REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE

MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE

N° d'ordre : ……………

MEMOIRE de fin d'étude

Présenté pour l'obtention du diplôme de MASTER

Domaine : Mathématiques et Informatique

Filière : Informatique

Spécialité : Systèmes d'Informations Avancés

Par : Taisse Hassen

Ben Loglayad Ali

SUJET :

Une Approche MDA pour la génération du code OWL2 à partir des diagrammes de classe

Soutenu publiquement le :/   /2021 devant le jury composé de :

………………………	**Université de M'sila**	**Président**
Med. …………………	**Université de M'sila**	**Rapporteur**
……………………	**Université de M'sila**	**Examinateur**
……………………	**Université de M'sila**	**Examinateur**

Dédicaces

A mes parents

Je vous dois ce que je suis aujourd’hui grâce à votre amour, à votre patience et vos innombrables sacrifices. Que ce modeste travail, soit pour vous une petite compensation et reconnaissance envers ce que vous avez fait d’incroyable pour moi.

Que Dieu, le tout puissant, vous préserve et vous procure santé et longue vie afin que je puisse à mon tour vous combler.

A mes très chers frères

Aucune dédicace ne serait exprimer assez profondément ce que je ressens envers vous, je vous dirais tout simplement, un grand merci, je vous aime.

A mes très chers amis

En témoignage de l’amitié sincère qui nous a liées et des bons moments passés ensemble. Je vous dédie ce travail en vous souhaitant un avenir radieux et plein de bonnes promesses.

Remerciements

En tout premier lieu, je remercie Allah le tout puissant, à la sagesse et au savoir infinis, « Gloire à toi ! Nous n'avons de savoir que ce que Tu nous as appris. Certes c'est Toi l'Omniscient, le sage, le tout miséricordieux le très miséricordieux » (Sourate al-Baqarah, verset 32).

Je tiens à remercier mon encadreur DR BENOUIS MOHAMED pour le grand honneur qu'il m'a fait en me proposant le sujet de ce mémoire de fin d'étude. J'ai eu l'honneur et le privilège de travailler sous son assistance et de profiter de ses qualités humaines, professionnelles et de sa grande expérience, il m'a guidé tout au long de ce travail. L'élaboration avec amabilité et dynamisme le caractérisant. Que ce modeste travail puisse satisfaire mes examinateurs, pour qu'ils en témoignent ma gratitude et reconnaissance pour l'aide et les conseils qu'il ma prodigué, ainsi que pour la savoir qu'il ma inculqué.

Je remercie tous mes enseignants de l'université de M'sila.

Mes remerciements vont également aux membres de jury d'avoir accepté de juger mon travail.

Je remercie vivement toute ma famille, en particulier mes parents, pour m'avoir toujours soutenu au cours de mes études. Qu'ils trouvent ici le fruit de leur patience et du soutien permanent qu'ils m'ont prodigué pour affronter tous les moments difficiles.

Merci pour tous ceux qui, m'ont aidé de près ou de loin à réaliser ce travail.

Table des Matières

Table des matières I

Liste des figures II

Introduction générale......…………………………………………………………… IX

Chapitre 1:diagramme de classe

1/diagrammes de classe…………………………………………………………………………… 1

1.2 */Premier niveau de modélisation des données d'une application*…………… 2

1.2.1 Les éléments de modélisation……………………………………………………… 3

- ✓ Objectif………………………………………………………………………………. 4

1.2.2. Diagramme de classes et d'objets……………………………………………………… 5

- ✓ Objectif……………………………………………………………………………… 6

2/Diagramme de classes et d'objets………………………………………………………… 7

2.1/. Lien entre objets et associations entre classes………………………………………… 8

2.2/Multiplicités (cardinalités)………………………………………………………………. 9

2.3 *Les paquetages*…………………………………………………………………………… 10

3 */Les interfaces*…………………………………………………………………………… 11

4/ *Les classes*…………………………………………………………………………………… 12

4.1 *Classes abstraites*…………………………………………………………………………… 13

4.2 *Classes non abstraites* …………………………………………………………………… 14

4.3/ *Les relations entre les classes*……………………………………………………………. 15

9.2.4 *Héritage*…………………………………………………………………………………… 16

9.2.5 *Implémentation*……………………………………………………………………………… 17

9.2.6 *La relation d'agrégation*…………………………………………………………………… 18

9.2.7 *Relation de dépendance*…………………………………………………………………… 19

21

Chapitre02: les ontologie et owl2

1/Introduction ………………………………………………………………………… 22

2.1/ Ontologies et outils du Web Semantic ………………………… 23

2.2.1/ Web Sémantique……………………………………………………………….. 24

2 /Qu'est-ce que OWL 2 ?.. 25

3. Fondements de l'ingénierie ontologique …………………………….. 26

3.1/ Définition ……………………………………………………………………… 27

3.2/Processus…………………………………………………………………………… 28

4/ Codage de l'ontologie ……………………………………………………… 29

5. Intégration d'ontologies existantes……………………………………… 30

6.Les ontologies……………………………………………………………………… 31

Chapitre03: modélisation méta modélisation et transformation des modèles

1/Modèle…………………………………………………………………………………….. 32

2/Meta_modèle………………………………………………………………………….. 33

2.1 Un méta-modèle pour le diagramme de classe…………………………….. 34

3/La génération d'un outil pour le diagramme classe……………………….. 35

4/L'ingénierie dirigée par les modèles, ou Model-driven Architecture 36

5/MDA…………………………………………………………………………………… 37

Chapitre 04 :approche propose

1/Introduction………………………………………………………………………. 38

2/Processus de transformation des modèles avec TGG ………………………… 39

3/Méta-modélisation en EMF………………………………………………………. 40

41

42

43

4/ Génération d'un outil EMF pour « les diagrammes de classe » ………………………………

5/ Transformation du modèle …………………………………………………………………

6/ L'implémentations ………………………………………………………………………

7/ Conclusion………………………………………………………………………………

8/Conclusion générale………………………………………………………………………

9/ Résume ……………………………………………………………………………………

10/ Abstrait…………………………………………………………………………

11/ *Abstract*………………………………………………………………………

12/ Références……………………………………………………………………

44
45
46
47
48
49
50
51
52
53

API : *Application Programming Interface*.

BI : *Business Intelligence*.

CWM : *Commun Warehouse Metamodel.*

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

EBNF : *Extended Backus-Naur Form.*

EMF : *Eclipse Modeling Framework.*

EPL : *Eclipse Public License.*

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 Langage.*

XSL : ex*tensible Stylesheet Language.*

Chapitre 01 :

Introduction générale

Contexte

Aujourd'hui, le logiciel fait partie intégrante de notre quotidien vu les problèmes et les craintes engendrés par le passage au troisième millénaire.

Les systèmes modélisés sont de plus en plus complexe et leur taille ne cesse d'augmenter, ce qui a mis en évidence le besoin de construire des logiciels de qualité et de favoriser des techniques de développement rigoureuses et adéquates aux exigences de leurs utilisateurs.

Au cours des dernières années, un intérêt croissant du milieu académique et de l'industrie a accompagné cette prise de conscience afin de couvrir les phases amont du cycle de vie d'un logiciel, et particulièrement l'activité d'analyse des besoins et celle de spécification.

La première a pour rôle d'exprimer les exigences de l'utilisateur qui doit être validée par rapport aux objectifs poursuivis, alors que la deuxième vise à instaurer une première représentation du futur système. Toutefois, dans ces deux activités, la description des détails de réalisation doit être toujours évitée.

De nombreuses méthodes de conception de systèmes ont été introduites dans ce domaine, en s'appuyant sur des techniques, des théories ainsi que des notations différentes.

Parmi celles-ci, deux fameuses méthodes de conception qui ont vu le jour : les méthodes semi formelles à objets et les méthodes formelles.

La différence fondamentale entre ces deux familles est que la première propose des notations consistant à simplifier la compréhension d'une solution logicielle, tandis que la deuxième se distingue par sa capacité de fournir des bases mathématiques pour la vérification et la correction de la solution logicielle obtenue.

Non seulement notre satisfaction réside dans l'application des techniques formelles pour produire des logiciels sûrs, mais nous sommes également persuadés que les techniques semi formelles offrent à l'ingénieur logiciel un raisonnement abstrait sur les modèles, une lisibilité ainsi qu'une bonne compréhension de ces derniers. donc, dans le cadre de l'étude des notations et méthodes semi-formelles et de spécification formelle,

1/ *Les diagrammes de classe*

Le diagramme de classes constitue un élément très important de la modélisation : il permet de définir quelles seront les composantes du système final : il ne permet en revanche pas de définir le nombre et l'état des instances individuelles. Néanmoins, on constate souvent qu'un diagramme de classes proprement réalisé permet de structurer le travail de développement de manière très efficace; il permet aussi, dans le cas de travaux réalisés en groupe (ce qui est pratiquement toujours le cas dans les milieux industriels), de séparer les composantes de manière à pouvoir répartir le travail de développement entre les membres du groupe.
Enfin, il permet de construire le système de manière correcte (***Build the system right***).

*1.1. **Premier niveau de modélisation des données d'une application***

Objectif : définir un premier niveau de modélisation des données de l'application en faisant abstraction des aspects techniques (informatiques) qui se poseront lors de la réalisation de l'application. Les éléments de modélisation d'UML sont suffisamment généraux pour permettre aux utilisateurs non spécialistes en informatique (décideurs en entreprise, par exemple) de comprendre les éléments fondamentaux (essentiels) qui vont etre pris en compte lors de la réalisation de l'application.
Eléments de modélisation de ce premier niveau de modélisation : classe, attribut, association, multiplicité, rôle : diagramme de classes objet, donnée, lien : diagramme d'objets spécification formelle a l'aide d'expressions OCL complétant le diagramme de classes

1.2. Les éléments de modélisation

1.2.1. **Objet**

Objet : entité qui a des propriétés structurelles et comportementales.
Par exemple un compte bancaire a un numéro, un solde qui correspond au montant d'euros que le propriétaire du compte a confie a la banque. Sur ce compte ce dernier pourra déposer, retirer de l'argent ou effectuer des virements.

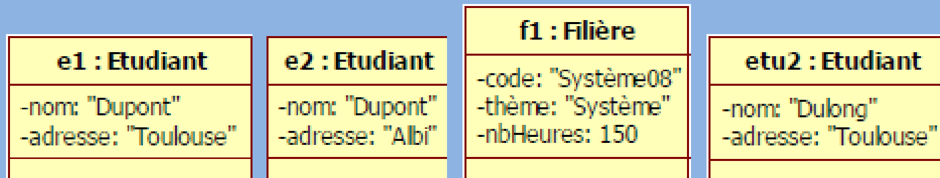
1.2.2. **Diagramme de classes et d'objets**

Diagramme de classes, diagramme d'objets : représentation graphique des classes et des objets.
Exemple : A l'Université, on souhaite mettre en place une application permettant d'enregistrer, pour chaque étudiant inscrit a l'Université, son nom et son adresse ainsi que pour chaque formation de L'Université son code, le thème des enseignements et le nombres d'heures d'enseignement.
Modéliser les données de cette application a l'aide d'un diagramme de classes et donner des exemples d'instances d'objets.
De ce texte, on en déduit le diagramme de classes suivant :

Etudiant
-nom: String
-adresse: String

Filière
-code: String
-thème: String
-nbHeures: Integer

et un exemple de diagramme d'objets, instance du diagramme ci-dessus :



Remarques :

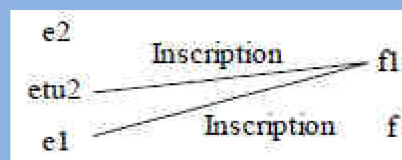
e2, e1, etu2, … sont appelés les noms (identifiants) des objets.

'Dupont', 'Albi', … sont des données de l'application.

e2 : Etudiant = ('Dupont', 'Albi') ou e2 sont des représentations simplifiées d'un objet.

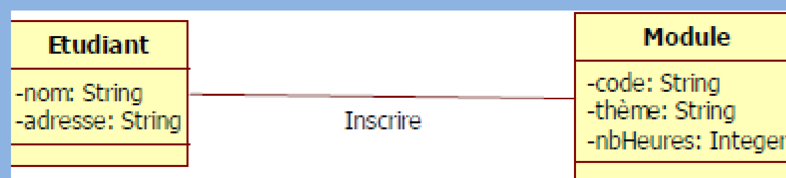
1.2.4. Lien entre objets et associations entre classes

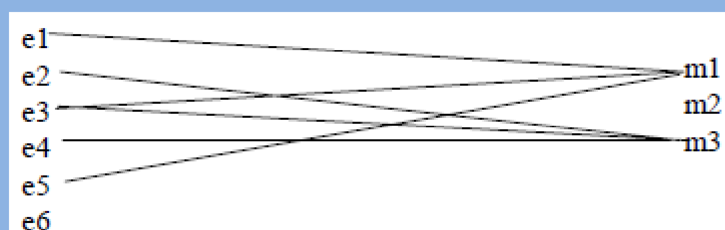
Liens entre objets (diagramme d'objets) : Dans un diagramme d'objets on peut établir des liens entre des objets. Par exemple, les étudiants e1 et etu2 sont *inscrits* en filière f2. Ce qui se représente graphiquement au niveau du diagramme d'objets de la manière suivante :



1.2.5. Multiplicités (cardinalités)

Soit le diagramme de classes et le diagramme d'objets suivants :





A chaque étudiant, on fait correspondre les modules ou il est inscrit :

- a e1, on fait correspondre {m1} (dans le cadre de l'association Inscription)
- de même, a e2, on fait correspondre {m3}
- de même, a e3, on fait correspondre {m1, m3}
- …

D'une manière générale, a tout objet e € Etudiant, on associe dans le cadre de l'association

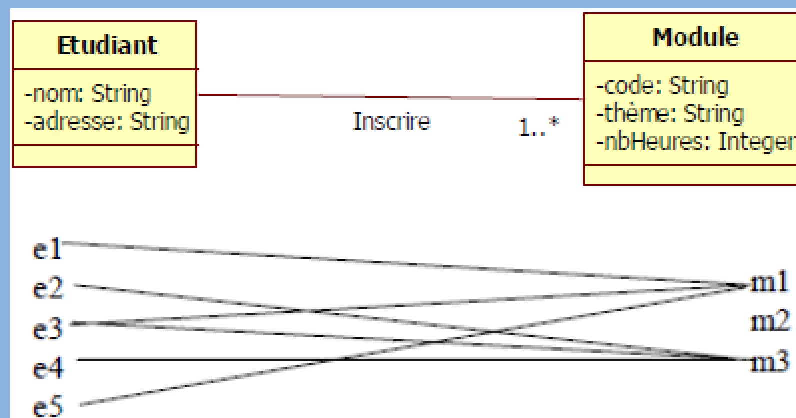
Inscription un ensemble de modules lies a e, dont le nombre peut varier selon les mises a jour effectuées sur le diagramme d'objets.

Au niveau du diagramme de classes, on doit indiquer le nombre minimum et maximum de modules qui pourront êtres lies a tout étudiant : un tel couple d'entiers est appelé multiplicités (cardinalités).

Ce couple d'entiers exprimant les multiplicités d'une association vis a vis d'une classe :

- est note m..M, avec m = le nombre minimum (généralement 0 ou 1), et M = le nombre maximum de modules (généralement 1 ou *, pour indiquer un entier > 1) auxquels tout étudiant e peut s'inscrire ;
- est situe sur l'une des extrémistes de l'association, près de la classe Module.

Le diagramme suivant indique que tout étudiant doit êtres inscrit a au moins un module, et le diagramme d'objets qui suit, est une instance pertinente du diagramme de classes, puisque l'on voit des étudiants qui sont inscrits a un module (e1, e2, e4 et e5) et que e3 est inscrit a plusieurs modules.



1.3 *Les paquetages*

Les paquetages permettent typiquement de définir des sous-systèmes. Un sous-système est formé d'un ensemble de classes ayant entre elles une certaine relation logique. Souvent, un paquetage fait l'objet d'une réalisation largement indépendante, et peut être confiée à un groupe, ou à un individu n'ayant pas un contact étroit avec les responsables d'autres paquetages.

• Ils regroupent des éléments de modélisation, selon des critères purement logiques.

• Ils permettent d'encapsuler des éléments de modélisation (ils possèdent une interface).

• Ils permettent de structurer un système en catégories (vue logique) et sous-systèmes (vue des composants).

• Ils servent de "briques

FIGURE 9.1 Notation utilisée pour un paquetage

On peut ouvrir un package; par défaut, le package choisi est "default".

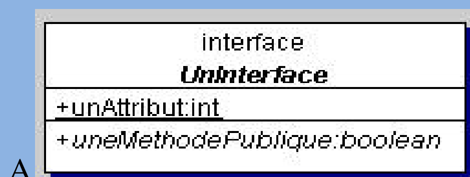
1.3 *Les interfaces*

Les interfaces représentent l'élément le plus abstrait du diagramme de classes.

La manière habituelle pour représenter un interface est représentée à la figure1.3.

Un membre est accompagné de son type, comme il est d'usage dans beaucoup de langages de projection.

FIGURE 1.2 Représentation d'un interface

A 

A cette représentation est associée du code, dans le langage de projection choisi. Dans le cadre de ce chapitre, nous utiliserons systématiquement Java comme langage de projection. Il est bien entendu que la modélisation est largement indépendante du langage de projection; néanmoins, certaines différences d'implémentation peuvent apparaître, comme par exemple l'absence de la notion d'interface en C++, et son remplacement par la notion de classe abstraite purement virtuelle.

FIGURE 1.3 Traduction de la représentation en code Java

```
package UnPackage;

public interface UnInterface {
    boolean uneMethodePublique();

    int unAttribut = 0;
}
```

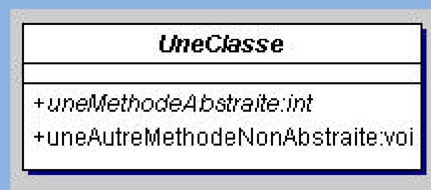
1.4 *Les classes*

La notion de classe est essentielle en programmation orientée objets : elle définit une abstraction, un type abstrait qui permettra plus tard d'instancier des objets. On distingue généralement entre classes abstraites (qui ne peuvent pas être instanciées) et classes "normales", qui servent à définir des objets.

1.4.1 *Classes abstraites*

Une classe abstraite ne peut donc pas être utilisée pour fabriquer des instances d'objets; elle sert uniquement de modèle, que l'on pourra utiliser pour créer des classes plus spécialisées par dérivation (héritage). Une classe abstraite est assez proche de la notion d'interface; d'ailleurs, la notion d'interface est généralement implémentée par une classe abstraite, dont toutes les méthodes sont purement virtuelles, en C++ (rappelons que C++ ne connaît pas la notion d'interface).

FIGURE 1.4 : Représentation d'une classe abstraite



Le + dénote la publication du membre concerné. Une classe abstraite peut posséder des membres privés ou des attributs privés ou publics; d'autre part, les méthodes peuvent faire l'objet d'une implémentation, même si la méthode est purement virtuelle.

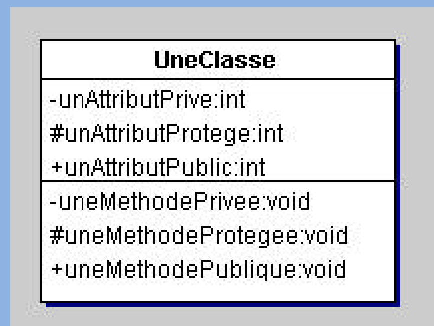
FIGURE 1.5 La même classe abstraite, projetée en Java

```
package UnPackage;

abstract public class UneClasse {
    public abstract int uneMethodeAbstraite();

    public void uneAutreMethodeNonAbstraite() {
    }
}
```

1.5.1 *Classes non abstraites*

Une classe "normale" ne contient pas de membres abstraits. Sa représentation en UML correspond

FIGURE 9.6 Représentation d'une classe



Noter les petits signes "cabalistiques" précédant l'identification des méthodes :

- \- dénote un membre privé.
- \+ dénote un membre public.
- # dénote un membre protégé

D'ailleurs, la projection en Java est parfaitement claire à ce sujet (figure9.7, page99).

FIGURE 9.7 Code correspondant

```
public class UneClasse {
    private int unAttributPrive;
    protected int unAttributProtege;
    public int unAttributPublic;
    private void uneMethodePrivee() {};
    protected void uneMethodeProtegee() {};
    public void uneMethodePublique() {};
}
```

9.2.3 *Les relations entre les classes*

Les diverses classes possèdent des relations de dépendance entre elles. Ces relations

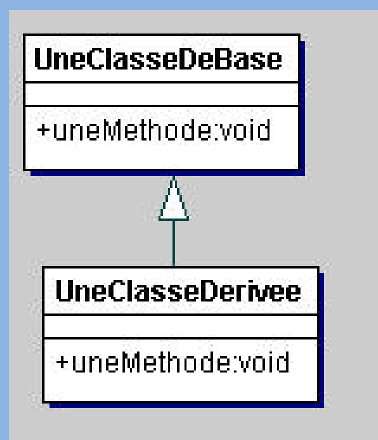
possèdent en principe un équivalent syntaxique dans le langage de projection. Les principales de ces relations sont énumérées dans la suite. Il est néanmoins important de noter que certaines

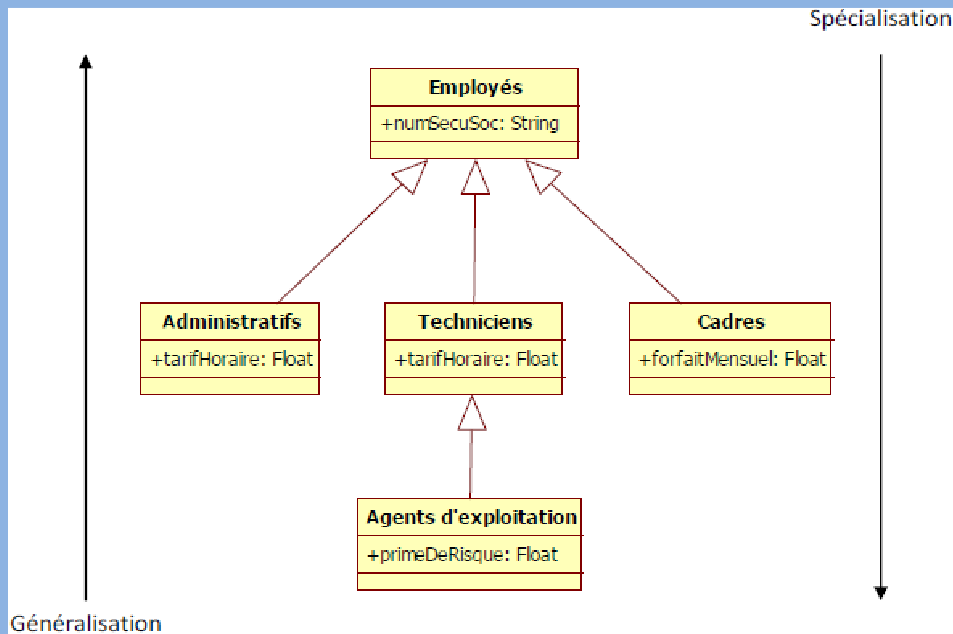
relations peuvent ne pas avoir d'équivalent dans le langage de projection considéré.
Ainsi, C++ introduit la notion de Template, ou classe paramétrable, qui peut être modélisée en UML, mais qui représente une notion inconnue en tant que telle en Java.
A l'inverse, Java permet de définir une classe qui implémente un interface, alors que la notion d'interface est inexistante en C++. Dans les deux cas, l'outil de modélisation, s'il génère du code, choisira le mode de représentation le mieux adapté en considération du langage de projection choisi.
Un interface UML pourrait, par exemple, se traduire par une classe abstraite en C++.

9.2.4 *Héritage*

L'héritage constitue une relation de spécialisation.
Elle est notée, en UML, par une flèche allant de la classe spécialisée vers la classe originale (de la classe vers la superclasse).

FIGURE 9.8 Relation d'héritage





On notera que la relation inverse n’est en principe pas documentée; c’est que cette relation n’est en principe pas connue au moment de la conception d’un produit, et qu’elle est susceptible

de changer au cours du temps. De plus, une superclasse n’est pas censée dépendre de ses dérivées : alors, à quoi bon le documenter ?

FIGURE 9.9 Code généré par la modélisation précédente

```
/* Generated by Together */

package UnPackage;

public class UneClasseDeBase {
    public void uneMethode() {
    }
}
```

```
/* Generated by Together */

package UnPackage;

public class UneClasseDerivee extends UneClasseDeBase {
    public void uneMethode() {
    }
}
```

9.2.5 *Implémentation*

Une classe peut implémenter un interface; elle peut aussi en implémenter plusieurs. En notation UML, cette relation est dénotée par une flèche en traitillés

FIGURE 9.10 Implémentation d'un interface

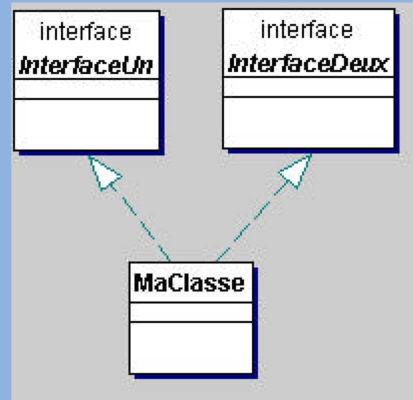


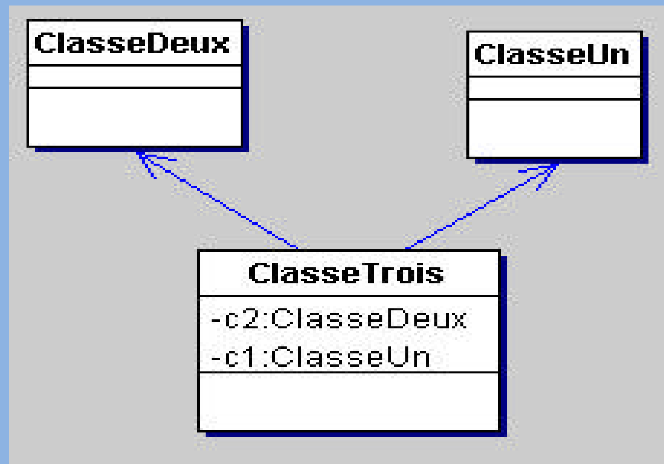
FIGURE 9.11 Code correspondant

```
/* Generated by Together */

package UnPackage;

public class MaClasse implements InterfaceUn, InterfaceDeux {
}
```

9.2.6 *La relation d'agrégation*

Lorsqu'un objet en contient d'autres, on parle d'agrégation. Le diagramme de classes d'UML décrit cette relation par une flèche pleine, comme indiqué à la figure9.12, page102. Il faut noter que l'agrégation est parfois appelée "relation de contenance". On peut signaler aussi que UML ne propose pas de représentation spécifique pour l'héritage privé ou protégé, notions propres au langage C++, ce qui de l'avis de l'auteur est une bonne chose.

FIGURE 9.12 Relation d'aggrégation et code correspondant




```
/* Generated by Together */

package UnPackage;

public class ClasseTrois {
    private ClasseDeux c2;
    private ClasseUn c1;
}
```

La relation d’agrégation est souvent (et de manière fort judicieuse) implémentée par des membres privés. Il va de soi que ces derniers peuvent être protégés ou publics.

9.2.7 *Relation de dépendance*

La notion de dépendance est plus floue que les précédentes. Il est difficile de faire une nomenclature complète des possibles relations de dépendance. Certains outils de modélisation offrent la possibilité de tracer automatiquement les relations directes : hélas, les relations indirectes sont souvent négligées, parceque difficiles à détecter; d’autre part, le schéma résultant

de ces outils est souvent encombré de relations évidentes qu’il eût peut-être mieux valu, pour des raisons de lisibilité, passer sous silence.

C’est pourquoi beaucoup d’outils laissent à l’utilisateur le soin de définir ce type de relation. La figure9.13, montre un exemple de dépendance, étant bien entendu que ceci n’est qu’un exemple parmi de nombreux autres possibles.

FIGURE 9.13 Relation de dépendance et exemple de code correspondant

Chapitre 02 :

I. Introduction

Rappelons que l'ontologie est un objet informatique qui résulte d'une modélisation des connaissances d un domaine particulier et qui a pour objectif de répondre à un problème spécifique.

Jusqu'a présent, l'élaboration, tout comme la validation, l'évaluation et la maintenance d'ontologies relève le plus souvent d'un savoir-faire que d'une démarche d'ingénierie.

Ainsi, les développements d'applications utilisant des ontologies, la visibilité et le travail collaboratif au sein de la communauté d'Ingénierie des connaissances sont retardés. La plupart des équipes de recherche dans le domaine travaillent de manière ad hoc. Bien que la plupart des méthodologies initient le processus de construction par l'identification, puis l'organisation et la structuration des concepts et des relations à représenter, les ontologies réalisées sont très différentes les unes des autres. Faut-il faire l'hypothèse qu'il y ait autant de manières de représenter les connaissances d'un domaine qu'il y a d'ontologues ? Auquel cas, le principal problème de la construction d'ontologies est celui de l'organisation des concepts et des relations en taxinomies.

Dans ce chapitre nous allons présenter les principales méthodes et méthodologies utilisées pour construire les ontologies ainsi que les outils de construction d'ontologies.

2.1/ Ontologies et outils du Web Semantic

Ontologies Dans le contexte de l'informatique et des sciences de l'information, une ontologie peut être définie comme un ensemble de concepts et de relations permettant de modéliser un domaine de connaissances (Gruber (2009)).

Dans concepts sont incluses la signification et les contraintes logiques d'application d'une représentation.

Nous pouvons distinguer plusieurs types d'ontologie, regroupés et commentés par Psyché et al. (2003). Cette classification va dépendre du but de l'ontologie, de l'étendue et de la précision du domaine modélisé :

– les ontologies de représentation des connaissances : formalisation des connaissances ;

– les ontologies supérieures ou de haut niveau : ontologies universelles qui ne dépendent pas de domaine, ni de problème particulier. Elles concernent des concepts très généraux comme le temps, les évènements, l'espace, les actions... Ces concepts doivent être consensuels entre une grande communauté ;

– les ontologies génériques ou méta-ontologies : moins génériques que les précédentes, mais assez pour être réutilisables à travers plusieurs domaines (exemple OBOE[1]) ;

– les ontologies de domaine : ensemble de vocabulaires et de concepts qui décrit

un domaine d'application, basé sur la connaissance du domaine où la tâche est réalisée ;
– les ontologies de tâches : pour conceptualiser des tâches (diagnostic, planification...)spécifiques dans les systèmes ;
– les ontologies d'application : les plus spécifiques, domaine restreint, destinées à l'exécution d'une tâche. Un concept correspond souvent à un rôle joué par une entité dans le système.

Les ontologies peuvent exploiter le fonctionnement du Web de plusieurs manières, Berners-Lee et al. (2001) :
– pour améliorer la pertinence des recherches, par concept,
– pour associer à une page des structures de connaissance et à des règles d'inférence.

Web Sémantique

L'approche actuelle du Web Sémantique, ou Web des données, est basée sur l'idée de travailler, au niveau du Web, à partir de concepts plutôt que de documents, qui contiendraient alors des informations, ou métadonnées, formalisées pour être traitées automatiquement par des agents logiciel. Le Web de données est fondé sur une pile de langages et protocoles issus du Web et propres au Web Sémantique (illustration figure 2.1), conçus pour être compris sémantiquement et utilisables par les programmes. Le W3C[2] supervise le développement des standards, propre aux Web Sémantique, comme RDF [3], RDFS [4], OWL[5] ou SPARQL[6] que nous allons décrire.

RDF : pour Resource Description Framework, Manola & Miller (2004), un langage pour représenter des informations à propos de ressources sur le Web. RDF est destiné aux situations où ces informations ont besoin d'être traitées par des applications au lieu d'être seulement affichées pour les personnes. Les informations peuvent être échangées entre les applications sans perte de sens.

RDF est fondé sur l'idée d'identifier les choses en utilisant des URI [7], et en décrivant les ressources en fonction de propriétés et de valeurs de propriétés.

De ce fait, on pourra traduire des informations simples sur les ressources sous forme de graphe, où les nœuds représentent des ressources et les arcs les propriétés.

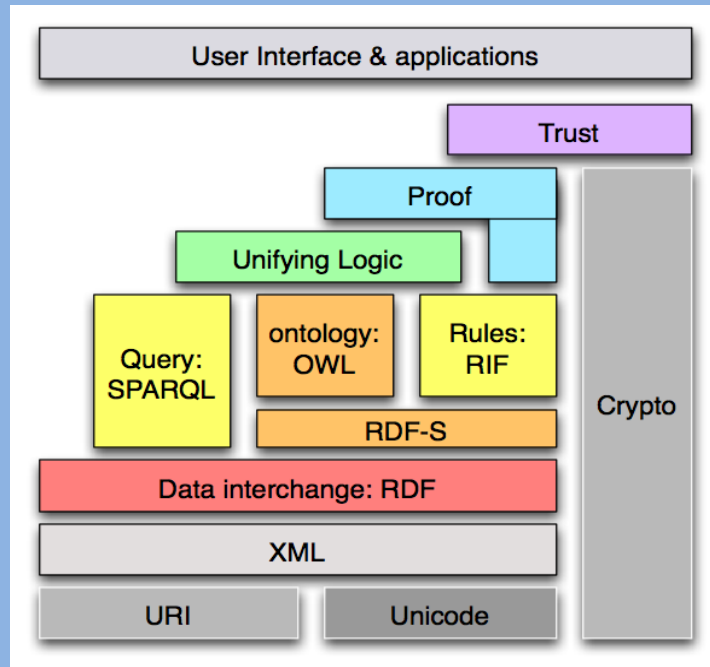


FIGURE 2.1 – Pyramide des technologies du Web Sémantique

RDF emploie une terminologie particulière, illustrée par la figure 2.2,

pour indiquer les diverses parties des déclarations :
– le sujet concerne ce qui est décrit,
– le prédicat correspond à la propriété,
– l'objet est la valeur de la propriété.

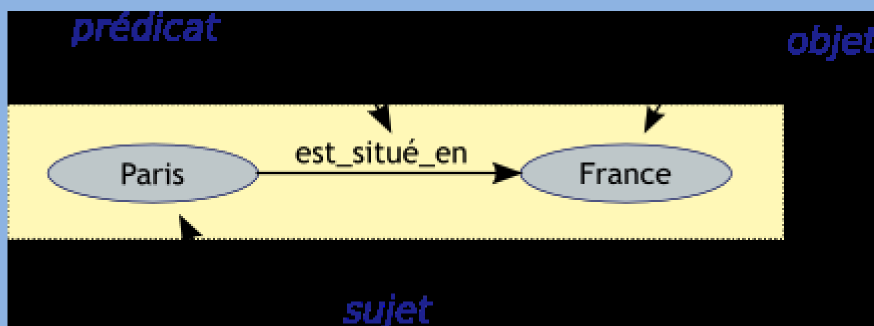


FIGURE 2.2 – Représentation graphique d'un triplet RDF

RDFS : pour RDF Schema, permet de décrire les ressources du vocabulaire RDF. Il défini la notion de classe et de propriété. Il permet de définir des hiérarchies de classes et de propriétés.
OWL: pour Web Ontologie Langage, McGuinness&van Harmel en (2004) et Segaran et al. (2009), est un langage RDF pour la définition de classes et de propriétés (RDFS) ,mais aussi pour permettre plus de raisonnements et des inférences basées sur les relations.

Ce langage est divisé en trois sous-langages dont la complexité et l'expressivité augmentent. OWL-Lite, le plus simple, OWL-DL, basé sur les logiques de description et OWL-Full, le plus expressif mais dont la calculabilité n'est pas garantie.

2 /Qu'est-ce que OWL 2 ?

OWL 2 est un langage d'expression d' ontologies .

Le terme ontologie a une histoire complexe à la fois dans et hors de l'informatique, mais nous l'utilisons pour désigne un certain type d'artefact informatique - c'est-à-dire quelque chose qui s'apparente à un programme, un schéma XML ou une page Web - généralement présenté comme un document.

Une ontologie est un ensemble d'énoncés descriptifs précis sur une partie du monde (généralement appelée domaine d'intérêt ou l' objet de l'ontologie). Des descriptions précises répondent à plusieurs objectifs : elles évitent notamment les malentendus chez les humains. communication et ils garantissent que le logiciel se comporte de manière uniforme et prévisible et qu'il fonctionne bien avec d'autres logiciels.

Afin de décrire précisément un domaine d'intérêt, il est utile de trouver un ensemble de termes centraux - souvent appelés vocabulaire - et de fixer leur sens.

Outre une définition concise en langage naturel, le sens d'un terme peut être caractérisé en indiquant comment ce terme est lié à les autres termes.

Une terminologie , fournissant un vocabulaire ainsi que de telles informations d' interdépendance , constitue une partie essentielle d' un OWL 2 document.

Outre cette connaissance terminologique, une ontologie peut également contenir des connaissances dites assertives qui traitent de des objets concrets du domaine considéré plutôt que des notions générales.

II. Fondements de l'ingénierie ontologique

II.1. Définition

L'ingénierie ontologique peut être définie comme une thématique de recherche visant à proposer des aspects pratiques, essentiellement des méthodes, des outils et des langages dédiés à l'application des résultats de la théorie des ontologies à la construction d'ontologie. **[GAND 02]**

II.2. Processus

Le processus de construction d'une ontologie repose sur deux étapes : l'ontologisation et l'opérationnalisation.

L'ontologisation consiste à construire une ontologie conceptuelle. Cette étape est réalisée à partir de différentes sortes de données telles que des glossaires de termes, d' autres ontologies, des textes, d'interviews d'experts, etc.

L'opérationnalisation consiste à coder l'ontologie conceptuelle obtenue à l'aide d'un langage de représentation de connaissances opérationnel (i.e. doté de mécanismes d'inférences).

Identification des objectifs et du contexte de l'ontologie : Le but de cet étape est de clarifier le pourquoi de la construction de l'ontologie, les utilisations prévues et la finalité (réutiliser, partager, utiliser comme une partie d'un KB, etc.) et les utilisateurs potentiels de l'ontologie.

Construction de l'ontologie : Cette étape est divisée en trois activités :

1. Capture de l'ontologie : cette étape se fait indépendamment d'un langage de représentation (conceptualisation). Elle commence par l'identification des concepts et des relations clés ensuite produire en langage naturel les définitions précises et non ambiguës pour les concepts et les relations et se termine par l'identification des termes dénotant les concepts et les relations pour ainsi essayer d'arriver à un agrément.
Pour la catégorisation et la capture de l'ontologie, Uschold et Grüninger **[USGRU 96]** proposent trois approches :
Approche descendante *(Top Down)* : l'ontologie est construite par généralisation en partant des concepts des basses couches taxinomiques. Cette approche encourage la création d'ontologies spécifiques et adaptées.

Approche ascendante *(Buttom Up)* : l'ontologie est construite par spécialisation en partant de concepts des hautes couches taxinomiques. Cette approche encourage la réutilisation d'ontologies.
Approche intermédiaire *(Middle Out)* : l'ontologie est construite à partir de concepts centraux puis les généralise ou les spécialise pour former les couches hautes et les couches basses de l'ontologie. Cette approche encourage l'émergence de domaines thématiques dans l'ontologie et favorise la modularité.

2. Codage de l'ontologie :

Cette étape implique deux tâches :

Représentation explicite de la conceptualisation (ex. classe, entité, relation).

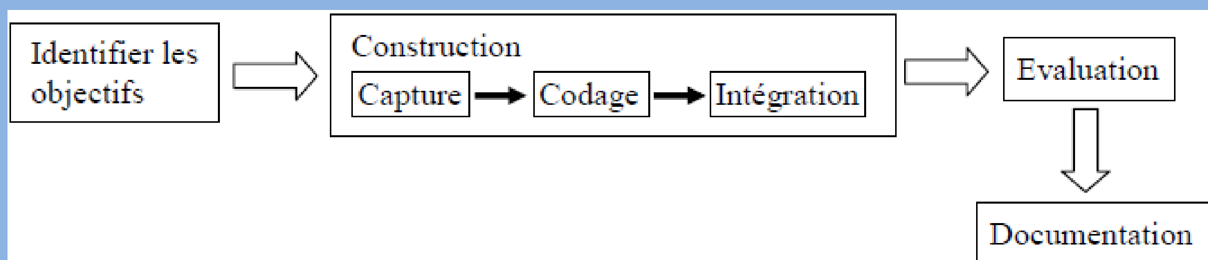
Ecriture du code dans un langage formel : Prolog, KL-One, OIL, CG, OWL…

3. Intégration d'ontologies existantes :

Cette étape fait référence à comment utiliser les ontologies qui existent. Elle peut être faite en parallèle avec les étapes de capture et/ou de codage. Par exemple ; The Frame Ontologie peut être réutilisée pour modéliser les ontologies de domaine qui utilise l'approche basée sur les frames.

Evaluation de l'ontologie : c'est la mise à l'épreuve de l'ontologie en la faisant confronter aux objectifs pour lesquels elle a été conçue et aux utilisateurs.

Documentation de l'ontologie : cette étape est essentielle pour l'acceptation de l'ontologie.



II.2.2. Les ontologies

Souvent les termes que l'utilisateur emploie pour décrire ses centres d'intérêts ne sont pas suffisants ou ont plusieurs sens (par exemple, JAVA : langage de programmation et style de danse). Pour résoudre ces problèmes dans la partie des ontologies du profil, l'utilisateur peut **définir les termes** et **les opérateurs employés**.

Les ontologies complètent la définition des centres d'intérêt en explicitant la sémantique de certains termes (donner aux mots un sens unique) ou de certains operateurs employés par l'utilisateur dans son profil ou dans ses requêtes ultérieures pour palier au problème le plus important de la recherche d'information qui est la grande quantité de données dans les résultats des requêtes due à l'utilisation de termes trop généraux. Cette catégorie comprend des synonymes des mots, leurs traductions dans les langues que l'utilisateur comprend et le sens que ce dernier donne à chacun de ces termes appelé plus souvent contexte. Le sens des termes doit être correct sinon les résultats renvoyés à l'utilisateur en présence de son profil risquent d'être tous inintéressants. Dans la partie de

définition des opérateurs, on met la définition des outils nécessaires pour décrire le sens des expressions de préférence de l'utilisateur. Parmi les opérateurs on retrouve les opérateurs de base (égalité, supériorité, infériorité, etc.), mais aussi des opérateurs plus complexes (Similaire, Environ, Entre, Meilleur que, Le plus grand, Le plus petit, Préféré

Chapitre 03 :

3.1 Modèle

Un **modèle** pourrait être défini comme une représentation abstraite de quelque chose. En pratique, les modèles sont présents partout autour de nous dans la vie courante, et les exemples sont nombreux.

Un plan de votre maison par exemple est le modèle à partir duquel on a construit la maison que vous habitez.

Ce plan regroupe des caractéristiques de dimension que votre maison respecte (en principe).

Dans le contexte de la modélisation, on dit alors que votre maison est **représentée par** son modèle (son plan).

Un modèle peut également représenter des choses plus abstraites, comme par exemple une séquence d'opérations qui s'enchaînent dans le temps, ou bien alors l'organisation et les interactions qui ont lieu entre les employés d'une entreprise.

Il est important de noter que plusieurs entités distinctes peuvent êtres représentées par un unique modèle donné.

Par exemple, dans un immeuble d'habitation, tous les appartements sont souvent construits à partir d'un nombre limité de modèles différents (un modèle de T3, un modèle de T2, etc.).

Intuitivement, les atouts de l'utilisation d'un modèle sont nombreux, surtout comme outil de communication.

Reprenons l'exemple de la maison. Le plan d'une maison est le fruit d'une concertation entre un architecte et le futur propriétaire, et permet de visualiser l'allure générale du bâtiment. Il permet également de faire des calculs de dimension pour les matériaux utilisés lors de la construction, mais aussi de présenter le projet à la mairie pour obtenir un permis de construire. Le plan est naturellement relu par l'entrepreneur et utilisé comme guide de montage pendant la construction. Enfin, lors d'un éventuel recours du propriétaire qui a noté une non conformité de l'ouvrage terminé avec le plan initial, c'est le document qui fait foi et qui est présenté à une instance judiciaire.

Cette notion peut s'appliquer à de nombreux domaines, et en particulier celui qui nous intéresse, l'informatique.

Dans ce cas, le modèle reste un outil de communication, mais les acteurs ne sont plus systématiquement humains.

L'exemple le plus connu est la programmation, où un opérateur humain (le programmeur) écrit un modèle de comportement ou de calcul (un programme), et le transmet à travers un

processus de compilation ou d'interprétation au système informatique qui produit un comportement ou des calculs décrits dans le modèle initial.

De la même façon que le plan de l'architecte est écrit avec une syntaxe, une nomenclature et des symboles picturaux précis, de nombreux langages de programmation différents coexistent et peuvent chacun exprimer certains types de modèles de calculs.

On voit donc que même si le concept de modèle est très répandu, les langages pour les exprimer sont très variables.

3.5 Meta modèle

Un **méta modèle** est simplement une description d'un langage de modélisation. Il apporte les règles syntaxiques qui permettent de l'utiliser (d'écrire un modèle dans ce langage) et le sens de chaque item présent dans ce langage. Par exemple, la spécification du langage Java, c'est-à-dire sa grammaire et sa sémantique, constitue le méta modèle du langage Java. Dans le contexte de la modélisation on dit qu'un programme Java sans erreur syntaxique est **conforme** au méta modèle du langage Java.

4.2.1 Un méta-modèle pour le diagramme de classe

Pour définir un méta-modèle (ou une écore) EMF utilise le diagramme de classe. Donc, une DC (comme le montre le méta-modèle de la figure 4.1) est composée de sept classes : ***Package, Association, Entité, Facture, Attribut, Méthode, Paramètre***

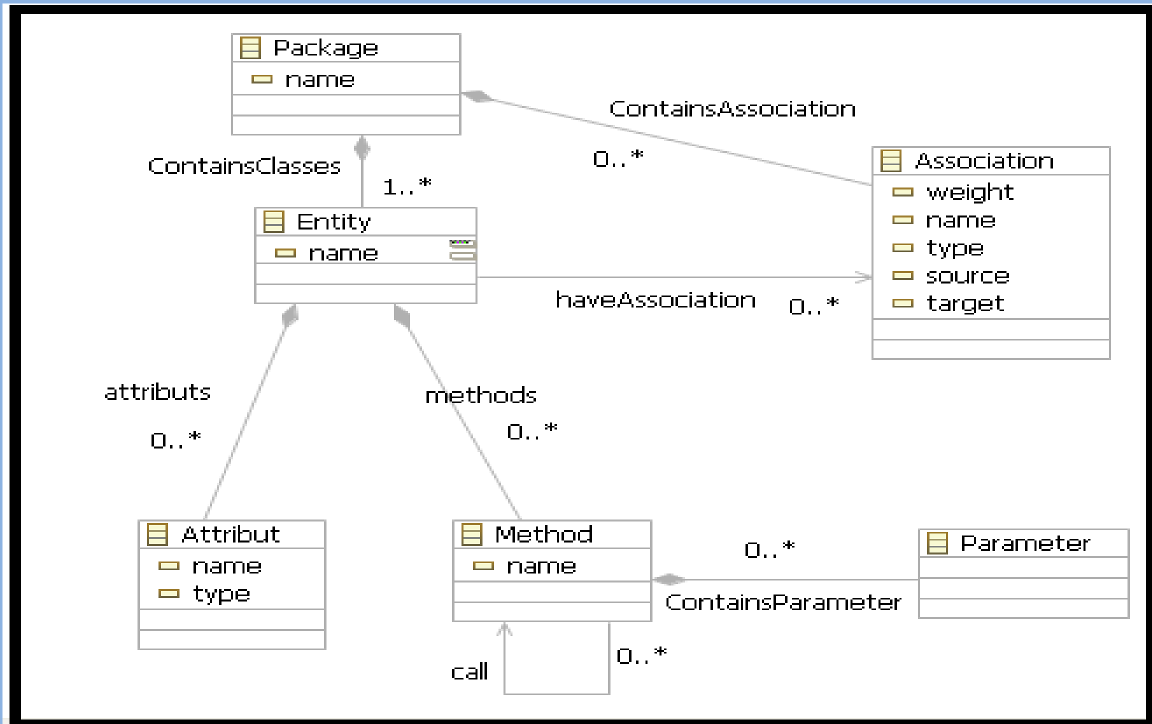


Figure 4.1 : le méta-modèle de diagramme de classe

4.2.2 La génération d'un outil pour le diagramme de classe

En se basant sur le méta-modèle (voir la figure 4.1) on peut générer un outil qui nous permettra de créer des exemples d'architectures logicielles (des instances) avec les étapes suivantes :

- **Création d'un projet EMF vide (File → New → Project…)**

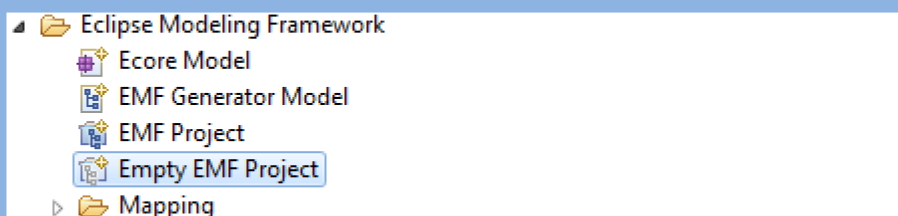
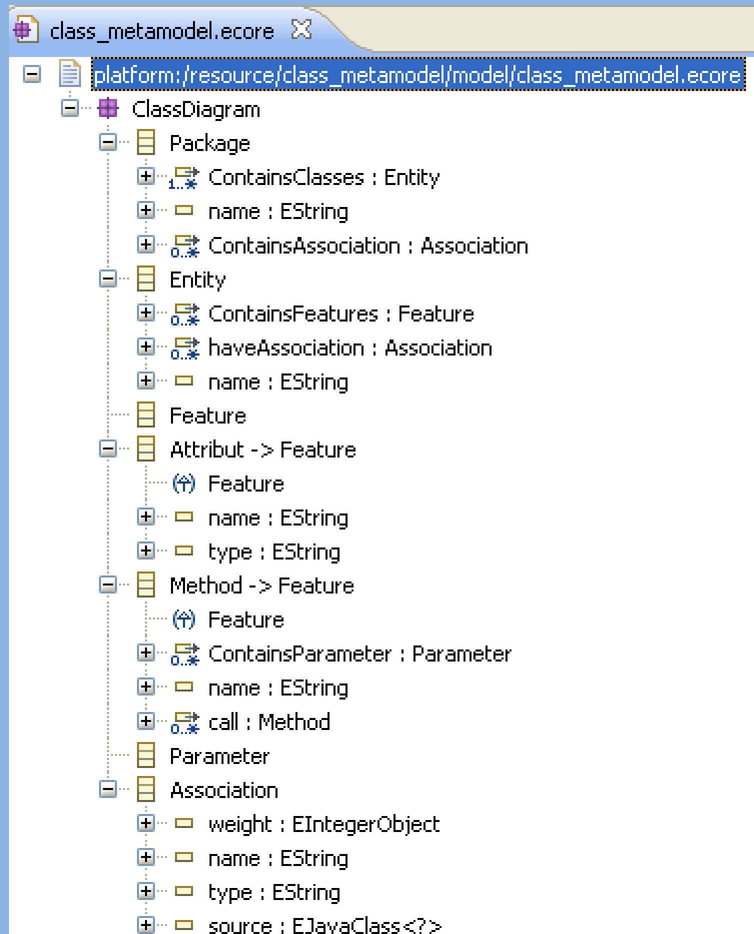


Figure 4.2 Crée projet EMF vide.

- **Création d'un diagramme Ecore (New → Ether→)**

Figure 4.3 Crée diagramme Ecore.

- **Défini les éléments de notre méta-modèle de DC**



- **Création de *Genoude***

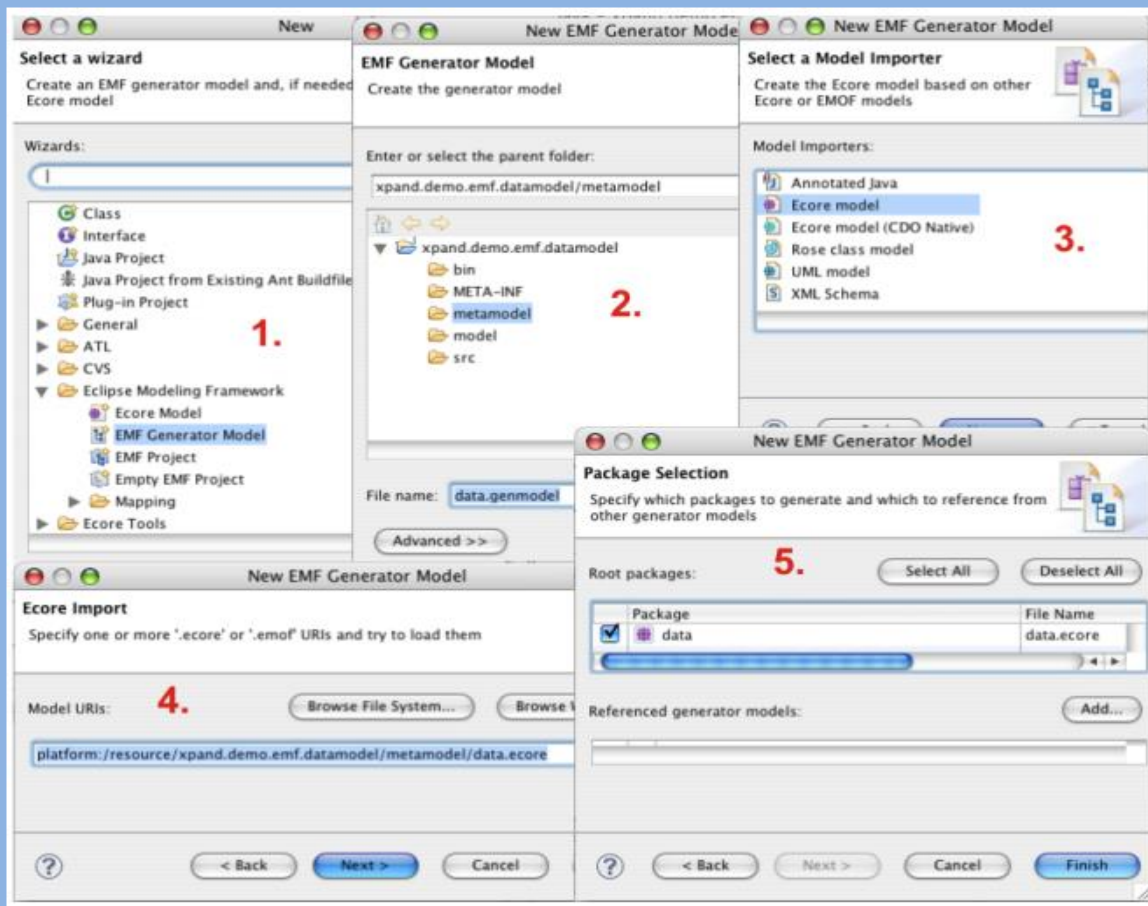


Figure 4.5 : génération de genmodel

➢ **Création les méta-modèles Ecore**

Nous allons créer les méta-modèles de DC et de RdP Ecore.

On a déjà vue comment faire ça.

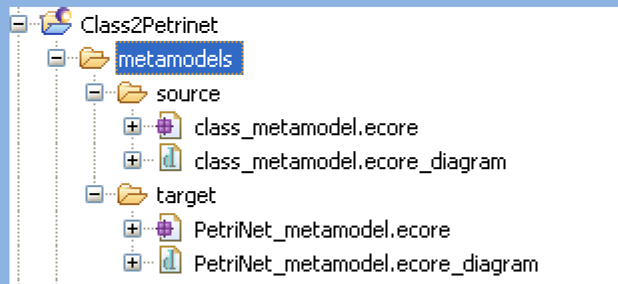
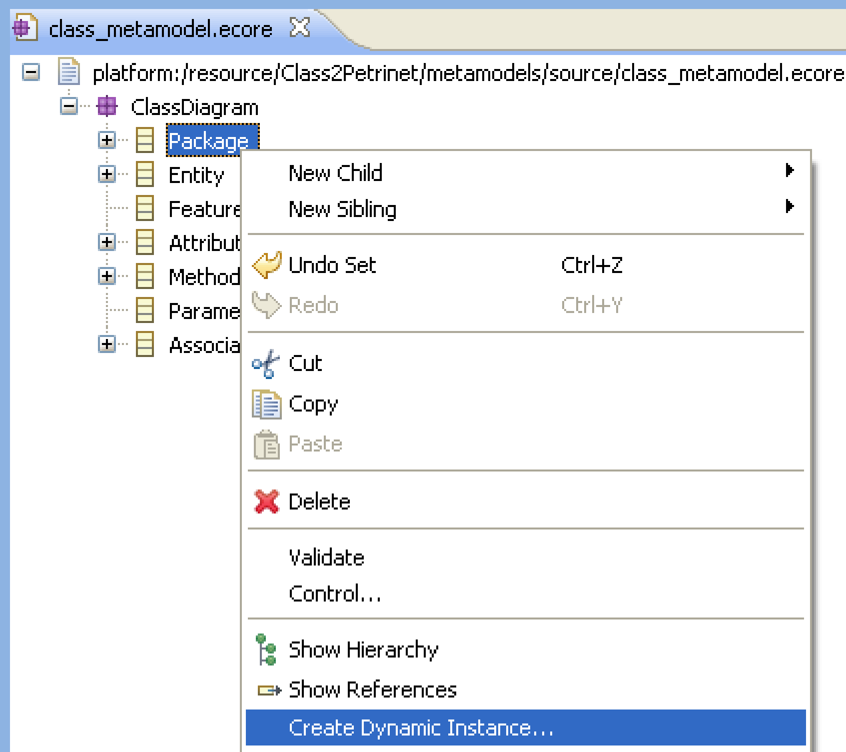


Figure 4.19 : création les méta-modèles

- **Génération de modèle le conforme à méta-modèles de DC**

On va générer le modèle conforme à la méta-modèle DC.



.

Attention, cette conformité ne garantit pas que le programme ne contient pas d'erreur d'exécution, ou même qu'en l'absence d'erreur d'exécution, les résultats obtenus à l'issu de son exécution seront ceux attendus par le programmeur

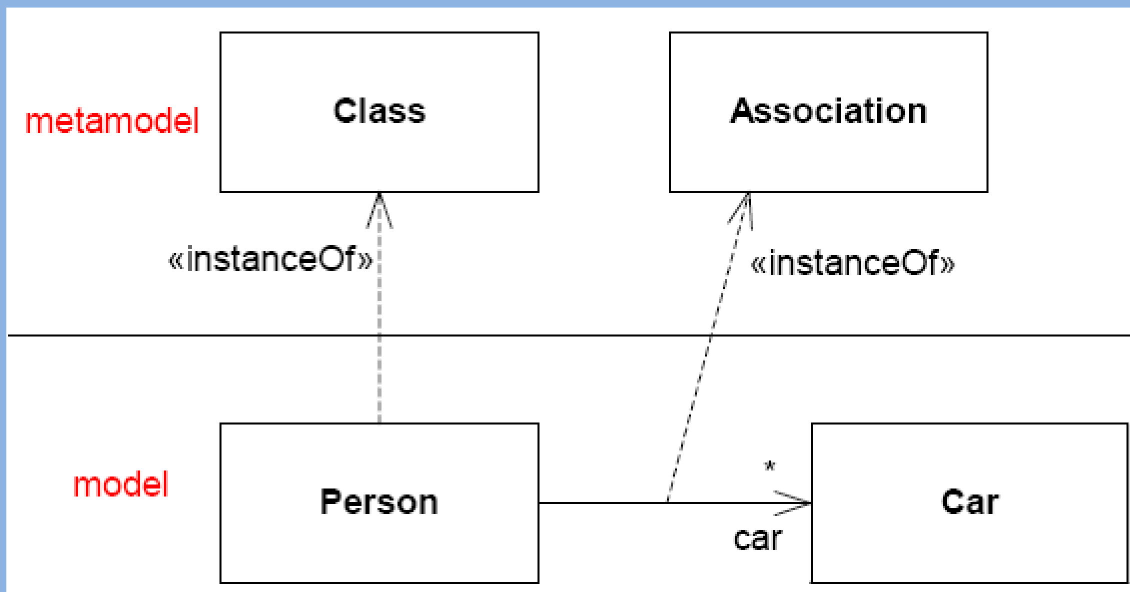


Figure 1 - Exemple de trois classes d'un modèle dont deux sont conformes à leur méta classe dans le méta modèle

L'ingénierie dirigée par les modèles, ou Model-driven Architecture (MDA)

Dans le contexte de l'ingénierie logicielle, l'OMG a depuis quelques années réfléchit sur la question de l'utilisation extensive de modèles, d'une part comme outil de dialogue avec les utilisateurs et/ou les concepteurs pour représenter les besoins fonctionnels, et d'autre part comme langage de représentation de la connaissance suffisamment proche d'une architecture logicielle donnée pour permettre un passage automatique d'un modèle à du code informatique. Cette réflexion reprend l'ambition fondamentale de séparer au mieux les problèmes de spécification des fonctionnalités d'un système, et les détails d'implémentation de ces fonctionnalités.

MDA

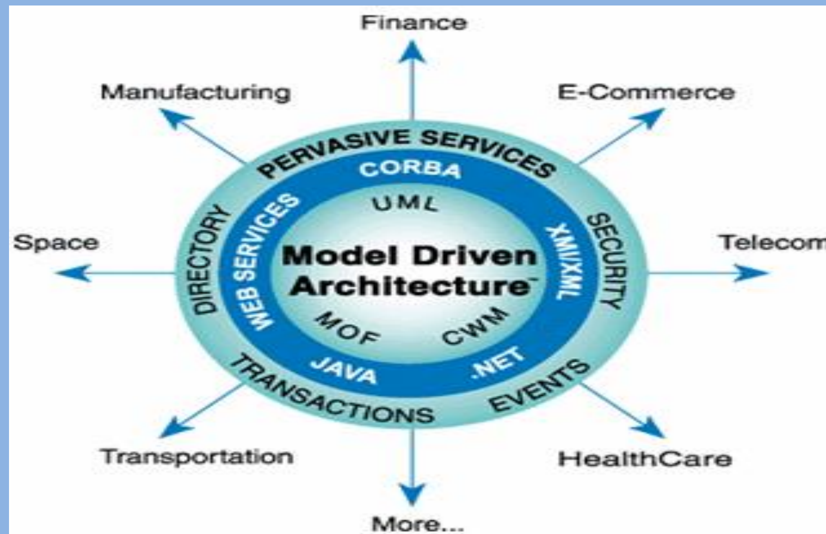


Figure 2 - Logo MDA

En 2001, l'OMG a présenté le fruit de ses réflexions dans un document [8], où le groupe introduit son architecture dirigée par les modèles (Model Driven Architecture en anglais). Il y est montré comment utiliser intensivement les modèles pour le développement logiciel, et un éventail d'exemples d'implémentation y est décrit.

L'OMG définit dans ce guide MDA plusieurs niveaux d'abstraction et d'indépendance vis-à-vis d'une implémentation particulière dans un environnement logiciel et matériel particulier. On retrouve ainsi essentiellement, du plus indépendant au plus spécifique, les niveaux « Computation indépendant », « Platform indépendant » et « Platform specific
».

Il est proposé d'exprimer (de modéliser) successivement un problème dans ces trois niveaux, d'abord au plus général pour capter le mieux possible les besoins souvent exprimés par des non-spécialistes, jusqu'au plus fin permettant ainsi dans l'idéal la génération directe de code fonctionnel pour une configuration logicielle et matérielle donnée.

Les deux niveaux les plus utilisés sont les deux derniers, « Platform indépendant » et « Platform spécifique ».

Pour permettre le passage ou la traduction de l'un vers l'autre, l'approche MDA sous-entend tout d'abord de modéliser le problème ou le système considéré au niveau le plus indépendant, puis d'utiliser différents types de **transformation de modèle** pour obtenir le modèle spécifique à la plateforme cible.

Définition

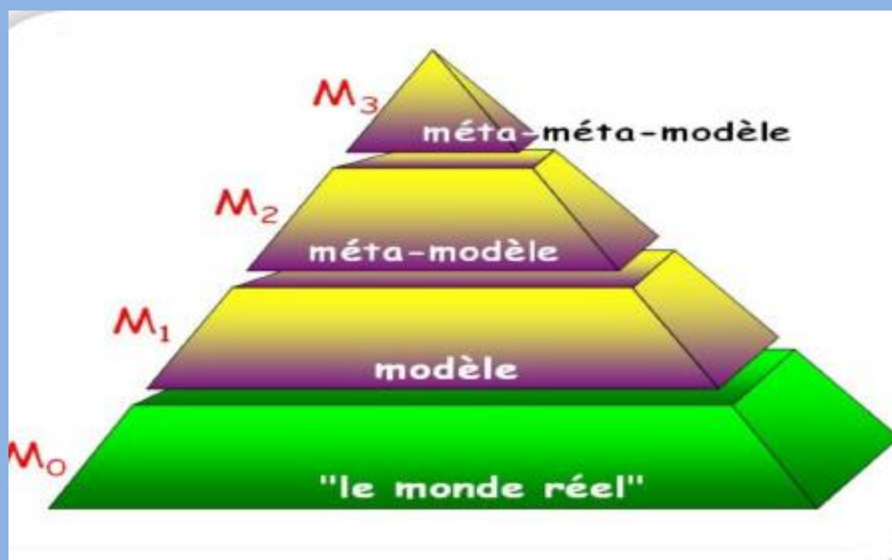
L'approche MDA permet de réaliser le même modèle sur plusieurs plates-formes grâce à des projections standardisées.

Elle permet aux applications d'interopérer en reliant leurs modèles et supporte l'évolution des plates-formes et des techniques.

La mise en oeuvre du MDA est entièrement basée sur les modèles et leurs transformations.

Architecture MDA à quatre niveaux:

L'OMG a défini une architecture à quatre niveaux d'abstraction, comme carde général pour l'intégration des méta-modèles, en se basant sur l'MOF comme le montre la figure 1.3. Dans cette architecture, les modèles de deux niveaux adjacents sont liés par une relation d'instanciation :



Les quatre niveaux d'abstraction pour MDA

Le niveau M0 : Niveau des instances des modèles. Il définit des informations pour la modélisation des objets du monde réel.

• ***Le niveau M1*** : Ce niveau représente toutes les instances d'un méta-modèle. Les Modèles du niveau M1 doivent être exprimés dans un langage défini au niveau M2. UML est un exemple de modèles du niveau M1.

• ***Le niveau M2*** : Ce niveau représente toutes les instances d'un méta-méta-modèle. Il est composé de langages de spécifications de modèles d'information. Le méta-modèle UML qui est

décrit dans le standard UML et qui définit la structure interne des modèles UML, appartient au niveau M2.

•***Le niveau M3*** : Ce niveau définit un langage unique pour la spécification des Méta-modèles. Le MOF élément réflexif du niveau M3, définit la structure de tous les Méta-modèles du niveau M2

Chapitre 04 :

3.1 Introduction

L‟approche des grammaires de graphes triples (Triple Graph Grammar : TGG), introduit par Andy Schürr [50] est une tentative de créer une méthode pour connecter différents systèmes/modèles par rapport à certains règles/critères prédéfinis, de sort que les changements dans un système/modèle conduirait inévitablement à des changements dans l‟autre.

TGG peut être utilisé dans différents transformation de modèles et les scénarios de synchronisation. TGG est spécifié pour les transformations bidirectionnelles (transformation à l‟avant et en arrière)

Dans ce chapitre, nous prenons les variétés de Diagramme de Class et nous y travaillons, où nous utilisons les outils Eclipse afin de représenter toutes les étapes, qui sont représentées dans le dessin des diagrammes méta modal, y compris l'ontologie
Ce chapitre présente la mise en œuvre de la création automatique de
Spécifications des diagrammes UML basées sur des grammaires à triple graphe (TGG).
Cette transformation automatique génère des opérations de simulation et vérification officielle.
Avant d'aborder les techniques de transformation de modèle et les règles de la transformation, il est nécessaire de définir le méta-modèle de « diagrammes de classe UML ».
Pour définir et réaliser notre méta-modèle nous allons utiliser le standard EMF

Processus de transformation des modèles avec TGG

Le formalisme TGG a été conçu pour générer et transformation des paires de graphes connexes (généralement appelés graphe source et cible) en vertu d‟un mécanisme de synchronisation bien formé (par exemple, un graphe de correspondance) qui permettrait de préserver les relations dans les graphes mis en correspondance après l‟application d‟une transformation.
La motivation initiale du formalisme était de fournir une modélisation et une spécification d‟outil graphique de haut niveau pour les problèmes impliquant des diagrammes connexes (arbres de syntaxe, diagrammes de contrôle) et les structures d‟information (les exigences, les documents de conception et la traçabilité). Plus récemment les concepts clés du formalisme TGG ont été utilisés dans le langage de transformation standard de l‟OMG,

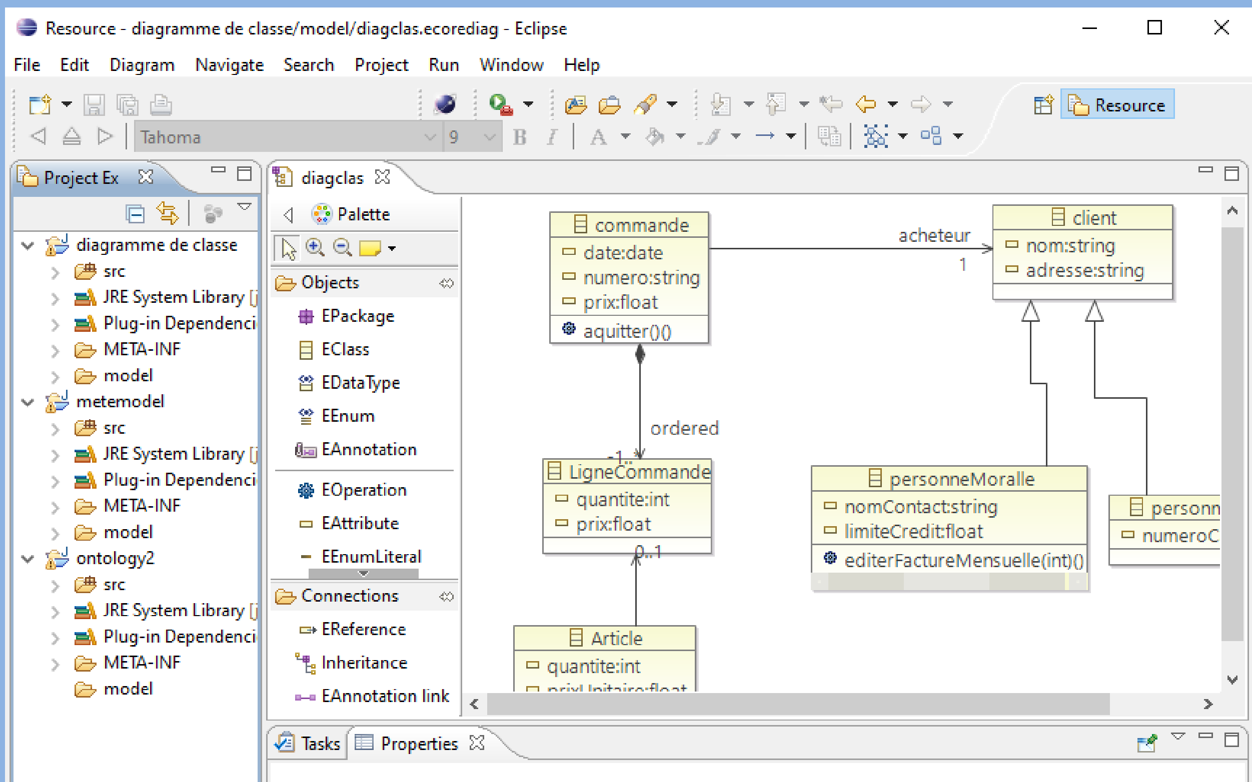
pour spécifier les « correspondances » d‟une transformation. En outre, les applications de TGG dans le domaine des spécifications de transformations bidirectionnelles de modèles sont de plus en plus fréquentes

Méta-modélisation en EMF

Grâce à l'outil standard de méta-modélisation EMF, nous implémentons les méta-modèles pour : les modèles sources constituant la partie LHS des règles de transformation TGG; les modèles cibles de ces règles (partie RHS) ; ainsi que ceux de la correspondance entre les deux Pour cela, nous proposons d'implémenter les sept méta-modèles suivant

Génération d'un outil EMF pour « les diagrammes de classe »

Basant sur le méta_modèl décrit dans la figure 3.1, nous avons généré un outil permettant de créer des diagrammes de classe sous format XMI.



3.3.1 Un méta-modèle pour le diagramme de classe

Pour définir un méta-modèle (ou une ecore) EMF utilise le diagramme de classe. Donc, un DC (comme le montre le méta-modèle de la figure 3.1) est composée de huit classes :

- ***ClassModel***
- ***Association***
- ***Class***
- ***Classifier***

- ***PrimitiveDataType***
- ***Attribute***
- ***Méthode***
- ***Paramètre***

- Pour construire un modèle EMF plusieurs formats disponibles
  - Modèle Ecore (voir la suite)
  - Classes Java annotées
  - Modèle de classes Rose
  - Modèle UML
  - XML Schéma

Nous utiliserons par la suite un modèle Ecore puisque l'outillage fourni par EMF facilite la construction.

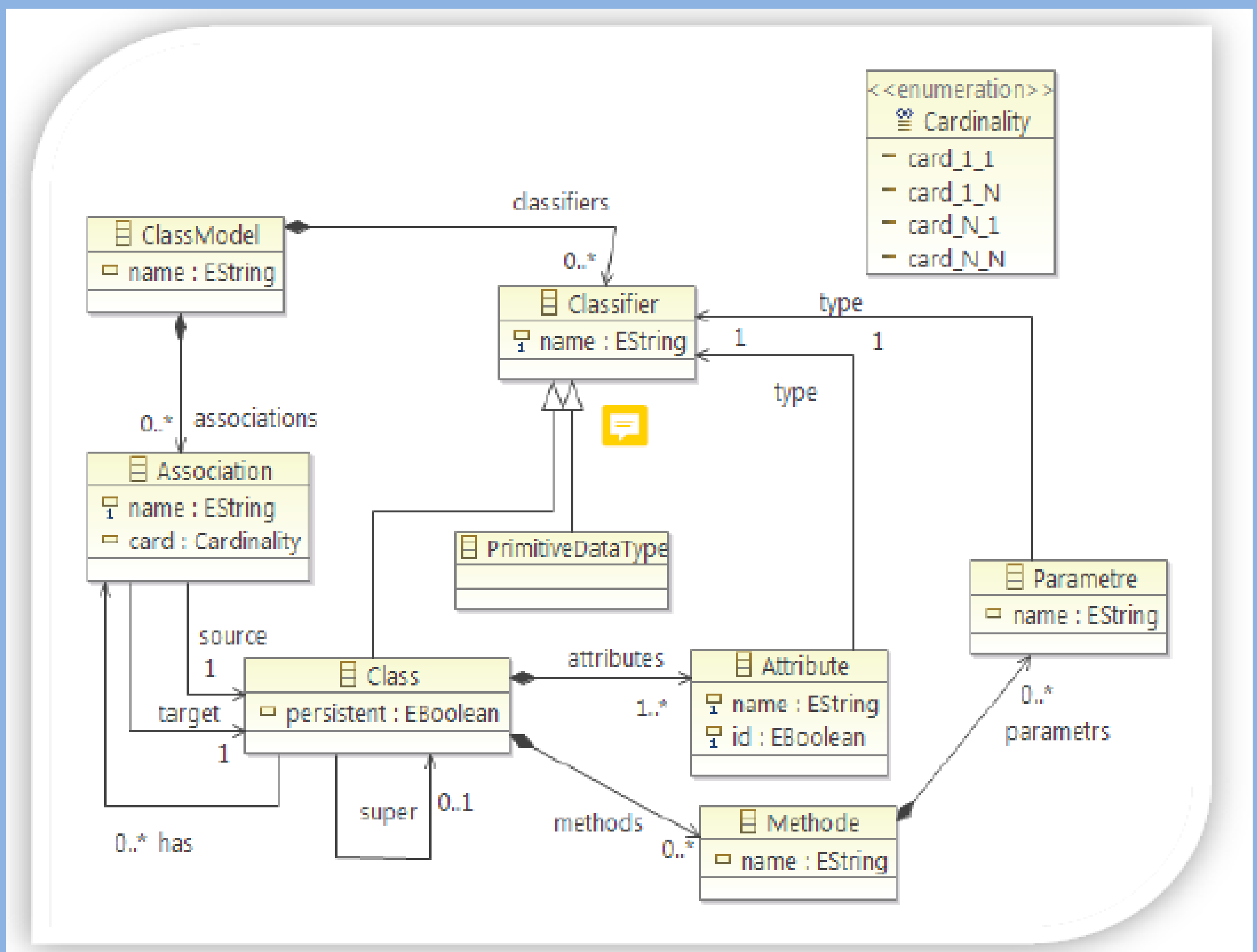


Figure 3.1 : un méta-modèle pour les diagrammes de classe

3.3.2 La génération d'un outil pour les diagrammes de classe

En se basant sur le méta-modèle (voir la figure 3.1) on peut générer un outil qui nous permettra de créer des exemples de diagramme de classe (des instances) avec les étapes suivantes :

- **Création d'un projet EMF vide (File → New → Project…)**

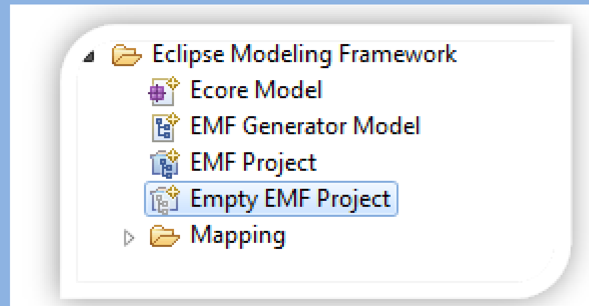


Figure 3.2 : Crée projet EMF vide.

- **Création d'un diagramme Ecore (New → Other →)**

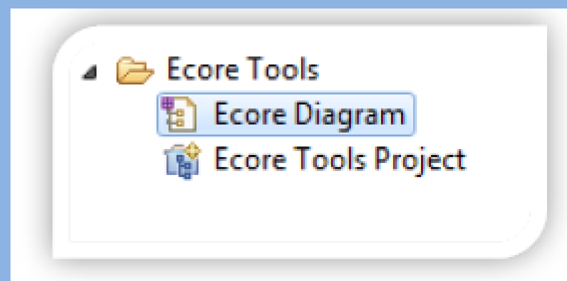


Figure 3.3 : Crée diagramme Ecore.

- **Défini les éléments de notre méta-modèle de DC**

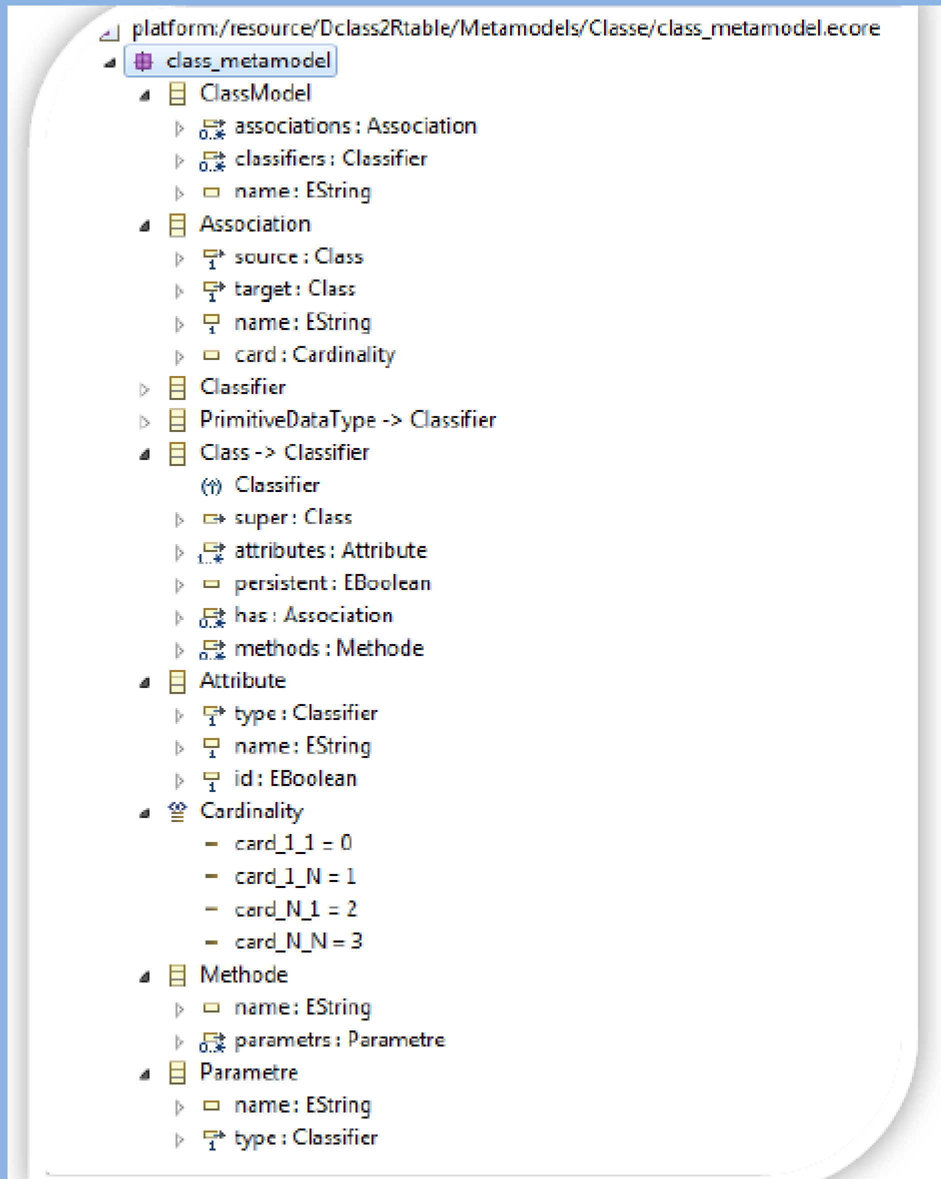
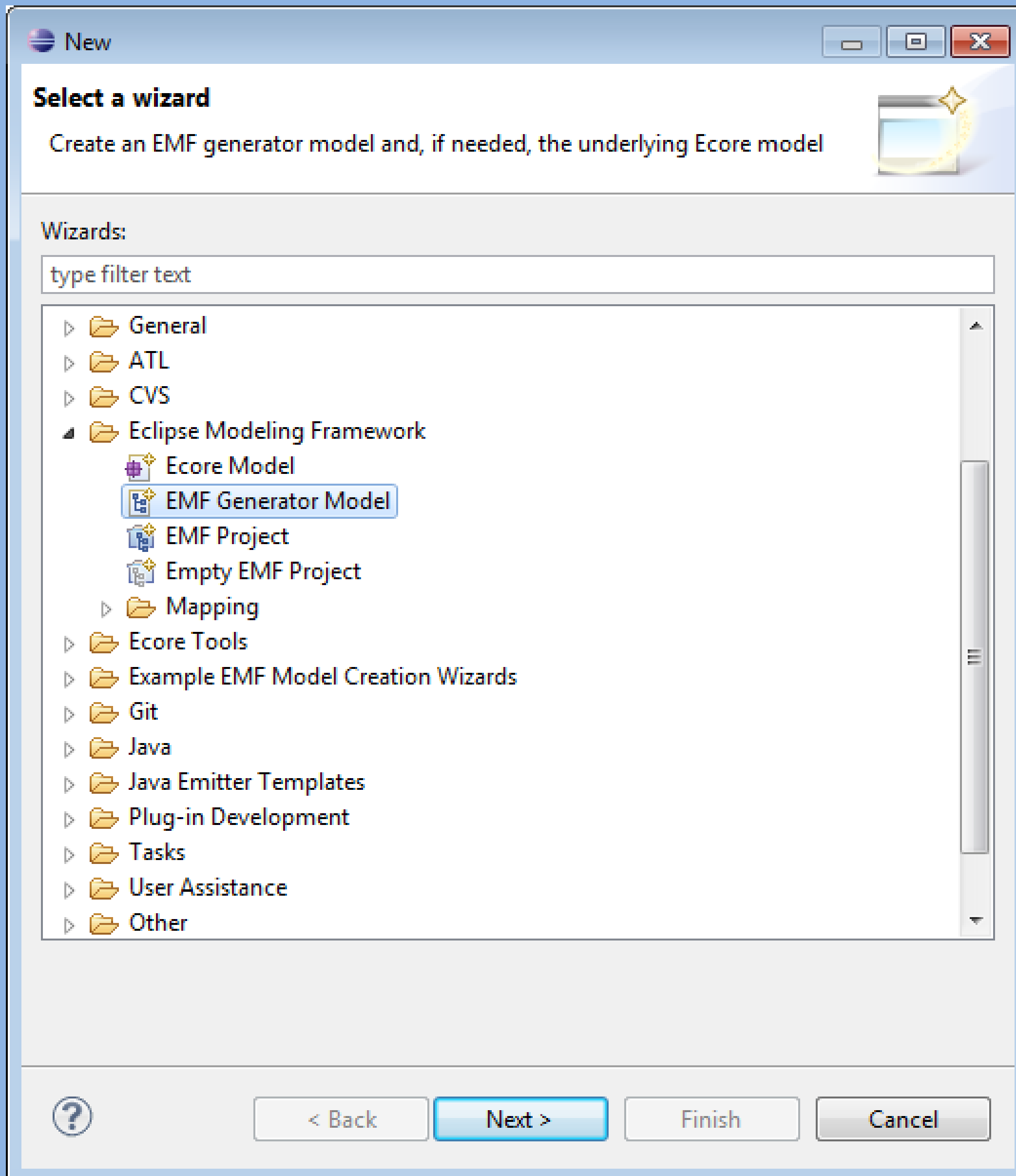


Figure 3.4 : les éléments du méta-modèle de DC.


New
Select a wizard
Create an EMF generator model and, if needed, the underlying Ecore model
Wizards:
type filter text
General
ATL
CVS
Eclipse Modeling Framework
Ecore Model
EMF Generator Model
EMF Project
Empty EMF Project
Mapping
Ecore Tools
Example EMF Model Creation Wizards
Git
Java
Java Emitter Templates
Plug-in Development
Tasks
User Assistance
Other
< Back
Next >
Finish
Cancel

Eclipse Modeling Framework (EMF)

EMF existe depuis 2002, est un framework qui traite des modèles c'est-à-dire qu'EMF offre à ces utilisateurs un cadre de travail pour la manipulation des modèles. EMF permet de stocker les modèles sous forme de fichier pour en assurer la persistance. EMF permet également de traiter différents types de fichiers : conformes à des standards reconnus (UML, XML, XMI) et aussi sous des formes spécifiques (code Java) ou tout simplement sur mesure (au bon gré du concepteur).

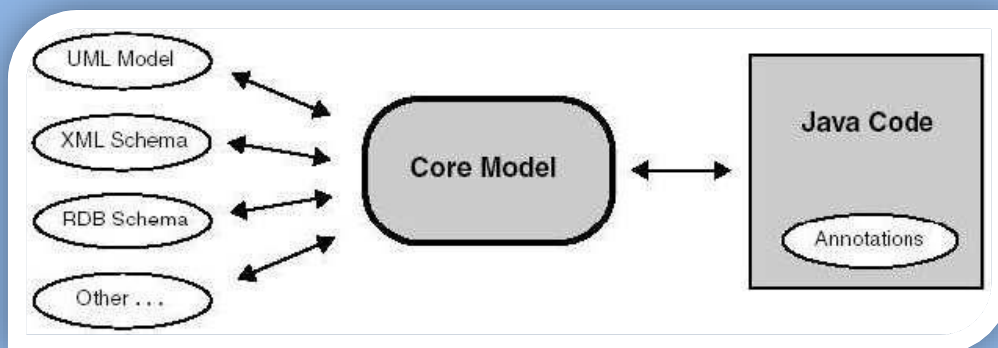
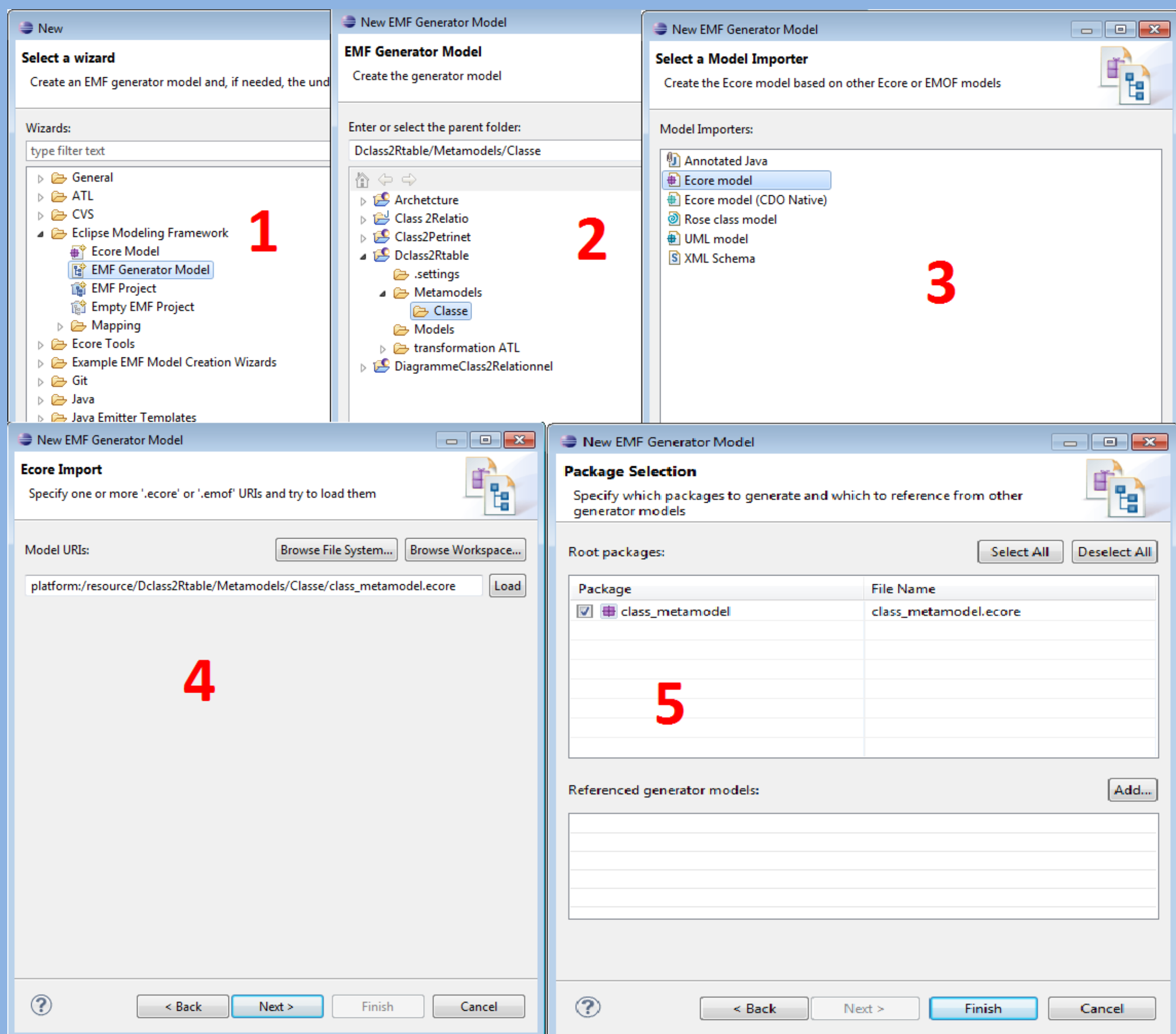
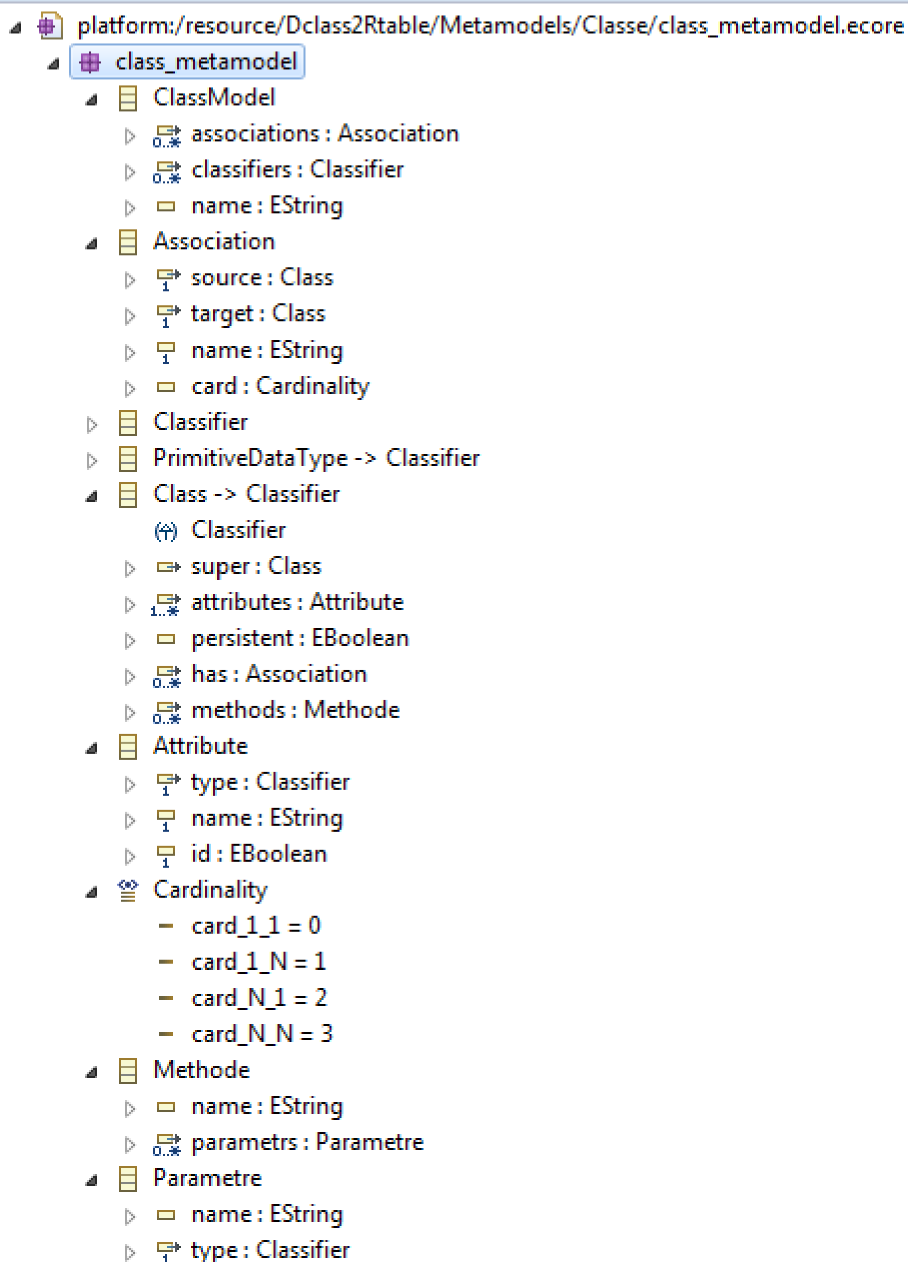


Figure :organisation générale d'EMF

Les etape pour represent diagramme de meta_model

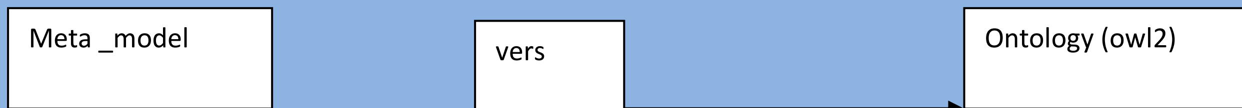




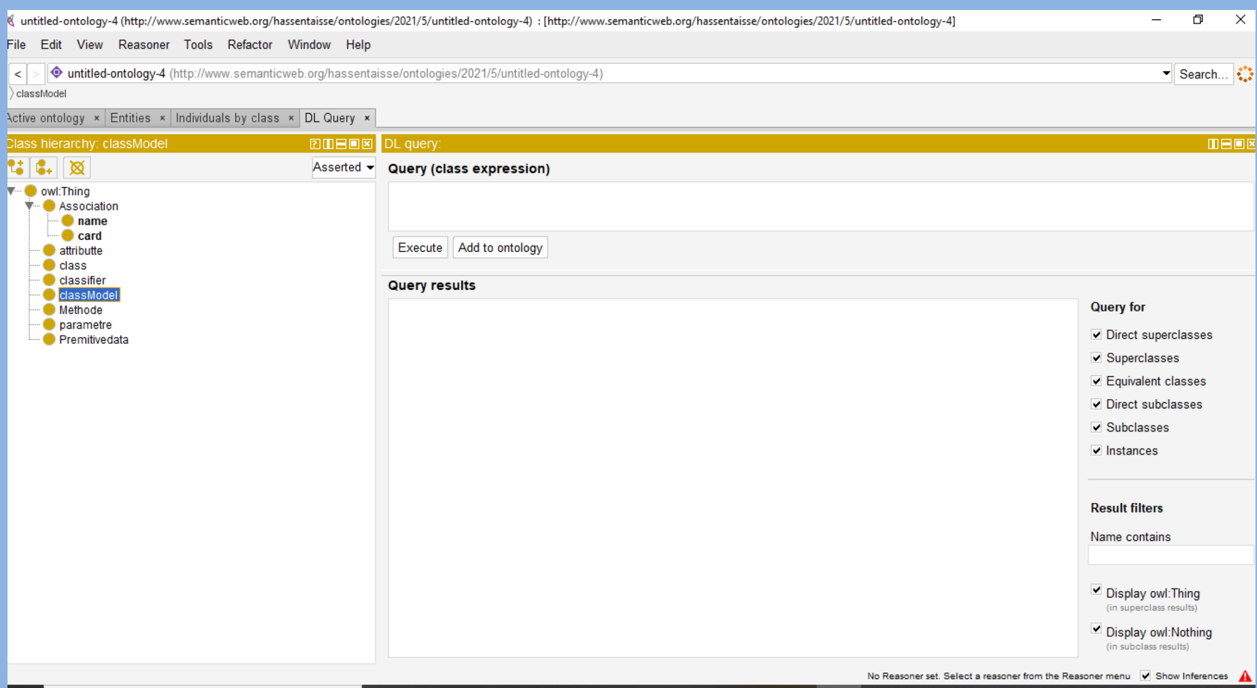
Transformation du modèle

L‟application du scénario le plus évident de TGG est la transformation d‟un modèle à un autre modèle.

Dans ce scénario, l‟un des deux modèles existe déjà, et l‟objectif est générer un modèle correspondant.
TGG est neutre par rapport à la direction de la transformation ou il ne dépend que de ce qui appelé le modèle source ou le modèle cible. Parfois, il n‟est même pas clair, ce qui devrait être la source et ce qui devrait être la cible, de sort que nous appelons simplement les domaines.
Ensuite, on doit indiquer quel domaine doit être utilisé en tant que source et cible pour une transformation



Les outile utilise pour generer le code OWL2.



C'est une application qui convertit un diagramme de classe et définit chaque vecteur et ses variables, puis convertit ou traduit du schéma en code.

L’implementation

```
@prefix dc: <http://purl.org/dc/elements/1.1/> .
@prefix grddl: <http://www.w3.org/2003/g/data-view#> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix xml: <http://www.w3.org/XML/1998/namespace> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

<http://www.w3.org/2002/07/owl> a owl:Ontology ;
     dc:title "The OWL 2 Schema vocabulary (OWL 2)" ;
     rdfs:comment """
  This ontology partially describes the built-in classes and
  properties that together form the basis of the RDF/XML syntax of OWL 2.
  The content of this ontology is based on Tables 6.1 and 6.2
  in Section 6.4 of the OWL 2 RDF-Based Semantics specification,
  available at http://www.w3.org/TR/owl2-rdf-based-semantics/.
  Please note that those tables do not include the different annotations
  (labels, comments and rdfs:isDefinedBy links) used in this file.
  Also note that the descriptions provided in this ontology do not
  provide a complete and correct formal description of either the syntax
  or the semantics of the introduced terms (please see the OWL 2
  recommendations for the complete and normative specifications).
  Furthermore, the information provided by this ontology may be
  misleading if not used with care. This ontology SHOULD NOT be imported
  into OWL ontologies. Importing this file into an OWL 2 DL ontology
  will cause it to become an OWL 2 Full ontology and may have other,
  unexpected, consequences.
   """ ;
     rdfs:isDefinedBy
          <http://www.w3.org/TR/owl2-mapping-to-rdf/>,
```

```
          <http://www.w3.org/TR/owl2-rdf-based-semantics/>,
          <http://www.w3.org/TR/owl2-syntax/> ;
     rdfs:seeAlso   <http://www.w3.org/TR/owl2-rdf-based-semantics/#table-
axiomatic-classes>,
                    <http://www.w3.org/TR/owl2-rdf-based-semantics/#table-
axiomatic-properties> ;
     owl:imports <http://www.w3.org/2000/01/rdf-schema> ;
     owl:versionIRI <http://www.w3.org/2002/07/owl> ;
     owl:versionInfo "$Date: 2009/11/15 10:54:12 $" ;
     grddl:namespaceTransformation <http://dev.w3.org/cvsweb/2009/owl-
grddl/owx2rdf.xsl> .


owl:AllDifferent a rdfs:Class ;
     rdfs:label "AllDifferent" ;
     rdfs:comment "The class of collections of pairwise different
individuals." ;
     rdfs:isDefinedBy <http://www.w3.org/2002/07/owl#> ;
     rdfs:subClassOf rdfs:Resource .

owl:AllDisjointClasses a rdfs:Class ;
     rdfs:label "AllDisjointClasses" ;
     rdfs:comment "The class of collections of pairwise disjoint classes."
;
     rdfs:isDefinedBy <http://www.w3.org/2002/07/owl#> ;
     rdfs:subClassOf rdfs:Resource .

owl:AllDisjointProperties a rdfs:Class ;
     rdfs:label "AllDisjointProperties" ;
     rdfs:comment "The class of collections of pairwise disjoint
properties." ;
     rdfs:isDefinedBy <http://www.w3.org/2002/07/owl#> ;
     rdfs:subClassOf rdfs:Resource .

owl:Annotation a rdfs:Class ;
     rdfs:label "Annotation" ;
     rdfs:comment "The class of annotated annotations for which the RDF
serialization consists of an annotated subject, predicate and object." ;
     rdfs:isDefinedBy <http://www.w3.org/2002/07/owl#> ;
     rdfs:subClassOf rdfs:Resource .

owl:AnnotationProperty a rdfs:Class ;
     rdfs:label "AnnotationProperty" ;
     rdfs:comment "The class of annotation properties." ;
     rdfs:isDefinedBy <http://www.w3.org/2002/07/owl#> ;
     rdfs:subClassOf rdf:Property .

owl:AsymmetricProperty a rdfs:Class ;
     rdfs:label "AsymmetricProperty" ;
     rdfs:comment "The class of asymmetric properties." ;
     rdfs:isDefinedBy <http://www.w3.org/2002/07/owl#> ;
     rdfs:subClassOf owl:ObjectProperty .

owl:Axiom a rdfs:Class ;
     rdfs:label "Axiom" ;
     rdfs:comment "The class of annotated axioms for which the RDF
serialization consists of an annotated subject, predicate and object." ;
     rdfs:isDefinedBy <http://www.w3.org/2002/07/owl#> ;
     rdfs:subClassOf rdfs:Resource .

owl:Class a rdfs:Class ;
```

```
     rdfs:label "Class" ;
     rdfs:comment "The class of OWL classes." ;
     rdfs:isDefinedBy <http://www.w3.org/2002/07/owl#> ;
     rdfs:subClassOf rdfs:Class .

owl:DataRange a rdfs:Class ;
     rdfs:label "DataRange" ;
     rdfs:comment "The class of OWL data ranges, which are special kinds of
datatypes. Note: The use of the IRI owl:DataRange has been deprecated as of
OWL 2. The IRI rdfs:Datatype SHOULD be used instead." ;
     rdfs:isDefinedBy <http://www.w3.org/2002/07/owl#> ;
     rdfs:subClassOf rdfs:Datatype .

owl:DatatypeProperty a rdfs:Class ;
     rdfs:label "DatatypeProperty" ;
     rdfs:comment "The class of data properties." ;
     rdfs:isDefinedBy <http://www.w3.org/2002/07/owl#> ;
     rdfs:subClassOf rdf:Property .

owl:DeprecatedClass a rdfs:Class ;
     rdfs:label "DeprecatedClass" ;
     rdfs:comment "The class of deprecated classes." ;
     rdfs:isDefinedBy <http://www.w3.org/2002/07/owl#> ;
     rdfs:subClassOf rdfs:Class .

owl:DeprecatedProperty a rdfs:Class ;
     rdfs:label "DeprecatedProperty" ;
     rdfs:comment "The class of deprecated properties." ;
     rdfs:isDefinedBy <http://www.w3.org/2002/07/owl#> ;
     rdfs:subClassOf rdf:Property .

owl:FunctionalProperty a rdfs:Class ;
     rdfs:label "FunctionalProperty" ;
     rdfs:comment "The class of functional properties." ;
     rdfs:isDefinedBy <http://www.w3.org/2002/07/owl#> ;
     rdfs:subClassOf rdf:Property .

owl:InverseFunctionalProperty a rdfs:Class ;
     rdfs:label "InverseFunctionalProperty" ;
     rdfs:comment "The class of inverse-functional properties." ;
     rdfs:isDefinedBy <http://www.w3.org/2002/07/owl#> ;
     rdfs:subClassOf owl:ObjectProperty .

owl:IrreflexiveProperty a rdfs:Class ;
     rdfs:label "IrreflexiveProperty" ;
     rdfs:comment "The class of irreflexive properties." ;
     rdfs:isDefinedBy <http://www.w3.org/2002/07/owl#> ;
     rdfs:subClassOf owl:ObjectProperty .

owl:NamedIndividual a rdfs:Class ;
     rdfs:label "NamedIndividual" ;
     rdfs:comment "The class of named individuals." ;
     rdfs:isDefinedBy <http://www.w3.org/2002/07/owl#> ;
     rdfs:subClassOf owl:Thing .

owl:NegativePropertyAssertion a rdfs:Class ;
     rdfs:label "NegativePropertyAssertion" ;
     rdfs:comment "The class of negative property assertions." ;
     rdfs:isDefinedBy <http://www.w3.org/2002/07/owl#> ;
     rdfs:subClassOf rdfs:Resource .
```

```
owl:Nothing a owl:Class ;
     rdfs:label "Nothing" ;
     rdfs:comment "This is the empty class." ;
     rdfs:isDefinedBy <http://www.w3.org/2002/07/owl#> ;
     rdfs:subClassOf owl:Thing .

owl:ObjectProperty a rdfs:Class ;
     rdfs:label "ObjectProperty" ;
     rdfs:comment "The class of object properties." ;
     rdfs:isDefinedBy <http://www.w3.org/2002/07/owl#> ;
     rdfs:subClassOf rdf:Property .

owl:Ontology a rdfs:Class ;
     rdfs:label "Ontology" ;
     rdfs:comment "The class of ontologies." ;
     rdfs:isDefinedBy <http://www.w3.org/2002/07/owl#> ;
     rdfs:subClassOf rdfs:Resource .

owl:OntologyProperty a rdfs:Class ;
     rdfs:label "OntologyProperty" ;
     rdfs:comment "The class of ontology properties." ;
     rdfs:isDefinedBy <http://www.w3.org/2002/07/owl#> ;
     rdfs:subClassOf rdf:Property .

owl:ReflexiveProperty a rdfs:Class ;
     rdfs:label "ReflexiveProperty" ;
     rdfs:comment "The class of reflexive properties." ;
     rdfs:isDefinedBy <http://www.w3.org/2002/07/owl#> ;
     rdfs:subClassOf owl:ObjectProperty .

owl:Restriction a rdfs:Class ;
     rdfs:label "Restriction" ;
     rdfs:comment "The class of property restrictions." ;
     rdfs:isDefinedBy <http://www.w3.org/2002/07/owl#> ;
     rdfs:subClassOf owl:Class .

owl:SymmetricProperty a rdfs:Class ;
     rdfs:label "SymmetricProperty" ;
     rdfs:comment "The class of symmetric properties." ;
     rdfs:isDefinedBy <http://www.w3.org/2002/07/owl#> ;
     rdfs:subClassOf owl:ObjectProperty .

owl:TransitiveProperty a rdfs:Class ;
     rdfs:label "TransitiveProperty" ;
     rdfs:comment "The class of transitive properties." ;
     rdfs:isDefinedBy <http://www.w3.org/2002/07/owl#> ;
     rdfs:subClassOf owl:ObjectProperty .

owl:Thing a owl:Class ;
     rdfs:label "Thing" ;
     rdfs:comment "The class of OWL individuals." ;
     rdfs:isDefinedBy <http://www.w3.org/2002/07/owl#> .

owl:allValuesFrom a rdf:Property ;
     rdfs:label "allValuesFrom" ;
     rdfs:comment "The property that determines the class that a universal
property restriction refers to." ;
     rdfs:domain owl:Restriction ;
     rdfs:isDefinedBy <http://www.w3.org/2002/07/owl#> ;
     rdfs:range rdfs:Class .
```

```
owl:annotatedProperty a rdf:Property ;
     rdfs:label "annotatedProperty" ;
     rdfs:comment "The property that determines the predicate of an
annotated axiom or annotated annotation." ;
     rdfs:domain rdfs:Resource ;
     rdfs:isDefinedBy <http://www.w3.org/2002/07/owl#> ;
     rdfs:range rdfs:Resource .

owl:annotatedSource a rdf:Property ;
     rdfs:label "annotatedSource" ;
     rdfs:comment "The property that determines the subject of an annotated
axiom or annotated annotation." ;
     rdfs:domain rdfs:Resource ;
     rdfs:isDefinedBy <http://www.w3.org/2002/07/owl#> ;
     rdfs:range rdfs:Resource .

owl:annotatedTarget a rdf:Property ;
     rdfs:label "annotatedTarget" ;
     rdfs:comment "The property that determines the object of an annotated
axiom or annotated annotation." ;
     rdfs:domain rdfs:Resource ;
     rdfs:isDefinedBy <http://www.w3.org/2002/07/owl#> ;
     rdfs:range rdfs:Resource .

owl:assertionProperty a rdf:Property ;
     rdfs:label "assertionProperty" ;
     rdfs:comment "The property that determines the predicate of a negative
property assertion." ;
     rdfs:domain owl:NegativePropertyAssertion ;
     rdfs:isDefinedBy <http://www.w3.org/2002/07/owl#> ;
     rdfs:range rdf:Property .

owl:backwardCompatibleWith a owl:AnnotationProperty, owl:OntologyProperty ;
     rdfs:label "backwardCompatibleWith" ;
     rdfs:comment "The annotation property that indicates that a given
ontology is backward compatible with another ontology." ;
     rdfs:domain owl:Ontology ;
     rdfs:isDefinedBy <http://www.w3.org/2002/07/owl#> ;
     rdfs:range owl:Ontology .

owl:bottomDataProperty a owl:DatatypeProperty ;
     rdfs:label "bottomDataProperty" ;
     rdfs:comment "The data property that does not relate any individual to
any data value." ;
     rdfs:domain owl:Thing ;
     rdfs:isDefinedBy <http://www.w3.org/2002/07/owl#> ;
     rdfs:range rdfs:Literal .

owl:bottomObjectProperty a owl:ObjectProperty ;
     rdfs:label "bottomObjectProperty" ;
     rdfs:comment "The object property that does not relate any two
individuals." ;
     rdfs:domain owl:Thing ;
     rdfs:isDefinedBy <http://www.w3.org/2002/07/owl#> ;
     rdfs:range owl:Thing .

owl:cardinality a rdf:Property ;
     rdfs:label "cardinality" ;
     rdfs:comment "The property that determines the cardinality of an exact
cardinality restriction." ;
     rdfs:domain owl:Restriction ;
```

```
     rdfs:isDefinedBy <http://www.w3.org/2002/07/owl#> ;
     rdfs:range xsd:nonNegativeInteger .

owl:complementOf a rdf:Property ;
     rdfs:label "complementOf" ;
     rdfs:comment "The property that determines that a given class is the
complement of another class." ;
     rdfs:domain owl:Class ;
     rdfs:isDefinedBy <http://www.w3.org/2002/07/owl#> ;
     rdfs:range owl:Class .

owl:datatypeComplementOf a rdf:Property ;
     rdfs:label "datatypeComplementOf" ;
     rdfs:comment "The property that determines that a given data range is
the complement of another data range with respect to the data domain." ;
     rdfs:domain rdfs:Datatype ;
     rdfs:isDefinedBy <http://www.w3.org/2002/07/owl#> ;
     rdfs:range rdfs:Datatype .

owl:deprecated a owl:AnnotationProperty ;
     rdfs:label "deprecated" ;
     rdfs:comment "The annotation property that indicates that a given
entity has been deprecated." ;
     rdfs:domain rdfs:Resource ;
     rdfs:isDefinedBy <http://www.w3.org/2002/07/owl#> ;
     rdfs:range rdfs:Resource .

owl:differentFrom a rdf:Property ;
     rdfs:label "differentFrom" ;
     rdfs:comment "The property that determines that two given individuals
are different." ;
     rdfs:domain owl:Thing ;
     rdfs:isDefinedBy <http://www.w3.org/2002/07/owl#> ;
     rdfs:range owl:Thing .

owl:disjointUnionOf a rdf:Property ;
     rdfs:label "disjointUnionOf" ;
     rdfs:comment "The property that determines that a given class is
equivalent to the disjoint union of a collection of other classes." ;
     rdfs:domain owl:Class ;
     rdfs:isDefinedBy <http://www.w3.org/2002/07/owl#> ;
     rdfs:range rdf:List .

owl:disjointWith a rdf:Property ;
     rdfs:label "disjointWith" ;
     rdfs:comment "The property that determines that two given classes are
disjoint." ;
     rdfs:domain owl:Class ;
     rdfs:isDefinedBy <http://www.w3.org/2002/07/owl#> ;
     rdfs:range owl:Class .

owl:distinctMembers a rdf:Property ;
     rdfs:label "distinctMembers" ;
     rdfs:comment "The property that determines the collection of pairwise
different individuals in a owl:AllDifferent axiom." ;
     rdfs:domain owl:AllDifferent ;
     rdfs:isDefinedBy <http://www.w3.org/2002/07/owl#> ;
     rdfs:range rdf:List .

owl:equivalentClass a rdf:Property ;
     rdfs:label "equivalentClass" ;
```

```
     rdfs:comment "The property that determines that two given classes are
equivalent, and that is used to specify datatype definitions." ;
     rdfs:domain rdfs:Class ;
     rdfs:isDefinedBy <http://www.w3.org/2002/07/owl#> ;
     rdfs:range rdfs:Class .

owl:equivalentProperty a rdf:Property ;
     rdfs:label "equivalentProperty" ;
     rdfs:comment "The property that determines that two given properties
are equivalent." ;
     rdfs:domain rdf:Property ;
     rdfs:isDefinedBy <http://www.w3.org/2002/07/owl#> ;
     rdfs:range rdf:Property .

owl:hasKey a rdf:Property ;
     rdfs:label "hasKey" ;
     rdfs:comment "The property that determines the collection of
properties that jointly build a key." ;
     rdfs:domain owl:Class ;
     rdfs:isDefinedBy <http://www.w3.org/2002/07/owl#> ;
     rdfs:range rdf:List .

owl:hasSelf a rdf:Property ;
     rdfs:label "hasSelf" ;
     rdfs:comment "The property that determines the property that a self
restriction refers to." ;
     rdfs:domain owl:Restriction ;
     rdfs:isDefinedBy <http://www.w3.org/2002/07/owl#> ;
     rdfs:range rdfs:Resource .

owl:hasValue a rdf:Property ;
     rdfs:label "hasValue" ;
     rdfs:comment "The property that determines the individual that a has-
value restriction refers to." ;
     rdfs:domain owl:Restriction ;
     rdfs:isDefinedBy <http://www.w3.org/2002/07/owl#> ;
     rdfs:range rdfs:Resource .

owl:imports a owl:OntologyProperty ;
     rdfs:label "imports" ;
     rdfs:comment "The property that is used for importing other ontologies
into a given ontology." ;
     rdfs:domain owl:Ontology ;
     rdfs:isDefinedBy <http://www.w3.org/2002/07/owl#> ;
     rdfs:range owl:Ontology .

owl:incompatibleWith a owl:AnnotationProperty, owl:OntologyProperty ;
     rdfs:label "incompatibleWith" ;
     rdfs:comment "The annotation property that indicates that a given
ontology is incompatible with another ontology." ;
     rdfs:domain owl:Ontology ;
     rdfs:isDefinedBy <http://www.w3.org/2002/07/owl#> ;
     rdfs:range owl:Ontology .

owl:intersectionOf a rdf:Property ;
     rdfs:label "intersectionOf" ;
     rdfs:comment "The property that determines the collection of classes
or data ranges that build an intersection." ;
     rdfs:domain rdfs:Class ;
     rdfs:isDefinedBy <http://www.w3.org/2002/07/owl#> ;
     rdfs:range rdf:List .
```

```
owl:inverseOf a rdf:Property ;
     rdfs:label "inverseOf" ;
     rdfs:comment "The property that determines that two given properties
are inverse." ;
     rdfs:domain owl:ObjectProperty ;
     rdfs:isDefinedBy <http://www.w3.org/2002/07/owl#> ;
     rdfs:range owl:ObjectProperty .

owl:maxCardinality a rdf:Property ;
     rdfs:label "maxCardinality" ;
     rdfs:comment "The property that determines the cardinality of a
maximum cardinality restriction." ;
     rdfs:domain owl:Restriction ;
     rdfs:isDefinedBy <http://www.w3.org/2002/07/owl#> ;
     rdfs:range xsd:nonNegativeInteger .

owl:maxQualifiedCardinality a rdf:Property ;
     rdfs:label "maxQualifiedCardinality" ;
     rdfs:comment "The property that determines the cardinality of a
maximum qualified cardinality restriction." ;
     rdfs:domain owl:Restriction ;
     rdfs:isDefinedBy <http://www.w3.org/2002/07/owl#> ;
     rdfs:range xsd:nonNegativeInteger .

owl:members a rdf:Property ;
     rdfs:label "members" ;
     rdfs:comment "The property that determines the collection of members
in either a owl:AllDifferent, owl:AllDisjointClasses or
owl:AllDisjointProperties axiom." ;
     rdfs:domain rdfs:Resource ;
     rdfs:isDefinedBy <http://www.w3.org/2002/07/owl#> ;
     rdfs:range rdf:List .

owl:minCardinality a rdf:Property ;
     rdfs:label "minCardinality" ;
     rdfs:comment "The property that determines the cardinality of a
minimum cardinality restriction." ;
     rdfs:domain owl:Restriction ;
     rdfs:isDefinedBy <http://www.w3.org/2002/07/owl#> ;
     rdfs:range xsd:nonNegativeInteger .

owl:minQualifiedCardinality a rdf:Property ;
     rdfs:label "minQualifiedCardinality" ;
     rdfs:comment "The property that determines the cardinality of a
minimum qualified cardinality restriction." ;
     rdfs:domain owl:Restriction ;
     rdfs:isDefinedBy <http://www.w3.org/2002/07/owl#> ;
     rdfs:range xsd:nonNegativeInteger .

owl:onClass a rdf:Property ;
     rdfs:label "onClass" ;
     rdfs:comment "The property that determines the class that a qualified
object cardinality restriction refers to." ;
     rdfs:domain owl:Restriction ;
     rdfs:isDefinedBy <http://www.w3.org/2002/07/owl#> ;
     rdfs:range owl:Class .

owl:onDataRange a rdf:Property ;
     rdfs:label "onDataRange" ;
```

```
     rdfs:comment "The property that determines the data range that a
qualified data cardinality restriction refers to." ;
     rdfs:domain owl:Restriction ;
     rdfs:isDefinedBy <http://www.w3.org/2002/07/owl#> ;
     rdfs:range rdfs:Datatype .

owl:onDatatype a rdf:Property ;
     rdfs:label "onDatatype" ;
     rdfs:comment "The property that determines the datatype that a
datatype restriction refers to." ;
     rdfs:domain rdfs:Datatype ;
     rdfs:isDefinedBy <http://www.w3.org/2002/07/owl#> ;
     rdfs:range rdfs:Datatype .

owl:oneOf a rdf:Property ;
     rdfs:label "oneOf" ;
     rdfs:comment "The property that determines the collection of
individuals or data values that build an enumeration." ;
     rdfs:domain rdfs:Class ;
     rdfs:isDefinedBy <http://www.w3.org/2002/07/owl#> ;
     rdfs:range rdf:List .

owl:onProperties a rdf:Property ;
     rdfs:label "onProperties" ;
     rdfs:comment "The property that determines the n-tuple of properties
that a property restriction on an n-ary data range refers to." ;
     rdfs:domain owl:Restriction ;
     rdfs:isDefinedBy <http://www.w3.org/2002/07/owl#> ;
     rdfs:range rdf:List .

owl:onProperty a rdf:Property ;
     rdfs:label "onProperty" ;
     rdfs:comment "The property that determines the property that a
property restriction refers to." ;
     rdfs:domain owl:Restriction ;
     rdfs:isDefinedBy <http://www.w3.org/2002/07/owl#> ;
     rdfs:range rdf:Property .

owl:priorVersion a owl:AnnotationProperty, owl:OntologyProperty ;
     rdfs:label "priorVersion" ;
     rdfs:comment "The annotation property that indicates the predecessor
ontology of a given ontology." ;
     rdfs:domain owl:Ontology ;
     rdfs:isDefinedBy <http://www.w3.org/2002/07/owl#> ;
     rdfs:range owl:Ontology .

owl:propertyChainAxiom a rdf:Property ;
     rdfs:label "propertyChainAxiom" ;
     rdfs:comment "The property that determines the n-tuple of properties
that build a sub property chain of a given property." ;
     rdfs:domain owl:ObjectProperty ;
     rdfs:isDefinedBy <http://www.w3.org/2002/07/owl#> ;
     rdfs:range rdf:List .

owl:propertyDisjointWith a rdf:Property ;
     rdfs:label "propertyDisjointWith" ;
     rdfs:comment "The property that determines that two given properties
are disjoint." ;
     rdfs:domain rdf:Property ;
     rdfs:isDefinedBy <http://www.w3.org/2002/07/owl#> ;
     rdfs:range rdf:Property .
```

```
owl:qualifiedCardinality a rdf:Property ;
     rdfs:label "qualifiedCardinality" ;
     rdfs:comment "The property that determines the cardinality of an exact
qualified cardinality restriction." ;
     rdfs:domain owl:Restriction ;
     rdfs:isDefinedBy <http://www.w3.org/2002/07/owl#> ;
     rdfs:range xsd:nonNegativeInteger .

owl:sameAs a rdf:Property ;
     rdfs:label "sameAs" ;
     rdfs:comment "The property that determines that two given individuals
are equal." ;
     rdfs:domain owl:Thing ;
     rdfs:isDefinedBy <http://www.w3.org/2002/07/owl#> ;
     rdfs:range owl:Thing .

owl:someValuesFrom a rdf:Property ;
     rdfs:label "someValuesFrom" ;
     rdfs:comment "The property that determines the class that an
existential property restriction refers to." ;
     rdfs:domain owl:Restriction ;
     rdfs:isDefinedBy <http://www.w3.org/2002/07/owl#> ;
     rdfs:range rdfs:Class .

owl:sourceIndividual a rdf:Property ;
     rdfs:label "sourceIndividual" ;
     rdfs:comment "The property that determines the subject of a negative
property assertion." ;
     rdfs:domain owl:NegativePropertyAssertion ;
     rdfs:isDefinedBy <http://www.w3.org/2002/07/owl#> ;
     rdfs:range owl:Thing .

owl:targetIndividual a rdf:Property ;
     rdfs:label "targetIndividual" ;
     rdfs:comment "The property that determines the object of a negative
object property assertion." ;
     rdfs:domain owl:NegativePropertyAssertion ;
     rdfs:isDefinedBy <http://www.w3.org/2002/07/owl#> ;
     rdfs:range owl:Thing .

owl:targetValue a rdf:Property ;
     rdfs:label "targetValue" ;
     rdfs:comment "The property that determines the value of a negative
data property assertion." ;
     rdfs:domain owl:NegativePropertyAssertion ;
     rdfs:isDefinedBy <http://www.w3.org/2002/07/owl#> ;
     rdfs:range rdfs:Literal .

owl:topDataProperty a owl:DatatypeProperty ;
     rdfs:label "topDataProperty" ;
     rdfs:comment "The data property that relates every individual to every
data value." ;
     rdfs:domain owl:Thing ;
     rdfs:isDefinedBy <http://www.w3.org/2002/07/owl#> ;
     rdfs:range rdfs:Literal .

owl:topObjectProperty a owl:ObjectProperty ;
     rdfs:label "topObjectProperty" ;
     rdfs:comment "The object property that relates every two individuals."
;
```

```
     rdfs:domain owl:Thing ;
     rdfs:isDefinedBy <http://www.w3.org/2002/07/owl#> ;
     rdfs:range owl:Thing .

owl:unionOf a rdf:Property ;
     rdfs:label "unionOf" ;
     rdfs:comment "The property that determines the collection of classes
or data ranges that build a union." ;
     rdfs:domain rdfs:Class ;
     rdfs:isDefinedBy <http://www.w3.org/2002/07/owl#> ;
     rdfs:range rdf:List .

owl:versionInfo a owl:AnnotationProperty ;
     rdfs:label "versionInfo" ;
     rdfs:comment "The annotation property that provides version
information for an ontology or another OWL construct." ;
     rdfs:domain rdfs:Resource ;
     rdfs:isDefinedBy <http://www.w3.org/2002/07/owl#> ;
     rdfs:range rdfs:Resource .

owl:versionIRI a owl:OntologyProperty ;
     rdfs:label "versionIRI" ;
     rdfs:comment "The property that identifies the version IRI of an
ontology." ;
     rdfs:domain owl:Ontology ;
     rdfs:isDefinedBy <http://www.w3.org/2002/07/owl#> ;
     rdfs:range owl:Ontology .

owl:withRestrictions a rdf:Property ;
     rdfs:label "withRestrictions" ;
     rdfs:comment "The property that determines the collection of facet-
value pairs that define a datatype restriction." ;
     rdfs:domain rdfs:Datatype ;
     rdfs:isDefinedBy <http://www.w3.org/2002/07/owl#> ;
     rdfs:range rdf:List .
```

Conclusion

La proposition proposée dans cet article est un processus qui vise à transformer les modèles UML vers un langage d'ontologie réalisé par TGG Interpreter, en s'appuyant sur la notion de TGGs.

C'est un approche algébrique, c'est pourquoi elle semble un bon choix pour automatiser la transformation.

De plus, le principal avantage de ces approches est qu'elles offrent une grande puissance de Abstraction.

Par conséquent, nous pouvons spécifier le comportement du système en nous concentrant uniquement sur ce que le système est censé faire mais pas comment il est censé le faire.

De plus, les TGG sont formalisme formel, naturel, visuel et de haut niveau pour représenter les transformations, et que

Contrairement à d'autres processus de transformation.

Les règles TGG proposées dans cet article sont destiné à automatiser la génération de spécifications d'ontologie pour diagrammes UML.

Par conséquent, nous avons fait un couplage systématique entre une modélisation objet

et une méthode formelle.

Nous pouvons conclure que nous avons donné à la recherche un grand avantage, puisque ce couplage est spécifiquement garanti par les angles complémentaires et les prises croisées de

ces deux techniques. On peut aussi dire que nous avons fourni la base de nos futurs travaux. Celui-ci, fournira un processus complet, à commencer par l'automatisation du graphe transformation et se terminant par la vérification des résultats obtenus de cette transformation. Puisque les spécifications de l'ontologie définissent un cadre sémantique pour les modèles objets, nous intégrons

la puissance de l'outil d'ontologie.

Conclusion générale

La démarche MDA, bien qu’assez récente, suscite un réel intérêt chez bon nombre d’industriels et de développeurs.

En effet, cette démarche est prometteuse et répond à des attentes légitimes non comblées par les technologies objet ou composant, elle autorise la séparation du logique métier de l’entreprise de son implémentation physique.

Pour que le MDA se diffuse auprès des développeurs, le savoir doit être essaimé de nouveaux outils doivent être développés. Pour produire ces outils qui font encore aujourd’hui défaut, plusieurs projets ont été lancés pour que l’approche MDA tienne ses promesses.

Ces outils permettront l’automatisation des transformations ainsi que la génération automatique de code à partir de modèles.

Le travail présenté dans ce mémoire s’inscrit dans le domaine de l’ingénierie dirigée par les modèles.

Il se base essentiellement sur l’utilisation combinée de méta –modélisation et de transformation de modèle. Plus précisément, la méta-modélisation et transformation des diagrammes de classe, à l’aide d’EMF et ATL.

L’EMF présente un Framework complet et extensible pour le développement MDA.

L’objectif de notre travail est de suivre cette démarche pour résoudre certains problèmes rencontrés lorsqu’on utilise le langage de modélisation le plus populaire (UML).

Aujourd'hui, lors de la conception de base de données, il devient de plus en plus courant d'utiliser la modélisation UML plutôt que le traditionnel modèle entités-association.

On peut donc transformer automatiquement des diagrammes de classe en BDD.

Nous cherchons dans un travail futur d’étendre notre approche pour couvrir tous les autres concepts utilisés dans les diagrammes de classe (héritage, composition, …etc) et de générer le code java convenable permettant d’accéder et de manipuler la base de données déjà générée.

Enfin nous souhaitons que ce travail puisse être une base pour d’éventuels travaux et puisse être amélioré et enrichi davantage.

ملخص:

نأمل من خلال هذه المذكرة, في تقديم مقاربة MDA بغية معالجة وتحويل مخططات الفئات UML إلى توليد كود (OWL2),التي تعتبر أداة فعالة لتحليل سلوكيات الأنظمة.

بالنسبة لمخططات الفئات, المشكل القار هو عدم القدرة على التنبؤ بسلوك النظام بعد تفعيله.

تعتمد المقاربة المطروحة على المعيار EMF ونموذجين تعريفيين: أحدهما خاص بمخططات الفئات والأخر خاص بكيفية تحويل مخطط فئات وكيفية تحويل عناصره وطريقة ربطها فيما بينها .

من اجل تحويل مخطط الفئات إلى كود موافقة,تدعو الحاجة إلى استخدام لغة Protege, ولإتمام العملية لا بد من تحديد مجموعة قواعد للتحويل بمقدورها انجاز الإجراء بصفة آلية.

كلمات مفتاحيه: IDM ,MDA,UML, diagramme de classe, réseau de Pétri, transformation, ATL, méta-modèle, MOF

Abstrait:

A travers cette note, nous espérons introduire une approche MDA pour traiter et transformer les diagrammes de classes UML en génération de code OWL2, qui est un outil efficace pour l'analyse du comportement des systèmes.

Pour les schémas de classes, le problème persistant est l'incapacité à prédire le comportement du système après son activation.

L'approche proposée est basée sur la norme EMF et deux modèles de définition : un pour les schémas de catégories et l'autre pour savoir comment transformer un schéma de catégories, comment transformer ses éléments et comment les lier les uns aux autres.

Afin de convertir le schéma de classe en un code d'approbation, il est nécessaire d'utiliser le langage Protege, et pour terminer le processus, il est nécessaire de définir un ensemble de règles de transformation qui peuvent exécuter la procédure automatiquement.

Mots clés : IDM, MDA, UML, diagramme de classe, réseau de Pétri, transformation, ATL, méta-modèle, MOF

Abstract:

Through this note, we hope to introduce an MDA approach to processing and transforming UML class diagrams into OWL2 code generation, which is an effective tool for systems behavior analysis.

For class schemas, the persistent problem is the inability to predict the behavior of the system after its activation.

The proposed approach is based on the EMF standard and two defining models: one for category schemas and the other for how to transform a category schema, how to transform its elements, and how to link them to each other.

In order to convert the class schema into an approval code, it is necessary to use the Protege language, and to complete the process it is necessary to define a set of transformation rules that can perform the procedure automatically.

Keywords: IDM, MDA, UML, diagramme de classe, réseau de Pétri, transformation, ATL, méta-modèle, MOF

Références

[01] BERNERS-LEE Tim, HENDLER James & LASILLA Ora (2001), The Semantic Web, Scientifique American.

[02] Delia Codruta ROGOZAN (2008), Gestion de l'évolution d'une ontologie : méthodes et outils pour un référencement sémantique évolutif fondé sur une analyse des changements entre versions de l'ontologie proposition de recherche doctorale en informatique cognitive (DIC 9410) Télé-Université de Québec.

[3] Riichiro Mizoguchi, mitsuru Ikeda, and Katherine Sinitsa (1997), Roles of Shared Ontology in AI-ED Research: Intelligence, Conceptualization, Standardization, and Reusability, In Proceedings of the 8th World Conference On Artificial Intelligence In Education AIED-97, Kobe, Japan, August, 1997, 99.537-544, <http://www.ei.sanken.osakau.ac.jp/ieee/Them.paper.html>,

[1]- Benoit combemale , ingénierie dirigée par les modelés (idm) état de l'art, université de toulouse article publie le 12 aout 2008.

[2]-Marcus Alanen and Ivan Porres. Difference and union of models. In UML Conference, pages 217, San Francisco, California, Octobre 2003. Springer-Verlag LNCS 2863.

[4]- Jamal abd-ali, méta modélisation et transformation automatique de psm dans une approche mda, mai 2006.

[5]- Hubert Kadima. Conception orienté objet guidée par les mod`eles. Dunod, 2005.

[6]- Mireille Blay-Fornarino Jean-Marie Favre, Jacky Estublier. L'ing´enierie dirig´ee parles mod`eles - Au-del`a du MDA. Herm`es - Lavoisier, 2006.

[7]- Jean Bezivin & Xavier blanc, mda : vers un important changement De paradigme en génie logiciel, Juillet 2002.

[8]- Jean Bézivin & Xavier Blanc, MDA : VERS UN IMPORTANT CHANGEMENT DE PARADIGME EN GENIE LOGICIEL, Université de Nantes.

[9]- Mohamed HADJ KACEM , Modélisation des applications distribuées à architecture dynamique : Conception et Validation , Novembre 2008.

[10]- NGUYEN Viet HOA, Capitalisation des architectures métiers pour une implémentation sur Différentes plates –formes techniques en utilisant la démarche MDA, Hanoï, Juin 2008.

[11]- Jacques barzic, model driven architecture (mda), conservatoire national des arts et métiers, fevrier 2007.

[12]- Benoit Combemale, Xavier Cr_egut, Marc Pantel , Transformation de Modèles Introduction _a ATL , IRISA CNRS Laboratory, University of Rennes 1,derniére mise _a jour le 24 octobre 2010

[14]- LAVIGNASSE Karen - LÉPINE Nathalie - MOLLIÈRE Hélène SROUR Youssef - SUDRE Raphaël, Génération de fichiers de paramétrage Projet opérationnel, Nantes, 24 mars 2007.

[15]- Jacques Barzic, Eclipse et ses plugins de modélisation (EMF – GEF – GMF), PDF créé le 12 janvier 2008.

[16]- XAVIER Blanc, MDA en Action, Groupe Eyrolles, 2005

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

[21]- Xin Jin, Applying Model Driven Architecture approach to Model Role Based Access Control System, Master of Science in System Science, University of Ottawa, Ontario, Canada ©2006 Xin Jin.

[22]- UML, diagrammes de classes, Introduction à la programmation orientée objets, chapitre 9

[23]- Méthodologie des systèmes d'information UML course dispenser par Annick Lassus. CNAM ANGOULEME 2000-2001

[24]- Laurent Audibert, UML 2 Edition 2007/2008,

[25]-http://laurent-audibert.developpez.com/cours-bd/.

[26]- Ass. MANYA F., Note du cours de SGBD, G2 INFO/U.KA. 2007 - 2008.

[27]- Guézelo, P. (2006). ModElisation des données : Approche pour la conception des bases des données (http://philippe.guezelou.fr/mcd.htm)..

[28]- Le bundle ATL pour Windows: http://www.sciences.univ-nantes.fr/lina/atl/atldemo/adt/.

[29]- JeffRothenberg, the Nature of Modeling, 1989 .

[30]- OMG, Object Constraint Language OMG Available Specification, Version 2.0, formal/06-05-01

[32] J.P. MATHERON, Comprendre Merise, Edition EYROLLES, 2005, p.210.