Java Annoté***

- Clic droit sur le répertoire *src* du projet.
- Menu : *Nouveau->Interface*.
- Renseigner la boîte de dialogue : le nom du paquetage et le nom de l'interface. Laisser les autres options aux valeurs par défaut, puis *Terminer*.
- Coder les autres interfaces conformément au modèle (attention *Categorie* est codée comme une classe).

À ce stade, le projet *bibliotheque* est contenu dans : *bibliotheque_2_8.2* de l'archive *eclipse_emf_gef_gmf_sources.zip.*

8.3. Générer le modèle *ecore*

Suivre la procédure ci-dessous.

- Clic droit sur le répertoire *model* du projet (*bibliotheque/model* pour l'exercice).
- Menu : *Nouveau->Autres...*
- Choisir : *Eclipse Modeling Framework (EMF)->Modèle EMF* puis *Suivant.*
- Donner un nom au modèle (le nom doit avoir l'extension *.genmodel*), puis *Suivant*.
- La boîte de dialogue suivante propose de choisir le type de modèle d'entrée (*Java annoté*, *Modèle de classe Rose*, *Modèle Ecore*, *XML Schéma*). Nous choisissons *Java annoté*.

L'arborescence dans l'*Explorateur de packages* d'*Eclipse* est complété de deux fichiers dans le répertoire *model* (le nom *bibliotheque* est celui de l'exemple) :

- *bibliotheque.genmodel :* c'est un modèle qui contient les paramètres de génération du code de l'application. Ce fichier est nommé le générateur (*generator* en anglais).
- *bibliotheque.ecore :* c'est le méta-modèle de l'application, utilisé ici comme modèle, conforme au méta-méta-modèle *Ecore.*

À ce stade, le projet *bibliotheque* est contenu dans : *bibliotheque_2_8.3* de l'archive *eclipse_emf_gef_gmf_sources.zip*

8.4. Descrire le modèles *.genmodel*

Lors de la génération du modèle *ecore* (paragraphe précédent), un onglet *blibliotheque.genmodel* est ouvert automatiquement dans la fenêtre principale de *Eclipse* (voir l'arborescence *Illustration 3*)

Cela tombe bien car nous allons nous en servir pour la génération du code.

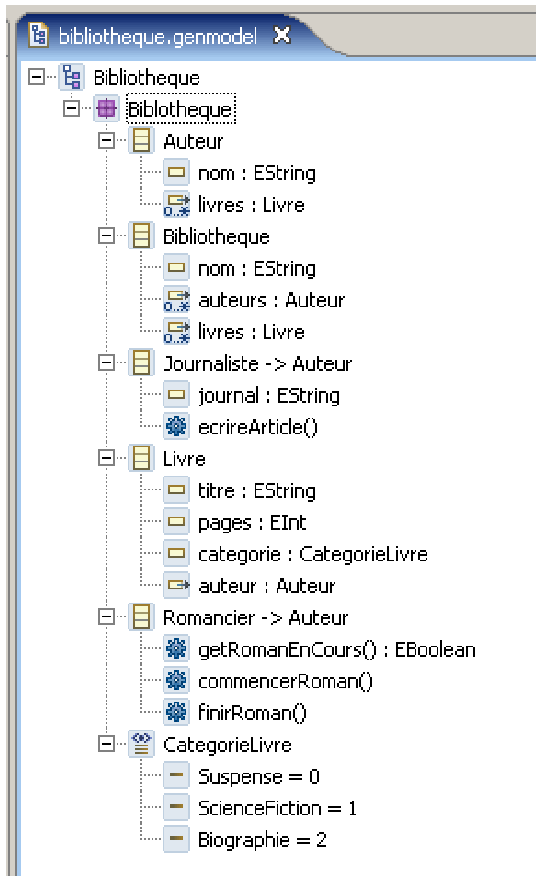

Illustration 3 - Arborescence du modèle Ecore de notre projet « Bibliothèque ».

- Le modèle a un enfant qui représente le paquetage,
- ce dernier a des enfants qui sont les classes, les types de données,...
- les enfants des classes sont les attributs, les références, les opérations.

En bas de l'espace de travail *Eclipse*, un onglet *Propreté* est normalement affiché.
Dans le cas contraire :

- clic droit sur la racine du modèle,
- choisir *Afficher la vue propriétés*.

Les propriétés permettent de contrôler complètement ce qui sera généré Dans la plupart des cas, les propriétés par défaut des différents éléments n'ont pas à être changées. L'étude détaillées de ces propriétés dépasse largement le cadre de cette étude.

8.5. Générer le code (vu ici comme un modèle)

Procédure

Elle est toute simple :

- faire un clic droit sur la racine du modèle,
- choisir *Générer le code de modèle,*
- l'opération se déroule automatiquement.

Que c'est il passé ?

1. Les interfaces déjà existantes (en Java Annoté, c'était notre modèle de départ créé § 8.2) sont complétées. Elles auraient été créées si le modèle original avait été importé de *Rationnal Rose*.
2. Deux nouvelles interfaces ont été créées :
   - une pour le paquetage lui même (*BiblothequePackage.java* pour notre projet),
   - une pour la *factory* (*BiblothequeFactory.java* pour notre projet).
3. Deux nouveaux paquetages ont été créés (ils ont le même nom que l'original avec respectivement les suffixes *.impl* et *.util*.) :
   - *.impl* : contient les classes qui implémentent les interfaces définies dans le modèle de génération,
   - *.util* : contient des classes utilitaires.
4. Différents fichiers « *manifest* » sont générés : plugin.xml,...

MCours.com

Un avantage de EMF

Si on modifie le modèle, on eut le re-générer en suivant la même procédure. Dans ce cas, les changements sont fusionnés avec les changements manuels que l'on à pu faire sur le code (sans perte de ces derniers).
On peut aussi générer uniquement une partie du code : il suffit, lors de la procédure ci-dessus, de faire un clic droit sur l'élément que l'on souhaite (re)générer (paquetage, classe,...) et non pas sur la racine du modèle.

À ce stade, le projet *bibliotheque* est contenu dans : *bibliotheque_2_8.5* de l'archive *eclipse_emf_gef_gmf_sources.zip*

8.6. Autres générations possibles

Le menu de génération du modèle propose, en plus de la génération du code, quatre possibilités :

- *Générer le code d'édition,*
- *Générer le code de l'éditeur*,
- *Générer le code du test*,
- *Générer tout* : génère en une seule commande le code et les trois modèles ci-dessus.

Les deux premiers vont servir à faire fonctionner l'application que l'on vient de développer comme un plugin d'*Eclipse*. On va ainsi, pouvoir créer une instance de l'application, telle que le ferait un de ces futurs utilisateurs : on va, dans le cas de notre exemple, créér une bibliothèque et la remplir de livres (avec leurs auteurs, leurs catégories,...). et la faire fonctionner.
Le dernier générera un plugin de test compatible *JUnit*.

9. Conclusion

À travers cette étude théorique et introductive de EMF, ainsi que par l'exemple décrit dans le paragraphe précédent , nous avons pu mieux cerner les buts et le fonctionnement du plugin.

Ce que l'on peut retenir :

1. EMF intervient au moment de la génération du code à partir d'un modèle : son domaine est le langage Java (donc aussi le monde objet) en standard,
2. EMF permet de prendre en entrée plusieurs formats de modèles (UML, Java Annoté, Schéma XML, XMI) : il peut donc être compatibles avec d'autres outils utilisés en amont dans le cycle de vie du logiciel,
3. une fois que l'on dispose du modèle d'entrée, les transformations se passent très facilement : quelques clics suffisent,
4. le code généré sera aussi précis et complet que le sera le modèle d'entrée : EMF pourra donc s'adapter à l'évolution des recherches (et donc des outils) qui fabriquerons les modèles du futur,
5. la génération du code à partir du modèle n'empêche pas son raffinage manuel : une re-génération après une modification du modèle n'écrase pas les compléments.

3ème Partie : Graphical Editing Framework (GEF)

4ème Partie : Graphical Modeling Framework (GMF)

5ème Partie : EMF, GEF et GMF en action

Conclusion

EntréeIndex

EntréeGlossaire

Annexe 1 : le méta-méta-modèle Ecore (simplifié)

Les digrammes ci-dessous sont issus de [VANWORMHOUDT 2004].

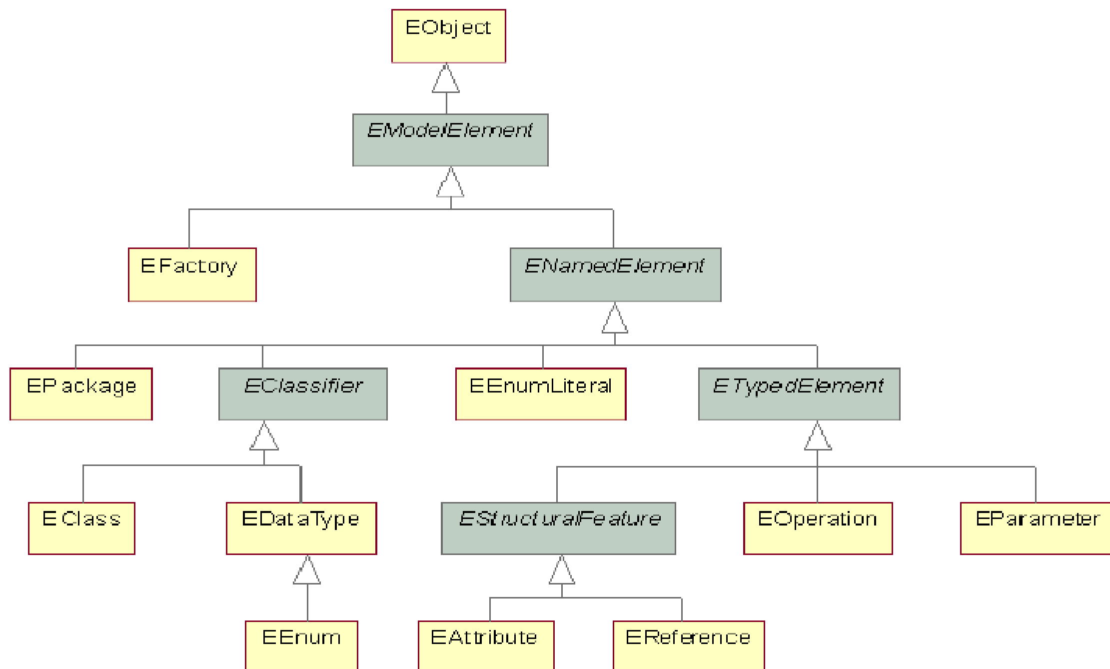

Illustration 4 - Méta-méta-modèle Ecore : hiérarchie des classes

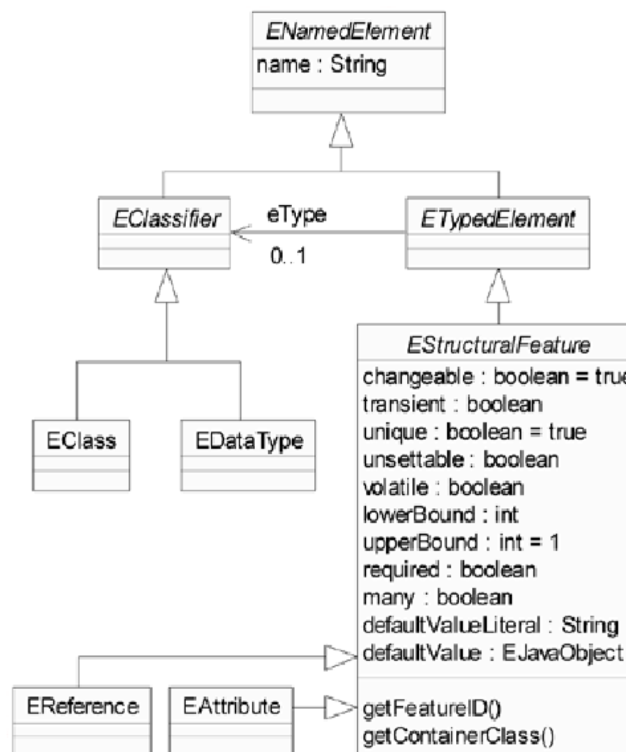

Illustration 5 - Méta-méta-modèle Ecore : éléments (1)

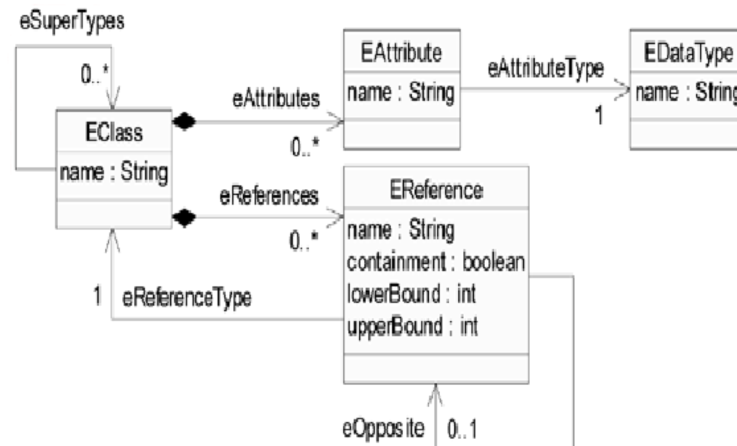

Illustration 6 - Méta-méta-modèle Ecore : éléments (2)

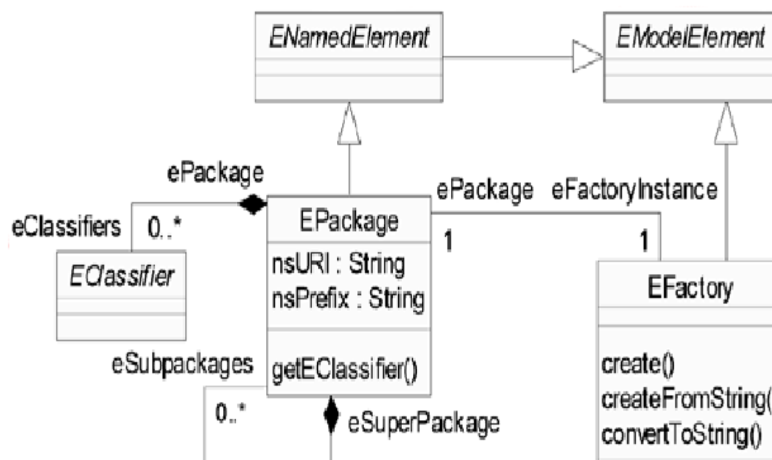

Illustration 7 - Méta-méta-modèle Ecore : éléments (3)

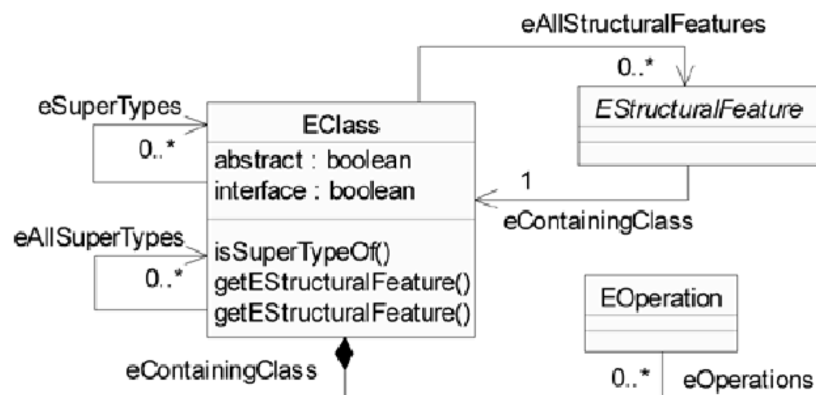

Illustration 8 - Méta-méta-modèle Ecore : éléments (4)

Bibliographie

[BARZIC 2006] BARZIC J., Model Driven Architecture (MDA), 2006. Disponible à l'adresse : http://www.289eme.fr/pdf/mda.pdf

[BLANC 2005] BLANC X., 2005. *MDA en action. Ingénierie logicielle guidée par les modèle*. . Eyrolles, Architecte logiciel, PARIS. 269 p, .

[BUDINSKY et al. 2004] BUDINSKY F., STEINBERG D., MERKS E., ELLERSICK R., GROSE T.J., 2004. *Eclipse Modeling Framework*. . Addison Wesley Professional, The Eclipse series, Lebanon. 720 p, 2ème chapitre disponible ici : http://www.eclipse.org/emf/docs/.

[ECLIPSE TOTALE] , 2006. . http://www.eclipsetotale.com.

[ECLIPSE-ORG-EMFa] , 2006. Generating an EMF Model. http://dev.eclipse.org/viewcvs/indextools.cgi/org.eclipse.emf.doc/tutorials/clibmod/clibmod.html.

[SUN JAVA a] , 2006. Page de téléchargment Java. http://developers.sun.com/resources/downloads.html.

[VANWORMHOUDT 2004] VANWORMHOUDT G., Vérification de modèles avec EMF et OCL, Présentation PPT, 2004. disponible ici : http://www.enic.fr/people/Vanwormhoudt/siteEMFOCL/documents/EMFOCLpresentation.pdf.

[WIKIPEDIA-FR 2006] Collectif, 2006. Divers documents. http://fr.wikipedia.org.

Index

EntréeIndex 20

Glossaire

EntréeGlossaire 20

RESUME

Mots clés : France, Bretagne,

SUMMARY

Key words: France, Britanny,

MCours.com