Atelier de conception pour l'évolution des systèmes PLM : une approche d'ingénierie dirigée par les modèles

Onur Yildiz

UNIVERSITÉ GRENOBLE ALPES

THÈSE

Pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ DE GRENOBLE ALPES

Spécialité : **Génie industriel**

Arrêté ministériel : 7 août 2006

Présentée par

Onur YILDIZ

Thèse dirigée par **Michel TOLLENAERE**
et codirigée par **Lilia GZARA et Philippe PERNELLE**

préparée au sein **du laboratoire G-SCOP (Sciences pour la Conception, l'Optimisation et la Production de Grenoble)**
et de **l'Ecole Doctorale I-MEP2 (Ingénierie - Matériaux Mécanique Energétique Environnement Procédés Production)**

Atelier de conception pour l'évolution des systèmes PLM : une approche d'ingénierie dirigée par les modèles

Thèse soutenue publiquement le **21 septembre 2015**,
devant le jury composé de :

Monsieur Dominique DENEUX
Professeur, Université de Valenciennes, Président
Monsieur Alain BERNARD
Professeur, École Centrale de Nantes, Rapporteur
Monsieur Samuel GOMES
Professeur, Université de Technologie de Belfort-Montbéliard, Rapporteur
Monsieur Michel TOLLENAERE
Professeur, Grenoble INP, Directeur de thèse
Madame Lilia GZARA
Docteur, Maître de conférences, HDR, Grenoble INP, Examinatrice
Monsieur Philippe PERNELLE
Docteur, Maître de conférences, Université Claude Bernard Lyon, Examinateur
Monsieur Kamal CHEBALLAH
Docteur, Société Audros Technology, Examinateur
Monsieur Alaeddine ZOUARI
Docteur, Maître assistant, ISGI Sfax, Examinateur

Remerciements

Il me sera très difficile de remercier tout le monde car c'est grâce à l'aide de nombreuses personnes que j'ai pu mener cette thèse à son terme.

En premier lieu, je tiens à remercier Monsieur Michel Tollenaere, Professeur à l'institut national polytechnique de Grenoble, qui fût pour moi un directeur de thèse attentif et disponible. Merci pour ta patience et tes encouragements qui m'ont permis de conclure un travail commencé il y a longtemps. Merci de ne pas avoir cessé de me motiver.

J'adresse de chaleureux remerciements à mes co-encadrants de thèse Madame Lilia Gzara de l'institut national polytechnique de Grenoble, Maître de conférence, HDR et Monsieur Philippe Pernelle de l'université Claude Bernard Lyon, Maître de conférence, pour leurs disponibilités, leurs attentions et leurs recommandations. Votre soutien et vos conseils m'ont été d'un immense réconfort au cours de mes moments difficiles. Un grand merci pour votre patience et votre écoute pendant toute la durée de ce projet.

Je souhaite exprimer ma gratitude à la société Audros Technology et tout particulièrement à Monsieur Kamal Cheballah, PDG de la société, qui m'a permis de réaliser cette thèse CIFRE. Je le remercie pour m'avoir attribué sa confiance et les moyens qu'il a mis en œuvre pour l'aboutissement de ce projet. Je remercie aussi l'ensemble des employés de la société Audros Technlogy de m'avoir aidé et facilité mon intégration dans le monde du travail. Merci pour ces 3 années que nous avons passés ensemble.

J'exprime tous mes remerciements à l'ensemble des membres de mon jury qui ont accepté d'évaluer ce travail de thèse. Merci à Monsieur Dominique Deneux, Professeur à l'université de Valenciennes, d'avoir accepté de présider le jury de cette thèse. Merci à Monsieur Alain Bernard, Professeur à l'école centrale de Nantes, et à Monsieur Samuel Gomes, Professeur à l'université de technologie de Belfort-Montbéliard, d'avoir accepté d'être les rapporteurs de ce manuscrit. Je remercie enfin Monsieur Alaeddine Zouari, Maître assistant à l'institution supérieur de gestion industrielle de Sfax, pour avoir accepté d'examiner cette thèse.

Je remercie toutes les personnes avec qui j'ai partagé ces 3 années de travail, mes collègues du laboratoire G-SCOP et tous ceux qui, de près ou de loin, ont contribué à la réalisation de ce projet.

Mes remerciements s'adressent enfin à ma famille qui m'a toujours épaulé dans ce projet, particulièrement à ma femme, qui m'a beaucoup soutenu dans la dernière phase de cette thèse.

A mon père,
pour son soutien depuis le début dans mes études.

Sommaire

Table des figures

Liste des tableaux

Glossaire

ACMS : Atelier de Conception et Modélisation de Système d'information
AP : Application Protocol
ATL : Atlas Transformation Language
AUPL : AUdros Programming Language
BD : Base de Données
CAD : Computer-Aided Design
CAO : Conception Assistée par Ordinateur
CIFRE : Convention Industrielle de Formation par la REcherche
CIM : Computation Independant Model
CRM : Customer Relationship Management
EMF : Eclipse Modeling Framework
ERP : Enterprise Resource Planning
FAO : Fabrication Assistée par Ordinateur
GED : Gestion Electronique des Documents
GEDT : Gestion Electronique des Documents Techniques
GMF : Graphical Modeling Framework
IDM : Ingénierie dirigée par les modèles
MES : Manufacturing Execution System
M2M : Model to Model
M2T : Model to Text
MDA : Model Driven Architecture
MDE : Model Driven Engineering
MOF : MetaObject Facility
MRP : Material Requirements planning
MVC : Modèle-Vue-Contrôleur
OCL : Object Constraint Language
OMG : Object Management Group
PDM : Product Data Management (contexte PLM)
PDM : Platform Definition Model (contexte MDA)
PIM : Platform Independant Model
PME : Petite et Moyenne Entreprise
PMI : Petite et Moyenne Industrie
PLM : Product Lifecycle Management
PSM : Platform Specific Model
RCP : Rich Client Platform

SCM : Supply Chain Management
SE : Standard Édition
SGDT : Système de Gestion de Données Techniques
SI : Système d'information
SLM : Simulation Lifecycle Management
STEP : STandard for the Exchange of Product model data
SQL : Structured Query Language
SysML : Systems Modeling Language
UML : Unified Modeling Language
XMI : XML Metadata Interchange
XML : eXtensible Markup Language

Introduction générale

Contexte des travaux

Lors des 30 glorieuses qui ont marqué l'après-guerre, l'industrie a permis une croissance économique qui a engendré une part de la prospérité de la société ; le premier et le second choc pétrolier ont été l'entame d'une lente érosion de l'emploi dans le secteur industriel. Cette dernière période s'est caractérisée par trois phénomènes marquants qui ont impacté l'évolution de l'industrie :

- un développement sans précédent des échanges commerciaux au niveau mondial et l'apparition d'acteurs industriels majeurs en Asie,
- l'émergence d'une offre très diversifiée permettant aux clients de choisir le produit adapté à son juste besoin,
- enfin la diminution drastique des coûts des solutions informatiques (matérielles mais aussi logicielles) permettant de dématérialiser les échanges d'informations.

L'usage des outils de calcul électronique s'est d'abord orienté sur le calcul scientifique (langage FORTRAN) et les situations et applicatifs de gestion (COBOL), avant de s'introduire dans les bureaux d'études pour la modélisation géométrique et les représentations graphiques (CAO), 2D et 3D. La profusion des informations techniques à gérer dans un projet a conduit à la prise en compte des processus de développement, en particulier l'ingénierie collaborative (concurrent engineering), dans la structuration des informations supportées par les outils [Sohlenius, 1992]. L'intégration des connaissances techniques mise à disposition du concepteur est devenue un concept clé de la recherche sur les processus d'ingénierie : cette intégration, en raccourcissant les cycles de développement, vise à éviter les remises en cause tardives de décisions précoces [Tichkiewitch, 1994].

Après les trois premières révolutions industrielles, en Europe puis aux Etats-Unis, enfin au Japon pour l'automatisation, nous sommes à l'aube d'une quatrième révolution désignée par certains auteurs comme l'Entreprise 4.0 [Siemens, 2015].

La standardisation du système d'information (SI) des entreprises industrielles est un phénomène qui n'est pas récent mais qui s'est accentué avec le déploiement de systèmes comme l'ERP (Enterprise Resource Planning), le PLM (Product Lifecycle Management), la SCM (Supply Chain Management), ou le MES (Manufacturing Execution System). En effet, pour la grande majorité des entreprises actuelles, le développement d'un SI ad-hoc et global, avec la rigidité qui l'accompagnerait nécessairement, n'a plus de sens. Elles ne sont plus prêtes à investir, comme par le passé, des sommes très importantes dans la conception

d'un système d'information spécifique qui s'est souvent avéré trop coûteux à réaliser et surtout à maintenir [Paviot, 2010]. Force est de reconnaître que le principal intérêt d'une standardisation du système d'information est bien entendu directement lié à son coût de développement mutualisé entre les clients de l'éditeur et à sa pérennisation. Pour autant, la standardisation du SI par des logiciels métiers configurables comme l'ERP ou le PLM, est directement confrontée aux limitations de la configuration de chacun de ces outils. La démarche de standardisation, dont l'objectif est la maitrise des coûts, est souvent antagoniste avec les besoins spécifiques de l'entreprise, dans l'écosystème qu'elle forme avec ses partenaires. De nos jours, les entreprises industrielles se doivent donc de trouver le juste compromis entre la standardisation des fonctionnalités de leur système d'information et les développements spécifiques destinés à les adapter au fonctionnement interne de l'entreprise, comme aux spécificités des partenaires de la supply chain [CIMdata, 2005].

Le contexte général de ce travail de thèse se situe dans le domaine de la construction de modèles customisés et évolutifs pour les systèmes d'information des entreprises industrielles. Dans nos travaux, nous nous focalisons sur les systèmes d'information de type PLM [Softech, 2015] [CIMdata, 2015] et plus particulièrement d'un point de vue éditeur/intégrateur de ces systèmes. En effet, dans le cadre de leur démarche d'innovation, les éditeurs de solutions PLM sont confrontés à l'extension des capacités fonctionnelles de leur solution logicielle tout en assurant la cohérence globale des implémentations chez leurs clients. Les PLM sont suffisamment génériques pour proposer des modèles standards adaptés aux besoins des entreprises en ajoutant seulement quelques développements supplémentaires. Cependant, dans le cadre d'un nouveau projet PLM dans une entreprise, la mise en place d'un système conforme à l'activité de l'entreprise nécessite souvent la création d'un ou plusieurs modèles de données particuliers, ainsi qu'un ensemble de fonctionnalités (traitements) spécifiques. Il est assez courant que les entreprises souhaitent conserver leurs méthodes de travail pour offrir le meilleur service à leurs clients et fournisseurs, bref mettre en place des "best practices". La question qui est parfois posée alors est de savoir si c'est l'entreprise qui s'adapte à l'outil ou l'inverse. La question n'est pourtant pas aussi simple, car plus qu'un choix tranché, les réponses sont souvent des compromis. D'abord parce que les capacités d'adaptation sont intrinsèquement infinies avec les systèmes PLM. De plus, les choix posés à un instant donné, doivent pouvoir évoluer dans le temps.

Dans nos travaux, nous souhaitons apporter des éléments de réponse à la problématique de construction et de gestion des évolutions des modèles métier au sein des systèmes d'information qui sont déployés dans les entreprises industrielles. Nous définissons ici par "modèle métier" les éléments manipulés et caractérisés par l'activité et le métier de l'entreprise industrielle. Nous utiliserons à nouveau ce terme dans les chapitres suivants, il correspond aux concepts métier d'une entreprise. Cette thèse a été réalisée dans le cadre d'une convention CIFRE avec un éditeur de PLM (Audros Technology). Nos travaux s'inscrivent dans une démarche de recherches où la problématique et la solution proposée sont définies en fonction des usages et pratiques industriels des déploiements de ce type de système.

Positionnement de la problématique

Le processus de déploiement d'un système PLM comme Audros dans une entreprise est réalisé en plusieurs étapes et fait intervenir plusieurs acteurs. Les différentes étapes de ce type de projet sont définies dans la figure ci-dessous :

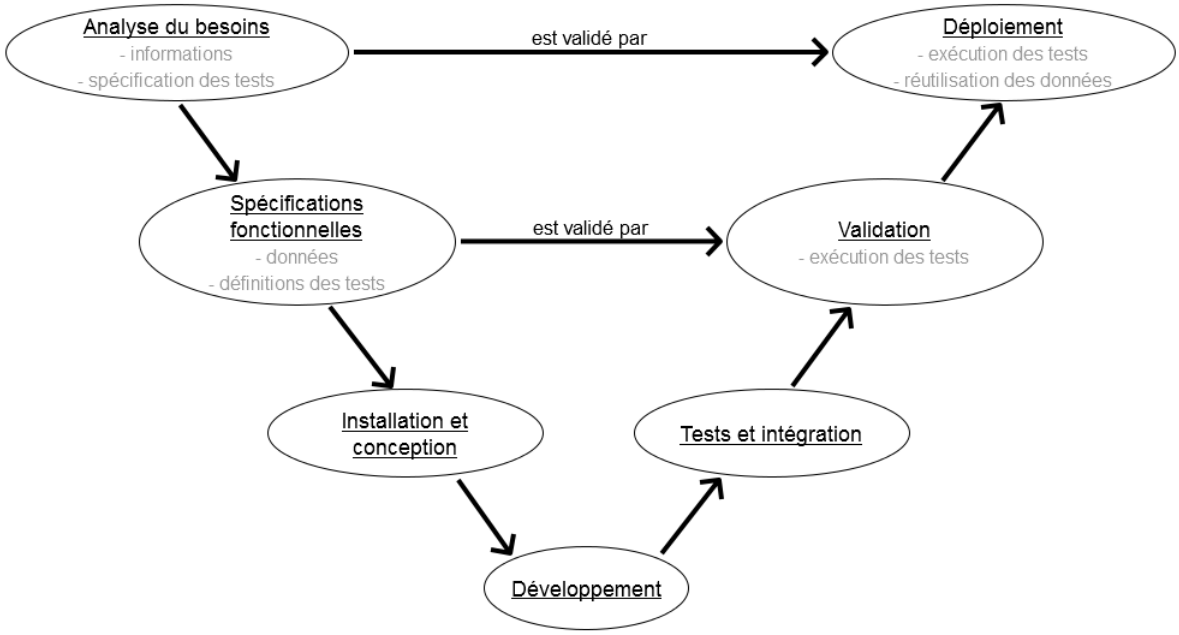

FIGURE 1 – Cycle de développement en V

Parmi les acteurs intervenant dans le cycle de déploiement, nous identifions l'éditeur de la solution PLM, l'intégrateur de cette solution (dans certains cas l'intégrateur et l'éditeur peuvent appartenir à la même entité organisationnelle) et l'entreprise qui représente le client final. Chacun de ces intervenants a un rôle à tenir dans le cycle de déploiement du PLM. En effet, les "stakeholders" du projet sont contraints d'interagir les uns avec les autres afin de mener le projet à son terme dans les délais définis. L'interaction et les rôles des différents acteurs sont illustrés par un schéma (fig. 2). Cependant, ce schéma n'est pas générique, il peut y avoir des différences d'un projet à un autre pour les entreprises clientes. En effet, le rôle de l'intégrateur peut être réalisé par l'éditeur de la solution ou inversement.

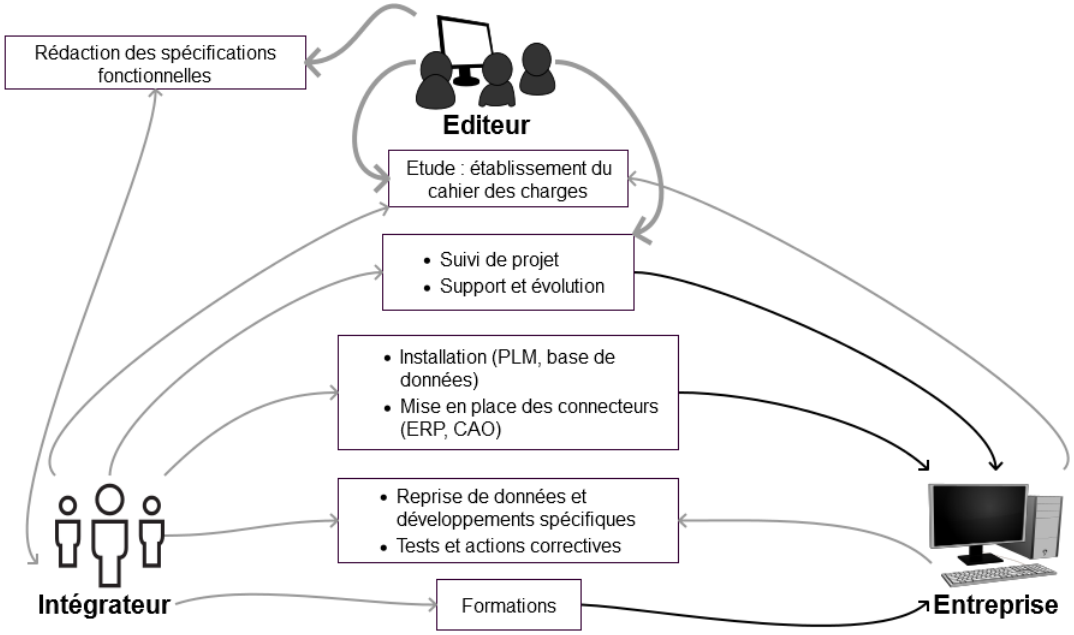

FIGURE 2 – Rôles et interactions des acteurs dans le cycle en V

La définition et la configuration d'une solution logicielle adaptée aux besoins de l'entreprise cliente peuvent difficilement s'envisager en une seule phase tant l'impact organisationnel des outils PLM est grand. L'évolution de l'organisation qui accompagne le déploiement des solutions logicielles est un processus qui prend généralement du temps par la mise en place et l'adoption des best practices. Dans ce processus conjoint d'évolution des pratiques de l'organisation et du déploiement logiciel, trois acteurs sont parties prenantes : l'intégrateur en charge de la convergence des bonnes pratiques et de l'outil logiciel, l'éditeur de l'outil logiciel et l'entreprise cliente. Chaque phase d'évolution des outils fait l'objet d'un certain nombre de documents classiques en génie logiciel, du cahier des charges aux spécifications, en passant par la spécification et les résultats obtenus lors des tests. Afin de proposer une solution adéquate aux besoins de l'entreprise cliente, les acteurs du projet définissent ensemble le cahier des charges pendant la phase d'étude du projet. Ensuite, le début du processus de déploiement du système PLM est réalisé par l'intégrateur après avoir pris connaissance des spécifications fonctionnelles rédigées par l'éditeur. Cette phase initiale comprend l'installation du système PLM et de sa base de données, le paramétrage de la solution PLM (mise en place du modèle de données, workflows, définition des droits utilisateurs...) et la mise en place des différents connecteurs entre le système PLM et les autres outils internes à l'entreprise (ERP, CAO (Conception Assistée par Ordinateur)...) amenés à interagir avec le PLM. L'intégrateur de la solution PLM se charge de réaliser la reprise de données et les développements spécifiques avec la participation de l'entreprise cliente. Ensuite la phase de test permet au client et à l'intégrateur de valider le fonctionnement de l'outil avant sa mise en production. La dernière étape d'un processus de déploiement consiste à apporter une activité de support et d'évolution du système PLM déployé. Le processus de déploiement d'un système PLM peut être à l'origine d'incohérences au sein du modèle de données de l'entreprise cliente aussi bien lors

d'un déploiement initial que lors de d'une évolution du système. Ces problèmes de cohérence sont illustrés par des exemples précis qui seront présentés au chapitre (cf chapitre 1).

Ces quelques exemples illustrent les problématiques liées à la création des modèles métier au sein des systèmes PLM et à la gestion de leur cohérence au cours du temps. La principale problématique qui se pose pour ces entreprises clientes, est double :

- d'une part il s'agit de les aider à construire un modèle métier adapté à ses besoins et à ses spécificités tout en assurant sa compatibilité avec les outils métiers et son instanciation dans le système PLM.
- d'autre part, il s'agit de disposer de moyens pour gérer les adaptations apportées au modèle standard proposé par l'éditeur tout en conservant la cohérence globale du système.

Ainsi, la nécessaire reconfiguration (évolution de l'activité de l'entreprise, changement des méthodes de travail...) du système est contrainte par deux types de cohérence :

- entre l'application PLM déployée chez un client et le système PLM standard (défini par l'éditeur).
- entre les éléments de l'application PLM déployée chez le client (modèle produit - modèle organisationnel - modèle processus - scripts).

Démarche proposée basée sur l'ingénierie des modèles

La motivation principale de ce travail de recherche est de permettre aux différents clients de solutions PLM, de faire évoluer leurs modèles "métier" **sans l'intervention d'un éditeur/intégrateur**. Cela peut être possible seulement si ces clients disposent d'un outil suffisamment complet qui leur permette de connaître les impacts des différentes modifications effectuées au sein de leur modèle "métier" (données, processus, droits d'accès, scripts). Cette notion d'impact est la partie centrale de ce travail de thèse, chaque modification d'élément du modèle, que ce soit un ajout, une mise à jour ou une suppression, doit impérativement engendrer une étape de validation avant sa mise en production. Un modèle sera considéré comme non valide s'il ne respecte pas l'ensemble des contraintes mises en place et sera donc synonyme d'incohérence future dans le cas d'une mise en production sans correction.

Dans notre travail de thèse, nous avons opté pour une démarche globale (mixte) où le modèle métier qui est implémenté dans le PLM est placé au coeur de l'entreprise. Ce modèle doit être construit selon une approche mixte, c'est-à-dire par une approche descendante ("top-down") et une approche ascendante ("bottom-up") :

- avec une approche descendante lors de la phase de déploiement d'un projet PLM où il convient de construire un modèle adapté aux besoins mais conforme aux principes du PLM.
- avec une approche ascendante lors de la phase d'usage du système PLM où il est souvent nécessaire de faire évoluer incrémentalement les choix initiaux face aux "adaptations naturelles" de l'entreprise à son contexte (technique, économique...).

Pour répondre à ces objectifs, nous avons fait le choix d'adopter une approche globale basée sur les principes de l'IDM (Ingénierie Dirigée par les Modèles) et plus particulièrement MDA (Model Driven Architecture) [OMG, 2001] [Bezivin and Gerbe, 2001] [Bezivin, 2004] [Blanc, 2005] [Favre et al., 2006]. L'architecture MDA est très proche des besoins de modélisation au sein des systèmes PLM. En effet, cette architecture impose une approche par niveau (niveau métier, niveau d'exécution...). De ce fait, elle correspond à nos besoins en termes de résolution à la problématique présentée plus haut. L'approche MDA peut être utilisée lors de différentes étapes d'un projet :

- lors du déploiement initial d'un système PLM dans un contexte métier spécifique : dans ce cas, il s'agit de modéliser les objets gérés dans le PLM avec les concepts métier du secteur d'activité de l'entreprise.
- lors des évolutions nécessaires pour s'adapter au contexte économique ou métier : dans ce cas, il s'agit de permettre la reconfiguration des modèles tout en garantissant la cohérence structurelle et comportementale de l'existant.

Afin de comprendre l'intérêt d'une approche d'ingénierie dirigée par les modèles pour notre problématique, nous étudierons dans le chapitre (cf chapitre 2) les différents travaux réalisés autour de ce type d'approche. Avant cela, nous présenterons les systèmes PLM de manière plus détaillée.

Plan de lecture de la thèse

Le premier chapitre présente d'abord les systèmes PLM et son positionnement par rapport aux systèmes d'information des entreprises puis il décrit le PLM Audros édité par la société Audros Technology. Il aborde ensuite les différentes typologies de modèles au sein des systèmes PLM et détaille le déploiement de ce type de système. Pour finir, ce chapitre identifie les problèmes industriels rencontrés par les entreprises utilisant un système PLM. L'ensemble de ces éléments permet d'établir la problématique scientifique de nos travaux. Le deuxième chapitre présente d'abord les travaux de recherches réalisés autour des modèles dans les systèmes PLM. Ensuite, il présente la démarche choisie pour répondre à la problématique proposée dans le premier chapitre, l'ingénierie dirigée par les modèles (IDM). L'ensemble des concepts d'ingénierie dirigée par les modèles sont décrits, ainsi que l'architecture dirigée par les modèles. Enfin nous présentons quelques travaux de recherches ayant mobilisé l'IDM.

La suite de de notre travail présente notre proposition basée sur MDA. Elle nous permet de proposer une résolution à la problématique de la thèse et de la mettre en pratique avec la réalisation d'un prototype basé sur le Framework Eclipse. Le chapitre trois introduit les concepts des différents niveaux d'une approche basée sur MDA et décrit le niveau d'exécution de notre démarche MDA par une analyse du système d'information du PLM Audros. Le chapitre 4 présente dans un premier temps une typologie de contraintes pouvant être affectées sur les modèles dans différents niveaux d'abstraction. La notion de validation de modèles est décrite par l'intermédiaire d'un cas d'utilisation. Dans un second temps, nous décrivons plus précisément les diverses étapes de transformation au sein de

notre approche MDA pour les systèmes PLM. Nous terminons le quatrième chapitre par les concepts de transformation inverse et de comparaison de modèles. Pour finir, le cinquième chapitre est consacré à la présentation du prototype qui illustre notre proposition. Nous présentons d'abord l'architecture de notre atelier de conception et de modélisation de systèmes d'information. Par la suite, nous illustrons le fonctionnement du prototype avec un exemple de création de modèle d'une entreprise dans le domaine de la plasturgie.

Chapitre 1

Positionnement des travaux et problématique

Sommaire

1.1 Introduction

Dans ce chapitre, nous précisons le contexte de nos travaux ainsi que la problématique scientifique qui sera traitée par la suite. Comme nous l'avons indiqué dans l'introduction générale, nos travaux concernent le déploiement et l'évolution des systèmes PLM au sein des entreprises industrielles. Ainsi donc, dans la première section (cf sec. 1.2), nous rappelons les principaux concepts des systèmes PLM. Puis, dans la section suivante, nous présentons la solution PLM de l'éditeur Audros (cf sec. 1.3) pour ensuite illustrer les problèmes rencontrés lors du déploiement et de l'évolution de la solution PLM à l'aide

d'exemples tirés des clients d'Audros (cf sec. 1.4). Enfin, nous déclinons, dans la dernière section (cf sec. 1.5), la problématique scientifique de cette thèse.

1.2 Les systèmes PLM

1.2.1 Origine et définition

De nos jours, les systèmes PLM sont largement déployés dans de nombreux secteurs de l'industrie. Ils constituent un des piliers du SI des entreprises industrielles principalement dans les grands groupes ou les entreprises de taille moyenne qui développent et maintiennent une offre produit. Ils sont présents dans de très nombreux secteurs industriels (automobile, aéronautique, ferroviaire, horlogerie, médical, naval, plasturgie...) [CIMdata, 2003] [AberdeenGroup, 2006] ou même de service (énergie, ingénierie, logiciels...). Initialement, les premiers systèmes sont apparus à la suite de l'introduction de la conception assistée par ordinateur (CAO)[1] dans les années 1980. En effet, le besoin de gérer des données issues du Bureau d'Etudes (BE) a rendu obsolète la traditionnelle "armoire à plans" qui renfermait les dernières versions des documents d'études. Ces outils génèrent des données complexes en termes de structuration. Chaque outil possède sa structure sémantique de description des éléments géométriques. En revanche, dans tous les cas, la structure d'un produit se décompose sous la forme d'un ensemble de fichiers et de liens entre les fichiers. Finalement, le besoin de gestion de ces données techniques complexes a permis l'émergence dans les années 80 de démarche d'intégration aux bases de données [Cheballah, 1992] [Randoing, 1995] en fédérant les données techniques des produits autour d'un système centralisé (de type base de données) [Maurino, 1993].

Le développement de ce type de système a été progressif et reste encore en constante évolution. Selon Debaecker [Debaecker, 2004], la genèse des systèmes PLM remonte aux 3 dernières décennies (fig. 1.1). On peut distinguer 3 grandes phases d'évolution :

1. Dans les années 1980, le besoin de gérer des données techniques issues du bureau d'étude a permis la création des premiers SGDT (Système de Gestion de Données Techniques) ou PDM (Product Data Management). Cependant, l'échange des données avec les premières versions de ces outils est limité aux acteurs du bureau d'études de l'entreprise, l'exploitation des données est limitée à la production.
2. Dans les années 90, avec l'évolution des SGDT (grâce à l'avènement de l'Internet) l'ensemble des services de l'entreprise (achats, bureau d'études, fabrication, logistique, marketing, ventes...) peuvent collaborer à travers l'outil. Cette amélioration des processus transversaux a permis d'augmenter la productivité de l'entreprise.
3. Les années 2000 marquent un tournant dans le développement des SGDT. En effet, les données techniques sont maintenant échangées entre l'entreprise et ses collaborateurs (fournisseur, partenaire, sous-traitant...). C'est grâce à ce partage d'informations internes et externes à l'entreprise que le SGDT devient le PLM (Product

1. Les applications de type CAO permettent de modéliser la géométrie ou la description spatiale d'un produit en cours de conception [Fenves et al., 2005]

Lifecycle Management). La gestion des données techniques est partagée pendant tout le cycle de vie des produits (de l'idée à sa fin de vie). L'apparition du terme PLM n'a pas été soudaine, elle est le fruit d'une évolution constante des méthodes de travail dans l'industrie et d'un besoin constant de progression [Lamouri, 2006].

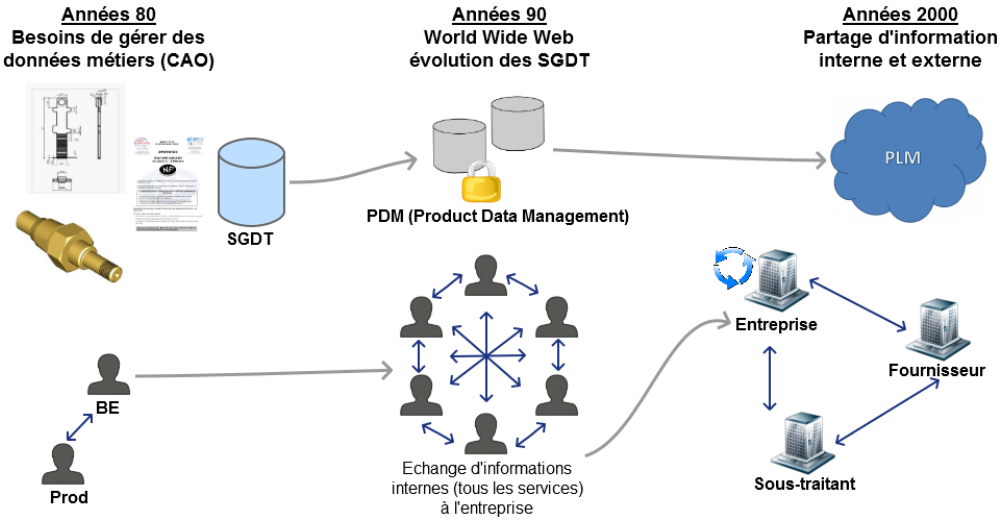

FIGURE 1.1 – Les 3 grandes phases de l'origine des systèmes PLM

La définition du PLM est sujet à des appréciations différentes suivant le point vue des auteurs. Comme le précisent [Abramovici, 2007] [Liu et al., 2009], il est difficile de trouver une définition unique admise par toute la communauté. Toutefois, nous pouvons distinguer deux périmètres dans les définitions proposées.

Le premier périmètre concerne la gestion des informations produit. Les auteurs mettent en avant les principaux piliers constitutifs d'un système PLM, à savoir le produit, les processus et l'organisation [Pernelle, 2002] [Sudarsan et al., 2005] [Saaksvuori and Immonen, 2008]. Le système PLM possède ainsi les caractéristiques d'un système d'information, à savoir un ensemble organisé de ressources (matériels, personnes, logiciels et procédures) permettant d'acquérir, de traiter, de stocker, de diffuser des informations (sous forme de données, textes, images, sons, etc.) dans et entre les organisations. Cette gestion des informations produit au sein de l'entreprise étendue est réalisée autour de trois concepts fondamentaux [Dutta and Wolowicz, 2005] :

1. Assurer un accès et une utilisation sécurisés des informations produit.
2. Maintenir l'intégrité des informations produit tout au long de son cycle de vie.
3. Gérer et maintenir les processus métier afin de créer, disséminer, partager et utiliser les informations produit.

Le deuxième périmètre est plus large, le PLM est alors défini comme une approche stratégique de l'entreprise pour la gestion et l'utilisation efficace du capital intellectuel de l'entreprise. Les auteurs mettent en avant la définition d'une stratégie produit (identification à long terme des produits, clients, nouvelles technologies, partenariats...) puis soulignent l'importance d'une mise à plat des processus métier (pour être conforme à la

vision stratégique) et d'une gestion des informations produit adaptée [CIMdata, 2003] [Stark, 2005] [Marin, 2009]

Dans nos travaux de thèse, nous abordons le PLM avec une vision système d'information. L'ensemble des définitions précédemment référencées nous permet d'identifier les concepts principaux autour du système PLM : le produit, le cycle de vie et l'acteur. Le rôle d'un système PLM est alors double : gérer l'ensemble des informations du cycle de vie des produits industriels d'une entreprise et fournir aux différents acteurs de celle-ci les informations des produits à différents stades de leurs cycles de vie selon leurs profils.

Dans le paragraphe qui suit, nous positionnons le système d'information PLM dans le système d'information global de l'entreprise.

1.2.2 Positionnement du PLM dans le SI

Rappelons que le Système d'Information au sens systémique de Le Moigne [Le Moigne, 1977] [Le Moigne, 1990] est un des systèmes de l'entreprise qui participe à son fonctionnement. Cette vision de l'entreprise du type "boite noire" et "flux" reste pertinente pour analyser les grands principes de fonctionnement d'une entreprise. Elle a donné lieu à de nombreuses définitions [Rolland et al., 1988] [Cauvet and Rosenthal-Sabroux, 2001]. Nous retenons en particulier la définition proposée par Reix [Reix, 2002] qui assimile le système d'information à un objet tridimensionnel avec :

1. une dimension informationnelle, identifiée par la notion de représentation [2], la notion d'information [3] et la création de données [4].
2. une dimension technologique, qui permet d'effectuer les processus de collecte, de mémorisation et de traitement de l'information.
3. une dimension organisationnelle, qui consiste en un ensemble de processus (un ensemble d'activités ou d'opérations fonctionnellement liées par la production d'un résultat identifiable)

La dimension technologique proposée par Reix est particulièrement intéressante. En effet, l'évolution technologique de ces vingt dernières années n'a pas eu l'effet d'homogénéité escompté. D'ailleurs, elle a même probablement eu l'effet inverse. Le développement informatique des bases de données et du génie logiciel a permis de pérenniser des outils spécifiques qui se sont imposés comme des standards de fait, prenant le pas sur les tentatives de normalisation. Force est de constater que pour les systèmes d'information des entreprises industrielles, l'évolution technologique portée par les éditeurs s'est fortement stabilisée autour de quelques sous-systèmes dont le PLM. Cette vision du PLM comme un composant ou un sous-système du SI nous parait pertinente. D'ailleurs, Le Duigou [Le Duigou, 2010] présente le PLM comme un système qui aide à transmettre les informations de définition des produits à d'autres solutions du SI. De même que Marin

2. Une représentation est une image du monde réel passé, présent et futur : cette image est composée de signaux pouvant être perçus par nos sens

3. un ensemble de données qui permettent de changer la représentation que l'on a de l'environnement

4. l'échange de l'information dans l'entreprise s'effectue avec des données numériques ou alphabétiques, décrivant des objets ou des évènements

[Marin, 2009], qui considère que le PLM *"constitue une structure ouverte à la connectivité et au partage d'informations du produit avec d'autres applications de l'entreprise"*, ou encore Terzi [Terzi, 2005] qui définit le PLM comme : *"un morceau de technologie qui peut interagir avec d'autres solutions"*. Soulignons enfin que l'hétérogénéité du SI peut être transposée au PLM. De nombreux chercheurs considèrent en effet les PLM comme étant des systèmes hétérogènes composés d'un grand ensemble d'outils formant une chaîne pour le développement des produits [Pernelle, 2002] [Danesi et al., 2008].

Parmi les autres composants du SI de l'entreprise, nous trouvons notamment :

L'ERP (Enterprise Ressource Planning) : cette famille de solutions, introduite il y a une vingtaine d'années, tient son origine du MRP (Material Requirements planning) utilisée dans la manipulation de la nomenclature de production. Une solution ERP est un logiciel intégré qui permet de gérer les principaux processus opérationnels d'une entreprise (achats, gestion comptable, gestion des stocks, gestion des ressources humaines, distribution...). Le but d'un ERP est de coordonner l'ensemble des activités de l'entreprise aussi bien de façon verticale (production, approvisionnement) que de façon horizontale (ressources humaines, marketing...).

Le MES (Manufacturing execution system) : cette famille de solutions est considérée comme le "Système d'information de l'Atelier". Une solution MES fournit les informations nécessaires pour les activités de production et ce en temps réel, de la création de l'ordre de fabrication jusqu'à l'obtention du produit fini temps réel. Les fonctions du MES sont normalisées par la nome ISA 95.

Le SCM (Supply Chain Management) : le SCM est lié à la chaîne d'approvisionnement complète de l'entreprise, c'est-à-dire à sa logistique (gestion des flux de matières et de matériaux, en entrée comme en sortie, des stocks et du transport des produits). D'ailleurs, Baglin décrit la SCM comme étant *"l'ensemble de la chaîne de flux et de services qui permet à une entreprise de répondre à la demande de ses clients en fournissant le produit ou le service voulu dans les délais prescrits"* [Baglin et al., 2005]. La finalité d'une solution SCM est l'optimisation des outils et des méthodes d'approvisionnement de l'entreprise afin de diminuer les coûts grâce à une réduction des délais de livraison et des stocks (organisation en flux tendu).

Le CRM (Customer Relationship Management) : La gestion de la relation client est l'ensemble des outils et techniques destinés à capter, traiter, analyser les informations relatives aux clients et aux prospects, dans le but de les fidéliser en leur offrant le meilleur service. La connaissance de chaque client à titre individuel est indispensable pour développer avec lui une relation profitable et durable en lui proposant des offres pertinentes. L'historique de ses achats, ses moyens de communication préférés, ses modes de paiement préférés, ses intérêts et ses services préférés (etc) sont autant d'informations strictement nécessaires au développement de relations à long terme. Les informations précédemment citées sont généralement dispersées dans les systèmes d'information et les entreprises généralement les regroupent dans une base de données client.

Le SLM (Simulation Lifecycle Management) : Plus récemment certains auteurs et éditeurs ont introduit le concept de SLM afin de gérer l'ensemble des simulations

numériques effectuées lors du développement d'une gamme de produits. L'objectif est de permettre de lier explicitement ces simulations à la formulation des exigences et à leur validation. En termes de fonctionnalités, le SLM permet de documenter les exigences, documenter tout ce qui a été fait pour les vérifier, documenter l'environnement (logiciels, matériels de test...) utilisé pour le faire, enfin garder une trace des modifications des exigences et des documents. Adressant cette problématique particulière des simulations numériques, de nombreux auteurs rattachent le SLM au PLM [De Fontaine, 2014].

L'ensemble de ces solutions (PLM, ERP, SCM, CRM) constituent l'ossature du SI de l'entreprise, avec des poids différents selon l'activité et le secteur d'activité de l'entreprise. La figure suivante (fig. 1.2) positionne le PLM vis à vis des autres composants du Système d'information et permet de mettre en évidence la difficulté d'un cloisonnement strict du périmètre fonctionnel des systèmes PLM.

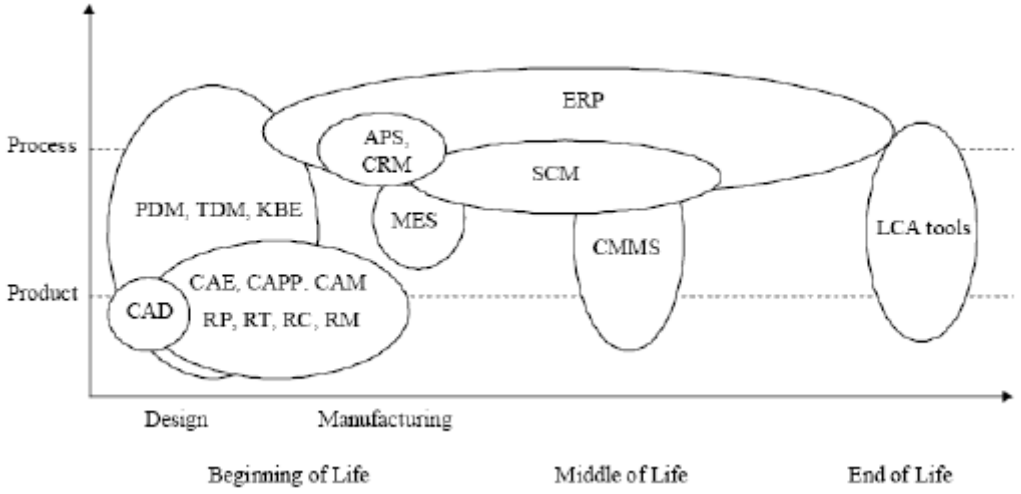

FIGURE 1.2 – Positionnement des systèmes PLM d'après [Terzi et al., 2008]

1.2.3 Fonctionnalités des systèmes PLM

Même s'il n'existe pas de convergence entre les définitions proposées dans la littérature et les systèmes réellement développés par les éditeurs, nous pouvons isoler les grandes fonctionnalités d'un système PLM [Debaecker, 2004] [Liu et al., 2009] [Le Duigou, 2010] [Abid et al., 2014] à partir des principes énoncés précédemment :

- Le coffre-fort : il sert d'espace de stockage sécurisé pour l'ensemble des données liées aux produits, en s'appuyant sur un système centralisé, sur une gestion des méta-données et sur un calcul temps réel des droits d'accès.
- La visualisation permet des accès et des aperçus rapides de documents aux formats divers, y compris des données issues de la CAO.
- Les vues utilisateur : les données sont présentées à un utilisateur de la façon dont sont conçues les vues et selon ses droits. Par exemple, un utilisateur du BE peut avoir une vue différente d'un utilisateur d'un autre service pour le même objet.

Les vues concernent également l'affichage des nomenclatures produits ou des cas d'emploi.
- La gestion des modifications contrôle la modification des données par des mécanismes simples de réservation (check-in/check out), les études d'impact [Rivière, 2004] ou des processus complexes, tels que ceux préconisés dans le modèle de maturité CMII [Silventoinen et al., 2009].
- La gestion de traçabilité, permet de garder les traces et les instances des données dans le temps.
- La gestion des structures produit permet de gérer des nomenclatures selon divers points de vue (eBOM : engineering Bill Of Material, mBOM : manufacturing Bill Of Material) et des objets métier en relation les uns avec les autres [Izadpanah, 2011].
- La gestion des cycles de vie contrôle l'espace d'état de l'ensemble des données et les processus qui y sont associés.

1.2.4 Les modèles au sein des PLM

La question de la modélisation au sein des systèmes PLM est centrale pour nos travaux. En effet, ces modèles permettent de structurer l'ensemble des informations gérées par le système. Et, comme nous le verrons par la suite, le déploiement d'un PLM en entreprise et la gestion de son évolution ont une incidence directe sur les modèles. De ce fait, la caractérisation des modèles implantés effectivement dans le système est un des résultats finaux des processus de déploiement et d'évolution. Pour autant, la vision d'un modèle global et générique porté par un système est peu réaliste au vu de l'ensemble des usages attendus d'un PLM. Les différents usages permettent de mettre en évidence divers points de vues des acteurs du PLM comme le montre Paviot [Paviot, 2010]. Ainsi, comme le propose de nombreux auteurs [Harani, 1997] [Thimm et al., 2006] [Noël and Roucoules, 2008] [Schuh et al., 2008], la question des modèles au sein des PLM doit être abordée avec une approche multi-modèles orientée autour de trois principaux domaines : le Produit, le Processus et l'Organisation. Dans la suite de cette section, nous présentons chacun de ces modèles.

Le modèle Produit

Dans le contexte PLM, le terme Produit est à considérer au sens général du terme : le produit réalisé qui est le résultat d'un travail humain (non naturel) sous la forme d'un bien ou d'un service destiné à un client [Nollet et al., 1994], mais aussi le produit imaginé (idée) par les équipes de conception. Le terme produit peut également avoir un sens différent selon son contexte d'utilisation [Zina et al., 2006]. Comme le précise [Paviot, 2010] dans ses travaux, *"le département conception verra le produit comme un système dont les fonctions doivent satisfaire le besoin du client, la production comme un système qu'il faut fabriquer et stocker, la maintenance comme un système dont il faut assurer le maintien en conditions opérationnelles, les achats comme un système dont il faut acheter certains composants à des fournisseurs etc"*. Dans le cas de produits complexes (système mécatronique), le produit est considéré selon plusieurs vues technologiques (électrique, mécanique, informatique, thermique...).

Le modèle Produit peut être défini comme étant une structure qui permet de collecter, d'organiser et de suivre l'évolution de l'ensemble des informations en relation avec le produit aux différentes étapes de son cycle de vie. La diversité des points de vue métier (conception, production, achats...) et technologiques (électrique, mécanique, informatique...) conduit à des modèles multi-vues ou multi-physiques. En effet, la structuration du produit est fortement liée au métier du modélisateur et à son activité. De ce fait, l'information à traiter est logiquement de différents types, formats et natures selon le métier.

Le modèle Processus

Les systèmes PLM permettent de structurer l'ensemble des données d'ingénierie et offrent désormais une couverture ou une interaction fonctionnelle importante des différents processus d'entreprise [Eynard, 2005]. Un processus peut être lancé pour diverses raisons, par exemple, pour une demande d'évolution d'un produit ou la validation des modifications. Comme pour le produit, le cycle de vie des produits s'exprime dans la gestion des processus. Il y a deux phases dans la gestion des processus :

- la modélisation du processus
- l'exécution du processus sous la forme de workflow

La modélisation (ou définition) des processus de l'entreprise permet d'*obtenir une abstraction d'un système hautement complexe afin d'en assurer un meilleur contrôle* [El Kadiri, 2009]. La modélisation permet d'élaborer une vision partagée des diverses actions sur les produits (demande de modification, demande de validation, création d'un nouveau projet...) afin d'automatiser la mise en oeuvre de ces actions. De plus, elle aide les différents acteurs à avoir une meilleure communication entre eux.

L'exécution du processus sous la forme d'un workflow permet d'*"automatiser un processus, de manière partielle ou totale, en faisant passer d'un participant à l'autre des documents, informations ou tâches pour une action spécifique, selon un ensemble de règles procédurales"* [WFMC, 1996]. Le workflow permet de faire circuler les lots de travaux relatifs à un processus donné d'un acteur ou un groupe d'acteurs vers un autre sous forme de "paquets" organisés selon une logique de découpage du processus qui correspond aux différents sous-objectifs du processus considéré. Cette vision du workflow est essentiellement procédurale : toutes les actions ainsi que leur enchaînement sont prédéfinis et toute information ne peut être transférée d'un acteur à l'autre que si elle a changé d'état. Ce workflow peut être organisé manuellement mais dans la pratique la plupart des workflow sont organisés dans un système de gestion de workflow (WorkFlow Management systems –WFMS). Le WFMS est défini comme étant *"Un système qui définit, crée et gère l'exécution de workflows à travers l'usage de logiciels, fonctionnant sur un ou plusieurs moteurs de workflow, capable d'interpréter la définition du processus, d'interagir avec les participants et, si nécessaire, d'invoquer l'utilisation d'autres outils et applications informatiques"* [WFMC, 1996]. Le moteur de workflow a pour objectif de transférer des documents entre les acteurs d'un processus en leur assignant des tâches à réaliser. Une tâche peut être une simple approbation (l'agent chargé de la tâche donne ou non son accord pour la poursuite du circuit), impliquer une action automatisée (envoi de mails...)

ou non (action matérielle, configuration...) ou l'attente d'un autre événement (date). La validation de l'étape est alors juste un simple acquittement signifiant que l'action a été effectuée. Ceci permet de faciliter la coordination entre les différents acteurs.

Le modèle Organisationnel

Les systèmes PLM permettent à tous les interlocuteurs qui interviennent tout au long du cycle de vie du produit de créer, visualiser ou modifier les données et documentations techniques, selon leur activité, avec une vue métier spécifique. L'une des fonctionnalités fondamentales des systèmes PLM réside dans la possibilité de filtrer les accès à des données confidentielles. Les droits d'accès évoluent dynamiquement en fonction de l'état d'avancement des projets et une gestion logicielle personnalisée des modifications et des changements de version garantit le respect des procédures courantes dans une entreprise [Bissay, 2010].

Le modèle organisationnel est un moyen de lier les différentes informations (ressources, infrastructure et pilotage) pour la mise en œuvre d'un projet produit. Il permet de définir la politique d'accès sur les classes d'objets (du modèle produit et du modèle processus) et ce par acteur, groupe d'acteurs ou rôle. Les objectifs de ce modèle sont :

- la définition des rôles et des groupes (équipes) pour les acteurs
- la traçabilité des actions des acteurs, rôles ou groupes
- la sécurisation des systèmes

1.2.5 Déploiement d'un PLM

Le déploiement d'un système PLM dans une entreprise industrielle est une opération assez complexe qui nécessite souvent plusieurs mois voire des années de travail. En effet, les modèles mis en place dans le système PLM se doivent de représenter le métier de l'entreprise. Avoir un modèle de représentation pertinent s'avère être compliqué lorsque l'entreprise a un métier très spécifique ne correspondant pas aux solutions standards proposées par les éditeurs. Cette situation nécessite souvent des développements spécifiques et/ou des adaptations dans la mise en œuvre du projet [Bissay, 2010]. Il n'existe pas de méthode standard pour le déploiement d'un système PLM ; le déroulement de ce type de projet dépend du secteur d'activité de l'entreprise et des fonctionnalités qu'elle souhaite implémenter. Malgré les différences notées lors des déploiements dans diverses entreprises, des grandes lignes communes peuvent être dégagées. La conduite d'un projet de déploiement de système PLM implique de maîtriser les points suivants :

- l'analyse des besoins,
- l'installation du logiciel,
- la mise en place des modèles métiers [5] et le paramétrage [6],
- l'incorporation des processus (workflows),
- l'intégration avec les autres composants du SI (ERP, CAO...),

5. un modèle métier est un modèle de données représentant le métier de l'entreprise (produits, processus et organisations mise en place)

6. sélection et personnalisation des éléments du modèle standard pour obtenir les modèles métiers souhaités

- la reprise de données,
- la conduite de changements et les formations.

Nous portons un intérêt particulier à la mise en place des modèles métiers qui représente une étape longue nécessitant une réflexion sur l'utilisation de modèles standards. Durant cette étape, un expert métier de l'entreprise fournit à l'expert PLM les éléments nécessaires à la création des modèles métiers. Cela nécessite de rassembler les informations produits, processus et organisationnelles représentant l'activité de l'entreprise. Ces informations sont transcrites de manière formelle sous différents types de données : des classes d'objets, des relations entre les classes, des étapes dans le cycle de vie de produits (statuts), des processus utilisés dans l'entreprise, la gestion des vues, la gestion des droits utilisateurs pour les données produits et les nomenclatures... Durant cette étape du processus de déploiement, il est important de partir des modèles standards proposés dans les solutions PLM, plutôt que de créer les modèles métiers à partir de zéro. Cette option est plus raisonnable pour des questions de temps (et donc de coûts) et d'anticipation sur les évolutions des modèles métiers. En effet, les éditeurs des solutions PLM basent leurs modèles standards sur leurs expériences successives (meilleures pratiques des clients et éléments récurrents). Ils proposent aujourd'hui des modèles assez complets couvrant la plupart des secteurs d'activité. Ces modèles standards peuvent évoluer au fil du temps pour s'adapter au contexte du marché. Les entreprises clientes ayant implémenté des solutions standards peuvent alors bénéficier au moindre coût des évolutions apportées par les éditeurs. Cependant, il est assez rare qu'un modèle standard d'une solution PLM corresponde parfaitement à l'activité (parfois assez spécifique) d'une entreprise cliente. Il est souvent nécessaire de procéder à des adaptations des modèles métiers standards. Ces adaptations représentent les éléments spécifiques de l'entreprise. Les modèles métiers obtenus sont alors un mixte entre les modèles standards et les adaptations spécifiques des entreprises.

1.3 Le PLM Audros

La suite PLM Audros est la solution éditée par la société Audros Technology [AudrosTechnology, 1993]. La société a été créée suite aux travaux de K. Cheballah en 1992 [Cheballah, 1992] et leur solution PLM est déployée aujourd'hui chez plus de 150 entreprises françaises et internationales (Belgique, Suisse, Irlande...) ; elle compte plus de 10 000 utilisateurs dans le monde. Depuis vingt ans, le PLM Audros est à destination des entreprises industrielles, principalement les PME/PMI. Ces PME/PMI sont issues de quelques grands secteurs d'activités industrielles dont l'énergie et l'électronique, l'automobile et le transport, l'aéronautique et la défense, le luxe... Comme tout système PLM, Audros est un outil de centralisation des informations et des données techniques des produits industriels. Il permet à l'ensemble des services d'une entreprise de disposer des informations relatives aux produits à divers stades de leurs cycles de vie.

Le PLM Audros est globalement structuré autour de deux configurations :
- Audros Standard Edition (SE)

– Enterprise Edition

Nous présentons ces deux éditions dans ce qui suit.

1.3.1 Standard édition

La standard édition est la version préconfigurée de la suite PLM Audros livrée avec un modèle de données dit "prêt à l'emploi". En effet, ce modèle de données comporte des classes d'objets, des relations, des statuts et des formulaires prédéfinis. Ce package comporte notamment :

– des automatismes : pour la codification, la génération de documents, des contrôles...
– des outils d'administration pour modifier le modèle dans la limite de la standard édition et des outils de maintenance.
– deux workflows prêts à l'utilisation : une demande d'évolution et une demande de validation.

Cette version de la SE couvre les cinq grands domaines d'application des bases fondatrices d'un système PLM (fig. 1.3) :

– Audros SE GEDT : Gestion électronique des données/documents techniques
– Audros SE PROCESS : Automatisation des processus
– Audros SE CAD : Gestion des données CAO
– Audros SE PROJECT : Gestion et suivi de Projets
– ERP Connect : Échanges avec les ERP

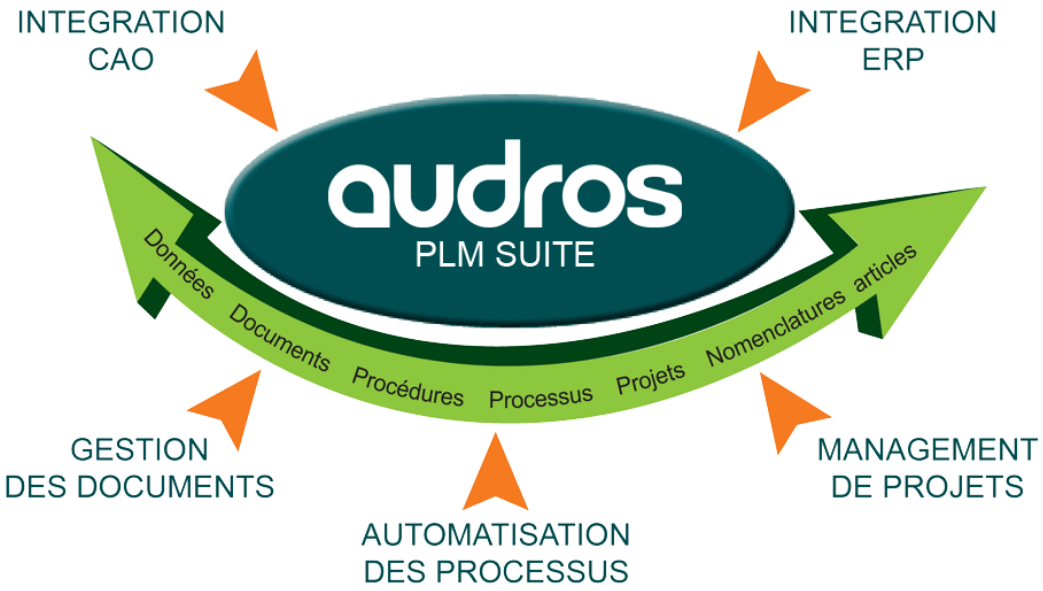

FIGURE 1.3 – Offre de la SE du PLM Audros

L'avantage de la standard édition de la suite Audros est d'être immédiatement opérationnelle. Elle se démarque par sa simplicité et sa facilité d'utilisation. Seule la (délicate) reprise de données de l'existant et quelques configurations sont nécessaires après l'installation avant de débuter avec le PLM. Les fonctions métiers de ce package ont été mises en place grâce à l'expérience de la société Audros en se basant sur les meilleures pratiques rencontrées chez ses clients.

1.3.2 Enterprise Edition

La solution SE peut vite être limitée pour des entreprises souhaitant avoir un contrôle total sur le PLM. En effet, certaines de ces entreprises subissent beaucoup d'évolutions de leurs modèles ou souhaitent automatiser un plus grand nombre de tâches proposées par une SE. C'est pourquoi Audros SE est évolutif : une extension vers la suite Audros Enterprise Edition permet d'accéder aux API et au macro-langage AUPL pour la personnalisation sur-mesure du modèle de données ou pour la réalisation d'applications métier.

1.3.3 Exemples d'applications

Afin d'illustrer le processus de déploiement de la solution Audros, nous considérons l'exemple de PumaProd, une PME agile et maître en développement de ses produits, qui souhaite implémenter le PLM Audros pour pouvoir gérer l'ensemble de ses documents de manière automatisée. L'objectif est d'améliorer l'organisation des informations et d'offrir d'autres avantages à PumaProd comme :

- un accès immédiat et facile à l'information par les utilisateurs,
- une réduction de l'espace de stockage,
- une réduction de la perte d'informations,
- une sécurisation plus élevée des données,
- la facilitation des mises à jour des informations.

La phase d'étude entre les acteurs du projet permet d'identifier le besoin de PumaProd et de prendre une décision sur le seul module à déployer qui correspond aux objectifs du client : la gestion documentaire (GEDT). Cependant, malgré la bonne couverture fonctionnelle du modèle standard du PLM, des adaptations sont à prévoir. La première consiste à adapter les codifications spécifiques des objets proposés par la SE du PLM Audros. PumaProd peut créer ses propres scripts ou utiliser celles de la SE d'Audros qui peuvent être ajustés ou supprimés. Ainsi, la codification des classes d'objets documentaires est adaptée de la façon suivante : Cpt-(RefProjet)-TypeDoc-AAMMJJ. Cette codification est définie au niveau de la classe mère (Document) et est donc héritée par l'ensemble des classes filles (Doc_Qual, Doc_Meth, Doc_Essai, Doc_BE, Doc_Achat). Ces adaptations sont décrites dans le tableau (tab. 1.1). Le modèle métier obtenu est présenté Figure (fig. 1.4).

Codification	Description
Cpt	représente un compteur automatique.
(RefProjet)	représente la référence du projet si le document est créé dans le cadre d'un projet (élément optionnel). Sinon la référence du document débutera par l'élément suivant.
TypeDoc	représente l'acronyme de l'attribut 'doc-type' de la classe documentaire. Par exemple pour un document commercial, l'acronyme sera COMM.
AAAAMMJJ	représente la date du jour de création.

TABLE 1.1 – Description de la codification des objets pour PumaProd

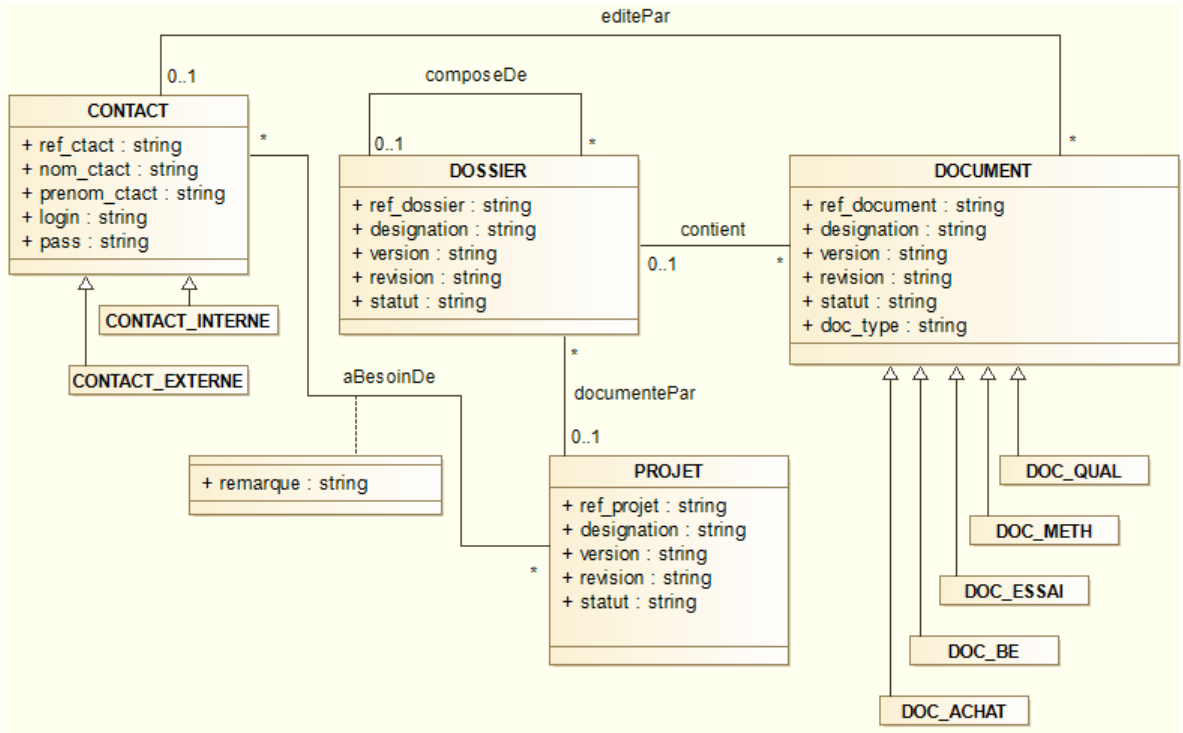

FIGURE 1.4 – Modèle de données au déploiement pour PumaProd

La deuxième adaptation consiste à modifier les profils utilisateurs afin d'avoir une gestion des droits plus élaborée que la standard édition. Les profils utilisateurs de la SE d'Audros étant limités à 3, PumaProd décide d'en créer d'autres afin de rendre l'application plus souple. Cette nouvelle gestion permettra d'avoir des droits plus spécifiques par services (BE, Achats, Qualité...). La dernière adaptation consiste à modifier certains attributs du lien 'aBesoinDe' entre la classe PROJET et CONTACT (cf. (fig. 1.4)). Les ajouts permettront aux utilisateurs de connaître plus de détails sur leur rôle dans ce projet. Par exemple, un nouvel attribut 'remarques' qui permettrait aux utilisateurs de saisir leurs remarques sur le projet en cours. L'attribut 'fonctionActeur' présent dans le modèle

standard est supprimé car inutile dans la gestion documentaire (il est utilisé seulement dans le cadre du module de gestion des processus). Ainsi, lors du déploiement et étant donné que l'entreprise n'a pas retenu le module de gestion des processus, la modification du lien 'aBesoinDe' utilisé également dans ce module n'a pas d'impact sur le fonctionnement standard du PLM.

Le modèle métier obtenu pour le PLM après la phase d'initialisation du projet est un modèle de données spécifique basé principalement sur la standard édition d'Audros. En effet, toutes les classes d'objets et de liens sont celles issues de la SE. La spécificité du modèle se situe au niveau des modifications effectuées dans la gestion des droits, les codifications spécifiques des documents et la modification d'une relation ('aBesoinDe'). Cette première intégration de la solution Audros chez PumaProd étant réalisée grâce à l'accompagnement de l'éditeur (dans d'autres cas, l'intégrateur peut l'assurer) ; aucun problème de cohérence ou de redondance n'a été signalé lors du test et l'usage du système PLM déployé.

1.4 Les problèmes industriels rencontrés

Afin d'illustrer la problématique de ce travail de thèse, nous montrons par des exemples concrets les différents problèmes rencontrés dans les entreprises industrielles pendant et après un premier déploiement d'un système PLM. Ces exemples sont tirés des problèmes rencontrés chez les clients de la société Audros Technology.

1.4.1 Problèmes rencontrés lors du déploiement initial

Dans cette section, nous nous focalisons sur 3 axes du déploiement d'un PLM, parmi ceux listés dans la section précédente : l'incorporation des processus, l'intégration avec les outils CAO et la reprise des données.

Incorporation des processus (workflows)

Habituellement, Les entreprises qui ne disposent pas de systèmes PLM gèrent leurs processus par des documents papiers à faire signer par les intervenants des processus. Cette méthode est confrontée à beaucoup de problèmes. En effet, les délais des processus sont souvent allongés (oubli sur le bureau, perte du document...), le suivi est mal géré (on ne sait pas qui est l'intervenant actuel)... La mise en œuvre d'un projet PLM est une opportunité pour ces entreprises de réduire les délais des processus et d'avoir un meilleur suivi des actions en cours. En effet, la gestion des processus permet de standardiser et d'automatiser la circulation de l'information dans l'entreprise. Elle permet aussi de garder une trace des décisions prises et de relancer les acteurs en cas de dépassement du délai imparti pour une tâche. Cependant l'intégration des processus dans le système PLM n'est pas une tâche aisée, surtout pour les processus très spécifiques et très longs qui nécessitent des développements supplémentaires :

- la modélisation des processus de l'entreprise représente une étape importante dans le déploiement d'un système PLM. L'automatisation des processus manuels est souvent une étape délicate, surtout pour la définition des étapes et des transitions. Le piège dans cette étape est de vouloir modéliser les processus tels qu'ils sont utilisés de façon manuelle. En effet, les processus deviennent vite très longs et moins souples qu'une gestion manuelle. Il est nécessaire à ce niveau de faire des compromis et d'alléger dans la mesure du possible les processus.
- les droits utilisateurs ont besoins d'être définis avec beaucoup d'attention et d'anticipation : par exemple les acteurs d'une étape d'un processus de demande d'évolution ont besoin d'avoir les droits de visualisation et/ou de modification sur les documents concernés par cette demande. De même, afin de pouvoir suivre le processus ou de le faire avancer (changement de statut), les acteurs doivent avoir les droits de modification sur l'objet processus et sur les statuts. Dans le cas contraire, le processus se retrouve bloqué dû à des problèmes de paramétrage lors du déploiement initial du PLM.

Intégration avec les outils CAO

Dans de nombreux déploiements initiaux, le besoin classique de gérer des données CAO est contraint par le type de logiciel CAO utilisé par l'entreprise cliente. De plus, certaines de ces entreprises souhaitent gérer leurs structures produit en fonction de leurs activités (utilisation de modèles CAO paramétrés, squelette de nomenclature, gestion des exigences...). Cette structure de produit ne correspond pas forcément à celle de la solution générique proposée par les éditeurs PLM. Il est alors nécessaire d'adapter la solution générique au mode de fonctionnement du client. La définition du modèle de données doit être validée afin de ne pas créer d'incohérence après les diverses adaptations survenue au sein de la solution générique.

La reprise de données

Le déploiement d'un système PLM nécessite très souvent un processus de reprise de données depuis un ou plusieurs systèmes vers le système PLM. Il est important pour les utilisateurs d'avoir du contenu et un historique des données avant de commencer à travailler avec un nouvel outil. Ce point n'est pas à sous-estimer car il représente une charge de travail non négligeable et demande des ressources importantes. En effet, dans une majorité des cas les fichiers à intégrer dans le PLM sont éparpillés dans des répertoires (coffre de fichiers). La réalisation d'une reprise de données soulève néanmoins plusieurs interrogations :

- Quel volume doit être traité ? Doit-on reprendre l'intégralité des données (sur plusieurs années) ou seulement une partie (à partir d'une date précise) ?
- Comment procéder à cette tâche ? Manuellement ou automatiquement ?
- Quel temps cela va-t-il prendre ?

Il est important d'avoir une phase d'étude au préalable avant de prendre une décision sur la manière de procéder. En effet, il est logique de déterminer le volume des données à

migrer et cela se traduit souvent par une grosse masse de données. Il est inconcevable de réaliser une reprise de données manuellement à partir du moment où le volume de données à intégrer est conséquent. De plus, la nécessité d'organiser les données est une notion importante pour la codification et la création des nomenclatures dans le système PLM. Dans la majorité des cas, la solution la plus cohérente est l'utilisation d'itérations par script pour insérer l'intégralité des données. La mise en place d'une bonne stratégie de reprise est essentielle, celle-ci doit pouvoir gérer les codifications (identifier les doublons), attacher les fichiers aux objets et créer les nomenclatures. Le calcul du temps de la reprise de données et de sa validation peut être déterminé avec l'utilisation d'une base de test. Cette solution permet d'éviter les surprises avant une mise en production. Cependant, malgré la bonne couverture d'une solution par itérations, celle-ci est souvent complétée par une intégration manuelle pour des fichiers non traités (mauvais répertoire, doublon dans la codification...).

Cette phase de reprise de données reste une étape très délicate pour les entreprises industrielles dans un projet de déploiement d'un système PLM. Les modèles métiers implémentés en amont doivent être réfléchis et donc permettre une reprise de données cohérentes [Izadpanah, 2011].

1.4.2 La gestion des évolutions de modèles

Après la mise en place d'un système PLM, des évolutions des besoins métiers apparaissent assez rapidement après un premier déploiement. De ce fait, l'entreprise est souvent amenée à faire évoluer les modèles qui ont été mis en place lors de la première intégration du PLM. L'évolution des modèles d'une entreprise pour adapter ou ajouter des fonctionnalités n'est pas sans risque. Nous identifions deux scénarios différents pour la mise en place de ces évolutions :

- la première est de faire appel à une société externe
- la seconde est de réaliser les modifications sans l'intervention d'une société externe

Nous discutons de ces deux scénarios ci-après.

Appel à une société externe

N'ayant pas assez de recul, certaines entreprises ne sont pas capables de réaliser les modifications correctement et se retrouvent dans le plus souvent des cas avec un système d'information instable. Afin de remédier à ce problème, elles sont obligées d'être accompagnées par une société externe (éditeur ou intégrateur) pour parvenir à leurs besoins. Cependant, cette solution s'avère assez coûteuse avec le temps à cause des quatre points suivants :

- peu de réactivité : le laps de temps entre l'identification du problème et l'intervention de l'éditeur ou de l'intégrateur est souvent préjudiciable à l'entreprise qui se retrouve obligé de continuer à travailler avec un SI instable. La création d'un ticket d'intervention de support technique (généralement via une interface web) est le premier moyen de communication entre l'entreprise et la société externe. Si le problème n'est pas résolu par une intervention à distance, un consultant de la société

externe intervient sur place (dans la mesure des disponibilités) pour la résolution du problème.
- problème d'expression de besoins : malgré une identification précise du problème, les entreprises décrivent assez mal les difficultés rencontrées au responsable du support. Par ailleurs, dans beaucoup de cas elles ne donnent pas assez de détails pour la résolution du problème : modification à l'origine du problème, comment reproduire le problème, les traces de l'outil pour le support technique... L'ensemble de ces éléments augmente le temps de résolution du problème rencontré.
- des effets de bord : une intervention sur des problèmes identifiés peut avoir des effets de bord sur d'autres éléments du modèle métier. En effet, ces corrections entraînent une réaction en cascade qui créée de nouveaux problèmes à corriger. Cet "engrenage" est très nuisible pour l'entreprise cliente qui se voit encore perdre du temps sur des nouvelles corrections qui auraient pu être évitées.
- réalisation des tests : il est préférable qu'après la correction d'un ou plusieurs problèmes, des tests soient réalisés afin de vérifier le bon fonctionnement de l'outil (pas d'instabilité). L'idéal dans cette situation est de disposer d'une base de tests pour ne pas interférer avec les données de production. Ces tests mobilisent des ressources de l'entreprise ou de la société externe (facturation supplémentaire).

Réalisation des évolutions sans l'intervention d'une société externe

Pour éviter les frais liés au scénario précédent, certaines entreprises tentent de réaliser elles-mêmes les modifications nécessaires. Ces modifications, sont généralement réalisées par un administrateur système interne à l'entreprise et préalablement formé à l'outil afin d'éviter l'intervention d'une société externe. Malgré cette formation, ces entreprises éprouvent des difficultés pour identifier les impacts des modifications ; aucun outil ne leur permet d'avoir une vision globale sur le système PLM en terme de dépendance entre les éléments qui le composent. Ces modifications peuvent donc causer des problèmes de cohérence ou des effets de bord au sein du système d'information de l'entreprise. La mise en production des nouvelles évolutions sans maîtriser leurs impacts peut s'avérer assez risquée. Lors de l'utilisation de l'outil, l'impact des modifications se fait ressentir par les utilisateurs.

Pour illustrer ces difficultés, revenons sur l'exemple de l'entreprise PumaProd présentée dans la section précédente. Celle-ci souhaite supprimer un attribut dans une classe documentaire et ajouter un nouveau statut dans le cycle de vie d'un objet.

La suppression de l'attribut 'doc-type' dans une classe d'objet documentaire peut avoir deux conséquences imprévues :
- l'oubli de mettre à jour le script de codification automatique qui se basait sur cet attribut pour codifier la référence du document (codification héritée). Au final, à chaque tentative de création d'une instance de la classe documentaire dans le PLM, une erreur d'exécution du script de codification sera déclenchée. En effet, la codification de la référence de cet objet se base sur l'attribut 'doc-type' qui a été supprimé.

– l'oubli de supprimer l'attribut de la vue métier (présentation des données d'un objet ou d'un lien dans le PLM Audros) des profils utilisateurs concernés alors que l'attribut a bien été supprimé dans la base de données. Ainsi, aucune instance de classe de ce type ne peut être créée par un utilisateur avec le stockage de la valeur de cet attribut. En effet, même si l'utilisateur saisit des informations dans ce champ, étant donné que l'attribut n'existe plus en base, il ne sera pas sauvé. Si l'utilisateur affiche de nouveau la vue métier de ce même objet créé précédemment, ce champ sera toujours vide.

La création d'un nouveau statut dans le cycle de vie d'un objet du modèle métier sans avoir modifié les droits ne permettra pas aux utilisateurs de mettre ce nouveau statut sur l'objet désiré. En effet, après l'ajout du statut, aucune gestion des droits n'a été modifiée par l'administrateur afin que les utilisateurs puissent modifier le cycle de vie de cet objet. L'ajout d'un nouveau statut doit être suivi d'une mise à jour des droits utilisateur afin de permettre aux utilisateurs d'utiliser ce nouveau statut.

Dans la suite de notre exemple, la même entreprise cliente est amenée à étendre le périmètre fonctionnel de son PLM en intégrant le module de gestion des processus écarté dans un premier temps. Elle souhaite en effet automatiser les processus de validation et d'évolution des documents, afin de réduire leurs délais d'exécution. D'autant plus que PumaProd n'est pas à l'abri de la perte d'un document de suivi (validation ou d'évolution) dans un service, ce qui obligerait à recommencer le processus depuis le début. Cette entreprise souhaite donc utiliser les workflows de demande de validation et de demande d'évolution proposés par la standard édition d'Audros afin de gagner du temps et d'avoir un meilleur suivi sur ces processus. L'intégration du nouveau module est réalisée par l'administrateur PLM. Le modèle de données obtenu après l'intégration de ce module complémentaire est visible sur la figure (fig. 1.5). Cette intégration implique l'introduction de 3 nouveaux éléments dans le modèle produit : la classe d'objet "PROCESSUS", la classe de lien 'concerne' pour exprimer qu'un processus donné (validation par exemple) porte sur tel objet métier et la classe de lien 'utilise' pour définir les acteurs d'un processus. Étant donné que le modèle produit de la SE a été légèrement modifié lors du déploiement initial du PLM afin de s'adapter aux spécificités de PumaProd, l'intégration du module de gestion des processus peut s'avérer plus compliquée que prévue. En effet, un problème de cohérence pourrait être rencontré car le lien 'aBesoinDe' qui contenait des attributs spécifiques pour permettre le fonctionnement du module de gestion des processus a été modifié lors de la mise en place du module gestion documentaire (cf. déploiement initial). De ce fait, le module gestion des processus ne peut pas fonctionner correctement suite à cette incohérence dans la base de données car l'attribut 'fonctionActeur' supprimé est obligatoire pour le fonctionnement de ce module. Il est utilisé pour définir la fonction des acteurs intervenant dans les processus. Afin de remédier à ces problèmes, la solution consiste à créer une nouvelle classe de liens qui aura la définition originelle du lien 'aBesoinDe', celle de la standard édition. Ce nouveau lien est représenté dans le diagramme UML par le lien 'utilise'. Cependant la création d'un nouveau lien pour un fonctionnement standard, rendra le modèle encore plus spécifique, un éloignement de la standard édition étant déconseillé pour éviter d'autres problèmes de ce genre.

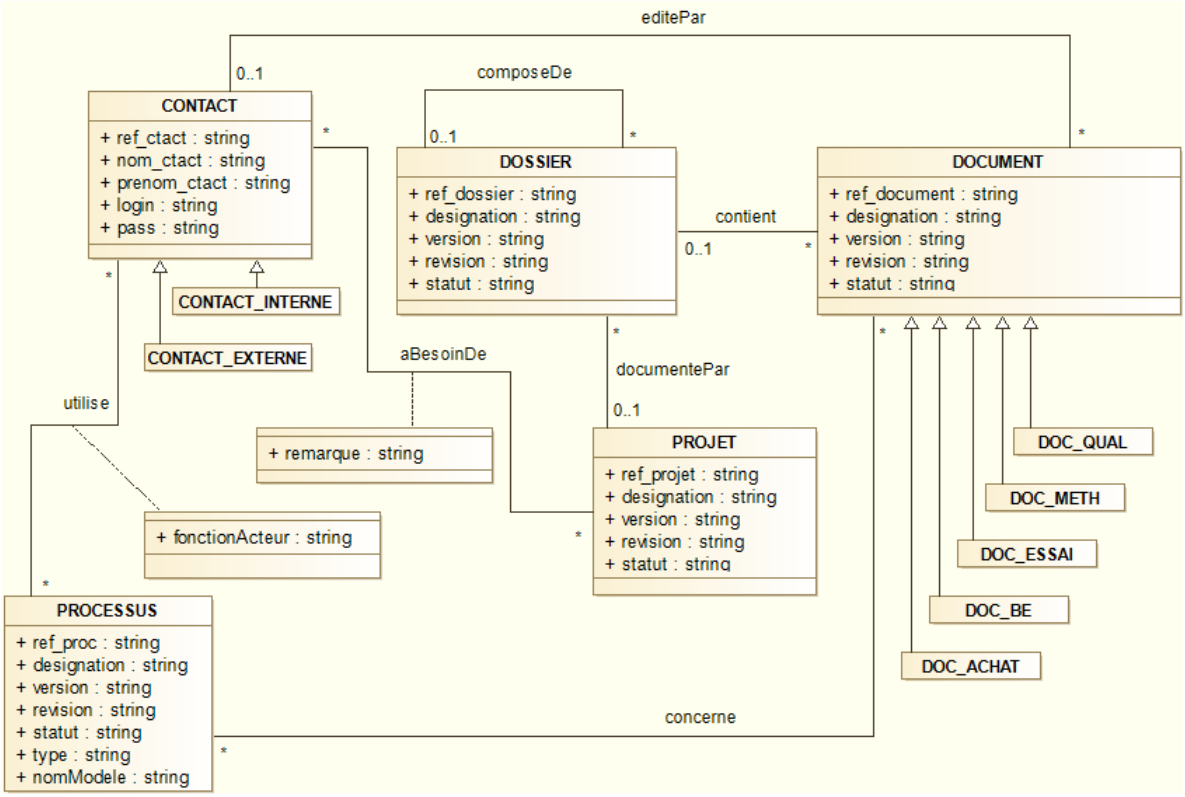

FIGURE 1.5 – Modèle produit obtenu après intégration du module de gestion des processus

De plus, après toute modification du modèle de données de la SE, l'entreprise peut être confrontée de la même façon à des problèmes de cohérence dans son modèle de données si elle désire bénéficier de nouvelles fonctionnalités. En effet, le modèle de données de la SE d'Audros est amené à évoluer au cours du temps afin de proposer de nouvelles fonctionnalités (pour répondre à des demandes récurrentes de clients ou pour s'adapter à l'évolution du marché) et également permettre à l'éditeur Audros de rester compétitif. De ce fait, si l'entreprise PumaProd souhaite intégrer une nouvelle fonctionnalité, elle doit faire attention car ses modèles métiers ne correspondent plus à la standard édition du PLM Audros. Généralement elle est obligée d'être accompagnée par l'éditeur ou l'intégrateur afin de mettre en place ces nouvelles fonctionnalités sans créer de problèmes de cohérence dans le SI de l'entreprise.

Enfin, dans le cas du PLM Audros, des tâches automatisées peuvent modifier les données en fonction d'évènements utilisateur. Par exemple nous pouvons citer la création de nomenclatures suite à un évènement création de projets et la codification automatique des documents selon leurs types. Ces tâches sont réalisées par des scripts AUPL (AUdros Programming Language) qui peuvent modifier le modèle métier ou interagir avec lui. Pour gérer ces diverses dépendances et maintenir une cohérence globale du système, il est nécessaire de contrôler, les dépendances (à la création et à la modification) entre les divers éléments des modèles métiers et disposer de mécanismes de synchronisation des scripts vis à vis des modifications des modèles.

Ces problématiques industrielles mettent en évidence des verrous scientifiques qui s'inscrivent dans un cadre plus global autour des systèmes d'information industriels et que nous décrivons dans la section suivante.

1.5 Problématique scientifique des travaux

Les différentes problématiques industrielles que nous avons illustrées résultent des retours d'expérience des clients de la société Audros. Elles permettent de mettre en évidence un ensemble de verrous et de problématiques scientifiques. Ainsi pour nos travaux, le contexte scientifique est indéniablement lié à la conception d'un système d'information et donc à la caractérisation des modèles qui le composent. Dans ce contexte, le système d'information est un élément pivot qui doit contribuer à la bonne marche de l'activité de l'entreprise. En tant que système, il permet de piloter les activités opérationnelles de l'entreprise, il se doit donc d'être adapté à ses activités et implicitement à ses métiers. Dans le cas des problématiques industrielles de déploiement, la problématique scientifique est double :

- pour l'utilisateur du PLM (entreprise cliente), il est nécessaire de définir le niveau métier de l'entreprise afin d'assurer un déploiement adapté. Ce niveau métier est représenté par un ensemble de modèles dont les concepts mobilisés sont conformes à l'activité de l'entreprise et manipulables par ses acteurs. Idéalement, les concepts spécifiques aux systèmes PLM devraient être intégrés et masqués.
- Pour le concepteur du PLM (éditeur), il s'agit de minimiser ses développements afin de ne pas développer des solutions spécifiques à chaque entreprise mais de se positionner dans une "personnalisation au plus tard". Pour ce dernier point, les travaux dans ce domaine en lien avec la ré-utilisabilité concerne les méthodologies de conception (ingénierie des composants, approche par parton, ingénierie des modèles).

Pour la construction des modèles métiers adaptés aux activités des entreprises clientes, il convient de distinguer deux catégories de construction : d'une part, la modélisation des éléments métiers. Les modèles obtenus permettent de structurer aussi bien sur les aspects données que les aspects processus, voire organisationnels. D'autre part, la modélisation des aspects comportementaux spécifiques du niveau métier. Ces aspects que nous dénommons les règles métiers caractérisent les usages et pratiques spécifiques en lien à l'activité de l'entreprise. Elles sont liées à un domaine d'activité mais aussi aux modes de fonctionnement même d'une entreprise. Leur identification, modélisation et mise en œuvre sont indispensables pour un déploiement initial et pour une évolution dans la mesure où elles impactent les modèles de données ou de processus.

En conclusion, la problématique scientifique de ce travail de thèse est déclinée selon deux questions de recherche :

- quels modèles et quelles méthodologies permettant la conception du niveau métier adapté aux entreprises et leur déploiement au sein d'un système d'information de type PLM ?

– quels modèles pour la définition des règles métiers et quels mécanismes de validation de ces règles au sein des modèles déployés dans un système PLM ?

Chapitre 2

Etat de l'art

Sommaire

2.1 Introduction

Dans le chapitre précédent nous avons introduit les concepts du PLM et positionné le contexte scientifique de ces travaux. Dans ce cadre nous présentons un état de l'art en lien avec le contexte de notre problématique. Dans la première section, nous présentons le contexte actuel des travaux de recherches dans le domaine du PLM, et plus particulièrement sur les aspects de modélisation et d'intégration. Dans la seconde section, nous présentons la démarche de l'ingénierie dirigée par les modèles

2.2 Etat de l'art des travaux autour des systèmes PLM

Ayant vocation à représenter les produits tout au long de leur cycle de vie, les systèmes PLM doivent s'appuyer sur des "représentations" techniques suffisamment riches pour permettre à chaque acteur (stakeholder) de disposer d'une vue propre cohérente. La question de la modélisation au sein des systèmes PLM est ainsi liée à des problématiques de représentation, de codage des applications PLM, de communication entre

applications, de déploiement et d'évolution. Les propositions académiques de modélisation et de représentation des produits en cours de conception, en vue d'une exploitation pour la production, la maintenance, le recyclage, sont nombreuses ; les premiers travaux sur ce sujet remontent au début des années 80 et ont dans un premier temps concerné le secteur du génie mécanique. Un remarquable état de l'art sur ces questions se trouve dans l'article de [Chandrasegaran et al., 2013] : les modèles les plus reconnus dans la littérature feront l'objet d'une description dans les deux sections suivantes. Sur un plan méthodologique, la conception à base de pattern [Gzara, 2000] a montré sa pertinence pour l'élaboration et la validation des modèles techniques de représentation, ainsi que la définition des processus qui permettent les interactions entre les acteurs de l'ingénierie [Gzara et al., 2003].

La structuration du produit qu'il soit virtuel ou physique a pour but d'organiser les informations d'un processus global de développement [Eynard et al., 2006] [Sudarsan et al., 2005]. Dans les phases initiales de conception, la structuration du produit est éminemment dépendante de la méthodologie de conception. Il existe de nombreux modèles produits [Pernelle, 2002], néanmoins le modèle de produit Fonction-Behavior-State (FSB) fait aujourd'hui consensus [Noël and Roucoules, 2008].

2.2.1 Interopérabilité de systèmes et échanges de données

L'interopérabilité est la capacité que possède un produit ou un système à fonctionner avec d'autres produits ou systèmes existants ou futurs, et ce sans restriction d'accès ou de mise en œuvre. Cette définition du GDT Interop place bien le problème à un un niveau global de la capacité à "opérer" ensemble de deux systèmes différents. Les problèmes d'interopérabilité sont à la source de nombreux problèmes techniques : dans le domaine ferroviaire par exemple, la compatibilité des voitures et des boogies avec les rails ou avec les ouvrages d'art, peut faire l'objet d'études techniques relativement complexes. L'interopérabilité peut aussi dans ce cas concerner les arrêts en gare, la longueur et la section des quais : les derniers avatars de la SNCF et de Réseaux Ferrés de France, avec le constructeur de train canadien Bombardier, illustrent la complexité de cette problématique. Sans vouloir rechercher ni identifier des responsabilités, il est clair qu'un certain nombre de "données techniques" n'ont, dans ce cas, pas été transmises aux bons interlocuteurs, et l'on peut dire que ce problème révèle un manque de maîtrise de la chaîne PLM du client au constructeur. Certaines erreurs très médiatisées, imputables aux outils de développement ou à leurs usages, militent pour une meilleure organisation des processus d'innovation d'ingénierie, afin de qualifier les améliorations de façon fiable. Par exemple, la perte de la sonde martienne Mars Climate Orbiter le 23 septembre 1999 a mis en évidence un manque de formalisation dans les processus de transfert de données entre des partenaires du projet, soit un problème d'interopérabilité de la chaîne de traitement.

STEP : interopérabilité des données d'ingénierie

Les industriels de l'ingénierie se sont, dès l'apparition des systèmes CAO au début des années 80, souciés de la pérennité de leurs données (pérennité des plans et des modèles 3D), de leur indépendance aux fournisseurs des solutions logicielles, enfin de leur

capacité à échanger des données avec des interlocuteurs externes à l'entreprise. Quand on pense que pour des raisons de certification, de maintenance et de fourniture de pièces détachées pour toute la vie série de l'appareil, les constructeurs aéronautiques sont tenus de maintenir à jour les plans de leurs produits sur une durée supérieure à une cinquantaine d'années. Les plans papiers faisaient ainsi l'objet d'un soin particulier en vue de leur conservation : usage contrôlé des calques et des encres, température et hygrométrie... Lors du passage à des formats numériques, il s'est agi de s'assurer de la possibilité de conserver les informations sur une durée aussi longue, aussi bien s'agissant des supports matériels (bande magnétique ?), des appareils de lecture (qui peut aujourd'hui lire une disquette 5"1/4 ?), que du format sémantique des fichiers qui encodent les informations graphiques. Cette contrainte de gestion sur une durée longue des informations techniques a été l'un des puissants moteurs de mise en place du standard d'interopérabilité STEP, alors que chaque éditeur faisait valoir la supériorité de sa solution propriétaire par rapport à la concurrence. Les grands éditeurs logiciels de cette époque avaient tous développé des applications et des formats de données propriétaires qui ne permettaient aucun échange entre systèmes différents. Les rares formats neutres éditables sous forme de texte se limitaient alors au standard de fait DXF d'AutoCAD (société AutoDesk), qui n'hébergeait qu'une partie de la sémantique des données de cet outil : ainsi lors d'un test de transfert bidirectionnel, l'utilisateur ne retrouvait qu'une partie de ses données, rendant une ingénierie collaborative très périlleuse et soumise à de multiples adaptations de données avec les risques d'erreurs correspondants. Bien évidemment, les sociétés qui abordaient à cette époque la mondialisation des échanges, ont donné aux efforts de normalisation des données techniques une portée planétaire : c'est ainsi qu'est né le projet STEP - norme ISO 10303 (Standard for the Exchange of Product Data Model). Aujourd'hui les protocoles applicatifs, tels que l'AP214, permettent de traiter de nombreux problèmes liés au partage de données CAO, voire leur intégration dans le PLM [Srinivasan, 2009].

BPMN : interopérabilité des processus d'entreprise

L'interopérabilité des processus est une exigence fondementale pour la maîtrise globale du processus de développement des produits. Ceci est d'autant plus nécessaire que les processus sont mis en oeuvre dans le SI par des composants hétérogènes, avec des niveaux de sémantique et de granularité très différents (cf chapitre 1). Dans ce contexte, la question des standards pour les processus métier se pose sur deux aspects : le premier concerne la définition des processus et le second aspect concerne leur exécution.

Pour la définition des processus, BPMN (Business Process Modeling Notation) [OMG, 2011a] est une notation graphique permettant de modéliser les processus métiers. BPMN est structuré autour d'un seul type de diagramme : BPD (Business Process Diagram). La démarche BPMN est centrée sur les activités et propose une approche dite 'top-down' de décomposition.

L'intérêt de BPMN est de proposer une notation simple permettant de modéliser des processus métier en choisissant le niveau de détail. Certes, leur définition, même à un faible niveau de détail, ne permet pas leur instanciation immédiate dans le PLM ou dans tout autre sous-système. C'est un des aspects qui différencie le choix de l'OMG par rapport à d'autres organisations comme la Wfmc avec XPDL où l'exécution est gérée par des

mécanismes de transformation vers un autre standard dénommé BPEL.

Dans [Pinel, 2013], l'auteure utilise BPMN en lien avec une démarche MDE pour définir les processus de définition de cahier des charges en lien avec une transformation de type MDE. Dans [Bissay, 2010], l'auteur utilise BPNN pour définir des processus métiers et pour modéliser une méthodologie de déploiement. D'autres travaux s'appuient sur BPMN pour initier des démarches de modélisation globale qui élargissent le cadre du PLM (par exemple dans une démache BPR) [Bertoni et al., 2009].

2.2.2 Gestion des connaissances et PLM

Dans le contexte PLM, la pérennisation du savoir faire d'ingénierie et des connaissances reste une préoccupation pour les chercheurs et les industriels. En effet, il est pertinent de s'interroger sur les capacités des systèmes PLM à capitaliser plus que la simple information autour des produits : les systèmes PLM peuvent-ils contribuer à capitaliser des connaissances métier ou des savoir-faire ? Actuellement, ces systèmes ne gèrent pas la connaissance en tant que telle, car, pour la création et/ou l'acquisition des connaissances, les données ne sont considérées que comme une « matière première » : la transformation d'un ensemble de données en connaissance nécessite un processus d'interprétation sémantique et d'appropriation des informations [Ermine, 2003]. Pourtant, les systèmes PLM recèlent en leur sein un capital de connaissances et de savoir-faire non-explicite qui est sous-exploité [Barcikowski, 2006]. D'ailleurs, il est probable que l'une des orientations futures des systèmes PLM concerne la création, l'intégration et l'utilisation de connaissances [Ameri and Dutta, 2005]. Des théories de la conception et de la gestion des connaissances, telles que C-K, sont aujourd'hui proposées par des groupes de recherches et mises en œuvre dans un certain nombre d'expérimentations industrielles.

La réutilisation des connaissances reste peu exploitée au sein de ces systèmes PLM. Toutefois, il existe de nombreux travaux portant justement sur l'utilisation des connaissances explicitées dans les phases de développement du produit permettant une intégration de cette connaissance au sein des systèmes d'information. Pour le cas particulier des PLM, cette intégration passe par l'usage de mécanismes complémentaires d'extraction et de de formalisation ainsi que par l'utilisation de systèmes spécifiques [Gomes, 2008]. On trouve d'ailleurs à ce sujet de nombreuses approches :

- les approches utilisant les systèmes multi-agents [Monticolo et al., 2008] [Mahdjoub et al., 2010]
- les approches utilisant des ontologies [Lalit et al., 2005] [Catalano et al., 2009] [Matsokis and Kiritsis, 2010] [Ben Miled, 2011]
- les approches à base d'annotations sémantiques [Felic et al., 2014] [Krima, 2013]
- les approches méthodologiques d'intégration [Srinivasan, 2009] [Bernard et al., 2009] [Bissay, 2010]
- les approches de plateforme de conception collaborative à base de paramètres et de connaissances [Zouari, 2007] [Zouari et al., 2015]

Quelles que soient les approches de représentation proposées, la nécessité des entreprises d'aller vers une meilleure maîtrise de leur savoir-faire est un facteur de levier important sur les systèmes PLM qui gèrent de fait les données issues de cette connaissance. Notons d'ailleurs que le besoin de caractériser une sémantique sur les données structu-

rées du PLM est multiple : on retrouve ce besoin pour une meilleure réutilisation du savoir-faire, mais aussi pour faciliter l'interopérabilité entre les systèmes.

2.3 Ingénierie Dirigée par les Modèles

2.3.1 Les concepts de l'IDM

L'ingénierie dirigée par les modèles (IDM) est un domaine de recherche qui a été à l'origine d'améliorations dans le développement du logiciel [Bezivin, 2004]. C'est une approche qui met les modèles au centre des processus d'ingénierie logicielle. En effet, avec cette démarche, tout ou une partie du code d'une application est générée directement à partir de modèles. Les modèles sont une abstraction d'un système et sont considérés comme des éléments de base pour la production, le test et l'exécution des systèmes d'information. Ils sont au centre des processus de développement de ces systèmes tout en étant riches et assez précis afin d'être aisément transformés ou exécutés. L'IDM vise donc à apporter des démarches permettant de maîtriser la complexité des systèmes d'information, mais également de l'évolution de ceux-ci. Contrairement à l'approche de la programmation objet, qui est basée sur les relations d'instances, l'ingénierie dirigée par les modèles utilise un concept et des relations différentes. En effet, elle est fondée sur la notion de niveau d'abstraction de modèles et sur des mécanismes de transformation entre ces modèles. L'ingénierie dirigée par les modèles doit sa notoriété aux travaux de l'OMG avec l'architecture dirigée par les modèles (MDA) [OMG, 2001], le but était de généraliser l'idée et les concepts de MDA.

Dans nos propositions, qui seront présentées dans la seconde partie, nous avons basé notre approche sur la démarche MDA. Dans le présent chapitre, nous abordons donc les principes de l'IDM et plus particulièrement la démarche proposée par l'OMG au travers MDA (source de l'IDM). Après les définitions des concepts de modèles et méta-modèles, et la démarche de l'IDM, nous présentons les initiatives l'OMG avec MDA et les standards comme le MOF, XMI ou l'OCL. Pour terminer, nous montrons l'étape clé d'une démarche IDM avec les principes des transformations de modèles.

La démarche d'IDM

L'ingénierie dirigée par les modèles (IDM) est une forme d'ingénierie générative qui permet la création d'une application informatique à partir de transformations de modèles [Muller, 2006]. Ainsi, avec l'IDM, le développement logiciel est processus itératif piloté par les modèles. Finalement, l'IDM pourrait être considérée comme une démarche globale du génie logiciel basée sur la définition de modèles à différents niveaux d'abstraction et d'usage [Favre et al., 2006]. La cohérence globale entre les modèles est alors fournie par des mécanismes de transformations.

Cette approche par niveau d'abstraction permet de dissocier de façon explicite, les spécificités d'un système, de sa mise en œuvre technologique. Ainsi la spécification abstraite d'un système permet au concepteur de se focaliser sur les concepts à identifier sans

être contraint par les éléments techniques. A ce niveau, on retrouve les principes de la méthodes Merise qui dissociait les niveaux logique et conceptuel d'une application.

En résumé, les trois piliers de l'IDM sont :

- le principe de conception par des niveaux d'abstraction de modèles,
- le principe de séparation des concepts et des technologies,
- le principe de transformation des modèles.

Modèle et méta-modèle

Les éléments principaux dans une approche d'ingénierie dirigée par les modèles sont comme son nom l'indique, les modèles. Il n'existe pas une seule définition de ce terme, nous retenons celle donnée par Minski [Minsky, 1965] pour définir un modèle : "Pour un observateur B, un objet A* est un modèle d'un objet A s'il permet à B de répondre à une question qu'il se pose sur A". Ici nous donnons une définition d'un modèle tel qu'il était perçu à l'origine. Un modèle doit servir à :

- décrire simplement et rapidement un système, accorder des vues multiples,
- spécifier, représenter le système sans ambiguïté,
- construire et tester le système.

Classiquement, à partir de cette définition nous pouvons dire qu'un modèle est une abstraction d'un système. Un modèle est composé d'entités sur lesquelles des opérations peuvent être appliquées de manière automatisées. Les modèles ont vocation à représenter la structure et le comportement d'un système complexe en apportant une simplification par l'usage de concepts formels. La question de la simplification est relative et pose la question de la contextualisation d'un modèle. Prenons par analogie, la démarche de modélisation des automaticiens pour asservir un système complexe. Certains vont modéliser de manière simplifiée le système, avec un modèle linéaire d'ordre 1 ou 2 et constituer un asservissement qui tienne compte de l'incertitude initiale. D'autres vont modéliser en utilisant des équations non linéaires pour être le plus proche possible du système. Ces deux approches mettent en évidence qu'un modèle unique n'existe pas, mais que le modèle est toujours le résultat d'un travail de contextualisation du concepteur. Dans le domaine du développement logiciel, cette question existe aussi sous une autre forme. Le système à modéliser est généralement virtuel et la complexité de la modélisation est implicitement liée à la performance et à la précision du système attendu. Dans le cas d'une approche objet, elle se caractérise par le nombre de classes, de méthodes et d'attributs en interaction. Cependant, la notion de modèle n'est pas suffisante en elle-même. En effet, un modèle doit être exprimé dans un langage spécifique respectant scrupuleusement une syntaxe qui correspond au langage de modélisation, le méta-modèle. Le modèle est une concrétisation (éléments) de ce méta-modèle.

Un méta-modèle est considéré comme un modèle de modèles. Il est donc lui-même considéré comme un modèle qui permet de définir le langage d'expression des modèles. Cette notion permet de définir la relation (estConformeA) entre le méta-modèle et un modèle. En effet, ce dernier doit être conforme aux concepts du méta-modèle à partir duquel il a été construit. La confusion entre méta-modèle et modèle générique est as-

sez courante notamment dans les propositions de méta-modèles à partir de diagramme UML de la relation de généralisation. Ainsi, en IDM, un méta-modèle spécifie en quelque sorte la syntaxe des modèles, comme par analogie, on peut dire qu'un langage spécifie sa grammaire [Cherif, 2013].

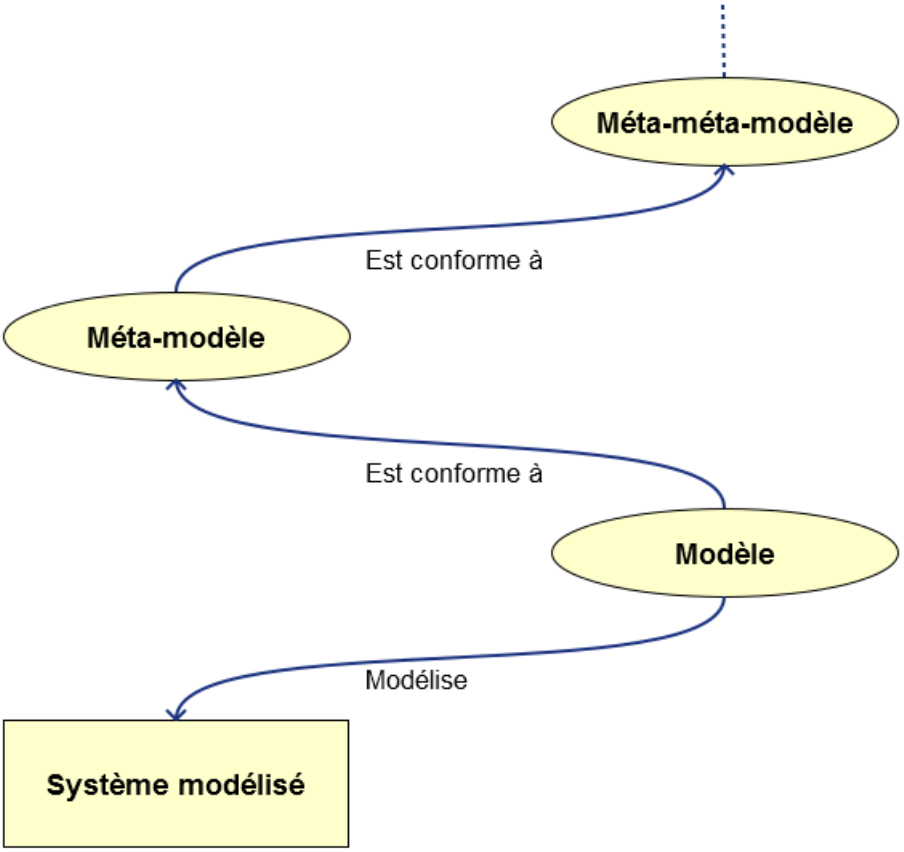

FIGURE 2.1 – Les bases de la démarche par l'IDM

Cette vision caractérise la relation de conformité qui lie le méta-modèle au modèle grâce auquel il a été construit. Cependant, le méta-modèle doit être lui-même écrit dans un langage de modélisation et donc être conforme à un méta-méta-modèle (niveau supérieur dans une démarche d'ingénierie dirigée par les modèles). Finalement la démarche d'IDM généralise une relation entre un niveau N et un niveau méta N+1.

2.3.2 L'approche d'architecture dirigée par les modèles (MDA)

Apparu en 2000 grâce aux travaux de l'OMG [OMG, 2001], le MDA est un standard (à l'origine de l'IDM) [Bezivin and Gerbe, 2001] qui a pour objectif d'initier une nouvelle façon de conception des applications en séparant l'architecture technique et la logique métier de l'entreprise, mettant ainsi en œuvre le double "track unified design" ou modèle en Y [Roques and Vallée, 2000]. En effet, contrairement à la logique métier qui ne subit pas beaucoup de modifications, l'architecture technique est plus instable car elle est souvent amenée à évoluer dans le temps. La séparation des spécifications fonctionnelles

du système de son implémentation sur une plate-forme est indéniablement la base de la démarche MDA. L'idée est donc de pouvoir gérer des modèles métiers hétérogènes indépendamment de toute plate-forme (PIM : Platform-Independant Models) et de passer par transformations successives à des modèles spécifiques dépendant de la plate-forme retenue (PSM : Platform-Specific Models). Le déploiement d'un modèle PIM sur plusieurs plateformes cibles est possible par l'intermédiaire d'une approche MDA. Le concepteur ne se soucie pas de la plateforme cible mais seulement du modèle métier de l'entreprise. Au final, à la fin du processus, MDA permet la génération de codes exécutables.

Finalement, l'IDM est une généralisation de la démarche MDA. On peut donc considérer MDA comme un processus spécique de l'IDM. Contrairement à l'IDM, MDA propose un nombre de niveaux d'abstractions fixes. Ceci est du, entre autres, au fait que sur le niveau d'abstraction le plus élevé, l'OMG propose le MOF [OMG, 2011b] qui est définit de manière réflexive.

Pour définir MDA, l'OMG s'appuie sur différents standards dans une architecture à quatre niveaux. Le schéma ci-dessous (fig. 2.2) présente l'ensemble des quatre niveaux de l'architecture MDA avec les différents standards.

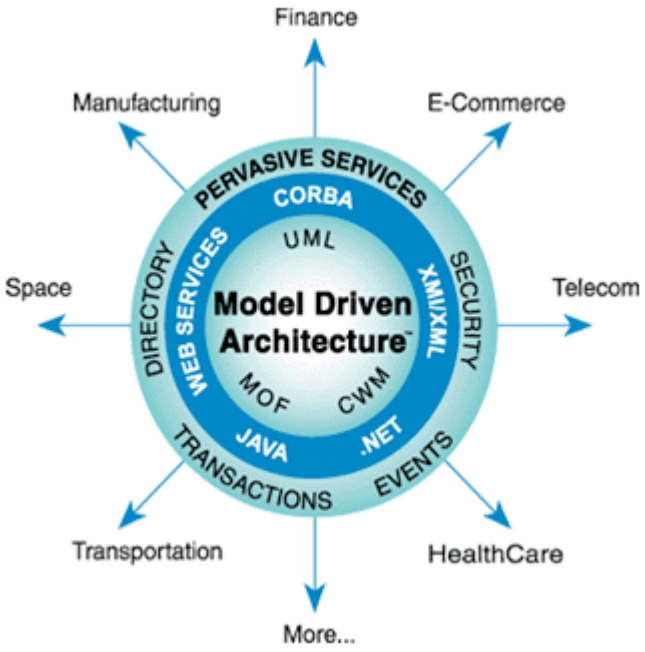

FIGURE 2.2 – Architecture de la démarche MDA

- Le niveau M3 caractérise le méta-méta-modèle. Il définit ainsi les notions de bases permettant la description formel des méta-modèles et des modèles et donc finalement les principes de conformité entre les niveaux. C'est le niveaux le plus élevé qui a la particularité de proposer une définition réflexive
- Le niveau M2 définit le méta-modèle et il est basé sur le méta-modèle d'UML 2. A ce niveau, on définit donc les règles de manipulation des modèles de niveau inférieur

(règles de transformation).
- Le niveau M1 définit les modèles qui doivent être conformes aux méta-modèles du niveau supérieur.
- Le niveau M0 donne une représentation "modélisée" d'un système réel.

Le MOF

Étant donné que les méta-modèles sont également considérés comme des modèles, il faut qu'ils soient eux même conformes à des « méta »-méta-modèles. L'OMG a mis en place un langage de définition des méta-modèles qui a lui aussi la forme d'un modèle, le MOF (Meta-Object-Facility) [OMG, 2011b]. Afin d'éviter un trop grand nombre de niveaux d'abstraction, ce dernier a la capacité de se décrire lui même. Il est donc réflexif, c'est-à-dire que le niveau supérieur est décrit à partir de lui-même. Ainsi, Le MOF se situe au sommet d'une architecture à quatre niveaux d'abstraction pour la modélisation des systèmes avec MDA présentée dans le schéma suivant (fig. 2.3) :

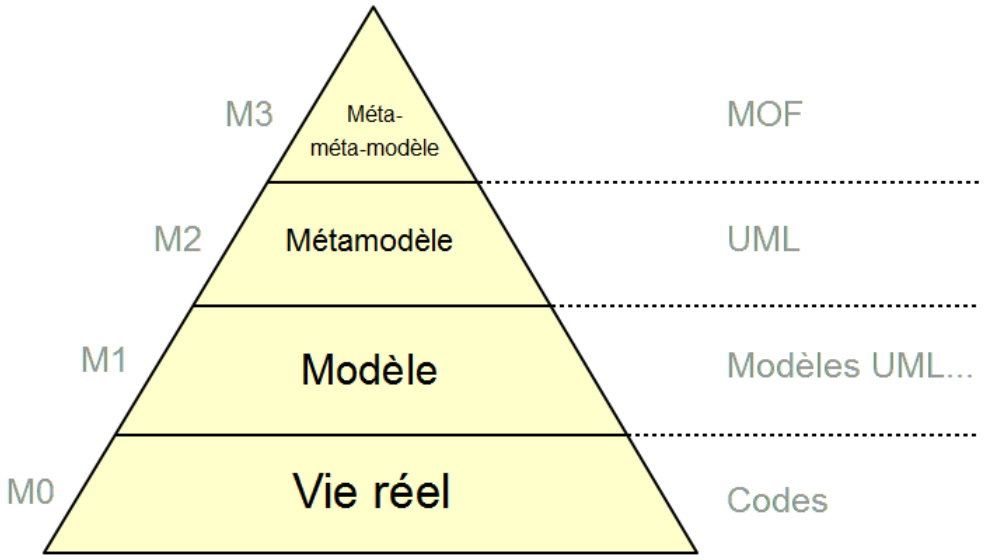

FIGURE 2.3 – Pyramide de la démarche MDA

Le MOF permet donc de définir la sémantique d'un langage et la syntaxe des méta-modèles ainsi que leur structure, c'est-à-dire les éléments principaux dans la construction des modèles. Elle permet notamment la manipulation et la comparaison des modèles et méta-modèles. Le MOF est un standard de méta-modélisation. Il définit par exemple la notation UML. Il est devenu aujourd'hui un pilier de l'architecture MDA. Il correspond à une architecture qui est basée sur une relation d'instanciation, c'est-à-dire que chaque élément d'une couche est une instance d'un élément de la couche supérieure. Ce standard permet de fédérer l'ensemble des langages de l'OMG.

Les niveaux CIM/PIM/PSM

Le principe fondamental de MDA est la séparation entre la logique métier et la logique d'implémentation. Il est composé de plusieurs modèles utilisés pour modéliser l'application puis à générer du code par les transformations successives. Dans l'approche des niveaux proposée par MDA, les modèles ou méta-modèles sont définis par une démarche d'abstraction vis à vis des plateformes cibles. Une description de chacun d'entre eux est donnée ci-dessous :

CIM le *Computation Independent Model* est un niveau de modélisation indépendant de tout système d'information. Il permet de représenter les concepts utilisés indépendamment de leur représentation ou de leur instanciation. Il permet de définir les sources de vocabulaires partagés avec d'autres modèles. L'indépendance technique de ce modèle lui permet de garder tout son intérêt au fil du temps et n'est modifiée que si le modèle de connaissance d'une entreprise a besoin de changement.

PIM Le *Plateform Independent Model* est un niveau de modélisation indépendant de toute plateforme technique (EJB, .NET...) et ne contient pas d'informations sur les technologies qui seront utilisées pour déployer l'application. Il s'agit d'un niveau de modèle qui représente une vue partielle d'un CIM.

PSM le *Plateform Specific Model* est le niveau de modélisation dépendant de la plateforme utilisée. Il est utilisé pour la génération de code. Le PSM décrit comment le système va utiliser ces plateformes. Il y a plusieurs niveaux de PSM, le premier issu de la transformation d'un PIM. Les autres sont obtenus par des transformations successives de même niveau (raffinement). Ils permettent d'obtenir le code dans un langage spécifique.

MDA fournit [Bezivin, 2004], un processus de conception et met en œuvre des outils pour :

1. spécifier un système indépendamment de la plate-forme qui le supporte, et donc réaliser un PIM,
2. enrichir ce modèle par étapes successives,
3. spécifier les plate-formes,
4. choisir une plate-forme particulière pour le système,
5. transformer la spécification du système (PIM) en une autre spécification pour une plate-forme particulière (PSM),
6. transformer un CIM en un PIM et un PIM en un autre PIM,
7. raffiner le PSM jusqu'à obtenir une implémentation exécutable.

Dans les diagrammes il est difficile, voire impossible dans certains cas de préciser de manière complète toutes les subtilités d'un modèle. Il faut donc pouvoir accompagner ces diagrammes de descriptions grâce à des contraintes qui permettent de les rendre plus précis et sans ambiguïté. L'objectif étant de les rendre beaucoup plus subtils et plus complets.

XML Medatada Interchange (XMI)

Avec MDA, au niveau conceptuel, les modèles restent des éléments abstraits, c'est pourquoi l'OMG a défini un format standard d'import et d'export des modèles, le XMI. Concrètement, ce standard permet le passage des modèles sous le format XML par l'intermédiaire de règles permettant la construction de schémas XML à partir de méta-modèles. La réciproque est vraie aussi, à partir d'un document XML il est possible de construire le méta-modèle.

Le XMI donne la possibilité aux modèles d'être encodé dans un document XML, il est donc un format d'échange standard de modèles pour les outils MDA. Cet encodage est réalisé par deux types de mécanismes (fig. 2.4) [Blanc, 2005] :

- génération : c'est une transformation des méta-modèles en DTD (Document Type Definition).
- sérialisation : c'est une représentation des modèles en XML.

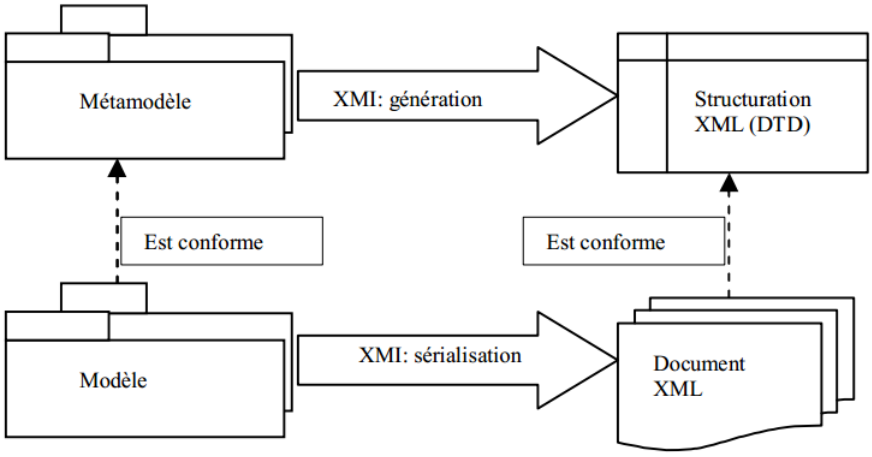

FIGURE 2.4 – Les mécanismes d'encodage

2.3.3 Les contraintes sur les modèles

Notion de contraintes

Avec UML 2.0 [OMG, 2005], l'OMG propose une extension des relations dans les modèles par la définition de contraintes (fig. 2.5). Elles sont utilisées dans la plupart des cas pour préciser des invariants (une contrainte prédicative sur un objet, ou un groupe d'objets, qui doit être respectée en permanence) sur les méta-classes et pour la description des extensions du méta-modèle grâce à des stéréotypes. Il existe plusieurs méthodes d'expression des contraintes, par un langage formel (en OCL par exemple), semi-formel ou en langage naturel. Dans [Cadavid et al., 2011], les auteurs rappellent que les méta-modèles existants présentent des anomalies qui empêchent l'analyse automatique, dans ce cas, la définition de contraintes apportent des précisions. Dans le cadre de MDA, la notion de contrainte représente une expression à valeur booléenne que l'on peut attacher à n'importe quel élément du modèle, elle apporte plus de précision sur les objets. Elle indique en

général une restriction ou donne des informations complémentaires sur un modèle. Pour définir des contraintes de modélisation, il est possible d'utiliser :

- des pré-conditions et post-conditions sur les opérations,
- des contraintes sur la valeur retournée par une opération,
- des règles de dérivation des attributs,
- des descriptions de cibles pour les messages et les actions,
- des expressions des conditions dans les diagrammes dynamiques,
- des invariants de type pour les stéréotypes.

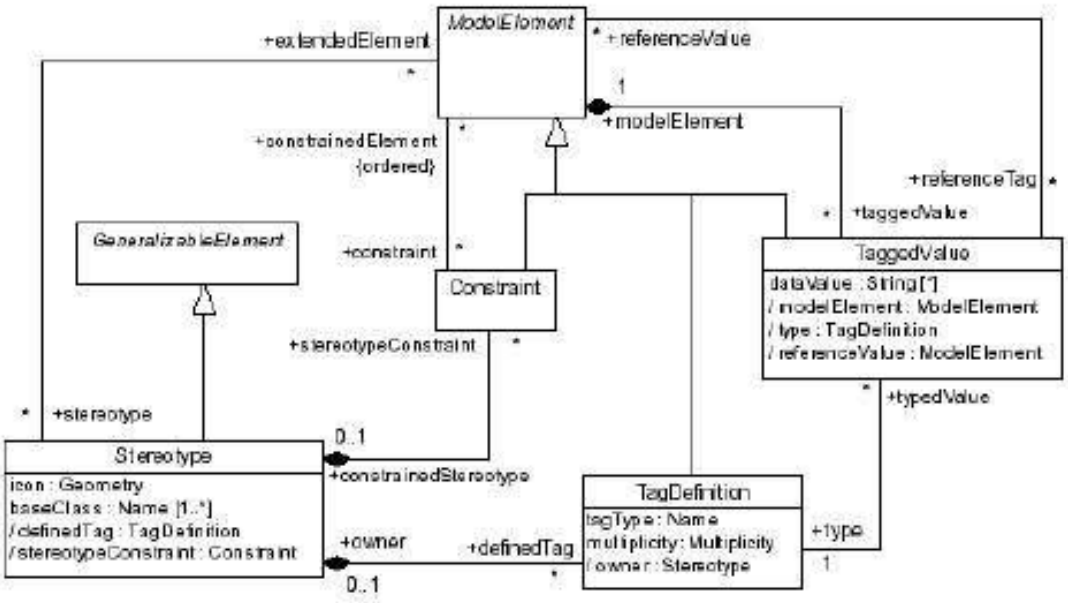

FIGURE 2.5 – Principes d'extension d'UML

Les contraintes permettent de caractériser les règles de vérification à chaque niveau de modélisation. Certaines contraintes sont implicites. Par exemple, les relations (composition, agrégation...) définies sur le méta-modèle impliquent d'être vérifiées dans les modèles PIM afin de garantir la conformité. D'autres nécessitent une validation explicite dans différents niveaux de modélisation. Ces contraintes complètent les diagrammes existants qui sont parfois peu précis et permettent de rendre les relations plus explicites et sans ambiguïté dans les modèles. Ils permettent également de contrôler la conformité et la cohérence au sein des différents modèles.

Object constraint language (OCL)

L'OCL [OMG, 2014] est un langage formel de définition des contraintes sur des méta-modèles (à travers les stéréotypes). Il a été créé en 1997 par IBM, sur les bases du langage IBEL (Integrated Business Engineering Language). Il a été intégré dans la définition d'UML 2.0 dès 2003 et est conforme au MOF 2.0. C'est un langage qui permet l'expression de contraintes sur n'importe quel élément d'un modèle sous forme de condition. Il ne

modifie pas les éléments du modèle, ce qui lui permet de n'avoir aucun effet de bord sur le modèle. Les fonctionnalités apportées par l'intégration de contraintes OCL sont les suivantes :

- stockage des modèles avec leurs contraintes.
- validation syntaxique et sémantique des contraintes.
- génération de code Java, C++ ou autres pour vérifier les contraintes à l'exécution.

L'OCL permet de restreindre une ou plusieurs valeurs d'un ou d'une partie d'un modèle. L'avantage d'un tel langage formel est qu'il est non ambigu et facile à utiliser. Concrètement, une contrainte OCL est une valeur booléenne. L'expression des contraintes une seule fois dans le méta-modèle peut être évaluée sur tous les modèles conformes au méta-modèle. OCL permet principalement l'expression de deux types de contraintes sur l'état d'un objet ou d'un ensemble d'objets :

- des invariants qui doivent être respectés en permanence.
- des pré-conditions et post-conditions pour une opération.

Nous pouvons citer quelques travaux en lien avec la conception de produit : [Monestel et al., 2002] [Ziadi et al., 2003] proposent des contraintes architecturales pour des lignes de produits, exprimées par des contraintes OCL sur les méta-modèles. Cherfi [Si-Said Cherfi et al., 2012] propose des règles de mappage OCL pour la mise en correspondance de modèles métier.

2.3.4 Les transformations de modèles

Concepts de base

La notion de transformation est primordiale dans une démarche de type MDA. C'est elle qui permet le passage d'un niveau à un autre (PIM vers PSM par exemple). Par définition, une transformation est une opération qui prend en entrée un modèle source et qui génère en sortie un modèle cible. Ces deux modèles doivent avoir une relation de conformité avec leurs méta-modèles respectifs (qui peuvent être identiques). Une transformation est donc un ensemble de règles à respecter pour représenter des correspondances entre les éléments du modèle source et les éléments du modèle cible.

Les étapes principales de l'approche MDA sont constituées des niveaux CIM, PIM et PSM qui contiennent des modèles nécessaires à la génération d'un environnement exécutable. Ce dernier est obtenu par génération à partir du PSM qui est lui-même obtenu par transformations entres les différents niveaux, du PIM vers le PIM et du PIM vers le PSM. En effet, un des objectifs majeurs du MDA est de rendre opérationnel les modèles en utilisant des transformations successives entre eux. Ainsi, le raffinement du PIM vers le PIM puis du PIM vers le PSM nécessite l'utilisation d'outils et de techniques de transformations. Le but est de préciser ce que sera l'application à partir de modèles génériques. Ce sont les transformations successives qui permettent aux modèles de devenir des éléments productifs de MDA.

Pour réaliser les transformations, il est important que les modèles (source et cible) soient exprimés dans un langage de modélisation. Il existe plusieurs types de transforma-

tions :

- endogène : il concerne des transformations s'appuyant sur le même méta-modèle. Le modèle cible obtenu par transformation est conforme au même méta-modèle cible qui est dans ce cas précis le même que le modèle source.
- exogène : il concerne des transformations s'appuyant sur des méta-modèles différents. Le modèle cible obtenu par transformation est conforme à un méta-modèle différent que le méta-modèle source.
- horizontale : il concerne des transformations entre deux modèles du même niveau d'abstraction. Par exemple une transformation entre deux PIM.
- verticale : il concerne des transformations entre deux modèles de différents niveaux d'abstraction, par exemple la transformation du CIM vers PIM.

Le fonctionnement du processus de transformation peut être décrit en 2 phases. La première phase consiste à définir un méta-modèle cible et à mettre en correspondance les éléments de celui-ci avec le méta-modèle source. La mise en correspondance est réalisée grâce à des règles de transformation. Ces règles se basent sur un modèle source (conforme au méta-modèle source) en entrée et génère en sortie un modèle cible (conforme au méta-modèle cible). La deuxième phase concerne l'exécution des règles de transformation dans un langage donné. Le modèle d'entrée permet ainsi la production du modèle cible par cette exécution. Le schéma ci dessous résume le principe de fonctionnement des transformations de modèles (fig. 2.6).

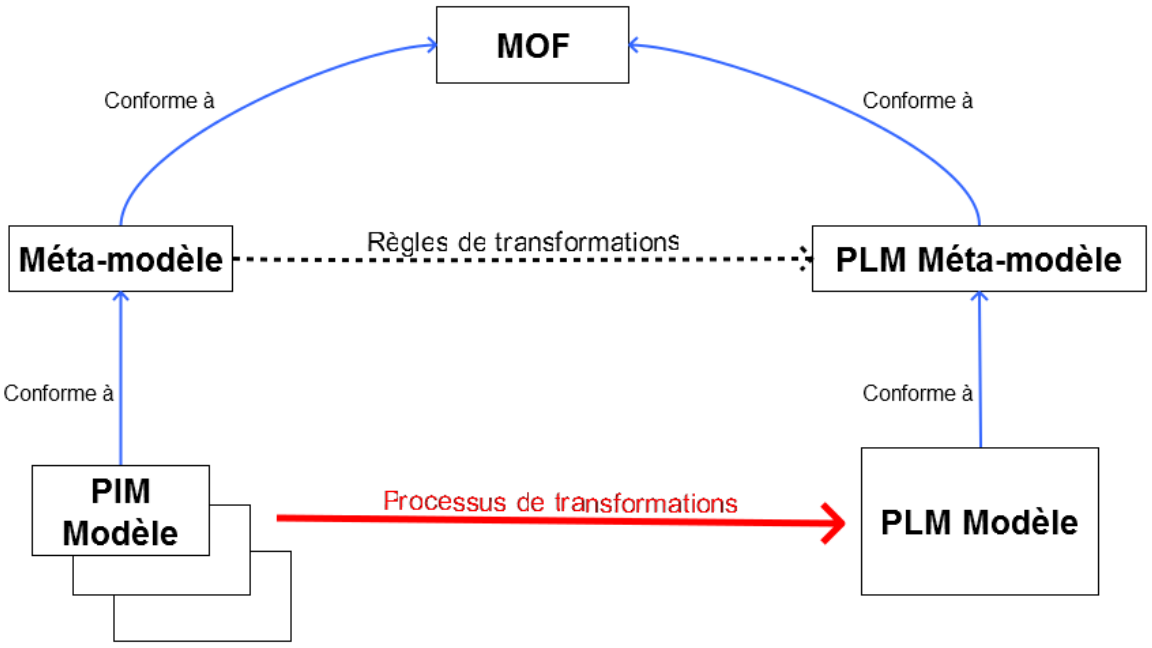

FIGURE 2.6 – Transformations de modèles

Approche Query View Transformation (QVT)

QVT est une spécification établie par l'OMG qui propose un standard dans la modélisation des transformations. Il permet de définir des langages de transformation et de

manipulation des modèles de transformation grâce à ses méta-modèles conformes et totalement immergés au MOF. C'est un standard pour la transformation de modèles vers modèles (M2M). Ce standard possède trois méta-modèles (langages) de transformation :

- relations : il correspond à un langage déclaratif et est basé sur le concept des patterns objets. Il est utilisé comme représentation des relations entre les modèles ou comme transformation (par la transformation RelationsToCore) en modèle core. Le langage Relations dispose également d'éléments de traces générées automatiquement lors de transformations entre modèles. Ainsi des mécanismes permettent la reconnaissance de motifs dans un modèle. De plus, il permet également l'interfaçage avec la partie impérative.
- operational mapings : langage impératif spécifique qui permet de décrire des transformations. Il peut être utilisé de manière autonome (en définissant des règles de transformation dans une boite noire) ou faire appel aux langages déclaratifs pour une utilisation hybride. L'OCL est le langage utilisé pour la définition des règles de transformation.
- core : langage déclaratif avec une syntaxe textuelle. Il permet de définir des règles de transformations. Contrairement au langage Relations, Core ne supporte pas les mécanismes de patterns. Les traces de la transformation doivent être générées par l'utilisateur.

L'OMG propose ainsi aux utilisateurs une approche hybride (déclaratif/impératif) dans le but de pouvoir satisfaire chacun d'entre eux. De plus, l'approche QVT est utilisée dans plusieurs langages dont Atlas Transformation Language qui est le plus courant et le plus utilisé dans les projets concernés par MDA.

Atlas transformation language (ATL)

Opérationnel et nature Atlas Transformation Language est un langage de transformation de modèles à modèles (M2M). Il permet de produire un ou plusieurs modèles cibles à partir d'un ou de plusieurs modèles en entrée. Il est très utilisé dans le milieu industriel et académique. C'est un langage avec une syntaxe proche de l'OCL (utilisation de l'OCL pour des requêtes de transformation) qui s'appuie sur le méta-modèle EMF (Eclipse Modeling framework). En effet, les outils de transformation ATL font partie de l'environnement de développement Eclipse sous la forme de plug-in. De plus, l'ATL utilise le format standard XMI pour la définition des méta-modèles.

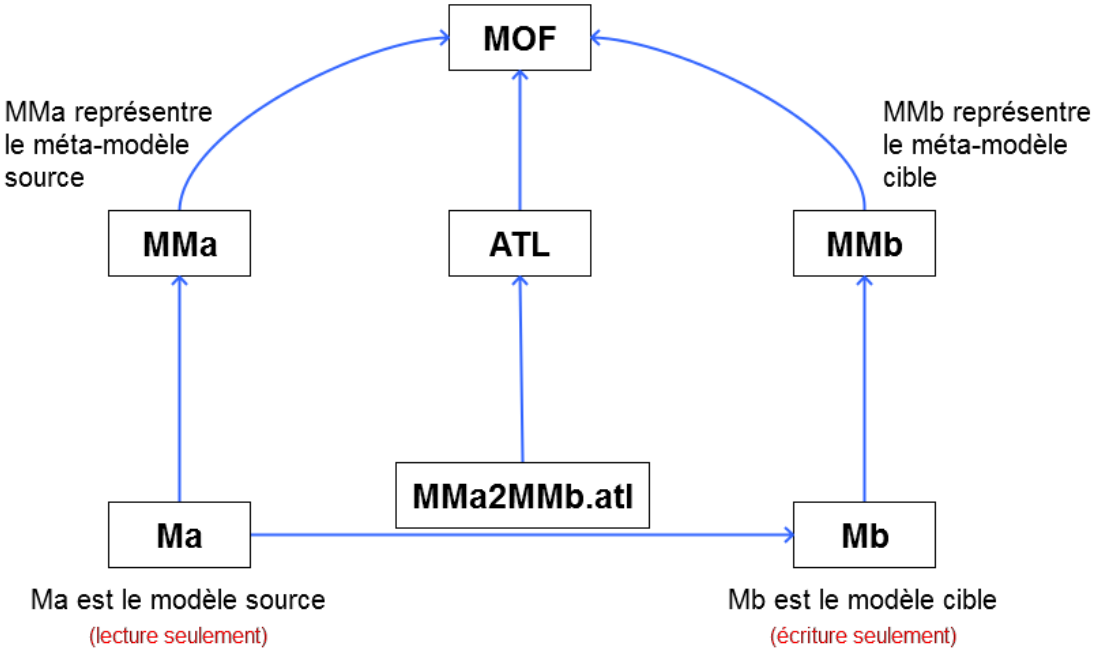

FIGURE 2.7 – Transformations avec ATL

Une transformation ATL consiste à créer un modèle cible (Mb) à partir d'un modèle source (Ma) en entrée. Cette transformation est basée sur des règles de correspondance entre le méta-modèle source (MMa) et cible (MMb). Ces règles sont regroupées dans le modèle ATL (MMa2MMb.atl) qui est lui aussi conforme au méta-modèle ATL. Tous les modèles (MMa, MMb et ATL) sont conformes aux méta-méta-modèle MOF de l'OMG. La transformation se déroule en parcourant le contenu du modèle en entrée (Ma), en comparant ce contenu avec les patterns sources (règles de correspondance définies dans le fichier .atl) et en produisant les patterns cibles dans le modèle cible en cas de correspondance.

Le langage ATL est un langage Hybride : déclaratif/impératif. La définition des règles de correspondance entre les modèles d'entrée et les modèles de sortie représente la partie déclarative du langage. La partie impérative est surtout utilisée pour les invocations des règles et pour les blocs d'actions (séquence d'instructions). Cependant, le paradigme déclaratif est celui recommandé par les concepteurs du langage.

Concepts d'ATL Les principaux concepts du langage ATL sont :

– entête de la transformation (header) : la convention veut qu'une transformation d'un modèle à un autre, soit nommée d'après les méta-modèles avec un 2 (to) ajouté entre les deux. OUT et IN sont les noms donnés aux modèles. Ils ne sont pas utilisés par la suite. Les modèles sources et cibles sont typés, ici DB et GEN qui correspondent respectivement aux méta-modèle DB_metamodel et Generic_metamodel. Le code ci-dessous montre le header d'une transformation avec ATL.

```
-- @path DB=/Audros/Metamodels/DB_metamodele.ecore
-- @path Gen=/Audros/Metamodels/Generic_metamodele.ecore
```

```
module Generic2db;
create OUT : DB from IN : Gen;
```

Listing 2.1 – Exemple d'un entête d'une transformation

- méthodes auxiliaires (Helpers) : un helper est défini sur un contexte et pourra être appliqué sur toute expression ayant pour type ce contexte (comme une méthode dans une classe en Java). Le contexte peut être celui du module ou bien celui d'un élément du module. Il peut prendre des paramètres (non obligatoires) et possède nécessairement un type de retour. Enfin, le code d'un helper est une expression OCL.
- les règles déclaratives (Declarative Rules) : elles spécifient un pattern source dans le modèle d'entrée et un pattern cible à créer dans le modèle de sortie. Elles sont de 3 types Standard (appliquée une fois pour chaque correspondance), lazy (appliquée pour chaque correspondance autant de fois qu'elle est référencée par une autre règle) et unique lazy (appliquée lorsqu'elle est référencée par une autre règle, pour chaque correspondance). Le code ci-dessous montre l'utilisation d'une règle déclarative en ATL.

```
rule Class2Table {
  from      -- source pattern
    c:ClassDiagram!Class
  to        -- target pattern
    t:Relational!Table
}
```

Listing 2.2 – Exemple de règle déclarative standard

2.4 Conclusion

Dans ce chapitre, nous avons présenté les concepts principaux de l'ingénierie des modèles. En fait, nous avons restreint ces concepts à la démarche MDA proposée par l'OMG. Cette démarche est au cœur de notre proposition, car nous pensons que le découpage par niveau est une bonne approche vis à vis des problématiques explicitées dans le premier chapitre. En effet, que ce soit pour le déploiement initial du PLM ou la gestion de ses évolutions, le principe de caractériser les niveaux métiers permet de séparer les concepts métier des contraintes techniques liées au PLM. Ainsi, comme le propose la démarche MDA, il s'agit de mettre en évidence des mécanismes permettant de simplifier et d'automatiser (grâce aux transformations) le travail du concepteur au sein des systèmes PLM.

Chapitre 3

Approche de construction de modèles métier au sein d'un système PLM

Sommaire

3.1 Introduction

Dans ce chapitre, nous décrivons la première étape de notre proposition. La démarche choisie étant basée sur MDA, nous présentons dans un premier temps les concepts qui vont être utilisés dans les modèles métiers. Ces concepts, du niveau CIM, sont décrits dans la première partie de ce chapitre par un modèle générique sous contraintes, ainsi que par les acteurs utilisant ce modèle. Ensuite, nous présentons la caractérisation des modèles métiers et d'exécution (niveau PIM et PSM). Pour finir, nous montrons la création d'un méta-modèle pour le PLM Audros afin de décrire le niveau PSM.

3.2 Méta-modèle du niveau CIM

Dans le contexte de ce travail, la modélisation du niveau CIM doit représenter les concepts d'un produit quel que soit son domaine ou sa technologie, les processus de production... On peut considérer qu'ils sont liés à leurs dimensions structurelles et comportementales. Dans tous les cas, le méta-modèle du niveau CIM n'est qu'un des éléments qui permet d'élaborer des modèles métiers consistants (niveau PIM), il doit donc être

construit pour un usage méthodologique dans une démarche de conception et de reconfiguration du système PLM. Les modèles du niveau CIM sont ici considérés comme les méta-modèles du niveau PIM.

Comme nous l'avons indiqué dans l'introduction générale et illustré par l'étude bibliographique, il n'existe pas de standard de méta-modèle, mais plusieurs travaux ont toutefois tenté de proposer des modèles de référence. Le méta-modèle proposé dans cette thèse est issu d'une démarche d'abstraction de ces modèles de référence. Dans la perspective CIM, le modèle est générique afin que la démarche proposée pour le support de l'évolution des modèles soit indépendante du domaine d'activité de l'entreprise et de toute solution PLM du marché. Il a pour but de définir les principaux concepts qui vont être utilisés dans les systèmes PLM. Ces concepts doivent être indépendants du système et du domaine métier de l'entreprise mais nous considérons qu'ils doivent faire apparaître les concepts spécifiques au domaine du PLM. Ils sont utilisés pour caractériser la conformité des modèles de niveau inférieur. Les éléments proposés dans ce modèle permettent de caractériser les concepts invariants structuraux et comportementaux qui doivent être utilisés par un système PLM.

3.2.1 Quels acteurs ?

La caractérisation des acteurs nous paraît fondamentale car elle pose la question de ceux qui vont utiliser les concepts du niveau CIM pour construire des modèles métiers. Les travaux en lien avec problématique de déploiement [Batenburg et al., 2006] [Bissay, 2010] [Luh et al., 2011] et les retours d'expériences [7] industrielles montrent la difficulté de faire dialoguer les experts métiers avec les architectes du PLM. Cette difficulté est en partie liée aux ambiguïtés d'usage de chacun [Minsky, 1969] : usage d'une terminologie parfois commune sans que les concepts soient les mêmes, construction des modèles en fonction de leurs implantations et non de leurs usages... Ainsi donc, la distinction des acteurs permettra de dissocier une partie de leur usage dans la construction des modèles métiers. Schématiquement, nous identifions deux types d'acteurs pour la construction des modèles métiers :

L'expert SI PLM , ou Maitrise d'oeuvre (MOE) qui manipule les concepts liés au système d'information PLM. Il utilise des concepts commun à tous les systèmes PLM et il définit les contraintes liées à ces concepts (exemple : tout objet du PLM a un identifiant unique, deux objets ne peuvent pas avoir le même nom).

L'expert métier , ou Maitrise d'ouvrage (MOA) qui décrit les concepts liés à son domaine d'activité. Il construit un ou plusieurs modèles qui doivent être déployés dans le PLM de l'entreprise compréhensible par l'utilisateur final ou les experts métiers.

Le diagramme (fig. 3.1) décrit les usages communs de ces différents acteurs. L'identification des concepts qui vont servir à construire un méta-modèle consistant est contrainte par leur usage. Certains concepts (objets métiers, relations...) sont utilisables par tous les types d'acteurs. Pour autant, d'autres concepts n'ont de sens que pour les experts métiers ou les experts SI.

7. http ://www.jdplm.fr

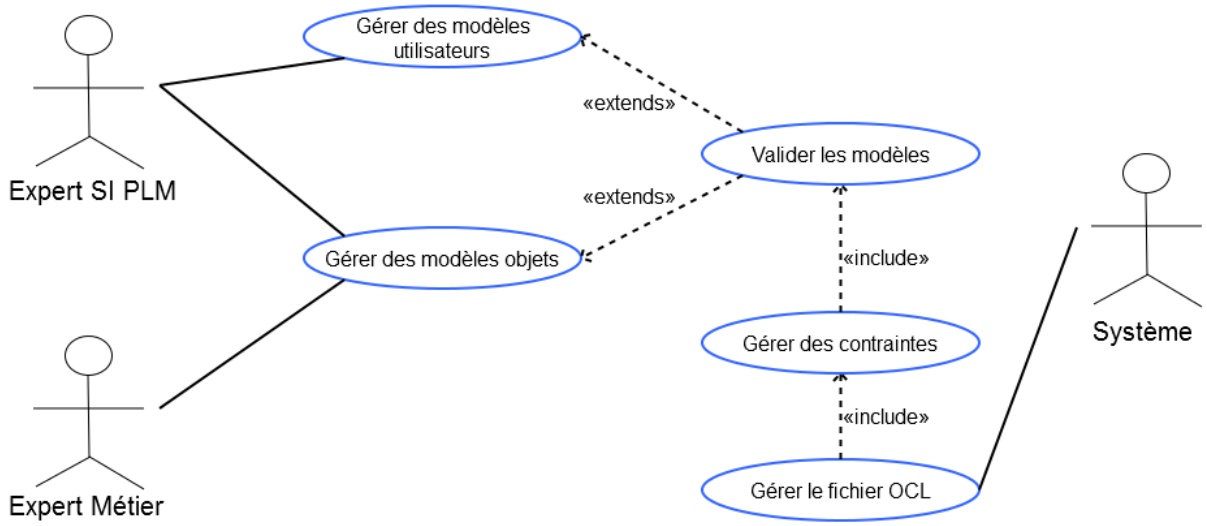

FIGURE 3.1 – Cas d'usage des acteurs pour le niveau CIM

3.2.2 Concepts du niveau CIM

Le diagramme (fig. 3.2) présente les principaux concepts proposés dans le méta-modèle du niveau CIM. Comme le montrent ces différents usages, chaque expert manipule des modèles composés de diagrammes et de contraintes. Pour autant les concepts manipulés par chacun des experts ne sont pas forcement disjoints. Si nous prenons le cas du concept BusinessObject, il sera utilisé par l'expert métier pour créer ses propres types d'objets mais il sera également utilisé par l'expert SI pour créer des objets liés au fonctionnement du PLM (par exemple les objets portant des modèles géométriques). A contrario, le concept d'AttrSystem sera manipulé uniquement par l'expert SI, car ils concernent les attributs systèmes des objets imposés par une application. Par la suite, nous montrerons que la distinction apparaît nettement mieux dans la définition des contraintes.

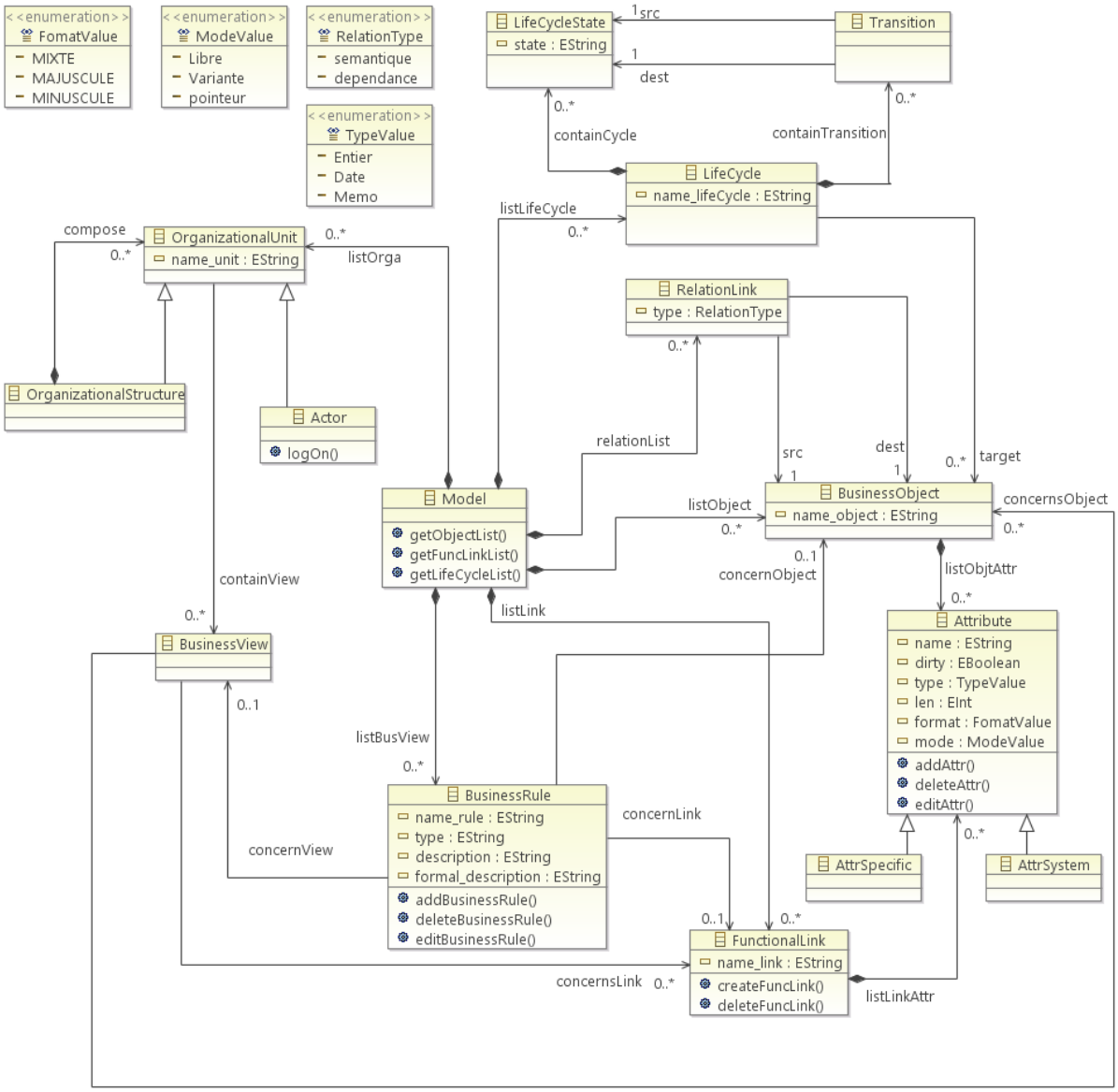

FIGURE 3.2 – Un méta-modèle pour les systèmes PLM

Les tableaux (tab. 3.1) et (tab. 3.2) récapitulent les principaux concepts de ce méta-modèle :

Concept	Description
Model	Stéréotype représentant le modèle créé, il est composé de plusieurs éléments : classe d'objets, classe de liens (direct ou indirect), de cycle de vie et de droits utilisateurs.
BusinesObject	Permet la représentation des classes d'objets (objets manipulables) dans un modèle de données de type PLM avec leurs attributs. Par exemple PIECE, SOUS-ENSEMBLE ou PRODUIT. Il peut représenter par exemple un modèle CAO.
FunctionalLink	Permet de décrire une relation directe entre deux objets (classe de lien), avec leurs attributs et la notion de père et fils. Par exemple un lien de type BOM entre un SOUS-ENSEMBLE (objet père) et une PIECE (objet fils).
Attribute	Décrit les attributs d'une classe d'objets ou de liens et peuvent être de deux types : AttrSystem ou AttrSpecific.
AttrSystem	Décrit les attributs systèmes d'une classe d'objets ou de liens, c'est-à-dire les attributs imposés par la solution PLM cible. Leurs définitions ne sont pas modifiables par les clients de la MOA. Par exemple : REFERENCE, DESIGNATION, VERSION et REVISION.
AttrSpecific	Décrit les attributs spécifiques d'une classe d'objets ou de liens, c'est-à-dire les attributs contrôlés par le client selon l'objet ou le lien. Par exemple : MATIERE, POIDS, TAILLE.
RelationalLink	Permet de décrire une relation entre deux objets de manière indirecte, avec objet source et objet destination.

TABLE 3.1 – Concepts objets du méta-modèle pour les systèmes PLM

Concept	Description
LifeCycle	Permet la définition d'un cycle de vie pour les objets. Il est composé d'un ou plusieurs états et des transitions entre les différents états.
LifeCycleState	Décrit les différents états possibles pour un objet. Par exemple VALIDE, OBSOLETE, APPROUVE, BROUILLON ou PROTO.
Transition	Décrit les différentes transitions entre les états (avec un état source et un état destination). Par exemple de l'état INITIAL -> EN COURS DE VALIDATION ou INITIAL -> ABANDONNE et EN COURS DE VALIDATION -> VALIDE.
BusinessRule	Permet de décrire des règles sur les objets et les liens (contraintes).
BusinessView	Décrit la présentation des objets et des liens (vue objet ou lien) à l'utilisateur selon ses droits.
OrganizationalUnit	Utilisé pour définir plusieurs groupes de personnes dans la gestion des droits. Par exemple le groupe Bureau d'études ou Achat.
OrganizationalStructure	Permet la création de niveau sous les groupes. Par exemple un niveau Administrateur BE et utilisateur BE dans le groupe bureau d'études.
Actor	Décrit les acteurs intervenant sur le PLM (Administrateurs ou utilisateurs). Un acteur doit faire partie d'un OrganizationalUnit et d'un OrganizationalStructure

TABLE 3.2 – Concepts organisationnels du méta-modèle pour les systèmes PLM

3.3 Caractérisation des modèles métier et d'exécution

3.3.1 Caractérisation du niveau PIM

La perspective PIM doit présenter les modèles décrivant le métier indépendamment de toutes plate-formes. Dans ce contexte, le modèle métier s'appuie sur la caractérisation de stéréotypes issus du méta-modèle. Ainsi, le modèle obtenu devra être conforme au méta-modèle et représenter les concepts métiers qui sont manipulés dans l'entreprise. Le contrôle de la conformité du modèle est assuré par un ensemble de contraintes (exemple : tout objet métier doit posséder au moins quelques caractéristiques systèmes comme la référence, la désignation...). A ce niveau, le concepteur doit identifier ses principaux concepts métiers et les liens entre ces concepts. Il doit par ailleurs caractériser les contraintes de conformité en associant chacun de ces concepts aux éléments du méta-modèle CIM. Cette étape consiste à enrichir ou de spécialiser un modèle PIM avec des concepts métiers standards. Par exemple, compléter un modèle avec des objets courants comme des objets documentaires

ou CAO (fig. 3.3). Au final, le niveau PIM doit permettre au concepteur de produire un modèle adapté aux besoins de l'organisation cible.

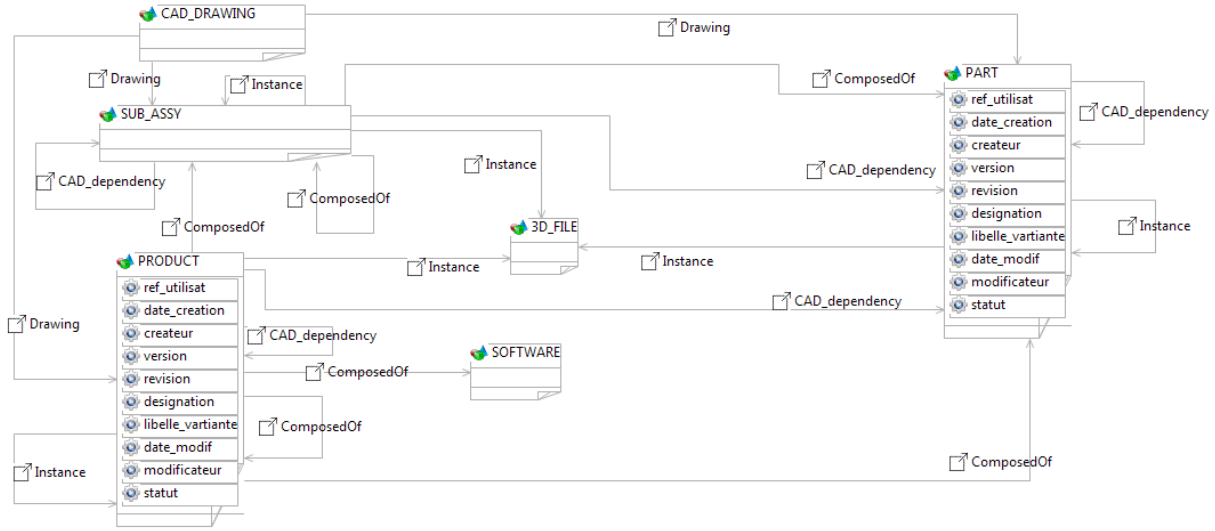

FIGURE 3.3 – Exemple d'un modèle PIM conforme au méta-modèle PLM

Ce modèle est basé sur la standard édition du PLM Audros et a été construit grâce à un framework développé dans le cadre de nos travaux ; nous le présenterons dans le chapitre (cf chapitre 5), les détails étant décrits dans les annexes. Les éléments identifiés par un quadrilatère correspondent à des classes d'objets et les relations entre ces objets correspondent à des classes de liens. Le PIM est un niveau de modélisation qui permet de mettre en œuvre des concepts métiers au sein d'un système PLM. Au niveau PIM, la modélisation n'est pas unique et peut évoluer avec le temps. Les concepts métiers définissant les modèles de ce niveau caractérisent une terminologie métier contextuelle. Ici le sens contextuel implique un faible niveau d'invariance pour deux raisons :

- les concepts manipulés sont très spécifiques d'un secteur d'activité.
- les concepts manipulés sont évolutifs dans le temps.

La suite de la démarche consiste à construire un modèle PIM pour un système PLM correspondant à des besoins spécifiques. Il doit correspondre à des éléments du méta-modèle et respecter toutes les contraintes. La construction du modèle peut être réalisée par un éditeur graphique prévue à cet effet afin qu'il soit plus facile et plus rapide aux concepteurs d'utiliser les concepts du méta-modèle. De plus, un éditeur graphique permet une visualisation plus rapide et plus pratique d'un modèle. Ainsi, la transformation est implicite et est déterminée par les règles du module graphique. La construction du modèle est contrôlée, les règles implicites ne permettent pas de créer des éléments non définis dans le méta-modèle source. En effet, il est seulement possible de créer des classes d'objets représentées par le stéréotype "BusinessObject" du méta-modèle (représenté par un rectangle sur la figure) et des liens entre ces derniers identifiés par le stéréotype "FonctionalLink" (représenté par une flèche orienté sur la figure). Un modèle est donc construit en se basant sur les propriétés du méta-modèle CIM, de ce fait, il est conforme à celui-ci.

Cette démarche ne vise en aucun cas à restreindre la créativité lors de la modélisation mais au contraire à l'inscrire dans un cadre qui garantisse la conformité du modèle PIM avec les standards du PLM, bref à garantir sa cohérence en vue de son implémentation dans un outil PLM identifié.

3.3.2 Caractérisation du niveau PSM

L'analyse du méta-modèle du PLM Audros est une étape importante pour permettre une réelle mise en œuvre au sein de la plateforme élaborée dans cette thèse et dédiée à la gestion de cohérence des objets et des évolutions. Pour cela, nous proposons un méta-modèle spécifique du PLM Audros contenant les concepts manipulés par ce système. En effet, la définition de la plateforme cible (ici le PLM Audros) est nécessaire pour la transformation d'un modèle PIM (indépendant de toute plateforme) vers un modèle PSM (dépendant de la plateforme Audros).

Définition du méta-modèle Audros

Le méta-modèle Audros [Cheballah, 1992] se structure autour de concepts qui seront utilisés dans les trois types de modèles suivants :

- modèle produit : permet de structurer la base de données. Il est constitué d'éléments communs à la base, définition des articles, des documents, des nomenclatures, définition des statuts, définition des thésaurus, définition des natures des liens avec leurs attributs.
- modèle organisationnel : permet la gestion des droits et des vues métiers par groupe de personnes (lors de la définition des droits en lecture/écriture, les droits appliqués par défaut à un utilisateur sont ceux de son groupe).
- modèle processus : permet de mettre en place les processus métiers de l'entreprise en se basant sur le modèle organisationnel et produit afin d'automatiser certaines tâches et de fluidifier l'exécution des processus.

L'analyse du système d'information du PLM Audros a permis de caractériser un méta-modèle (fig. 3.4) avec des concepts très proches de sa base de données que nous avons définie comme niveau d'exécution.

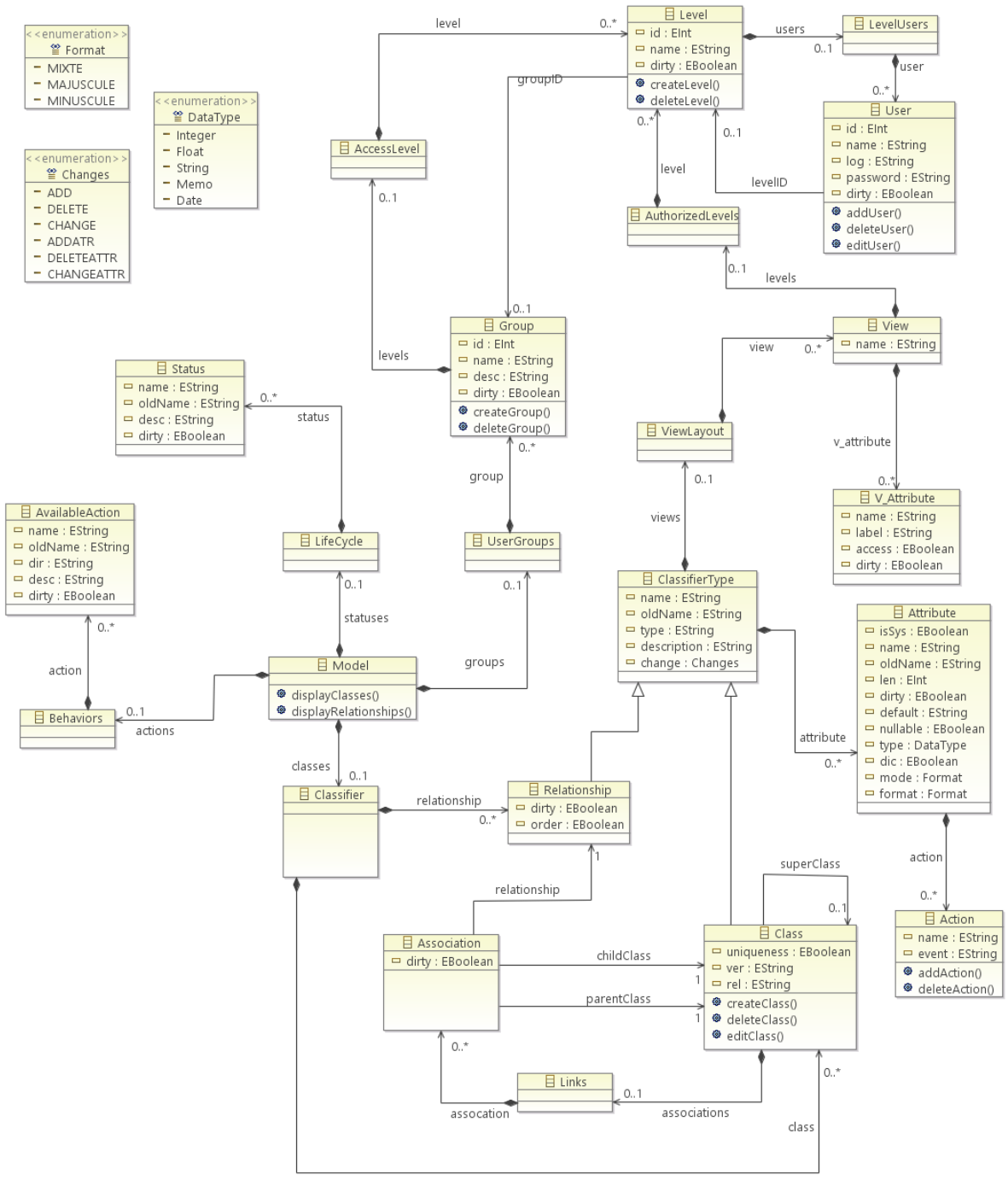

FIGURE 3.4 – Méta-modèle pour le système Audros

Du PSM vers le niveau d'exécution PLM

Rappelons ici que dans la notion initiale de MDA, le niveau le plus bas est un code exécutable dans un système d'exploitation. Dans notre démarche, nous fixons le niveau d'exécution au niveau système PLM cible. Comme le montre la figure suivante (fig. 3.5), l'exécution des modèles au sein du système Audros est définie par un ensemble d'éléments

que nous définissons par la suite. Dans la figure, le PDM (Plateform Definition Model) permet d'avoir une définition du système cible (Audros dans notre cas). Cette définition est impérative afin de rendre un modèle dépendant d'une plateforme. En effet, un modèle PIM est indépendant de toute plateforme ce qui le rend inutilisable. De ce fait, c'est la transformation du modèle PIM avec la définition de la plateforme cible qui génère un modèle dépendant de la plateforme, le PSM.

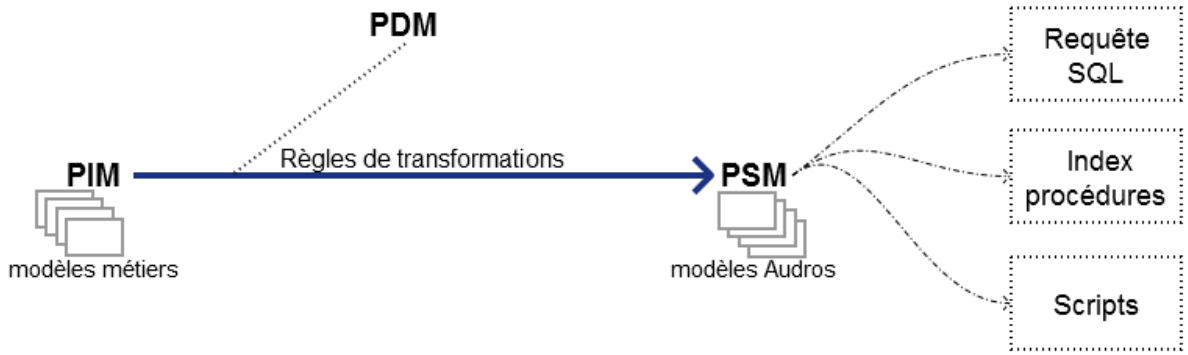

FIGURE 3.5 – Principe des transformations PIM -> PSM dans le cadre du PLM Audros

Dans notre démarche, l'objectif de la transformation d'un modèle PSM vers le niveau d'exécution du PLM consiste à obtenir des éléments exécutables au sein du système considéré. La notion "d'exécutable" que nous proposons n'est pas celle initialement définie par MDA mais c'est une solution que nous pensons compatible. Elle est caractérisée par des éléments composant le système d'information du PLM : des requêtes SQL (insertions, suppressions ou modifications selon les opérations réalisées dans le modèle PSM), création ou de mise à jour d'index et de procédures cataloguées dans la base de données (pour les automatisations), et la création ou de mise à jour de scripts dans le langage de programmation du PLM cible. Ces scripts sont généralement utilisés pour des actions d'automatisation, c'est-à-dire d'accélérer la production (codification, génération de fichier...), l'interfaçage avec des outils tiers (ERP, CRM..) ou de contrôle (vérification, restriction...).

Nous caractérisons donc le niveau d'exécution du système PLM par :
- un ensemble de fonctions sur les caractéristiques structurelles de la base de données (requêtes, index, procédures).
- un ensemble de fonctions sur les caractéristiques comportementales du système (scripts...).

L'implémentation du premier ensemble (requêtes, Index, procédures stockées) nécessite une analyse de la structuration du système cible [Cheballah, 1992]. Cette analyse nous à permis d'en savoir plus sur le fonctionnement du modèle de données du PLM Audros. En effet, le SI du PLM Audros comporte deux types de tables dans sa base de données :
- les tables systèmes
- les tables spécifiques

Les tables systèmes ne sont pas modifiables par les concepteurs PLM. La présence de la totalité de ces tables est impérative pour que l'outil puisse fonctionner correctement (sans

incohérence majeure). Elles sont destinées à contenir des informations pour le bon fonctionnement du PLM et définissent les caractéristiques minimalistes de l'outil. Les tables spécifiques sont créées (modifiées et/ou supprimées) par des concepteurs (administrateur du système) et représentent le modèle métier de l'entreprise. Les concepteurs ont plus de liberté sur la création de ces tables dans le cas d'une solution "From Scratch" ou limité dans le cas d'une solution "Packagée". Ces tables représentent les différents concepts métiers destinés à être utilisés dans l'outil PLM. Elles sont instanciées de deux manières différentes :

- dans des tables préfixées par dt_NomObjet pour les classes d'objets.
- dans des tables préfixées par ds_NomLien pour les classes de liens.

Ainsi donc, chaque concept créé par le concepteur dans le modèle engendre la création d'une table dt_NomObjet et/ou d'une table ds_Nomlien dans la base de données du PLM Audros, c'est la personnalisation de l'outil. Par exemple : La création du concept de PART et du lien ComposedOf dans le modèle métier créera une table dt_part et une autre ds_composedof dans la base de données du PLM. Avant la création des tables spécifiques, les informations des classes (d'objets ou de liens) sont d'abord insérées dans les tables systèmes (fig. 3.6). Un succès permet alors la génération des tables spécifiques dt_ou ds_ sinon aucune des tables n'est créée.

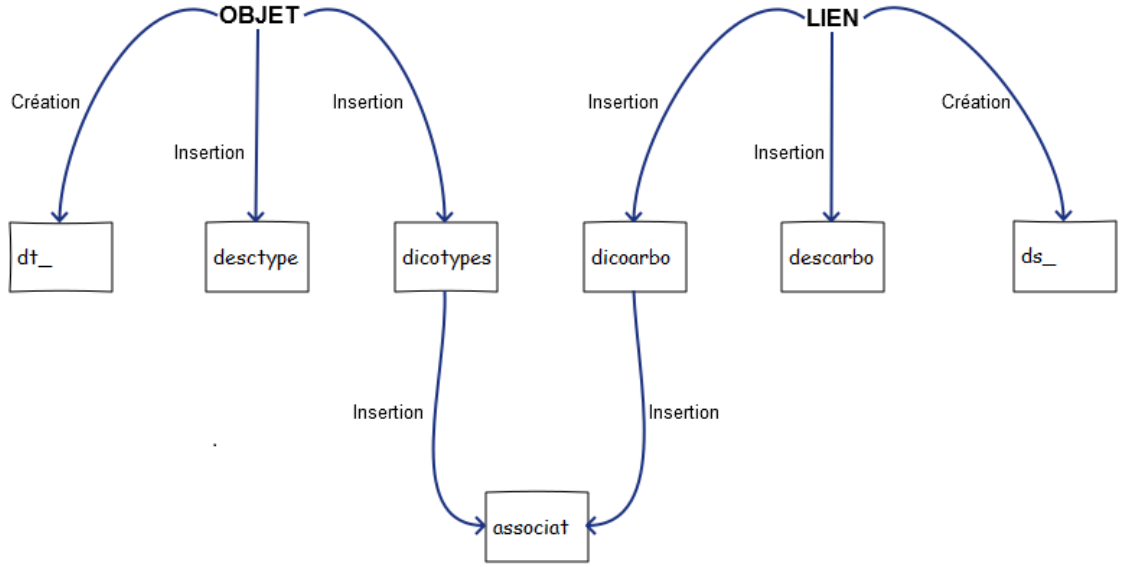

FIGURE 3.6 – Fonctionnement de la BD du PLM Audros (création d'objet et de lien)

Une description des différentes tables systèmes présentées dans le schéma précédent (fig. 3.6) comporte :

desctype : description de tous les attributs de la classe d'objets (nom attribut, type, taille, attribut hérité...).

descarbo : description de tous les attributs de la classe de liens (nom attribut, type, taille...).

dicotypes : description de la classe d'objets (unicité, version et révision initiale, autorisation d'attachement d'un fichier, gestion de l'historique, statut initial...).

dicoarbo : description de la classe de liens (ordonnancement, commentaire...).

associat : liste les associations entre les objets par des liens (type lien, père, fils...).

Par exemple, la création du modèle suivant (fig. 3.7) dans le système d'information du PLM Audros provoque l'enchaînement des actions suivantes :

- génération des requêtes SQL de création des tables dt_ et ds_ correspondants au modèle métier,
- insertion des attributs (spécifiques et systèmes) des objets dans la table desctype,
- insertion de la description (statut initial, première version/révision...) des objets dans la table dicotypes,
- création des tables dt_PART et dt_PRODUCT,
- création des procédures cataloguées concernant les objets en création, modification et suppression,
- insertion des attributs (spécifique et système) du lien dans la table descarbo,
- insertion de la description (type de lien...) du lien dans la table dicoarbo,
- création de la table ds_ComposedOf,
- insertion de la définition (objet père, objet fils...) du lien dans la table associat.

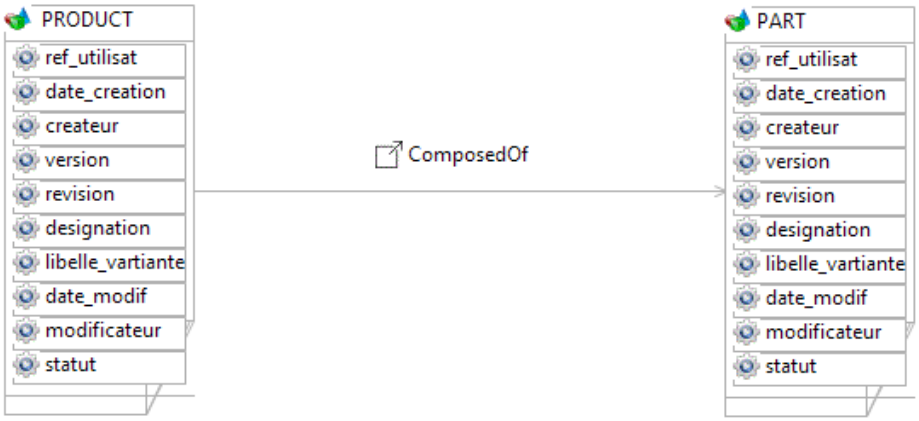

FIGURE 3.7 – Un exemple du fonctionnement du système d'information du PLM Audros

L'ordre des opérations est très important car elle permet de garder une cohérence au sein du système informatique. En effet, l'échec d'une étape entraîne la suppression ou l'annulation des événements en corrélation avec cette étape. Le fonctionnement avec les liens est identique à celui des objets, les seules différences sont situées au niveau des tables systèmes.

L'implémentation du second ensemble (caractéristiques comportementales du système) se réalise à partir d'un macro langage interprété AUPL (Audros Programming Language). Les commandes de ce macro langage sont regroupées au sein d'un script qui peut être associé à un évènement sur une action élémentaire :

- création, modification, suppression d'un objet
- modification de la version, révision ou du statut d'un objet

- création, modification, suppression d'un lien
- création, modification, suppression d'un attribut
- création, modification, suppression sur les attachements
- ouverture, fermeture de l'application

De plus, un objet peut engendrer la création de scripts dans le système cible. Dans notre exemple, une règle métier (cf chapitre 4) sur l'objet "PRODUCT" permettra de générer un script de codification automatique. En effet, la règle métier contient la codification à appliquer pour les instances d'un objet "PRODUCT". Celle-ci est traitée par les mécanismes de transformation (cf chapitre 4) et produit en sortie un script. Le déclenchement du script est prévu lors de la création d'une instance d'un "PRODUCT" dans l'outil PLM.

3.4 Conclusion

Dans notre contexte, nous avons adapté la démarche MDA sur plusieurs aspects. Tout d'abord, au niveau du CIM, nous avons introduit des concepts liés au PLM même si en théorie les concepts sont indépendants de leur implémentation. Cela permet de définir une abstraction suffisamment dense pour permettre un réel contrôle de conformité au niveau des modèles PIM. De plus, au niveau du PSM, le niveau d'exécution cible se positionne au sein d'un système PLM et non au sein du code exécutable dans un système d'exploitation. Ainsi, l'idée générale est d'être capable de gérer des modèles hétérogènes PIM, indépendamment de toute plateforme mais conformes aux concepts du PLM et de passer par des transformations à des modèles PSM plus spécifiques dépendant de la plateforme sélectionnée.

Chapitre 4

Modélisation sous contraintes et transformations

Sommaire

4.1 Introduction

Dans le chapitre précédent, nous avons établi les éléments permettant de modéliser un contexte métier et sa réalisation au sein d'un système PLM. Ces éléments de modélisation sont définis à partir des méta-modèles de niveau CIM (ou du MOF). Ils permettent de construire une représentation cohérente de modèles sans pour autant permettre de spécifier l'ensemble des règles métier devant être mises en place dans le PLM. Ce chapitre aborde donc ce point en caractérisant des contraintes métier et les différents mécanismes de transformation. Dans la première partie de ce chapitre, nous présentons une structure de contraintes cohérente pour notre démarche MDA. Puis dans la seconde partie, nous présentons les différents mécanismes de mise en œuvre des modèles métier par des règles de transformation (PIM vers PSM et PSM vers du code exécutable).

4.2 Contraintes sur les modèles et méta-modèles

La notion de contrainte a pour objectif de créer un cadre restrictif afin de faciliter le travail des experts dans la conception des modèles. Une restriction sous forme de contraintes, telle que nous la proposons, permet donc de définir des règles spécifiques qui ne peuvent être représentées par les capacités explicites d'UML. En effet, les restrictions standards d'UML sont liées aux relations entre les classes et à leurs multiplicités (cardinalités). Cette logique a du sens dans la mesure où la complexité se gère directement dans le contexte comportemental des classes (méthodes de classes, diagrammes d'activités et de séquences).

4.2.1 Une typologie des contraintes

Dans la mesure où nous avons plusieurs niveaux de modélisation différents (CIM, PIM, PSM), il convient de distinguer les types de contraintes. Pour autant, nous pensons que la présentation par niveaux n'aide pas le concepteur, notamment lorsque les acteurs sont différents (experts SI et experts métiers). Aussi, nous proposons une typologie des contraintes par usage, plutôt que par niveaux :

- Les contraintes de conformité : elles doivent garantir la conformité d'un modèle métier et d'un modèle PLM vis à vis de leurs concepts respectifs.
- Les contraintes de support : ces contraintes sont spécifiques à la mise en œuvre d'un système PLM. Ce sont des contraintes sur les concepts intrinsèques au PLM (unicité, traçabilité...).
- Les contraintes métier : ces contraintes permettent d'exprimer sur les modèles des règles métier sur les classes ou les relations. Elles sont définies par les experts métier.

Les règles explicites catégorisées sont définies sur les différents niveaux de modélisation par le langage OCL [Warmer, 2003] [Cabot and Teniente, 2007]. La figure (fig. 4.1) montre les points d'application de ces contraintes dans la démarche MDA [Kleppe et al., 2003].

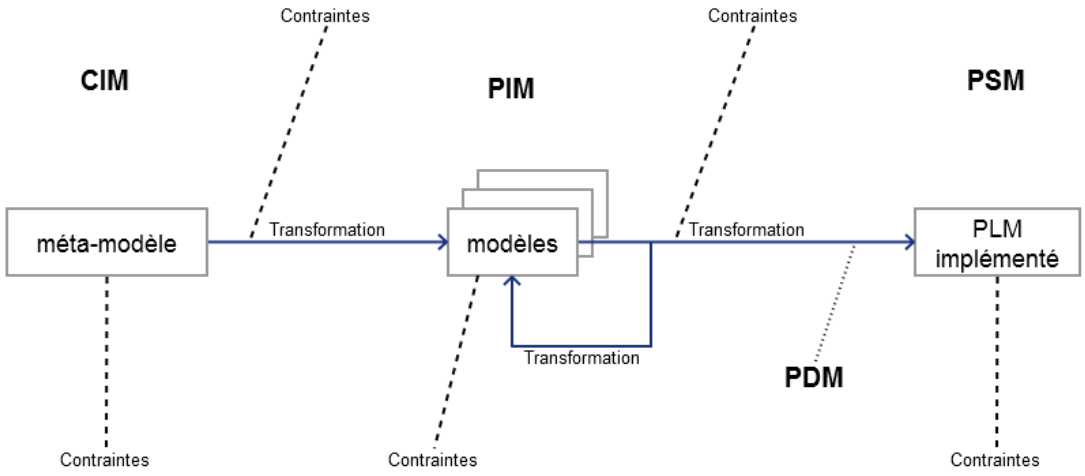

FIGURE 4.1 – Positionnement des contraintes sur les niveaux de modélisation

Cette figure illustre le fonctionnement des contraintes dans notre démarche MDA. Les contraintes sont des éléments éditables par les différents experts, à l'exclusion des contraintes définies sur le méta-modèle. En effet, celles-ci ne sont modifiables que par les administrateurs de haut niveau, car elles identifient des concepts de conformité générique pour les systèmes PLM. Une modification de ces contraintes nécessite une régénération du module graphique pour la construction des modèles PIM. La présence des contraintes sur plusieurs niveaux enrichit le fonctionnement d'un framework utilisant la démarche MDA (cf chapitre 5). Effectivement, cette manière de fonctionner donne la possibilité aux experts d'ajouter leurs propres contraintes sur les niveaux PIM et/ou PSM. De plus, des contraintes peuvent être ajoutées sur les règles de transformation pour compléter les contraintes définies sur les modèles. Elles peuvent également être utilisées dans les transformations pour la création de scripts de contrôle ou de codification.

4.2.2 Les contraintes de conformité

Les contraintes de conformité sont des contraintes définies au niveau des concepts CIM. Elles doivent permettre de compléter, si besoin, les contraintes implicites (relations, multiplicités) de la structure du méta-modèle. Elles sont applicables pour tous les modèles qui seront créés. En effet, étant donné qu'une contrainte de conformité est définie au niveau du méta-modèle, l'ensemble de ces contraintes doit être respecté lors de la validation d'un modèle conforme à ce méta-modèle. Elles représentent en quelque sorte les contraintes générales, c'est-à-dire des contraintes qui assurent une cohérence logique dans la mise en production des modèles, indépendamment des métiers. Nous pouvons par exemple citer quelques exemples de contraintes de conformité (tab. 4.2.2) :

Titre	Description
C1 : Contrainte sur le modèle	Il est impossible d'avoir deux BusinessObject ayant le même nom sur un même modèle. En effet, chaque BusinessObject est identifié par son nom dans un modèle.
C2 : Contrainte sur les relations	Les FunctionalLink doivent impérativement avoir un élément (BusinessObject) père et un élément fils.

Table 4.1 – Exemple de contraintes de conformité

Ces exemples de contraintes de conformité sont implémentés avec le langage OCL :

```
context Model
inv BO_uniqueness : self.ListObject->select(b1, b2 : BusinessObject |
b1.name_object <> b2.name_object)
```

Listing 4.1 – C1 : Exemple d'unicité pour le nom des BusinessObject dans un modèle

```
context FonctionnalLink
inv FL_end : self.father.name_object -> notEmpty()
and self.son.name_object -> notEmpty()
```

Listing 4.2 – C2 : Vérification des extrémités des FunctionalLink

4.2.3 Les contraintes de support PLM

Les contraintes de support PLM sont établies par l'expert SI. Elles permettent de définir un ensemble de règles qui caractérise le fonctionnement global du PLM. L'identification des contraintes de support (tab. 4.2.3) permet de rendre les modèles métier transformables vers un modèle PSM. On peut distinguer deux sous-types de contraintes de support :

- les contraintes de support globales. Elles permettent de définir des règles applicables indépendamment de toute plateforme, elles s'appliquent au niveau PIM.
- les contraintes de support locales qui sont spécifiquement liées à une plateforme PLM.

Type	Description
C3 : Contrainte globale	Le lien de Composition (de type 'BOM') permet de relier des objets CAO. Les extrémités ne contiennent que ('père' avec 'fils') : (ASM,PRT), (ASM,ASM), (ASMPRT,ASMPRT), (ASM,ASMPRT) (ASMPRT,ASM) et (ASMPRT,PRT).
C4 : Contrainte globale	Le lien MiseEnPlan permet de décrire les modèles référencés par le Plan CAO (de type 'DOC') et les extrémités ne contiennent que : (DRW avec ASM) (DRW avec PRT) (DRW avec ASMPRT)(DRW avec PLT).
C5 : Contrainte locale	Chaque "BusinessObject" doit avoir exactement deux attributs système ("SystemAttr"), la référence et la désignation.

TABLE 4.2 – Exemple de contraintes de support

Dans le tableau suivant, nous présentons quelques extraits de ces contraintes pour un modèle d'une PME (dans le cadre du PLM Audros). Ces exemples descriptifs de contraintes peuvent être modélisés avec OCL.

```
context FonctionnalLink
inv bom_link : self.link_type = 'BOM'
and (self.father.type = 'ASM' or self.father.type='ASMPRT')
and (self.son.type = 'ASM' or self.son.type='PRT'
```

```
  or self.son.type='ASMPRT')
```

Listing 4.3 – C3 : Exemple d'une contrainte OCL : le lien Composition

```
context FonctionnalLink
inv draw_link : self.link_type = 'DOC'
and self.father.type = 'DRW'
and (self.son.type = 'ASM'
     or self.son.type='PRT'
     or self.son.type='ASMPRT'
     or self.son.type = 'PLT')
```

Listing 4.4 – C4 : Exemple d'une contrainte OCL : le lien MiseEnPlan

```
context BusinessObject
inv test_syst_attr : size(self.attrSystem) = 2
```

Listing 4.5 – C5 : Exemple d'une contrainte OCL : les attributs systèmes

4.2.4 Les contraintes métier

Afin de rendre la génération des modèles PIM plus consistante et plus cohérente au regard du fonctionnement des systèmes PLM, des contraintes ont été ajoutées dans les éléments du méta-modèle CIM. Ils permettent de caractériser les règles de vérification à chaque niveau de modélisation. Certaines contraintes sont implicites. Par exemple, les relations (composition, agrégation...) définies sur le méta-modèle CIM impliquent d'être vérifiées dans les modèles PIM afin de garantir la conformité. Comme nous le verrons dans le chapitre suivant, les contraintes que nous proposons se positionnent sur l'ensemble des niveaux de modélisation (CIM, PIM, PSM).

Les contraintes métier ont naturellement vocation à être mises en œuvre dans la plateforme PLM cible. Dans ce cadre, nous identifions deux types de contraintes métier :

- les contraintes métier structurelles. Elles sont définies sur les modèles métier PIM. Elles complètent les mécanismes implicites et permettent de valider la structure métier du modèle.
- les contraintes métier dynamiques. Elles sont définies sur les modèles métier mais elles doivent être validées au niveau des instances des objets.

4.3 Validation des modèles par les contraintes

Une validation de modèles consiste à savoir si l'ensemble du modèle est conforme par rapport à l'ensemble des contraintes implicites et explicites. Cette validation est une étape clé dans une approche d'architecture dirigée par les modèles. En effet, l'utilisation d'un modèle dans le SI cible dépend fortement de sa validation. Ainsi, un modèle conforme à son méta-modèle peut être considéré comme cohérent et sans ambiguïté. Au contraire, un

modèle non conforme peut comporter des ambiguïtés et de ce fait engendrer des problèmes de cohérence dans le SI cible. Nous définissons les notions suivantes :

Définition d'une contrainte valide : une contrainte est dite valide si l'expression du prédicat associé à la contrainte est vraie.

Définition d'un modèle conforme : un modèle est considéré comme conforme si toutes les contraintes de conformité associées aux éléments de ce modèle sont valides.

Définition d'un modèle cohérent : un modèle est considéré comme cohérent si toutes les contraintes métiers associées aux éléments de ce modèle sont valides.

Définition d'un modèle déployable : un modèle est considéré comme déployable si toutes les contraintes de support associées aux éléments de ce modèle sont valides.

Définition d'un modèle valide un modèle est considéré comme valide s'il est conforme, cohérent et déployable.

Le langage OCL a vocation de permettre de faciliter la mise en place de mécanismes de vérification automatique. Ainsi, dans les modèles, nous pouvons valider la syntaxe d'une ou plusieurs contraintes définies en OCL. Ainsi, il sera possible de connaître si un modèle est conforme, cohérent et déployable. Cela nécessite de réaliser une demande de validation de l'ensemble du modèle. Par exemple, un modèle M1 conforme au méta-modèle MM1 ne pourra être validé automatiquement seulement si toutes les contraintes sur le méta-modèle MM1 et le modèle M1 sont respectées. Dans le cas contraire, le modèle est considéré comme invalide, les messages d'erreurs (contraintes non validées) permettront d'identifier les concepts invalidés par l'étape de validation. Dans l'absolu, un modèle invalide correspond à un modèle inutilisable car son utilisation pourra engendrer des incohérences au sein du SI cible. Cependant, la réalité industrielle tolère des exceptions. En effet, malgré les problèmes pouvant être causés par la mise en production d'un modèle invalide, certaines entreprises pourront être bloquées par les diverses corrections à apporter (temps de traitement, blocage de l'ensemble du modèle pour une contrainte affectant seulement une petite partie...). De ce fait, nous avons identifié plusieurs cas d'utilisation afin d'éviter de bloquer l'ensemble du processus de transformation :

Cas 1 : si un modèle n'est pas déployable alors les étapes de transformation successives seront bloquées. En effet, une contrainte de support PLM est trop essentielle pour le fonctionnement global du PLM. De ce fait, afin de garder une logique totale au sein du SI, le modèle nécessite d'être déployable.

Cas 2 : de la même façon, si un modèle n'est pas conforme alors les étapes de transformation successives seront bloquées. Effectivement, tout comme les contraintes de support, la validation des contraintes de conformité est indispensable pour garder la logique au sein du système PLM.

Cas 3 : si un modèle est conforme et déployable mais non cohérent, alors c'est à l'expert métier de décider si oui ou non il souhaite poursuivre. Les contraintes métier sont créées et modifiées par cet expert, et c'est donc à lui que revient la responsabilité de considérer le modèle comme valide même s'il n'est pas cohérent. Ce fonctionnement permet de définir des priorités et d'éviter de bloquer un modèle à cause de contraintes métier qui ne sont pas primordiales dans le fonctionnement de l'application.

Ces cas d'utilisation permettent de définir le champ d'application des contraintes et leur importance vis-à-vis du modèle à implémenter. Le blocage des étapes de transformation dans le cas d'un modèle non déployable ou non conforme est une sécurité afin de garantir la logique globale du système d'information cible. En effet, cette sécurité empêche la mise en production d'un modèle "défaillant" pour ne pas exposer les utilisateurs à créer des incohérences successives ayant comme origine une contrainte non respectée lors de la modélisation.

Si un modèle est valide, c'est qu'il est conforme, cohérent et déployable. La suite logique est de déclencher les transformations qui vont permettre de générer le code exécutable pour l'application cible.

4.4 Transformation de modèles

Les modèles créés doivent être capables de répondre aux questions que l'on se pose sur le système modélisé, de la même manière que le système aurait répondu lui-même [Minsky, 1969]. MDA est une démarche qui fait intervenir des séquences de transformations de modèles. Chaque transformation prenant un modèle en entrée (voir deux dans le cas d'une comparaison de modèle) et produisant un modèle en sortie, jusqu'à l'obtention du code exécutable de façon automatique [Kleppe et al., 2003]. Il est nécessaire pour cela de choisir un outil puis un langage de transformation et de caractériser les transformations comme des activités particulières, dont les produits en entrée et en sortie se doivent d'être des modèles conformes à un même méta-modèle (transformation endogène) ou à plusieurs méta-modèles (transformation exogène).

4.4.1 Transformation PIM vers PSM

Les modèles métier identifiés dans le niveau PIM doivent finalement être implémentés dans une plateforme PLM. Cette étape clé de la démarche MDA consiste à créer des modèles PSM spécifiques à une plateforme grâce à des transformations de modèles PIM indépendants de toutes plateformes. La spécification d'un modèle PIM à une plateforme est réalisée grâce à des informations techniques de la plateforme d'exécution définies dans la transformation. En effet, toutes les spécifications techniques de la plateforme d'exécution doivent être intégrées dans les processus de transformation. Les informations de la plateforme cible sont définies dans un modèle de description de la plateforme, le PDM (Plateforme Définition Model). Cette démarche peut être illustrée selon un processus en Y [Roques and Vallée, 2000] visible dans le schéma (fig. 4.2). Le PSM obtenu en sortie de transformation est spécifique de la plateforme d'exécution. Le PSM est le niveau qui

fournit les informations utiles à la génération de code de l'application. C'est justement à partir de ces modèles que sera généré le code source.

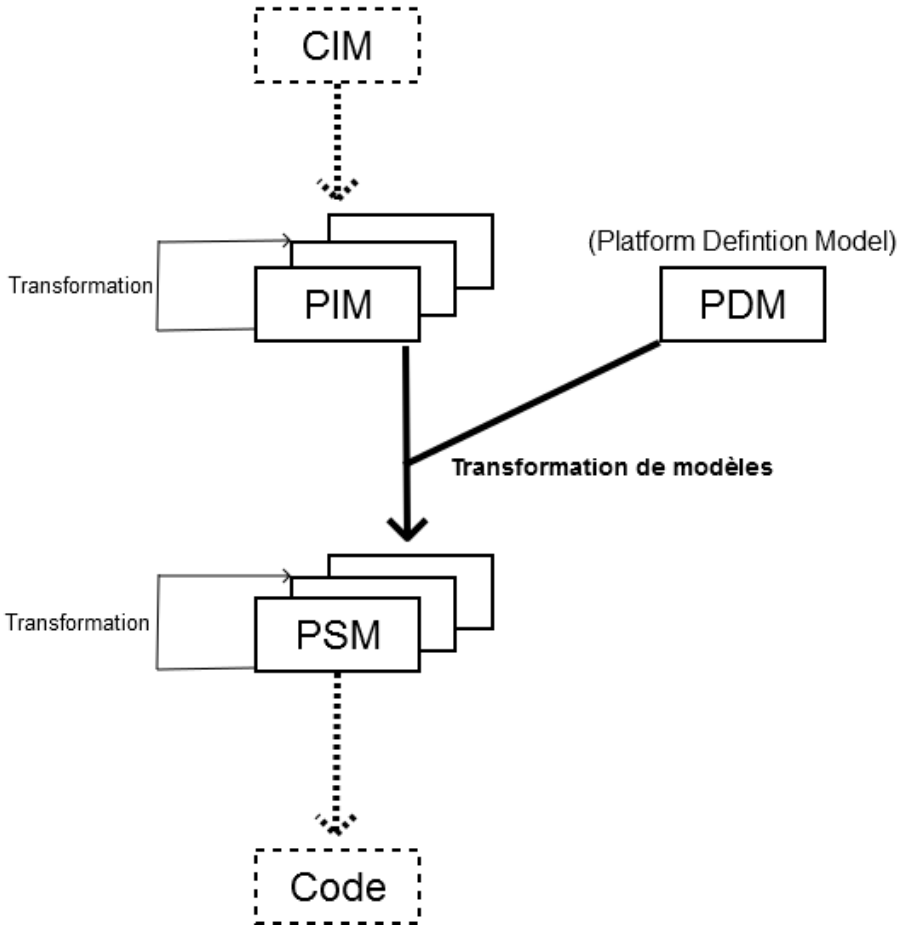

FIGURE 4.2 – Processus en Y de la démarche MDA

Le procédé d'intégrer le modèle de la plateforme cible dans la transformation permet de faciliter le développement d'applications multi-plateformes et de migration de logiciel. La transformation d'un modèle métier conforme au méta-modèle PLM vers un modèle Audros consiste à mapper chaque concept métier à son correspondant de la BD Audros. Cette transformation se passe au niveau M1 de l'approche MDA : c'est à dire la transformation d'un PIM vers un PSM (avec deux méta-modèles). Cependant, les règles de transformation (basées sur un modèle de transformation) sont définies entre les méta-modèles des modèles concernés : PLM et Audros. Dans la suite, nous présentons ces règles de transformation (Mapping entre les différents concepts).

Afin de disposer d'un prototype fonctionnel, nous avons décidé de ne prendre en considération seulement qu'une partie des méta-modèles pour le mapping. Les règles suivantes représentent donc une partie du module de transformation :

- Model (PLM) -> Model (Audros)
- BusinessObject -> Class

- FunctionnalLink -> Relationship
- Attribute, AttrSystem, AttrSpe -> Attribute
- BusinessView -> View
- OrganisationalUnit -> UserGroups
- LifeCycle (PLM) -> LifeCycle (Audros)

Les figures suivantes (fig. 4.3) et (fig. 4.4) mettent en évidence les correspondances entre les différents concepts des deux méta-modèles (représentation graphique). Les concepts ayant la même couleur et portant le même numéro sont mappés entre eux.

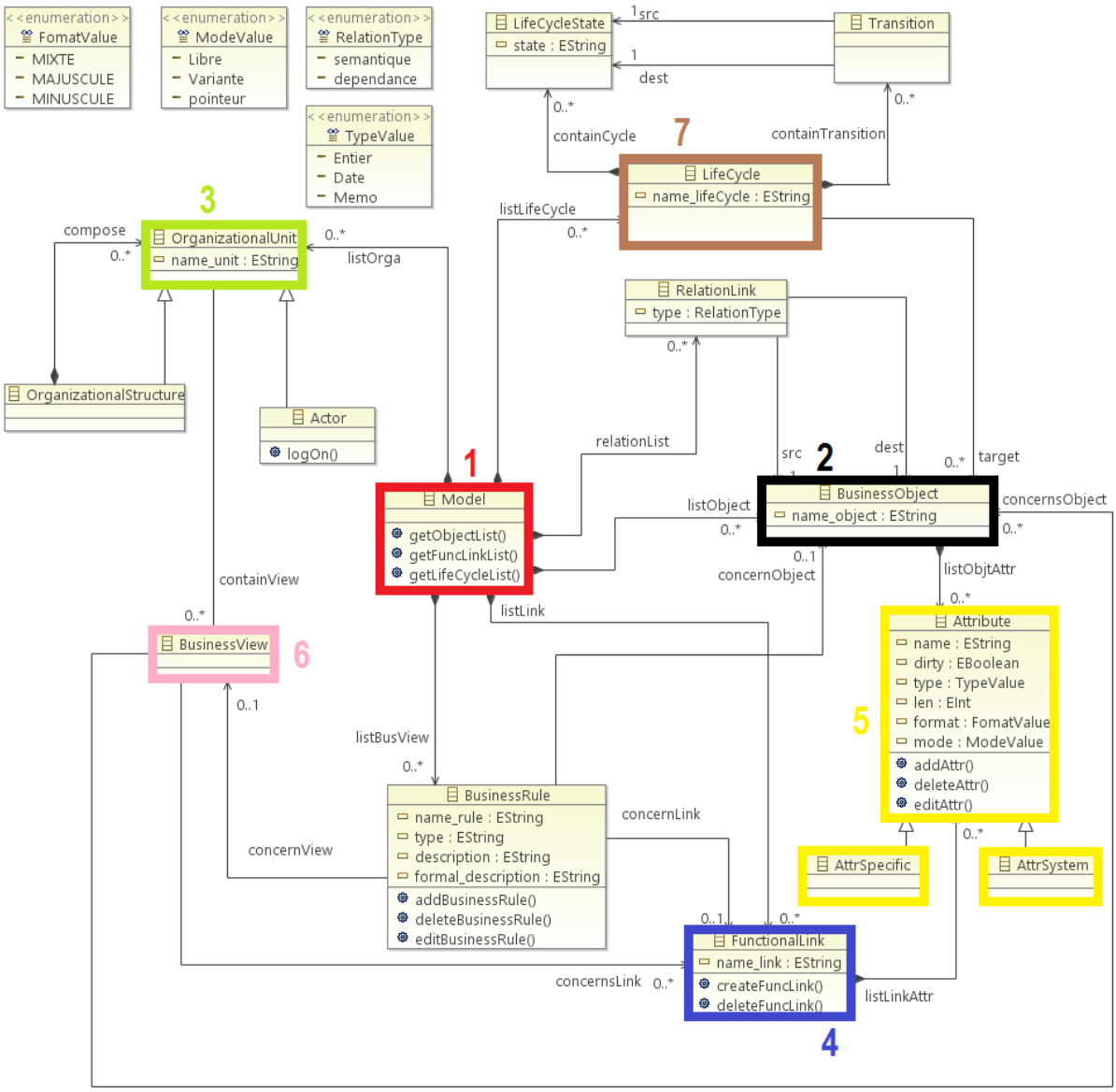

FIGURE 4.3 – Représentation graphique des correspondances entre les méta-modèles (partie 1)

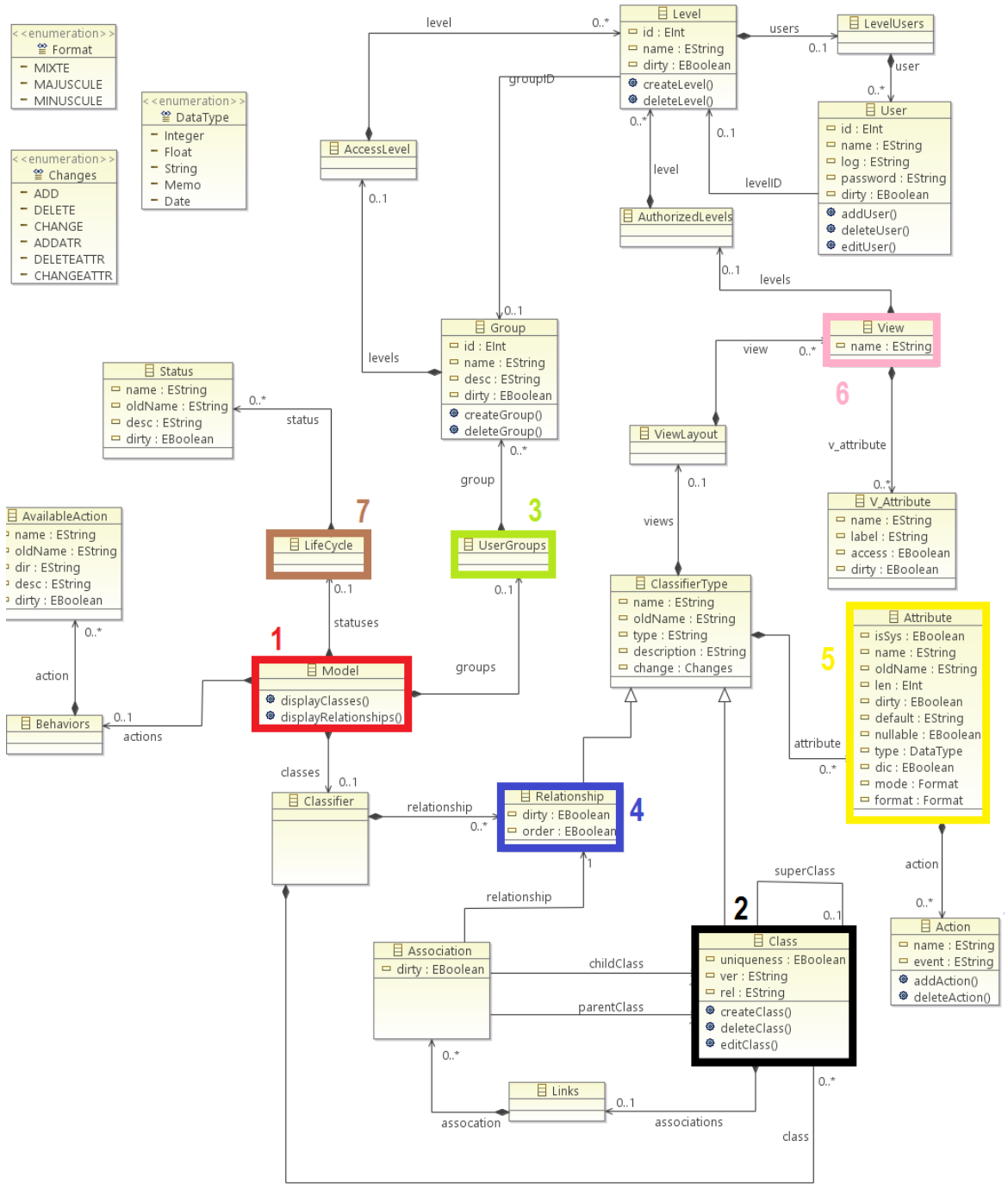

FIGURE 4.4 – Représentation graphique des correspondances entre les méta-modèles (partie 2)

Les règles de transformation d'un modèle métier PLM vers un modèle Audros ont été implémentées à l'aide du framework ATL [Jouault and Kurtev, 2005] [Jouault et al., 2008]. Chaque règle est traduite par une "Matched rule" ou une "lazy rule" au niveau du module ATL. Le module de transformation réalisé supporte la transformation des éléments BusinessObject, FunctionalLink, Attribute, AttrSystem, AttrSpecific et Model du méta-

modèle PLM vers leur correspondance dans le méta-modèle Audros. Ces transformations avec le langage ATL sont définies dans un fichier .atl, voici quelques extraits de code provenant de ce fichier :

Le premier exemple présente l'entête d'un fichier ATL. La transformation "business2db" prend en entrée un modèle métier (conforme au méta-modèle PLM) pour en sortir un modèle BD (conforme au méta-modèle Audros) :

```
-- @path Business=http://acms/1.0
-- @path DB=http://dbmetamodel/1.0

module business2db;
create OUT : DB from IN Business;
```

Listing 4.6 – Entête d'un fichier de transformation ATL

Une "lazy rule" pour la transformation d'un BusinessObject du modèle PLM en une Class du modèle BD.

```
lazy rule BusinessObject2Class {
  from
    b : Business!BusinessObject
  to
    c : DB!Class (
      name <- b.name_object,
      attribute <- b.ListObjAttr->collect(attr | thisModule.BusinessAttribute2Attribute(
          attr))
    )
}
```

Listing 4.7 – Exemple d'une "lazy rule" avec ATL

Une "lazy rule" pour la transformation d'un FunctionalLink du modèle PLM en Relationship du modèle BD.

```
lazy rule BusinessRelationship2Relationship {
  from
    br : Business!FunctionalLink
  to
    r : DB!Relationship (
      name <- br.name_link,
      attribute <- br.ListLinkAttr->collect(attr | thisModulz.BusinessAttribute2Attribute(
          attr))
    )
}
```

Listing 4.8 – Exemple d'une "lazy rule" avec ATL (2)

Une "lazy rule" pour la transformation d'un Attribute du modèle PLM en Attribute du modèle BD.

```
lazy rule BusinessAttribute2Attribute {
```

```
  from
    ba : Business!Attribute
  to
    a : DB!Attribute (
      name <- ba.Nom,
      format <- ba.Format,
      len <- ba.longueur,
      type <- ba.Type,
      mode <- ba.Mode,
      nullable <- ba.Obl
    )
}
```

Listing 4.9 – Exemple d'une "lazy rule" avec ATL (3)

La règle ("matched rule") ci-dessous présente la transformation de la racine du modèle, c'est à dire la transformation d'un Model métier en un Model base de données :

```
rule businessModel2DBModel {
  from
    bm : Business!Model
  to
    bdm : DB!Model (
      classes <- classifier
    ),
     classifier :DB!Classifier (
      class <- bm.ListObjects->collect(object | thisModule.BusinessObject2Class(object)),
      relationship <- bm.ListLinks->collect(relation | thisModule.
          BusinessRelationship2Relationship(relation))
    )
}
```

Listing 4.10 – Exemple d'une "Matching rule" avec ATL

4.4.2 Transformation PSM vers PLM

La transformation d'un modèle PSM vers du code exécutable pour le PLM utilise des générateurs de code qui permettent de produire le code dans un langage de programmation. En effet, la compilation du code dans le produit cible (PLM) rendra possible l'utilisation du modèle par les utilisateurs dans le PLM. Dans notre cas, il s'agit d'obtenir du code SQL pour créer la définition du modèle dans la base de données du PLM Audros. Le générateur de code prend en entrée le modèle pour produire en sortie du texte correspondant au modèle dans le produit cible. Cela consiste à prendre en entrée un modèle PSM conforme au méta-modèle Audros créé grâce à l'atelier ACMS (cf chapitre 5) afin de générer en sortie du texte (en XML dans notre cas, préféré au langage SQL direct). Cette préférence pour un format neutre nous avantage dans la suite. En effet, ce choix nous permet de gérer différents types de bases de données grâce à des outils tiers qui se chargent de transformer les données XML en requête exécutable dans la BD cible. Le

fichier XML obtenu en sortie de la transformation correspond aux différentes informations à insérer dans la base de données cible (du PLM). De plus, ce fichier XML peut identifier les changements qui ont été effectués sur un modèle déjà existant via la fonctionnalité de comparaison de modèles. En effet, plusieurs outils open source sont capables d'identifier et d'appliquer les modifications intervenues sur un modèle (cf chapitre 5). Ensuite, il suffit d'analyser le fichier XML de sortie afin d'insérer, de modifier ou de supprimer les données dans la base de données Audros par l'intermédiaire d'autres outils ou de scripts automatiques. L'avantage d'utiliser un outil pour ce type d'opération est de faire abstraction de la BD cible, c'est à dire d'appliquer les modifications quelle que soit la base de données cible (Oracle, SQL-Server...).

4.5 Transformation inverse et comparaison

La transformation des modèles PSM en code exécutable consiste à générer le code source de l'application, de façon totale ou partielle, à partir des modèles PSM de l'application. Cette étape n'est pas à proprement dit considérée comme une transformation par MDA. En effet, une transformation selon MDA est définie par la transformation d'un modèle vers un autre modèle, chacun d'eux étant structuré par son méta-modèle. Or le code source n'a pas de méta-modèle, la transformation des PSM vers le code est plutôt considérée comme une retranscription textuelle du modèle PSM. L'objectif d'une transformation inverse est de permettre à des applications déjà existantes d'être intégrées dans une démarche MDA.

Prenons l'exemple d'une entreprise disposant d'un outil PLM. Après une certaine période d'utilisation et de modification au sein de son modèle de données, il peut être utile de vérifier sa cohérence globale. Il est important de disposer de mécanismes pouvant intégrer le modèle de données de cette entreprise dans notre démarche. La transformation inverse doit être capable d'analyser le système d'information de cette entreprise afin de la transcrire dans un modèle PIM. C'est une opportunité pour des entreprises n'ayant pas pu profiter de cette vérification lors de la première implémentation du produit.

Dans notre cas, la démarche de transformation inverse est plus complexe. En effet, dans le cas des systèmes PLM, les entreprises modifient la structure de données ou ajoutent de nouveaux scripts sans passer par une étape formalisée de modélisation : cette démarche s'apparente à une intervention sur un réseau électrique ou de plomberie sans en consulter les plans ! Cette démarche est assez courante mais pas exempte de risques comme nous l'avons décrit dans le premier chapitre de cette thèse. En effet, modifier ou ajouter des scripts sans passer par une étape de validation peut engendrer divers problèmes de cohérence sans connaître l'ensemble des impacts au niveau du SI. Ainsi donc, l'objectif des transformations inverses est de permettre une abstraction des réalisations effectives afin de vérifier qu'elles ne remettent pas en cause les règles de conformité, de cohérence et de déployabilité du modèle initial. Cela se traduit par des mécanismes de retranscription des scripts créés et/ou modifiés grâce aux règles de transformation inverse en règles métier (OCL). Ainsi, l'ensemble des modifications apportées par les scripts sont mis à disposition

des concepteurs dans un outil prévu à cet effet. Il sera ensuite possible de visualiser les nouvelles règles métier dans la partie modélisation et de vérifier leur cohérence en demandant une validation des contraintes. Le piège pour ces entreprises est de ne pas modéliser les diverses modifications car elles se retrouveront face à des pertes d'information. En effet, un concepteur qui apporte des modifications sur le modèle sans avoir au préalable réalisé une transformation inverse se retrouve avec le risque d'effacer du modèle de données les dernières modifications ajoutées au niveau du PLM. La transformation inverse est donc la solution qui permet de retranscrire l'ensemble des modifications (structure de données ou scripts) apporté au niveau du PLM. De ce fait, l'ensemble des modifications pourra être vérifié et sauvegardé sans perte d'information.

4.6 Conclusion

Les propositions de ce chapitre ont pour objectif de compléter la démarche du chapitre précédent afin de faciliter le travail du concepteur de modèle. En effet, la notion de contraintes et leurs modélisations avec OCL, permettent une mise en œuvre effective des mécanismes de contrôle selon les différents niveaux (CIM, PIM et PSM) dans MDA. Seule une validation de l'ensemble des contraintes permettra la mise en œuvre effective dans la logique MDA, c'est à dire qu'aucune transformation ne sera rendu possible avant la validation d'un modèle. Finalement, c'est l'étape de validation d'un modèle qui assure la cohérence globale des modèles et qui assure leur homogénéité. Ensuite, la mise en application des diverses transformations permet à ces modèles d'être utilisables dans une plateforme définie. Effectivement, la transformation d'un modèle PIM vers un modèle PSM rend le modèle dépendant d'une plateforme. Ce dernier n'a plus qu'à être utilisé pour la génération de code afin de le rendre applicable dans la plateforme cible. L'ensemble de la chaîne de transformation (PIM->PSM->PLM) constitue la démarche logique des experts PLM. Dans le cas d'une gestion évolutive du système dans le temps, la transformation inverse des modèles et les outils de comparaison de modèles permettent de revalider les modifications au regard des règles métier existantes.

Finalement, afin de vérifier la pertinence et la faisabilité de notre démarche, nous avons développé un framework spécifique (ACMS) que nous avons utilisé sur un cas industriel d'une PME dans le domaine de la plasturgie. La description de ce framework dans le contexte de ce cas est présentée dans le chapitre suivant.

Chapitre 5

Atelier de conception et de modélisation de systèmes d'information (ACMS)

Sommaire

5.1 Introduction

Dans ce chapitre, nous proposons un framework (Atelier de Conception et de Modélisation de Systèmes d'information - ACMS) afin de valider la démarche proposée dans les chapitres précédents (cf chapitre 3) et (cf chapitre 4). L'objectif principal de ce framework est la validation de l'ensemble des propositions présentées dans le cadre d'une démarche MDA. Pour cela, nous avons réalisé des composants logiciels spécifiques au sein de la plate-forme Eclipse [Foundation, 2015] permettant de manipuler des modèles, des

contraintes et des transformations de modèles. Le but est de permettre la mise en œuvre initiale et la modification de modèles pour le PLM Audros.

La première partie de ce chapitre est consacrée aux technologies et outils utilisés dans l'atelier de modélisation ACMS. Ensuite, nous illustrerons notre travail par la construction d'un modèle métier produit pour une entreprise dans le domaine de la plasturgie qui montre l'ensemble des étapes d'une démarche MDA présentée dans cette thèse.

5.2 Architecture du Framework ACMS

Comme nous l'avons indiqué en introduction, ACMS est un framework permettant la mise en œuvre d'une architecture MDA pour les systèmes PLM. Il est basé sur Eclipse et sur différents composants de gestion de modèles (EMF, GMF) et de génération de code (Acceleo, liquibase). Ainsi, dans cette section, nous présentons l'architecture applicative (fig. 5.1) qui a permis de mettre en œuvre nos différentes propositions pour faciliter le travail de l'architecte PLM.

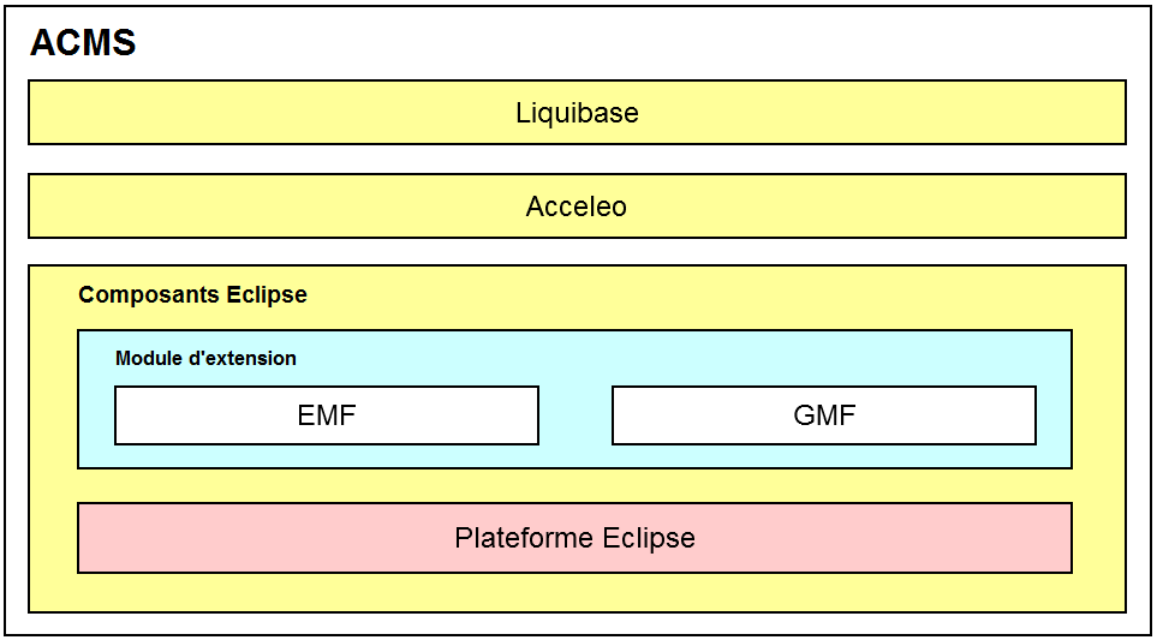

FIGURE 5.1 – Architecture du framework ACMS

5.2.1 La plateforme Eclipse

Eclipse est un environnement de développement intégré pour la production de logiciels qui possède deux caractéristiques essentielles : il est extensible et polyvalent. En effet, il est composé d'un noyau auquel peuvent s'ajouter des composants spécifiques. De plus il permet de manipuler un grand nombre de langages informatiques dont UML. Cet environnement permet de simplifier le travail d'un développeur informatique et propose la mise en œuvre de n'importe quel langage de programmation.

D'un point de vue simplifié, la plate-forme Eclipse est structurée autour de deux concepts principaux :

- **des plugins** : un plugin est un composant qui fournit un ou plusieurs services. Ce composant est un objet qui peut être configuré dans un système au moment de son déploiement.
- **un runtime** : Le runtime Eclipse fournit une infrastructure pour soutenir l'activation et le fonctionnement d'un ensemble de plugins.

Cette notion de composant (dénommé plugin) est fondamentale, car Eclipse est bâti sur une architecture totalement modulaire. La réutilisation des composants intégrés à cette plateforme permet le développement d'applications autonomes par l'intermédiaire du projet Rich Client Platform (RCP). La plateforme Eclipse permet donc de construire son propre environnement avec le socle Eclipse Runtime et l'agrégation de différents composants qui apportent de nouvelles fonctionnalités (fig. 5.2). On peut utiliser des composants existants ou définir ses propres composants spécifiques. Au final, pour créer ACMS, nous avons intégré des composants standards (EMF, GMF, Acceleo et Liquibase) et nos propres composants.

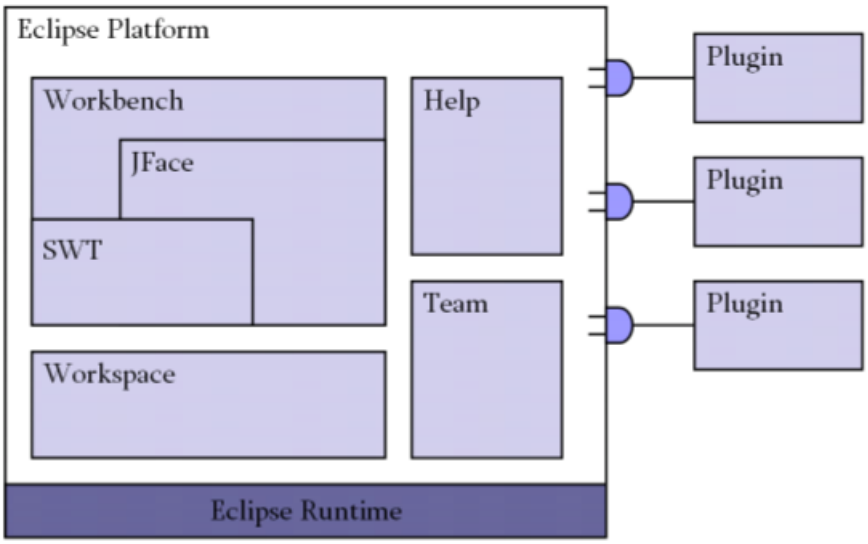

FIGURE 5.2 – Architecture globale de la plateforme Eclipse

Par ailleurs, il convient de noter qu'Eclipse comprend notamment :

- une gestion de modèles de projets
- une infrastructure de déboggage indépendante
- des langages de développement

5.2.2 Le framework EMF

Parmi les plugins utilisés par ACMS, EMF (Eclipse Modeling Framework) est un composant de modélisation et d'aide à la génération de code prenant en compte le cycle de vie des objets. Il contient une infrastructure de génération de code et des applications

basées sur des modèles de données structurées. Initialement, ils permettent de produire des classes Java représentant un modèle décrit généralement sous la forme d'un modèle en XMI. EMF est conçu pour faciliter la conception et la mise en œuvre d'un modèle structuré. Il est une version améliorée du MOF 2.0. Il fournit également des outils permettant de pouvoir visualiser les modèles, de les éditer avec un système de commandes et de les manipuler dans un éditeur. EMF permet de créer deux types de modèles :

- des modèles définissant des concepts (méta-modèles)
- des modèles en conformité avec ces concepts

Tout modèle EMF est une instance d'un modèle EMF avec pour racine commune le méta-modèle Ecore fourni par EMF (fig. 5.3). Ce framework permet non seulement de créer un méta-modèle représentant les concepts désirés par l'utilisateur et de créer des modèles issus de ce méta-modèle et de les manipuler avec un outillage adapté.

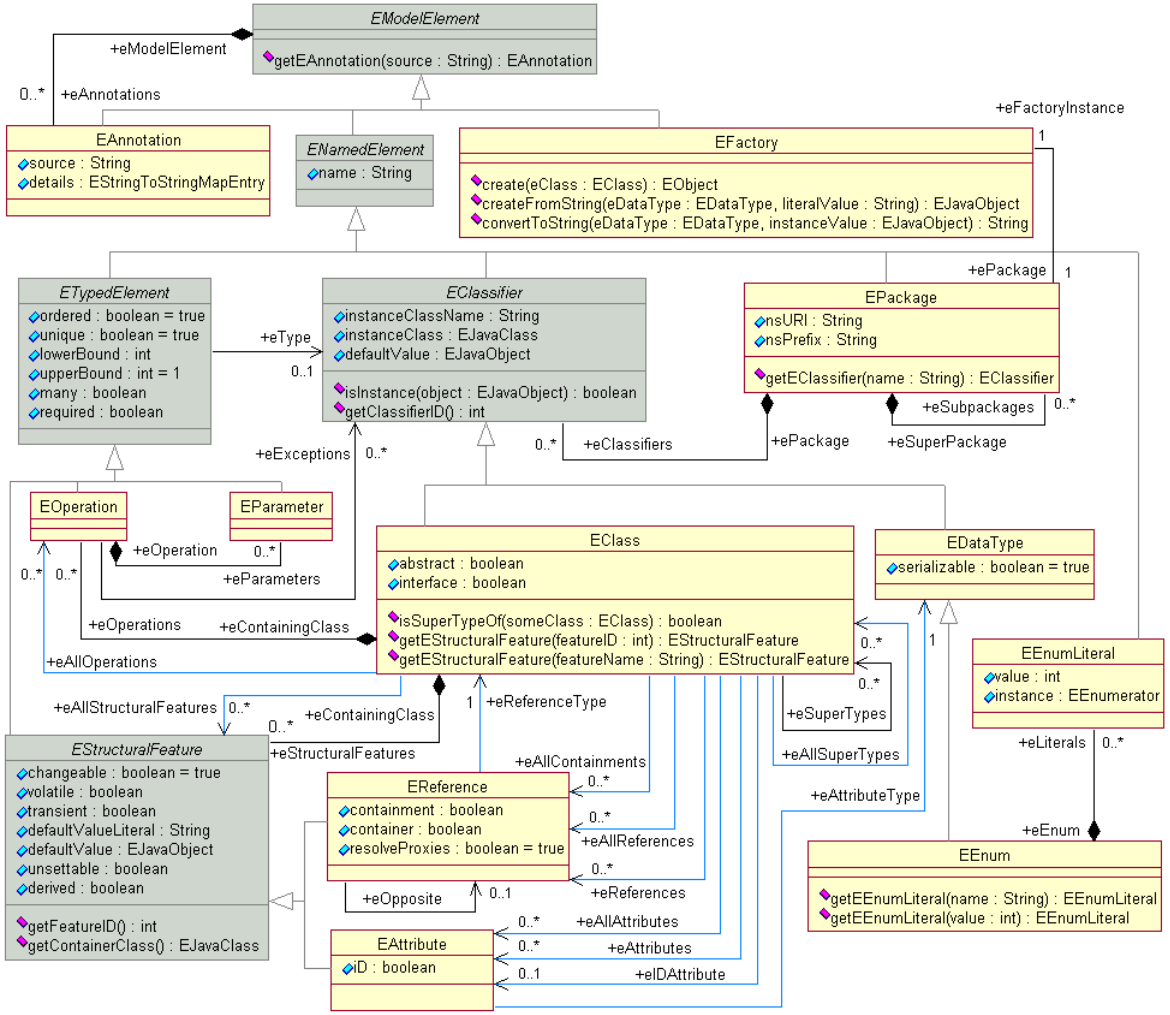

FIGURE 5.3 – Extrait du méta-modèle Ecore [Cariou, 2015]

Le méta-modèle Ecore se situe donc au sommet de la pyramide de notre architecture à quatre niveaux pour MDA, il est la représentation du MOF. Le tableau ci-dessous présente

quelques méta-éléments pour la construction d'un méta-modèle :

Concept	Description
EClass	permet de définir une classe.
EAttribute	permet de définir un attribut d'une classe, il concerne les types primitifs comme les entiers, les booléens, chaines...)
EReference	permet de définir une association entre deux classes.
EOperation	permet de définir une opération d'une classe.
EParameter	permet de définir un paramètre d'opération.

TABLE 5.1 – Quelques concepts du méta-modèle Ecore

5.2.3 Le framework GMF

Le GMF (Graphical Modeling Framework) est un plugin basé sur EMF. Il fournit un support dans la mise en œuvre des éditeurs graphiques pour la création de modèles conformes à un méta-modèle. Il s'appuie sur le design pattern Modèle-Vue-Contrôleur (MVC) qui permet de représenter les modèles avec des figures. GMF est généré depuis les fichiers .genmodel et .ecore établies par EMF. Ce plugin possède 4 fichiers spécifiques qui sont :

- .gmfgraph : ce fichier spécifie à chaque objet une présentation graphique (rectangle, cercle, arc, image...).
- .gmftool : il concerne la génération d'une palette de création pour l'éditeur graphique. Elle permet la création de composants pouvant apparaître dans le modèle (les concepts du méta-modèle).
- .gmfmap : il spécifie la correspondance entre l'élément observable du modèle, sa représentation graphique et l'outil figurant dans la palette permettant de l'instancier.
- .gmfgen : ce fichier englobe tous les fichiers mentionnés précédemment, il permet la génération du code décrivant tous les aspects du diagramme dans lequel le modèle sera établi.

5.2.4 Acceleo

Acceleo est un plugin de générateur de code qui permet de mettre en œuvre l'approche MDA pour réaliser des applications à partir de modèles basés sur EMF. Il s'agit d'une implémentation de la norme de l'OMG pour les transformations de modèles vers du texte (M2T). Acceleo fournit des outils pour la génération de code depuis des modèles. Grâce à ces outils, Acceleo permet notamment de réaliser des générations incrémentales. La génération incrémentale consiste à générer du code, puis à pouvoir modifier le code généré librement et à regénérer le code sans pour autant perdre les modifications réalisées à la main sur le code généré précédemment. Il permet aussi :

- une interopérabilité des méta-modèles d'entrée
- une syntaxe arborescente dédiée à la manipulation de modèles
- une personnalisation de la génération par Template

– l'indépendance du langage généré

5.2.5 Liquibase

Actuellement la gestion (ajouts, suppressions et modifications) des données dans une base de données s'avère être une opération compliquée en particulier en raison de la présence de liens. Les modifications peuvent par ailleurs impacter directement les données de production [Oussalah, 1997], avec des conséquences coûteuses sur les objets physiques (erreurs de cotes, etc). De plus, dans des projets de type PLM, que ce soit lors de la première mise en place ou lors d'une évolution, ces changements de données peuvent être rencontrés de façon très courante. De ce fait, il est important de disposer d'outils pouvant s'adapter aux divers changements et d'appliquer de manière cohérente les diverses modifications dans une base données d'un système d'information. Dans le cadre de notre projet de développement, le choix s'est porté sur Liquibase. Les critères de sélection ont été basés sur quelques paramètres importants pour nous. Liquibase est un outil open source, multi-plateforme, multi-bases de données (dont Oracle et SQL-Server) et permet de gérer l'historique des changements effectués dans la base.

Le but de cet outil est de rendre les modifications au sein d'une base de données beaucoup plus simples, surtout dans un projet de type PLM nécessitant des changements assez importants et fréquents. La gestion de plusieurs bases de données est rendue possible grâce à son fonctionnement par l'utilisation d'un langage standard, le XML. En effet, le fichier XML contient l'intégralité des modifications effectuées, puis Liquibase effectue les opérations de mise à jour de la base de données après lui avoir fourni les paramètres de connexion. Voici quelques exemples de fonctionnalités proposées par cet outil :

– ajouter des enregistrements dans une table
– ajouter ou supprimer une table
– ajouter ou supprimer un index
– ajouter ou supprimer d'un attribut ou colonne

5.3 Application industrielle avec ACMS

Comme nous l'avons indiqué dans les chapitres précédents, il s'agit d'outiller les experts métiers et SI, avec un environnement graphique permettant de manipuler des concepts métiers implémentables dans un PLM. Cela implique :

– la saisie d'un méta-modèle général
– la capacité à saisir et modifier des modèles métier à partir des concepts du méta-modèle et les contraintes métier associées.
– la capacité à valider ces modèles métier
– la mise en œuvre des mécanismes de transformation du PIM vers PSM et du PSM vers le PIM

5.3.1 Gestion des méta-modèles

La création des méta-modèles conformes au méta-méta-modèle Ecore (cf chapitre 3) a été réalisé grâce au composant EMF [García-Magarino et al., 2009] qui est intégré à ACMS. Comme nous l'avons indiqué précédemment, nous avons réalisé deux méta-modèles dans le cadre de notre projet ACMS, l'un pour la création de modèles métier et l'autre afin de spécifier le SI du PLM Audros. Ce sont ces méta-modèles qui seront utilisés dans le cadre de notre atelier ACMS. Ci-dessous un rappel des deux méta-modèles présentés dans le chapitre (cf chapitre 3) :

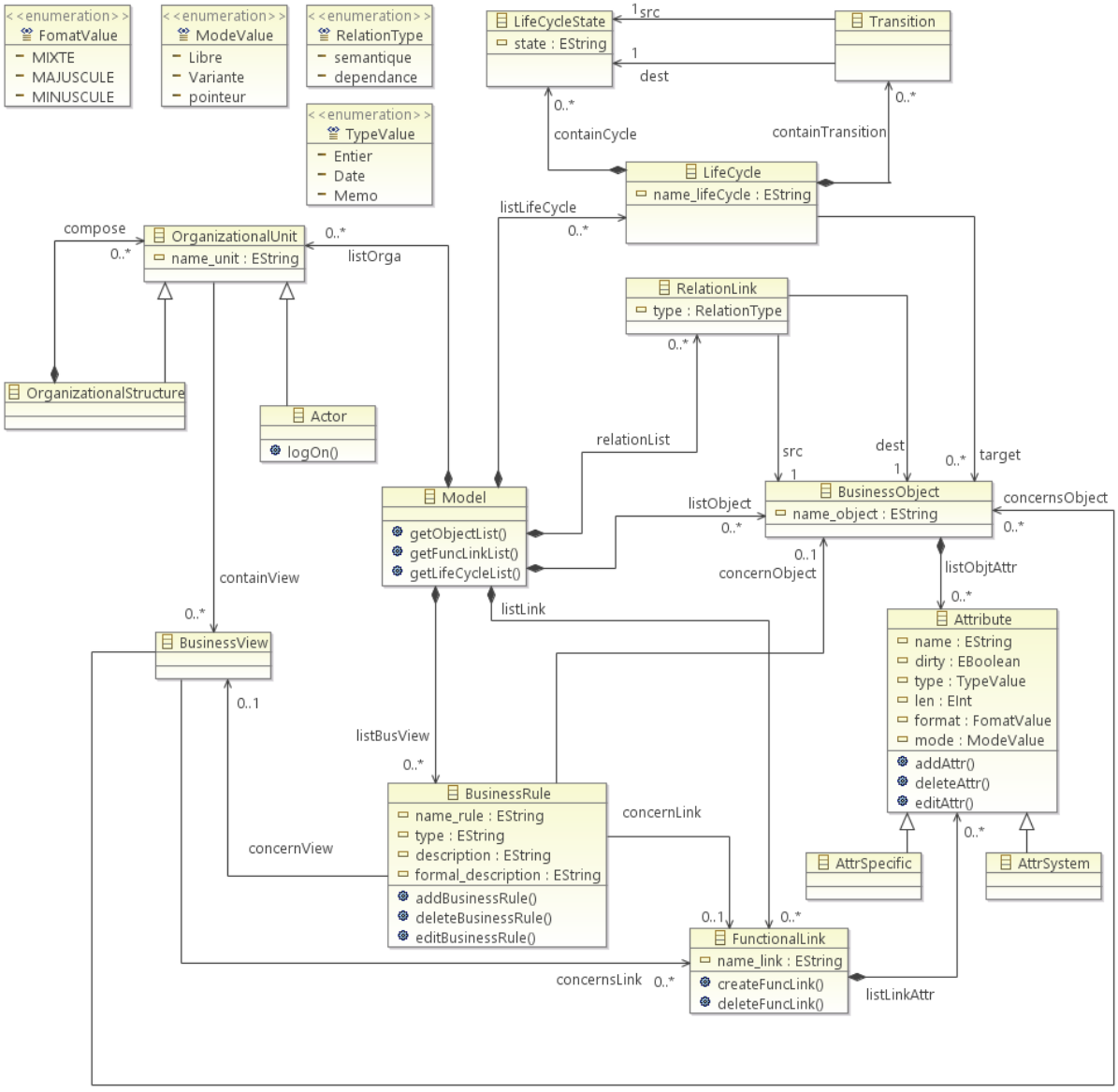

FIGURE 5.4 – Méta-modèle pour les systèmes PLM

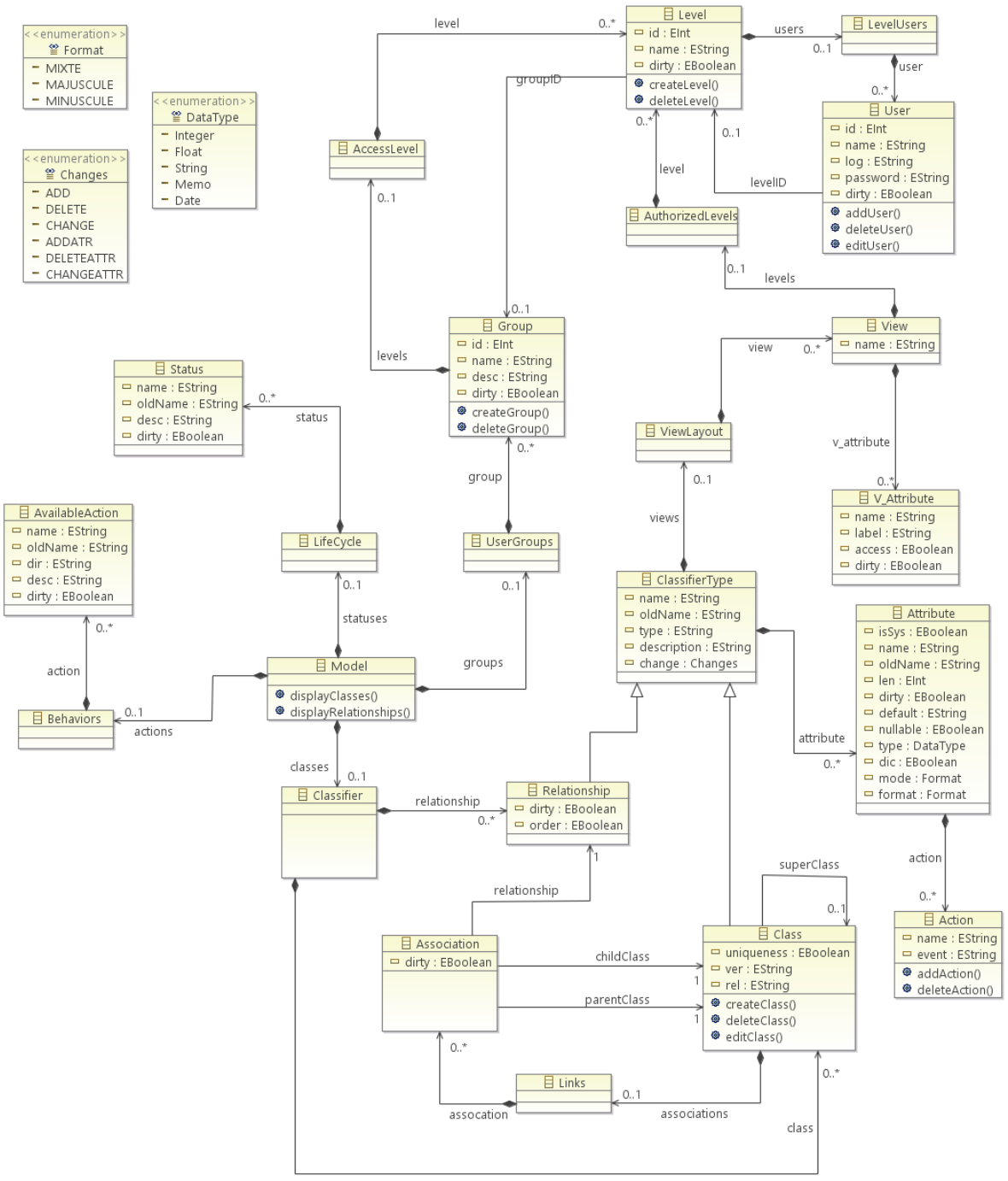

FIGURE 5.5 – Méta-modèle pour le système Audros

5.3.2 Construction des modèles métier

Afin d'axer les besoins, nous avons formalisé les usages des différents experts. Le premier cas d'utilisation (fig. 5.6) décrit les différentes étapes de création d'un modèle objet. Ici, les deux acteurs ont le même rôle dans la création de modèle objet. Il est précisé qu'un modèle objet peut être composé d'un ensemble d'objets reliés par des liens

fonctionnels et des cycles de vie contenant les différents états parcourus par les objets. De plus la création d'objets ou de liens inclut la gestion des attributs.

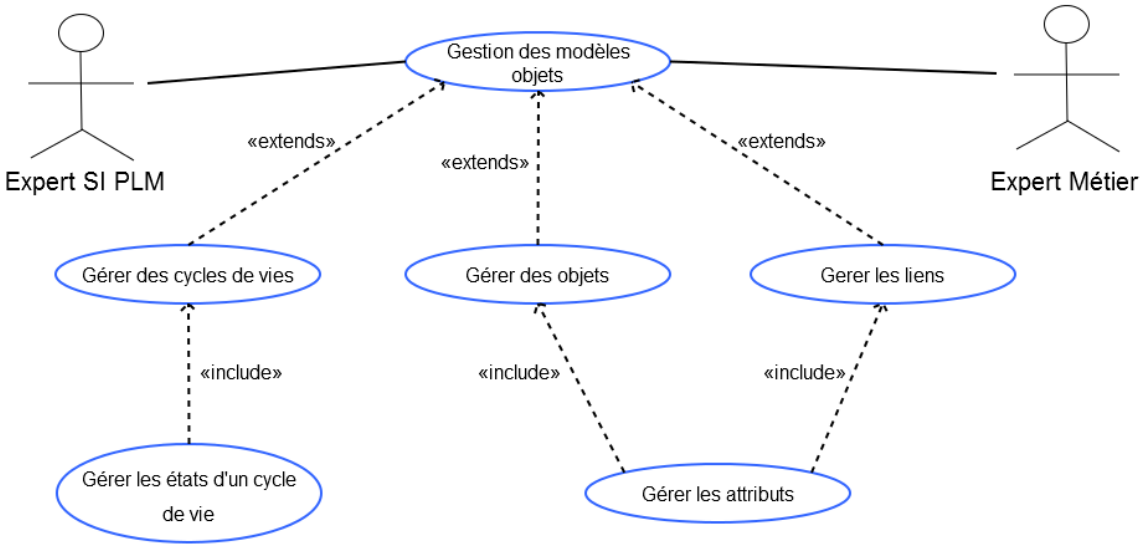

FIGURE 5.6 – Cas d'utilisation "création et édition d'un modèle"

Comme indiqué dans les usages, les experts créent les éléments graphiques à partir des concepts du méta-modèle. Dans notre exemple industriel, nous avons isolé la création des concepts liés aux acteurs du PLM et les objets qu'ils vont manipuler (fig. 5.7). Le concepteur de modèles a donc le choix entre :

- un modèle objet basé sur les concepts du méta-modèle liés aux objets (BusinessObject, FonctionalLink, RelationLink, Attribute...) qui peut être utilisé par les deux acteurs
- un modèle utilisateur basé sur les concepts liés aux droits et aux utilisateurs (BusinessView, OrganisatioalUnit...) plutôt orienté vers les experts SI

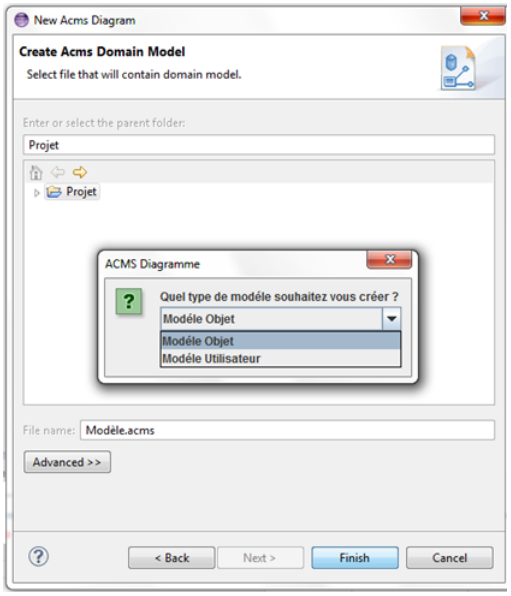

FIGURE 5.7 – Création d'un modèle dans ACMS

Selon le choix du concepteur, l'application ouvre un nouvel onglet avec une palette de création permettant de créer un modèle objet ou utilisateur. La palette de création est propre à chaque type de modèle créé. La figure (fig. 5.8) montre l'interface de création d'un modèle objet. Nous pouvons voir les différents concepts accessibles dans la partie droite contenue dans la palette.

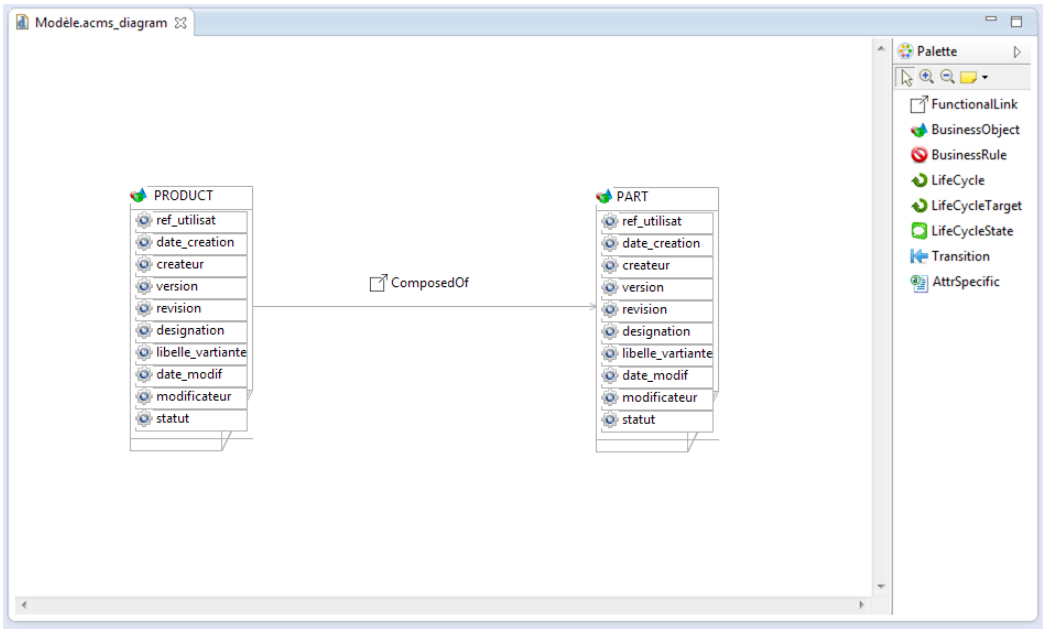

FIGURE 5.8 – Interface de création d'un modèle objet

Finalement, afin de valider la démarche de mise en œuvre de ces modèles métier, nous avons utilisé ACMS dans un contexte industriel de type PME/PMI. Ce contexte est basé

sur l'analyse de déploiement au sein d'une entreprise de plasturgie [Bissay, 2010]. Dans ses travaux, Aurélie Bissay illustre un processus de déploiement en s'appuyant sur des interviews d'experts métier du domaine de l'injection et de l'extrusion. A partir de ces travaux, nous avons établi le modèle métier de niveau PIM correspondant. La figure suivante (fig. 5.9) montre les différents objets métier créés à partir des concepts CIM proposés au chapitre (cf chapitre 3).

La société à l'origine de cette étude développe une activité basée principalement sur des réponses à des appels d'offres. Les objectifs stratégiques de cette entreprise sont le raccourcissement des délais de réponse aux clients et l'accroissement du taux d'acceptation des devis. Les problèmes rencontrés sont la difficulté de réutiliser des études faites précédemment, l'absence de vision de la hiérarchie de l'avancement des études, enfin les trop longues attentes d'approbation. Afin de pouvoir répondre à ces exigences stratégiques et de trouver une solution à ses problèmes, la société décide de déployer un système PLM. Dans sa thèse Aurélie Bissay étudie l'*activité du processus de gestion des appels d'offres et de son sous processus de réalisation du devis* afin d'identifier les concepts métier suivant :

- *appel d'offres*
- *devis*
- *commande*
- *pièce*
- *matière*
- *moule*
- *affaire*

Le but est de créer un modèle de données dans le PLM correspondant au fonctionnement de l'entreprise et d'avoir le maximum de connaissances afin d'optimiser la gestion du processus d'appel d'offre.

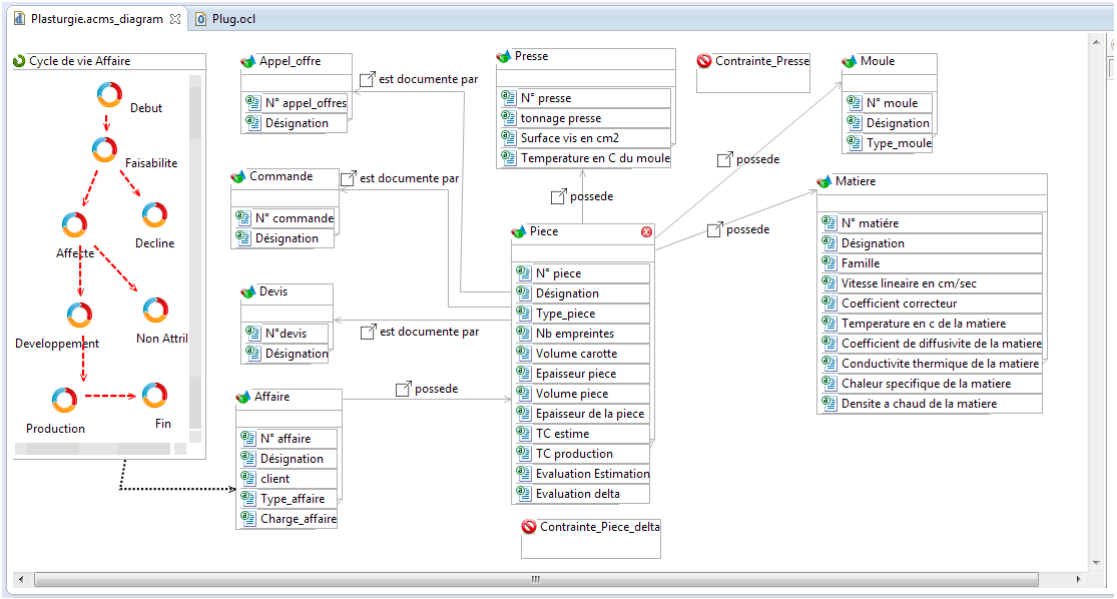

FIGURE 5.9 – Extrait du modèle métier plasturgie

La construction graphique d'un modèle caractérise la base des langages établis sur des diagrammes (comme UML). L'éditeur graphique permet donc de rendre la construction d'un modèle métier PLM (conforme au méta-modèle PLM) plus simple pour les experts. Le but de cet éditeur est de manipuler les différents concepts du méta-modèle PLM.

5.3.3 Définition des contraintes et validation

La modélisation métier présentée dans la section précédente doit être complétée par la mise en place de contraintes. Afin de contrôler la gestion des contraintes dans les modèles, nous avons créé un élément dans le méta-modèle, il est représenté par le stéréotype BusinessRule. Il permet notamment de contrôler et de manipuler les contraintes à appliquer sur le modèle.

Le cas d'utilisation (fig. 5.10) permet d'énumérer les différentes actions appliquées pour gérer des contraintes. Le concepteur peut ajouter des contraintes en créant des règles métier (BusinessRule) dans le modèle, modifier et supprimer des contraintes en les sélectionnant depuis la liste des contraintes. Les contraintes et les modifications sont enregistrées dans un fichier OCL qui regroupe l'ensemble des contraintes.

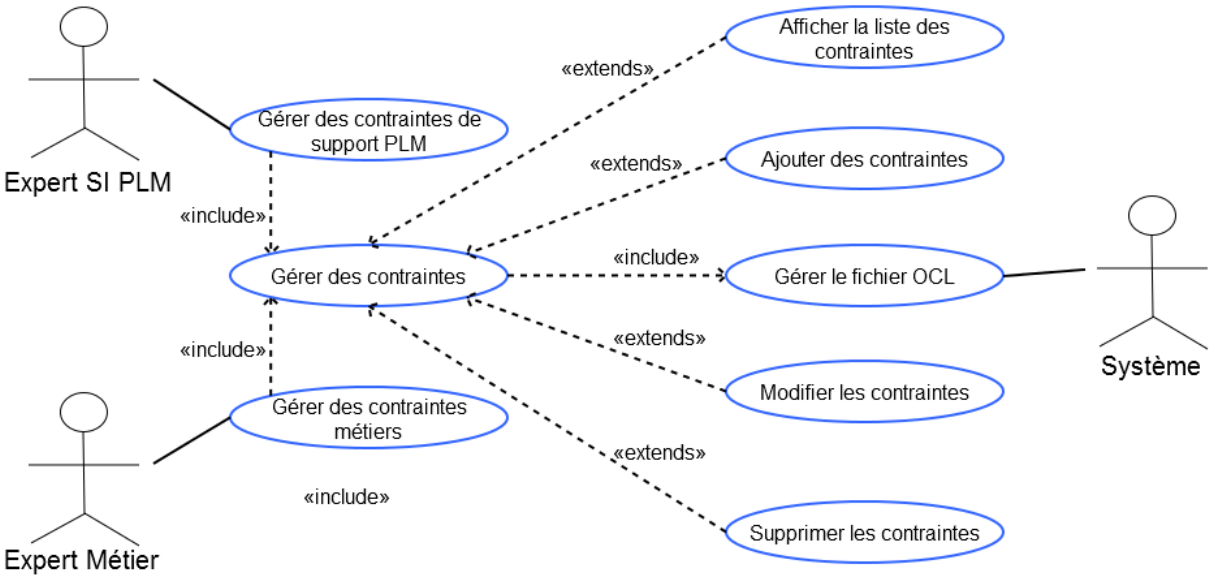

FIGURE 5.10 – Cas d'utilisation "gestion des contraintes"

Lors de la création d'un nouveau projet ACMS, un fichier .ocl créé automatiquement contiendra les règles et les contraintes OCL appliquées sur le modèle. La figure suivante (fig. 5.11) représente l'interface de gestion des contraintes. Cette gestion est étroitement liée au fichier OCL et au modèle du diagramme. Ils permettent la création, la modification et la suppression de contraintes pour le modèle.

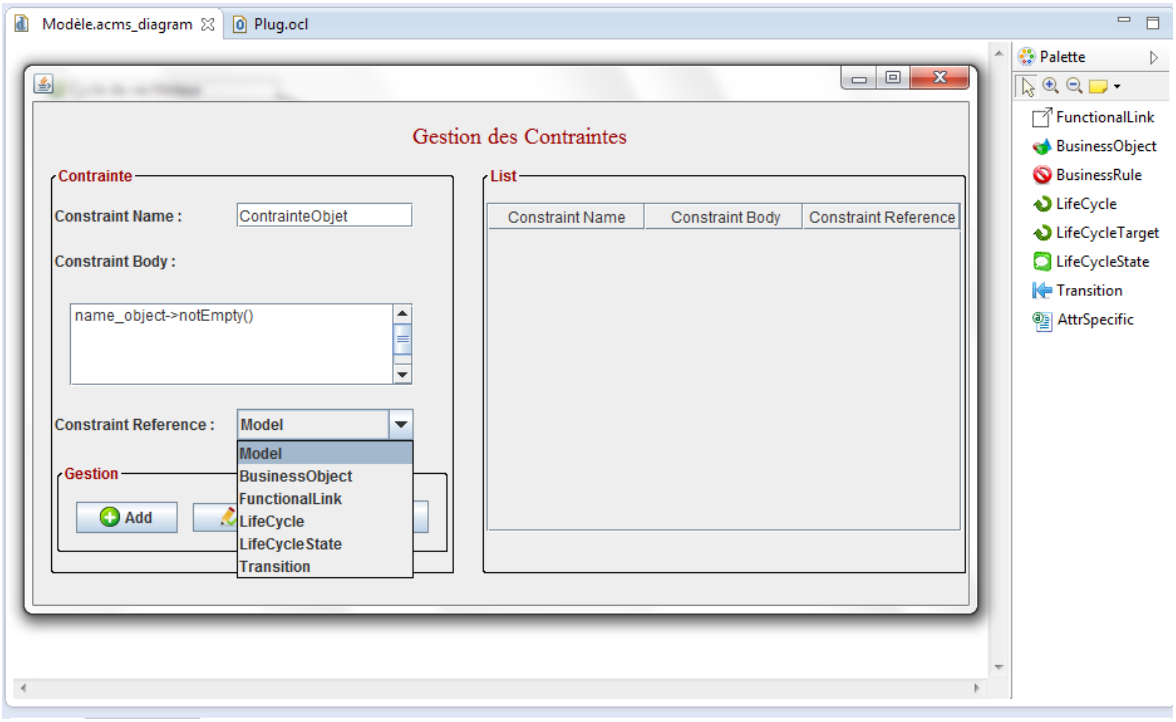

FIGURE 5.11 – Interface de gestion des contraintes dans ACMS

Dans le chapitre (cf chapitre 4), nous avons proposé une typologie de contraintes. Concernant les contraintes de support, l'exemple suivant (list. 5.1) présente une contrainte liée au PLM : "le nom des BusinessObject ne doit jamais être vide".

```
context BusinessObject
inv empty_name : name_object -> notEmpty()
```

Listing 5.1 – Contrainte OCL : un BusinessObject doit avoir un nom

La thèse d'Aurélie Bissay [Bissay, 2010] a identifié deux règles métier dans la construction d'un modèle dans le domaine de la plasturgie. En effet, plusieurs critères ont été étudiés afin de construire ce modèle. La première modélisation avec les différents objets et liens n'étant pas suffisante, des jeux d'essais ont été mis en place afin de déterminer les éléments manquants du modèle. L'analyse des processus métier de l'entreprise a permis de mettre en avant les méthodes de calculs et d'estimation du temps de cycle d'injection d'une pièce plastique. Ce temps de cycle dépend de plusieurs paramètres comme la matière utilisée, la forme de la pièce, le temps de solidification etc... Concrètement, "le temps de cycle (Tc) correspond à la somme du temps d'injection (Ti), du temps de solidification (Ts) et du temps de cycle à vide (Tcàv). Ti, Ts et Tcàv sont eux même le résultat de calcul que nous avons identifié". L'attribut delta dans la pièce est le résultat d'une soustraction entre TC production et TC estimé et doit être positif. Nous avons donc créé une contrainte qui prend en compte le calcul de ce delta. La contrainte sera considérée comme invalide si TC estimé et supérieur à TC production.

```
context BusinessObject
inv Contrainte_compare :
```

```
if self.name_object.equalsIgnoreCase('Piece') then
  self.ListObjAttr->select(p :
  Attribute| p.Nom.equalsIgnoreCase('TC estime')).oclAsSet()->last() > self.ListObjAttr
      ->select(p :
  Attribute| p.Nom.equalsIgnoreCase('Tc production')).oclAsSet()->last()
else true endif
```

Listing 5.2 – Contrainte OCL : différence entre TC estimé est TC production

Deux types de liens sont identifiés dans la thèse, le lien 'possede' et le lien 'est documente par'. D'après A. Bissay "Le lien 'possède' sera utilisé entre les objets dits 'structurants', alors que le lien 'est documenté par' sera utilisé pour les objets dits documentaires. Ainsi on aura des relations du type 'une affaire possède une pièce', 'une pièce est documentée par un appel d'offre', etc.". La contrainte (list. 5.3) présente l'application de cette contrainte en OCL. La contrainte concerne le stéréotype FunctionalLink, si son nom est 'possede' alors nous vérifions les extrémités (père et fils) du lien. La contrainte sera valide seulement si le père est un objet de type 'PIECE' ou 'AFFAIRE' alors l'extrémité fils doit être une 'PIECE', un 'MOULE', une 'PRESSE' ou une MATIERE.

```
context FunctionalLink
inv Contrainte_Lien_possede : if self.name_link.equalsIgnoreCase('possede') then (self.
    father.name_object.equalsIgnoreCase('Piece') or self.father.name_object.
    equalsIgnoreCase('Affaire')) and (self.son.name_object.equalsIgnoreCase('Piece') or
    self.son.name_object.equalsIgnoreCase('Presse') or self.son.name_object.
    equalsIgnoreCase('Moule') or self.son.name_object.equalsIgnoreCase('Matiere')) else
    true endif
```

Listing 5.3 – Contrainte OCL : le lien 'possede'

Finalement, la figure suivante (fig. 5.12) montre le modèle métier complété des différentes contraintes.

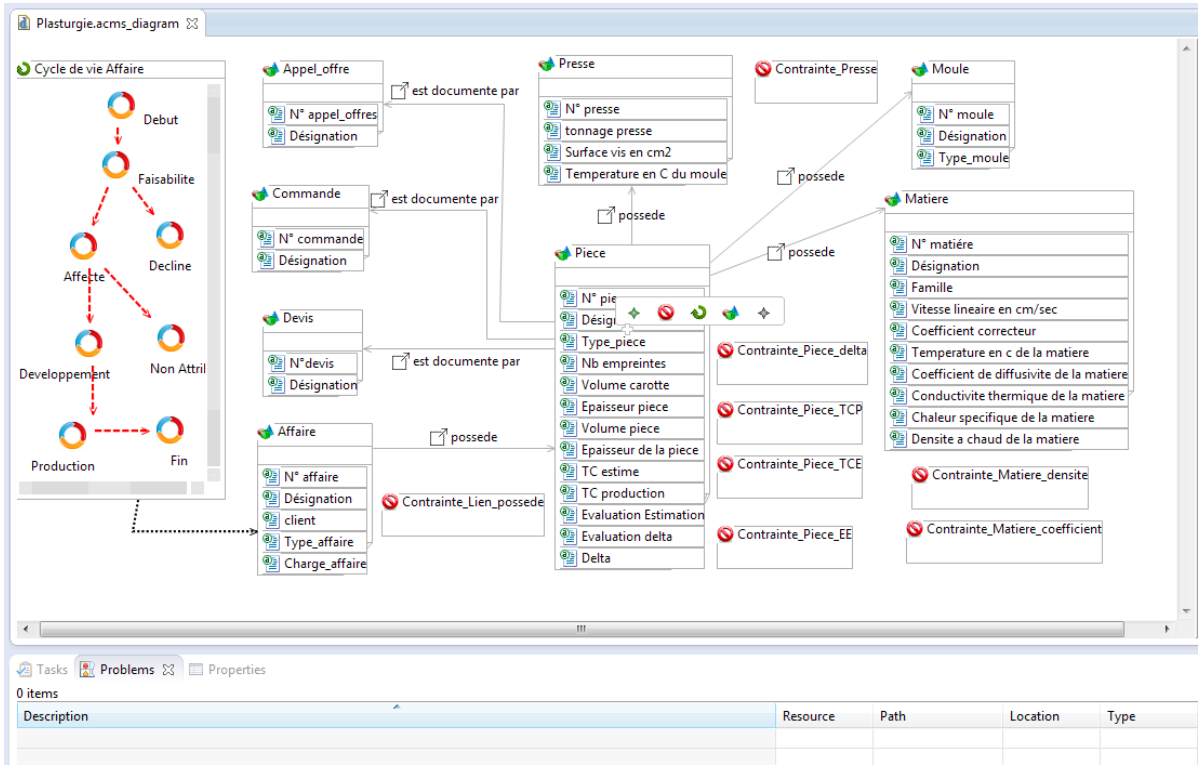

FIGURE 5.12 – ACMS - Extrait du modèle métier plasturgie avec des contraintes

L'étape de validation nécessite l'ajout d'un bouton dans le menu contextuel de l'application. L'utilisateur doit être capable de valider son modèle à n'importe quel moment. L'action de validation est alors suivie par un enchaînement d'évènements. En effet, l'application récupère le chemin du jar contenant le plug-in de validation, cela nous permet de décompresser le jar et d'ajouter le dernier fichier OCL à jour concerné en remplaçant l'ancien. Ensuite, il ne reste plus qu'à compresser les fichiers sous la forme du jar d'origine et à le replacer dans le répertoire contenant la totalité des plugins pour notre application Eclipse.

Par exemple, la figure (fig. 5.13) montre une erreur lors de la validation d'un modèle avec le nom d'un BusinessObject n'ayant aucune valeur. En effet, à la validation du modèle la contrainte présentée précédemment n'a pas été respectée. Un indicateur visuel (petite croix rouge) apparaît alors sur l'élément concerné par la contrainte afin que l'utilisateur puisse corriger cette incohérence.

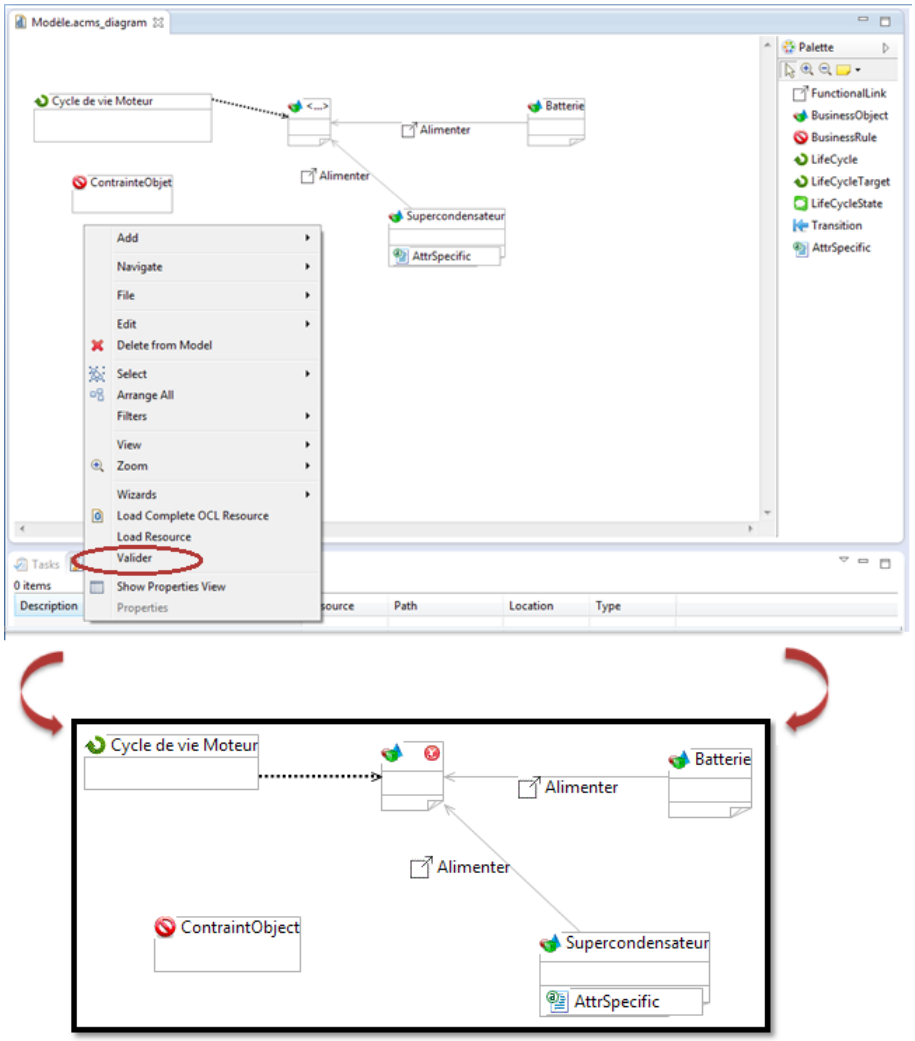

FIGURE 5.13 – Validation d'un modèle avec une contrainte non respectée

5.3.4 Les transformations de modèles

Cette section montre les deux transformations réalisées par notre application. La première est une transformation de modèle à modèle réalisée en ATL (M2M dans la figure (fig. 5.14)). Comme cela a été illustré dans le chapitre précédent, nous avons mis en correspondance les éléments du premier méta-modèle d'entrée (PLM) avec les éléments du deuxième méta-modèle (BD). Ces règles de transformation rendent alors possible la création en sortie d'un modèle PSM conforme au méta-modèle BD en ayant un modèle PIM conforme au méta-modèle PLM. La deuxième, correspondant à une transformation d'un modèle PSM vers du texte sera réalisé grâce à des outils existants, Acceleo et Liquibase (2 outils open source) présentés au début de ce chapitre. Dans notre cas, cette transformation (M2T dans la figure (fig. 5.14)) correspond à une transformation d'un modèle vers un langage standard, le XML. L'obtention du fichier XML contenant les informations à insérer dans la base de données est réalisée par Acceleo. Ensuite, Liquibase se charge

d'analyser le fichier XML obtenue afin d'insérer, de modifier ou de supprimer les données dans la base de données Audros. L'avantage de Liquibase est de faire abstraction de la BD cible et de pouvoir contrôler toutes les opérations de création et de mise à jour dans la BD.

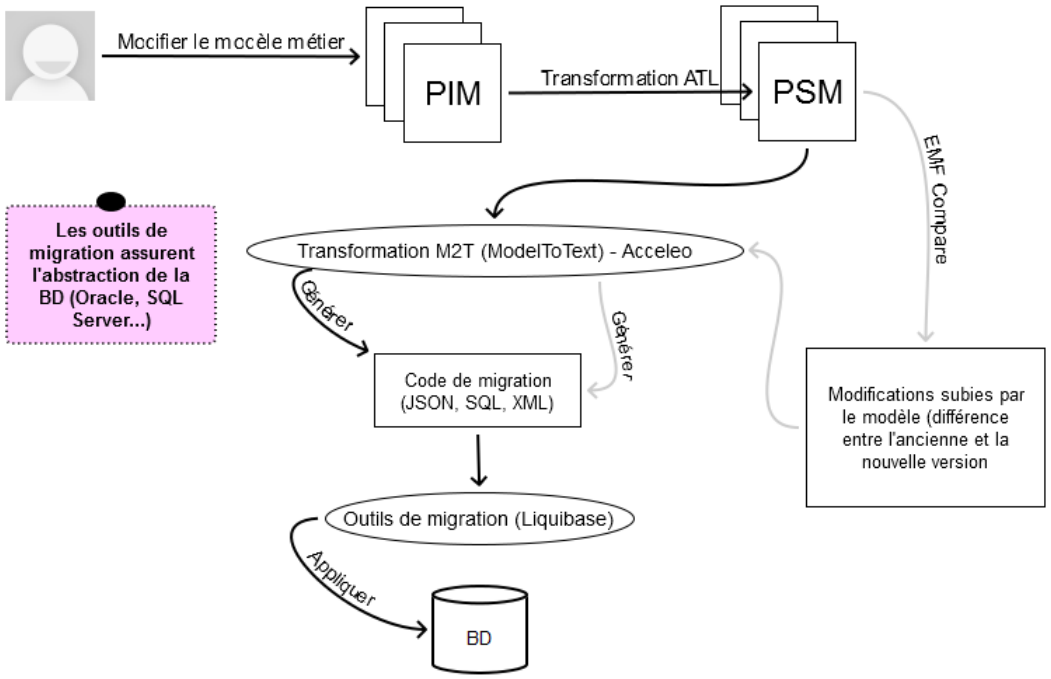

FIGURE 5.14 – Schéma récapitulatif des outils de transformation

Les règles de transformation des modèles Audros vers du XML (destinée à l'outil LiquiBase) ont été implémentées à l'aide du framework Acceleo. Pour chaque élément du méta-modèle Audros, plusieurs Templates Acceleo, correspondant chacun à une opération sur l'élément concerné (Création, modification, suppression) ont été créés. Le modèle en entrée est parcouru, et pour chaque élément rencontré, le Template correspondant est exécuté. L'ensemble des codes ci-dessous présente une partie des Templates correspondant à l'élément Attribute :

- La création d'un attribut dans une classe d'objet ou une relation revient à créer une colonne dans une table d'objet dt_NomObjet ou dans une table de lien ds_NomLien. Le template ci-dessous montre la création d'une colonne.

```
[template public create(anAttribute : Attribute)]
  <column name="[anAttribute.name.toUpper() /]" type="[convertType(anAttribute.
      type.toString()) /]([convertLength(anAttribute.type.toString(), anAttribute.len.
      toString()) /]
    <constraints nullable="[anAttribute.nullable /]" />
  </column>
[/template]
```

Listing 5.4 – Création d'une colonne dans une table

– L'ajout d'un attribut à une classe d'objet.

```
[template public add(anAttribute : Attribute, aClass:Class)]
  <addColumn tableName="DT_[aClass.name /]">
    [ self .create() /]
  </addColumn>
[/template]
```

Listing 5.5 – Ajout d'une colonne dans une table d'objet

– L'ajout d'un attribut à une relation.

```
[template public add(anAttribute : Attribute, aRelationship:Relationship)]
  <addColumn tableName="DS_[aRelationship.name /]">
    [ self .create() /]
  </addColumn>
[/template]
```

Listing 5.6 – Ajout d'une colonne dans une relation

– la suppression d'un attribut d'une classe d'objet.

```
[template public drop(anAttribute : Attribute, aClass:Class)]
  <dropColumn columnName="[self.name /]" tableName="DT_[aClass.name /]" />
[/template]
```

Listing 5.7 – Suppression d'un attribut dans un objet

L'exécution de ces templates par l'outil Acceleo donne en sortie un fichier XML contenant des balises 'SQL' qui seront traitées par le logiciel Liquibase. Le choix de ce langage a été préféré pour optimiser notre temps de travail. En effet, Liquibase est un outil capable de faire abstraction de la base de données cible. De ce fait, avec le même fichier standard XML généré par Acceleo, il est capable d'exécuter les requêtes dans une base de données Oracle ou SQL-Server. Par exemple, si nous avions choisi de transformer le modèle PSM en code SQL direct, nous aurions dû différencier les templates pour Oracle et SQL-Server. De plus, la société Audros propose à ses clients les deux types de bases de données traitées par l'outil Liquibase, ce qui renforce le choix du langage XML plutôt que le SQL pur. Voici une partie de ce fichier XML correspondant à la création de la table MOULE dans la base de données du PLM Audros. La première partie du code est identique quelle que soit la classe d'objet créée, ce sont les colonnes (invisibles dans l'outil PLM pour les utilisateurs) qui permettent de réaliser les jointures (liaisons) entre les différentes tables (systèmes ou spécifiques) de la base de données. Par exemple le 'NUM_ART' correspond à un identifiant (entier) unique pour un objet dans la base de données. Il est différent de la référence "métier" (codification) de l'objet.

```
<createTable tableName="DT_MOULE" schemaName="AUDROS" tablespace="ts_audros">
    <column name="AUDROS_PROVENANCE" type="java.sql.Types.VARCHAR (14)"
        defaultValue="">
        <constraints nullable="true" />
    </column>
    <column name="SUJET" type="java.sql.Types.INTEGER" defaultValueNumeric="0">
```

```
  <constraints nullable="true" />
</column>
<column name="AUDROS_SYSTEME" type="java.sql.Types.VARCHAR (12)" defaultValue="
    ">
  <constraints nullable="true" />
</column>
<column name="ID_STAT" type="java.sql.Types.INTEGER" defaultValueNumeric="0">
  <constraints nullable="true" />
</column>
<column name="ID_SU_CL_OB" type="java.sql.Types.INTEGER" defaultValueNumeric="
    0">
  <constraints nullable="true" />
</column>
<column name="NUM_ART" type="java.sql.Types.INTEGER">
  <constraints nullable="true" />
</column>
```

Listing 5.8 – Code XML pour la création de la classe MOULE (partie 1)

La deuxième partie du code identifie les attributs systèmes (obligatoires et imposés par le PLM) et spécifiques (gérés par les utilisateurs du PLM). C'est pour cela que nous retrouvons le 'TYPE_MOULE' à la fin du fichier. La référence et la désignation étant représentées par les attributs systèmes ('REF_UTILISAT' et 'DESIGNATION').

```
<column name="REF_UTILISAT" type="java.sql.Types.VARCHAR(64)" defaultValue="
    null">
  <constraints nullable="true" />
</column>
<column name="DATE_CREATION" type="java.sql.Types.DATE(9)" defaultValueDate="
    null">
  <constraints nullable="true" />
</column>
<column name="CREATEUR" type="java.sql.Types.VARCHAR(24)" defaultValue="null">
  <constraints nullable="true" />
</column>
<column name="VERSION" type="java.sql.Types.VARCHAR(10)" defaultValue="null">
  <constraints nullable="true" />
</column>
<column name="REVISION" type="java.sql.Types.VARCHAR(10)" defaultValue="null">
  <constraints nullable="true" />
</column>
<column name="DESIGNATION" type="java.sql.Types.VARCHAR(255)" defaultValue="
    null">
  <constraints nullable="true" />
</column>
<column name="LIBELLE_VARTIANTE" type="java.sql.Types.VARCHAR(64)" defaultValue
    ="null">
  <constraints nullable="true" />
</column>
```

```
  <column name="DATE_MODIF" type="java.sql.Types.DATE(9)" defaultValueDate="null
      ">
    <constraints nullable="true" />
  </column>
  <column name="MODIFICATEUR" type="java.sql.Types.VARCHAR(24)" defaultValue="
      null">
    <constraints nullable="true" />
  </column>
  <column name="STATUT" type="java.sql.Types.VARCHAR(24)" defaultValue="null">
    <constraints nullable="true" />
  </column>
  <column name="TYPE_MOULE" type="java.sql.Types.VARCHAR(64)" defaultValue="null"
      >
    <constraints nullable="true" />
  </column>
</createTable>
```

Listing 5.9 – Code XML pour la création de la classe MOULE (partie 2)

5.3.5 Comparaison de modèles

La comparaison de modèles est une problématique importante dans le cadre de notre atelier. En effet, comme cela a été précédemment dit, les modèles métier sont souvent amenés à être modifiés suite à diverses raisons. De ce fait, il est nécessaire d'analyser les modifications effectuées sur un modèle par rapport à sa précédente mise en production en effectuant une comparaison de modèles. Cela permet de traiter seulement les changements qui ont eu effet durant le passage d'une version à une autre du modèle métier de l'entreprise. Dans le cadre de notre prototype, nous avons mis en place le traitement suivant : l'ensemble des modifications est regroupé dans un modèle (DiffModel). Ce modèle est utilisé (conjointement avec le modèle métier) comme entrée pour le module ATL contenant les différentes règles permettant de transformer ces différences en un modèle BD afin de les injecter dans la base.

Voici des fragments de code ATL de ces transformations :

Vérification de la première exécution du module (pas de comparaison à faire si c'est le cas) :

```
module business2db;
create OUT : DB from IN : Business, IN2 : DiffModel;

helper def : isFirstRun() : Boolean =
  if DB!Model.allInstances()->size() = 0 then
    true
  else
    false
  endif;
```

Listing 5.10 – Code ATL première exécution

Vérification si la cible des changements est un BusinessObject :

```
helper context DiffModel!Diff def : isBusinessObject() : Boolean =
  if self.match.left.oclIsTypeOf(Business!BusinessObject.oclType()) then
    true
  else
    false
  endif;
```

Listing 5.11 – Code ATL de vérification de changements d'un businessObject

Transforme les changements ayant pour cible les BusinessObject :

```
lazy rule Diff2DBElement {
  from
    diff : DiffModel!Diff(diff.isBusinessObject())
  to
    a : DB!Class (
      name <- diff.value.name_object,
      attribute <- diff.value.ListObjAttr->collect(attr | thisModule.
          BusinessAttribute2Attribute(attr))
    )
}
```

Listing 5.12 – Code ATL de transformation de BusinessObject

Le processus de comparaison de l'EMF Compare (fig. 5.15) revient à séquencer le traitement en 6 sous parties :

1. **La résolution du modèle :** c'est à dire de trouver à partir du point de départ (dernier modèle mis en place) tous les fragments requis pour la comparaison de la totalité du modèle logique.
2. **La correspondance :** nécessite des itérations sur les deux modèles à comparer afin de trouver les correspondances entre les éléments des modèles en entrée. Cela revient à déterminer qu'un élément du modèle 1 et l'équivalent d'un élément du modèle 2. Par exemple, savoir si la classe C1 du premier modèle est l'équivalent de la classe C2 du deuxième.
3. **La Différenciation :** à la fin de la phase de correspondance, une liste identifiant les similitudes entre les éléments est obtenue. Dans la phase de différentiation, il s'agit de parcourir cette liste et de déterminer les différences entre les éléments correspondants. Par exemple, le nom de la classe C1 a changé en C2.
4. **L'équivalence :** pendant l'étape précédente, la différentiation a détecté les différences entre les éléments correspondants. Cependant, deux différences distinctes peuvent représenter en réalité un seul changement (par exemple, différences au niveau de références opposées). Dans cette phase, toutes les différences sont revues et unifiées afin de savoir si elles sont considérées comme étant équivalentes.

5. **L'exigence :** Il s'agit dans cette phase de déterminer les dépendances nécessaires entre les différences détectées avant de pouvoir les fusionner. Par exemple, l'ajout d'une classe C1 dans le package P1 dépend de l'ajout du package P1 lui-même).

6. **Les conflits :** lors de la comparaison des fichiers où l'un des deux est géré par un système de versionning (CVS, SVN, Git...), il peut y avoir des conflits entre les modifications faites en local et celles faites dans le dossier distant. Durant cette phase, toutes les différences détectées seront examinées à la recherche de ces conflits.

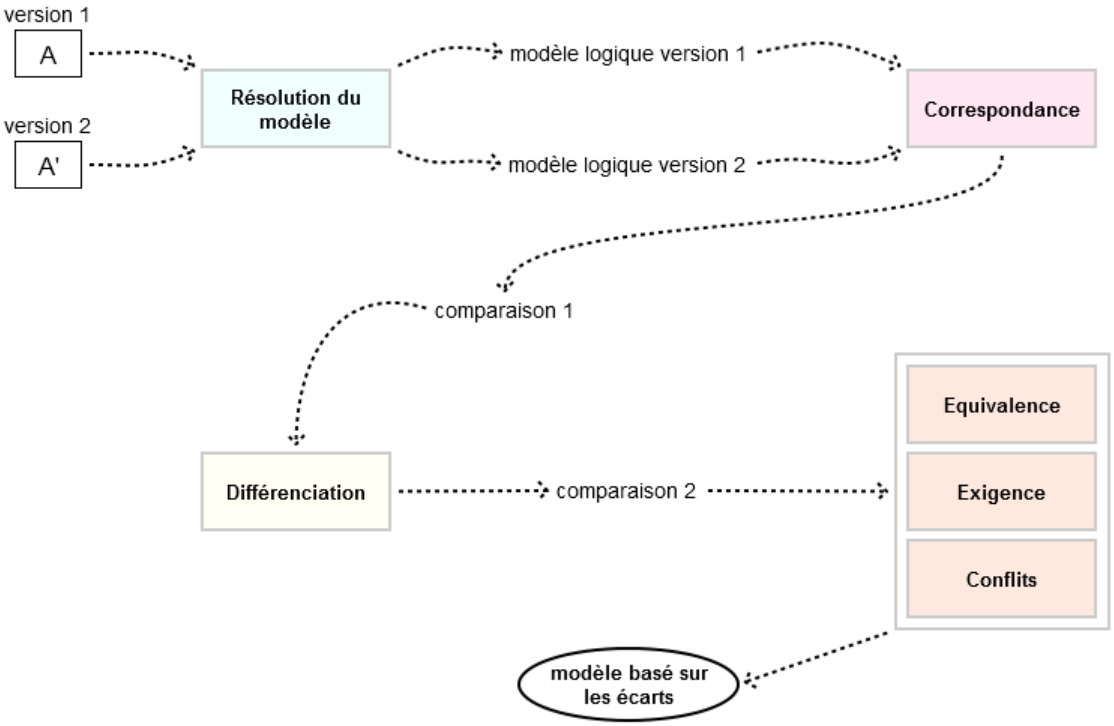

FIGURE 5.15 – Processus de comparaison de l'EMF Compare

Ci-dessous nous présentons un exemple d'un modèle DiffModel issu d'une comparaison entre deux modèles. Le nom d'un BusinessObject a été modifié au sein d'un modèle après une première mise en place de ce dernier. Les changements effectués sont alors visibles sur le modèle DiffModel obtenu après la comparaison :

```
<?xml version="1.0" encoding="ASCII"?>
<compare:ReferenceChange xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI"
    xmlns:compare="http://www.eclipse.org/emf/compare">
  <reference href="http://acms/1.0#//BusinessObject/ListObjAttr"/>
  <value href="/Audros/plasturgie.acms#//@ListObjects.2/@ListObjAttr.10"/>
</compare:ReferenceChange>
```

Listing 5.13 – Exemple d'un modèle obtenu en sortie d'une comparaison de modèle

5.4 Conclusion

Le but de notre framework ACMS est de valider l'ensemble de notre proposition décrite dans les chapitres précédents ((cf chapitre 3) et (cf chapitre 4)). Nous avons donc implémenté les méta-modèles (PLM et Audros) et créé des contraintes (respectivement des transformations) à l'aide du langage OCL (respectivement ATL) afin de passer d'un modèle PIM conforme au méta-modèle PLM à un modèle PSM conforme au méta-modèle Audros. Afin de rendre ces modèles plus complets et plus cohérents (éviter les problèmes en production), des contraintes OCL peuvent être ajoutées à divers niveaux d'exécution (CIM, PIM ou PSM). Le test de l'atelier avec la création d'un modèle plasturgie pour une PME a permis de tester nos propositions, notamment les contraintes, les transformations et la comparaison de modèles. La démarche MDA a été mise en place avec, au niveau le plus bas l'exécution des modèles dans le PLM Audros.

Conclusion

L'objet des travaux que nous avons présenté dans ce mémoire était destiné à répondre à des problématiques industrielles en lien avec le déploiement et aux évolutions des systèmes PLM. Bien que les problématiques à l'origine de ce projet de recherche soient issues du retour d'expérience de la société éditrice du PLM Audros, elles sont toutefois suffisamment génériques, et souvent rencontrées par les éditeurs et leurs intégrateurs. De fait, elles induisent des problématiques scientifiques autour de la modélisation et de la conception de Système d'Information ainsi que sur la vérification de modèles.

Cette thèse est divisée en deux parties. La première partie est constituée des chapitres 1 et 2 : le premier chapitre caractérise les principes de base des systèmes PLM ainsi que les problématiques abordés (les problématiques industrielles et les problématiques scientifiques induites). Le deuxième chapitre dresse un état de l'art des recherches sur les systèmes PLM ainsi qu'un état de l'art sur l'ingénierie dirigée par les modèles (IDM) qui peut être considérée comme une forme d'ingénierie générative (permet la création d'une application informatique à partir de transformations de modèles). En particulier, nous présentons les approches d'IDM proposées par l'OMG et notamment MDA.

La deuxième partie de la thèse détaille notre proposition et est composée de trois chapitres. Dans le chapitre 3, nous avons présenté notre proposition de modélisation basée sur les niveaux d'abstraction (CIM, PIM et PSM) de MDA. A ce stade, nous avons introduit les concepts liés aux systèmes PLM dans le niveau CIM au sein d'un méta-modèle, indépendants de toute plate-forme d'exécution. La perspective PIM présente les modèles métier en s'appuyant sur les stéréotypes issus du méta-modèle. Cela permet de représenter les concepts métier qui seront manipulés dans le SI de l'entreprise. Le niveau suivant dans la logique MDA est le PSM obtenu par transformation du modèle PIM. A cette nouvelle étape de la démarche, le modèle devient dépendant d'une plateforme définie lors de la transformation et représente le niveau d'exécution cible. Dans notre cas, il se positionne au sein d'un système PLM. L'objectif est d'être capable de gérer des modèles hétérogènes PIM conformes aux concepts du PLM et de passer par des transformations à des modèles PSM spécifiques, dépendants de la plateforme sélectionnée.

Par la suite, dans le chapitre 4 nous avons détaillé le principe des contraintes et leur utilisation dans les modèles en amont des transformations de modèles qui sont à l'origine de la génération de code. En effet, notre proposition de méta-modèle est complétée par les concepts de contraintes et leurs modélisations avec le langage OCL. Ces contraintes de conformité, de support PLM (déployabilité), et métiers (cohérence), sont définies sur les méta-modèles et les modèles. Elles permettent de définir ainsi, des mécanismes de validation de modèles (valide, conforme, cohérent et déployable). La validité d'un modèle

permet d'initier les transformations descendantes vers une plateforme cible définie. Dans ce chapitre, nous présentons aussi le principe des transformations inverses et la comparaison de modèles afin de gérer les évolutions du système dans le temps.

Pour finir, dans le dernier chapitre nous avons mis en œuvre la démarche proposée autour d'un prototype réalisé avec le framework Eclipse, que nous avons mis en pratique sur un cas industriel réel utilisable dans le PLM Audros.

Finalement, concernant l'objectif de faciliter le déploiement d'un système PLM, nous avons proposé une démarche avec un double mécanisme. D'une part, l'usage de méta-modèles et de contraintes permet de valider une conformité souhaitée à un niveau souhaité, d'autres part, des règles de transformation automatiques permettent de conserver la cohérence initiale des modèles jusqu'à leur implémentation. Cette démarche *descendante* aide le concepteur du système PLM car elle lui donne les moyens de construire ses propres règles métier et les outils de vérification.

Implicitement, il devient possible de gérer les évolutions du système de façon descendante, c'est à dire en partant des modèles PIM vers les modèles PSM. En effet, toute évolution réalisée sur un modèle PIM doit rester conforme au méta-modèle CIM et aux contraintes OCL. Dans le cas où les évolutions ne concernent pas les contraintes métier, la mise en œuvre du procédé de validation des contraintes OCL est implicite. Dans le cas où les évolutions concernent aussi les contraintes métier, il est nécessaire de vérifier que cela n'introduit pas d'incohérence. A ce niveau, les incohérences peuvent provenir des contraintes supplémentaires qui sont appliquées de façon indifférenciées sur les nouveaux objets métier ou sur ceux définis initialement. Les erreurs éventuelles seront traitées de la même manière que lors de la conception initiale. Au final, le modèle PIM devra est conforme, vérifié et déployable.

Dans le cas ou les évolutions sont réalisées directement sur le système PLM, nous proposons un mécanisme *ascendant* partiel basé sur le même principe (transformation de modèle et vérification de contraintes). Dans nos propositions, le mécanisme est partiel car la bijection (PIM -> PSM) n'est pas garantie. Pour autant, cela permet de mettre en place un premier niveau de contrôle afin d'identifier les modifications réalisées directement sur le niveau d'exécution du PLM (modification de la structure, création de script...).

La démarche que nous avons proposé dans cette thèse peut être assimilée à une démarche méthodologique d'usage d'un atelier de modélisation sous contraintes. Nous pensons que l'objectif d'aide à la conception et au déploiement est globalement atteint, toutefois un certain nombre de points n'ont pas été abordés dans ces travaux ou partiellement résolus. Ces points constituent de fait, des perspectives potentielles de travaux futurs.

Cohérence et simulation La première perspective concerne le contrôle de cohérence et la simulation de certains mécanismes au sein du système PLM. En effet, dans nos travaux nous proposons des mécanismes de contrôle de cohérence descendant et ascendant. Concernant les mécanismes ascendants (du code vers les modèles), nous avons des règles génériques qui ne permettent qu'un contrôle de cohérence partiel. Afin d'avoir un contrôle de cohérence totale, il convient de pouvoir analyser chaque constituant instancié du PSM et notamment les capacités de script que l'on retrouve dans la plupart des systèmes PLM industriel. Quelque soit la structure identifiée au sein du niveau PSM, un script

se déclenche sur des évènements ou des interactions, et produit un ensemble d'actions qui impactent le système (les données et ou la structure). Il est donc nécessaire pour un contrôle de cohérence total, de mettre en place une analyse intrinsèque de chaque élément ajouté par un des acteurs, afin de permettre de vérifier les règles de cohérence. De fait, en disposant d'une telle information (analyse interne des éléments du code), il est tout à fait envisageable de mettre en place des mécanismes de simulation. L'intérêt de ce genre de chose est de permettre aux acteurs de valider la mise en place d'un script spécifique sans compromettre l'intégrité du système.

DSM pour des règles métier La démarche de validation basée sur les contraintes permet de mettre en place des mécanismes de vérification formelles. Toutefois, elle peut être considérée comme complexe à mettre en œuvre. Dans nos travaux, nous avons proposé de catégoriser les contraintes mais nous n'avons pas apporté de solution spécifique pour les modéliser dans un langage métier accessible aux experts d'un domaine. Une des perspectives serait de définir un langage spécifique de définition de règles métier en utilisant les DSM. La seconde perspective concerne la définition d'un DSL.

Fusion de modèles La troisième perspective concerne l'alignement de SI et la fusion de modèles. Dans le cas d'acquisition d'entreprise ou de filiale, la mise en place d'un système d'information cohérent est une tâche difficile car elle nécessite une analyse de l'existant afin d'isoler les redondances ou les incohérences entre des systèmes différents. En utilisant une démarche MDA inversée (ascendante), la comparaison des systèmes peut être réalisée à un niveau d'abstraction plus élevé. En l'état, même des transformations ascendantes partielles apporteraient un mécanisme initial de comparaison. Les pré-requis sont bien entendu d'identifier le niveau PSM de chaque SI.

Bibliographie

[AberdeenGroup, 2006] AberdeenGroup (2006). The mechatronic system design benchmark report. Technical report, AberdeenGroup.

[Abid et al., 2014] Abid, H., Pernelle, P., Noterman, D., Benamar, C., and Campagne, J. (2014). Sysml approach for the integration of mechatronics system within plm systems. *International Journal of Computer Integrated Manufacturing IJCIM*.

[Abramovici, 2007] Abramovici, M. (2007). *The Future of Product Development*, chapter Future Trends in Product Lifecycle Management (PLM), pages 665 – 674. Springers.

[Ameri and Dutta, 2005] Ameri, F. and Dutta, D. (2005). Product lifecycle management : closing the knowledge loops. *Computer-Aided Design and Applications*, 2(5) :577 – 590.

[AudrosTechnology, 1993] AudrosTechnology (1993). Site internet d'audros technology. http ://www.audros.fr/cms/easysite/audros.

[Baglin et al., 2005] Baglin, G., Bruel, O., Garreau, A., and Greif, M. (2005). *Management Industriel et Logistique : Conception et pilotage de la Supply Chain*. Economica.

[Barcikowski, 2006] Barcikowski, M. (2006). *Vers une évaluation de la robustesse des connaissances au sein d'une base de connaissances*. PhD thesis, Université Claude Bernard Lyon 1.

[Batenburg et al., 2006] Batenburg, R., Helms, R., and Versendaal, J. (2006). Plm roadmap : stepwise plm implementation based on the concepts of maturity and alignment. *International Journal of Product Lifecycle Management*, pages 333–351.

[Ben Miled, 2011] Ben Miled, A. (2011). *Vers un système de réutilisation des connaissances en ingénierie de conception*. PhD thesis, Université de Technologie de Belfort-Montbéliard.

[Bernard et al., 2009] Bernard, A., Laroche, F., and Da Cunha, Catherine, M. (2009). Models and methods for knowledge formalisation in a plm context. In *3rd International Congress Design and Modelling of Mechanical Systems CMSM'2009*.

[Bertoni et al., 2009] Bertoni, M., Bordegoni, M., Cugini, U., Regazzoni, D., and Rizzi, C. (2009). {PLM} paradigm : How to lead {BPR} within the product development field. *Computers in Industry*, 60(7) :476 – 484.

[Bezivin, 2004] Bezivin, J. (2004). In search of a basic principle for model driven engineering. *European Journal for the Informatics Professional*.

[Bezivin and Gerbe, 2001] Bezivin, J. and Gerbe, O. (2001). Towards a precise definition of the omg/mda framework. In *16th IEEE International Conference on Automated Software Engineering*, San Diego, USA.

[Bissay, 2010] Bissay, A. (2010). *Du déploiement d'un système PLM vers une intégration des connaissances*. PhD thesis, École doctorale de Lyon.

[Blanc, 2005] Blanc, X. (2005). *MDA en action*. Eyrolles.

[Cabot and Teniente, 2007] Cabot, J. and Teniente, E. (2007). Transformation techniques for ocl constraints. *Science of Computer Programming*, 68(3) :152 – 168.

[Cadavid et al., 2011] Cadavid, J., Baudry, B., and Combemale, B. (2011). Empirical evaluation of the conjunct use of mof and ocl. In *Experiences and Empirical Studies in Software Modelling (EESSMod 2011)*, Wellington, New Zealand. CEUR.

[Cariou, 2015] Cariou, E. (2015). La méta-modélisation. http ://ecariou.perso.univ-pau.fr/cours/idm/cours-meta.pdf.

[Catalano et al., 2009] Catalano, C. E., Camossi, E., Ferrandes, R., Cheutet, V., and Sevilmis, N. (2009). A product design ontology for enhancing shape processing in design workflows. *Journal of Intelligent Manufacturing*, 20(5) :553 – 567.

[Cauvet and Rosenthal-Sabroux, 2001] Cauvet, C. and Rosenthal-Sabroux, C. (2001). *Ingénierie des systèmes d'information*. HERMES Science Publications.

[Chandrasegaran et al., 2013] Chandrasegaran, S. K., Ramani, K., Sriram, R. D., Horváth, I., Bernard, A., Harik, R. F., and Gao, W. (2013). The evolution, challenges, and future of knowledge representation in product design systems. *Computer-aided design*, 45(2) :204 – 228.

[Cheballah, 1992] Cheballah, K. (1992). *Aides à la gestion des données techniques des produits industriels*. PhD thesis, Ecole Centrale de Lyon.

[Cherif, 2013] Cherif, S. (2013). *A model driven based approach for the design of dynamically reconfigurable systems : from MARTE to RECOMARTE*. PhD thesis, Université des Sciences et Technologie de Lille - Lille I.

[CIMdata, 2003] CIMdata (2003). Product lifecycle management «empowering the future of business». Technical report, CIMdata.

[CIMdata, 2005] CIMdata (2005). Plm for mid-sized manufacturing enterprises. http ://www.softech.com/media/pdf/ProductCenter-WP_Analyst_Whitepaper.pdf.

[CIMdata, 2015] CIMdata (2015). Site internet sur le plm. http ://www.cimdata.com/.

[Danesi et al., 2008] Danesi, F., Gardan, N., Gardan, Y., and Reimeringer, M. (2008). P4lm : A methodology for product lifecycle management. *Computers in Industry*, 59 :304 – 317.

[De Fontaine, 2014] De Fontaine, I. (2014). *Pilotage des innovations d'ingénierie par la valeur : une voie d'amélioration pour l'ingénierie des aéronefs*. PhD thesis, Grenoble INP.

[Debaecker, 2004] Debaecker, D. (2004). *PLM la gestion collaborative du cycle de vie des produits : Product Life-Cycle Management*. Hermès - Lavoisier, Paris.

[Dutta and Wolowicz, 2005] Dutta, D. and Wolowicz, J. P. (2005). An introduction to product lifecycle management (plm). In *12th ISPE International Conference on Concurrent Engineering : Research and Applications*, Worth/Dallas, USA.

[El Kadiri, 2009] El Kadiri, S. (2009). *Management des processus collaboratifs dans les systèmes PLM*. PhD thesis, Université Lumière Lyon 2.

[Ermine, 2003] Ermine, J. (2003). *La gestion des connaissances*. Hermès Science.

[Eynard, 2005] Eynard, B. (2005). *Gestion du cycle de vie des produits et dynamique des connaissances industrielles en conception intégrée*. Habilitation à diriger des recherches, Université de technologie de Troyes.

[Eynard et al., 2006] Eynard, B., Gallet, T., Roucoules, L., and Ducellier, G. (2006). Pdm system implementation based on uml. *Mathematics and Computers in Simulation*, 70(5-6) :330–342.

[Favre et al., 2006] Favre, J.-M., Estublier, J., and Blay-Fornarino, M. (2006). *L'ingénierie dirigée par les modèles. Au-delà du MDA*. Hermès.

[Felic et al., 2014] Felic, A., König-Ries, B., and Klein, M. (2014). Process-oriented semantic knowledge management in product lifecycle management. *Procedia CIRP*, 25 :361 – 368.

[Fenves et al., 2005] Fenves, S. J., Sriram, R. D., Subrahmanian, E., and Rachuri, S. (2005). Product information exchange : Practices and standards. *Journal of Computing and Information Science in Engineering*, 5(3) :238 – 246.

[Foundation, 2015] Foundation, E. (2015). Eclipse.

[García-Magarino et al., 2009] García-Magarino, I., Fuentes-Fernández, R., and Gómez-Sanz, J. J. (2009). Guideline for the definition of emf metamodels using an entity-relationship approach. *Information and Software Technology*, 51(8) :1217 – 1230.

[Gomes, 2008] Gomes, S. (2008). *Ingénierie à base de connaissances pour une conception productive, optimisée et innovante du système Projet-Produit-Process-Usage*. Habilitation à diriger des recherches, Université de Technologie de Belfort-Montbéliard.

[Gzara, 2000] Gzara, L. (2000). *Les patterns pour l'ingénierie des systèmes d'information produit*. PhD thesis, INP Grenoble.

[Gzara et al., 2003] Gzara, L., Rieu, D., and Tollenaere, M. (2003). Product information systems engineering : an approach for building product models by reuse of patterns. *Robotics and Computer-Integrated Manufacturing*, 19(3) :239 – 261.

[Harani, 1997] Harani, Y. (1997). *Une approche multi-modèles pour la capitalisation des connaissances dans le domaine de la Conception*. PhD thesis, Grenoble INP.

[Izadpanah, 2011] Izadpanah, S. H. (2011). *A pattern based approach for the evolution of PLM tools in the extended enterprise*. PhD thesis, INP Grenoble.

[Jouault et al., 2008] Jouault, F., Allilaire, F., Bézivin, J., and Kurtev, I. (2008). Atl : A model transformation tool. *Science of Computer Programming*, 72(1–2) :31 – 39.

[Jouault and Kurtev, 2005] Jouault, F. and Kurtev, I. (2005). Transforming models with atl. In *Proceedings of the 2005 International Conference on Satellite Events at the MoDELS*, MoDELS'05, pages 128 – 138, Berlin, Heidelberg. Springer-Verlag.

[Kleppe et al., 2003] Kleppe, A. G., Warmer, J., and Bast, W. (2003). *MDA Explained : The Model Driven Architecture : Practice and Promise*. Addison-Wesley Longman Publishing Co., Inc.

[Krima, 2013] Krima, S. (2013). *Semantic based framework for dynamic customization of PLM-related information models.* PhD thesis, Université de Bourgogne.

[Lalit et al., 2005] Lalit, P., Debasish, D., and Ram, S. (2005). Ontology formalization of product semanticsfor product lifecycle management. Technical report, NIST.

[Lamouri, 2006] Lamouri, S. (2006). *Synchronisation des prises de décisions dans une chaîne logistique : robustesse et stabilité.* Habilitation à diriger des recherches, Ecole Doctorale d'Informatique et Electronique de Paris.

[Le Duigou, 2010] Le Duigou, J. (2010). *Framework for PLM systems in extended enterprise. Application on mechanical SMEs.* PhD thesis, Ecole Centrale de Nantes (ECN).

[Le Moigne, 1977] Le Moigne, J.-L. (1977). *La théorie du Système Général : théorie de modélisation.* P.U.F.

[Le Moigne, 1990] Le Moigne, J.-L. (1990). *La modélisation des systèmes complexes.* Dunod.

[Liu et al., 2009] Liu, W., Zeng, Y., Maletz, M., and Brisson, D. (2009). Product lifecycle management : a survey. In *Proceedings of the ASME 2009 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, San Diego, California, USA.

[Luh et al., 2011] Luh, Y.-P., Pan, C.-C., and Chu, C.-H. (2011). A hierarchical deployment of distributed product lifecycle management system in collaborative product development. *International Journal of Computer Integrated Manufacturing*, 24(5) :471–483.

[Mahdjoub et al., 2010] Mahdjoub, M., Monticolo, D., Gomes, S., and Sagot, J.-C. (2010). A collaborative design for usability approach supported by virtual reality and a multi-agent system embedded in a plm environment. *Computer-Aided Design*, 42(5) :402 – 413.

[Marin, 2009] Marin, P. (2009). *L'usage des systèmes d'information PLM (Product Lifecycle Management) contribue t-il a l'innovation collaborative.* PhD thesis, HEC Paris.

[Matsokis and Kiritsis, 2010] Matsokis, A. and Kiritsis, D. (2010). An ontology based approach for product lifecycle management. *Computer in Industries*, 61(8) :787 – 797.

[Maurino, 1993] Maurino, M. (1993). *La gestion des données techniques technologie du concurrent engineering*, page 180. Springer.

[Minsky, 1965] Minsky, M. (1965). Matter, mind and models.

[Minsky, 1969] Minsky, M. L. (1969). *Semantic Information Processing.* The MIT Press.

[Monestel et al., 2002] Monestel, L., Ziadi, T., and Jézéquel, J.-M. (2002). Product line engineering : Product derivation. In *Workshop on Model Driven Architecture and Product Line Engineering, associated to the SPLC2 conference*, San Diego, United States.

[Monticolo et al., 2008] Monticolo, D., Hilaire, V., Gomes, S., and Koukam, A. (2008). A multi-agent system for building project memories to facilitate the design process. *Integrated Computer-Aided Engineering*, 15(1) :3 – 20.

[Muller, 2006] Muller, P.-A. (2006). *De la modélisation objet des logiciels à la metamodélisation des langages informatiques.* PhD thesis, Université Rennes 1.

[Noël and Roucoules, 2008] Noël, F. and Roucoules, L. (2008). The ppo design model with respect to digital enterprise technologies among product life cycle. *International Journal of Computer Integrated Manufacturing*, 21(2) :139 – 145.

[Nollet et al., 1994] Nollet, J., Kélada, J., and Diorio, M. (1994). *La gestion des opérations et de la production : une approche systémique*. G. Morin.

[OMG, 2001] OMG (2001). Omg model driven architecture [online]. http ://www.omg.org/mda/.

[OMG, 2005] OMG (2005). Uml version 2.0 [online]. http ://www.omg.org/spec/UML/2.0/.

[OMG, 2011a] OMG (2011a). Business process model and notation (bpmn), version 2.0[online]. http ://www.omg.org/spec/BPMN/.

[OMG, 2011b] OMG (2011b). Meta object facility (mof)[online]. http ://www.omg.org/spec/MOF/2.4.1/.

[OMG, 2014] OMG (2014). Object constraint language omg available specification version [online]. http ://www.omg.org/spec/OCL/2.4/.

[Oussalah, 1997] Oussalah, C. (1997). *Ingénierie objet : concepts et techniques*. InterEditions.

[Paviot, 2010] Paviot, T. (2010). *A Methodology for solving interoperability problems in the field of Product Lifecycle Management*. PhD thesis, Ecole Centrale Paris.

[Pernelle, 2002] Pernelle, P. (2002). *Système d'information Produit pour le PME/PMI : modélisation multi-niveaux d'entreprises engagées dans un travail coopératif*. PhD thesis, Ecole supérieure d'ingénieur d'Annecy.

[Pinel, 2013] Pinel, Muriel, M. (2013). *Product Lifecycle Management (PLM) in the subcontracting industry as a key for operational agility and product management*. PhD thesis, Université de Grenoble.

[Randoing, 1995] Randoing, J.-M. (1995). *Les SGDT*. Hermes Science.

[Reix, 2002] Reix, R. (2002). *Systeme d'information et management des organisations*. Vuibert.

[Rivière, 2004] Rivière, A. (2004). *Gestion de configuration et des modifications lors du développement de grands produits complexes en ingénierie concourante*. PhD thesis, Université de Grenoble.

[Rolland et al., 1988] Rolland, C., Foucaut, O., and Bency, G. (1988). *Conception des systèmes d'information*. Eyrolles.

[Roques and Vallée, 2000] Roques, P. and Vallée, F. (2000). *UML en action : de l'analyse des besoins à la conception en Java*. Eyrolles.

[Saaksvuori and Immonen, 2008] Saaksvuori, A. and Immonen, A. (2008). *Product lifecycle management*. Springer Science & Business Media.

[Schuh et al., 2008] Schuh, G., Assmus, D., and Zancul, E. (2008). Product structuring - the core discipline of product lifecycle management. *Computers in Industry*, pages 210 – 218.

[Si-Said Cherfi et al., 2012] Si-Said Cherfi, S., Ayad, S., and Comyn-Wattiau, I. (2012). Aligning business process models and domain knowledge : A meta-modeling approach. In *European Conference on Advances in Databases and Information Systems*, volume 186, pages 45 – 56, Poznan, Poland. Springer.

[Siemens, 2015] Siemens (2015). Industrie 4.0 – the fourth industrial revolution. http ://www.industry.siemens.com/topics/global/en/digital-enterprise-platform/videos/pages/video_industrie4.aspx.

[Silventoinen et al., 2009] Silventoinen, A., Papinniemi, J., and Lampela, H. (2009). A roadmap for product lifecycle management implementation in smes. In *ISPIM Conference*, pages 21 – 24.

[Softech, 2015] Softech (2015). Site inernet sur le plm. http ://www.softech.com.

[Sohlenius, 1992] Sohlenius, G. (1992). Concurrent engineering. *Cirp Annals-manufacturing Technology*, 41 :645 – 655.

[Srinivasan, 2009] Srinivasan, V. (2009). *Advanced Design and Manufacturing Based on STEP*, chapter STEP in the context of Product Data Management. Springer.

[Stark, 2005] Stark, J. (2005). *Product Lifecycle Management : 21st Century Paradigm for Product Realisation*. Decision engineering. Springer.

[Sudarsan et al., 2005] Sudarsan, R., Fenves, S., Sriram, R., and Wang, F. (2005). A product information modeling framework for product lifecycle management. *Computer-Aided Design*, 37(13) :1399 – 1411.

[Terzi, 2005] Terzi, S. (2005). *Elements of Product Lifecycle Management : Definitions, Open Issues and Reference Models*. PhD thesis, Université Henri Poincaré - Nancy I.

[Terzi et al., 2008] Terzi, S., Ball, P., Bouras, A., Dutta, D., Garetti, M., Gurumoorthy, B., Han, S., and Kiritsis, D. (2008). A new point of view on product lifecycle management. In *International Conference on Product Lifecycle Management, PLM 2008, University in Seoul, South Korea, July 2008*.

[Thimm et al., 2006] Thimm, G., Lee, S. G., and Ma, Y. S. (2006). Towards unified modelling of product life-cycles. *Computer in Industries*, 57(4) :331–341.

[Tichkiewitch, 1994] Tichkiewitch, S. (1994). De la cfao à la conception intégrée. *Revue internationale de CFAO et d'infographie*, 9(5) :609 – 621.

[Warmer, 2003] Warmer, J. (2003). The role of ocl in the model driven architecture. In *Revised Papers from the International Workshop on Scientific Engineering for Distributed Java Applications*, FIDJI '01, London, UK. Springer-Verlag.

[WFMC, 1996] WFMC (1996). Workflow management coalition terminology and glossary (wfmc-tc-1011). Technical report, Workflow Management Coalition, Brussels.

[Ziadi et al., 2003] Ziadi, T., Jézéquel, J.-M., and Fondement, F. (2003). Product line derivation with uml. In *Proceedings Software Variability Management Workshop, Univ. of Groningen Departement of Mathematics and Computing Science*, Rennes, France.

[Zina et al., 2006] Zina, S., Lombard, M., Lossent, L., and Henriot, C. (2006). Generic modeling and configuration management in product lifecycle management. In *Computational Engineering in Systems Applications, IMACS Multiconference on*, pages 1252 – 1258.

[Zouari, 2007] Zouari, A. (2007). *Proposition de mécanismes de versionnement et d'agrégation des connaissances de domaine en conception collaborative de produit industriels.* PhD thesis, Grenoble INP et ENIS Sfax.

[Zouari et al., 2015] Zouari, A., Tollenaere, M., Ben Bacha, H., and Maalej, A. Y. (2015). Domain knowledge versioning and aggregation mechanisms in product design processes. *Concurrent Engineering : Research and Applications*, pages 1 – 12.

Annexe A

ACMS : Organisation dans l'environnement Eclipse

Les différents composants de l'atelier de modélisation et de conception de système d'information (ACMS) ont été implémentés sous forme de plugin Eclipse. Les plugins développés sont généralement de deux types :

- des plugins définissant le noyau de la logique des fonctionnalités
- des plugins pour contribuer à l'interface graphique de la plateforme Eclipse afin de permettre à l'utilisateur d'atteindre ces fonctionnalités

Le déploiement des plugins implémentés se fait en les regroupant dans des Features. Cette méthode consiste à regrouper différents plugins développés et de les déployer vers un site local. Il est ensuite possible de les installer sur une distribution d'Eclipse. ACMS est un environnement construit sur l'infrastructure Eclipse et est très modulaire. En effet, les fonctionnalités offertes par ce prototype sont embarquées dans des plugins qui peuvent être montés ou démontés selon notre choix. Attention, des dépendances peuvent exister entre les plugins. Le projet ACMS est basé sur deux features :

1. le premier regroupe les différents plugins de création du méta-modèle pour les systèmes PLM (cf chapitre 3) et l'interface graphique pour la création des modèles.

2. le deuxième regroupe les différents plugins du méta-modèle Audros, des transformations Model to Model (M2M) et Model to text (M2T) et la persistance en base de données.

A.1 Présentation des projets du premier Feature

La figure (fig. A.1) et le tableau (tab. A.1) illustrent et décrivent le premier feature : le projet de création du méta-modèle pour les systèmes PLM et l'interface graphique pour la création des modèles.

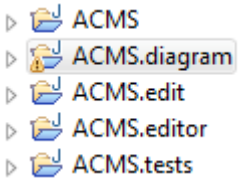

FIGURE A.1 – Liste des projets dans Eclipse pour le premier feature

Projet	Description
ACMS	Contient la définition du méta-modèle pour les systèmes PLM et fournit les interfaces et la description de chaque objet et l'implémentation des méthodes de chaque objet créé dans le méta-modèle.
ACMS.diagramme	Englobe toutes les fonctionnalités modifiable à propos de l'interface de création des modèles, de la palette de création et des objets qui existent dans le modèle.
ACMS.edit	Fournit les éléments du méta-modèle en aspect graphique.
ACMS.editor	Fournit une présentation générale du plugin et son initialisation.
ACMS.tests	Regroupe les tests unitaires de création de modèle.

TABLE A.1 – Description des différents projets du premier feature pour ACMS

A.2 Présentation des projets du deuxième Feature

La figure (fig. A.2) et le tableau (tab. A.2) illustrent et décrivent le deuxième feature : le projet de création du méta-modèle Audros, les projets pour la réalisation des transformations et de la persistance des données.

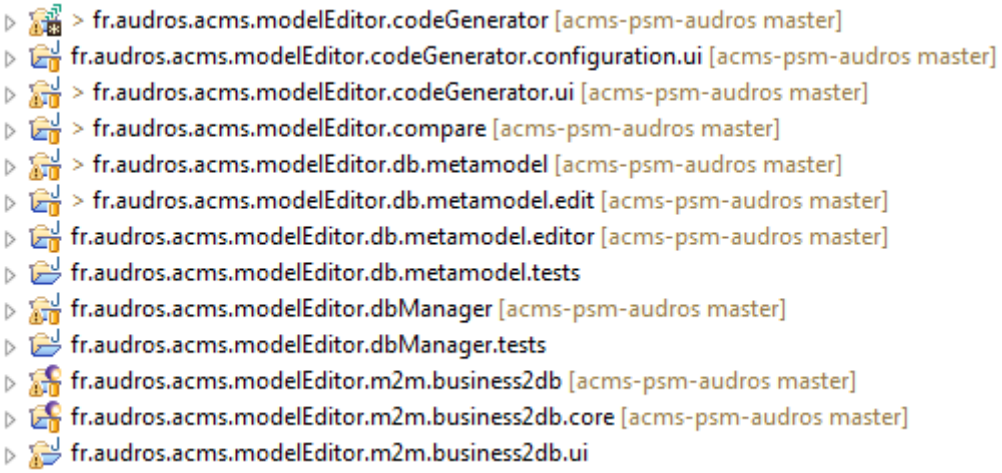

FIGURE A.2 – Liste des projets dans Eclipse pour le deuxième feature

Projet	Description
fr.audros.acms.modelEditor .codeGenerator	Encapsule le moteur de transformation et les différents templates Acceleo de transformation M2T.
fr.audros.acms.modelEditor .codeGenera- tor.configuration.ui	Ajoute la fonctionnalité de configuration de la connexion à la base de données.
fr.audros.acms.modelEditor .codeGenerator.ui	Ajoute une contribution au menu contextuel permettant de lancer la transformation Acceleo et de persister le modèle métier dans la base de données.
fr.audros.acms.modelEditor .compare	Offre les fonctionnalités de comparaison et de gestion des différentes versions des modèles métiers.
fr.audros.acms.modelEditor .db.metamodel	Contient l'implémentation EMF du méta-modèle Audros.
fr.audros.acms.modelEditor .db.metamodel.edit	Contient l'infrastructure de manipulation des concepts définis dans le méta-modèle (noyau de l'éditeur des modèles Audros).
fr.audros.acms.modelEditor .db.metamodel.editor	Contient le code associé à l'UI de l'éditeur des modèles Audros.
fr.audros.acms.modelEditor .db.metamodel.tests	Regroupe les tests unitaires de création de modèle.
fr.audros.acms.modelEditor .dbManager	Une bibliothèque de gestion de la base de données (test de connexion, paramètre de connexion...).
fr.audros.acms.modelEditor .dbManager.tests	Regroupe les tests unitaires de connexion à la BD.
fr.audros.acms.modelEditor .m2m.business2db	Contient le moteur de transformation et les modules ATL de transformation des modèles PLM en modèles de Audros.
fr.audros.acms.modelEditor .m2m.business2db.ui	Encapsule le code de contribution au menu contextuel permettant de lancer la transformation ATL.

TABLE A.2 – Description des différents projets du deuxième feature pour ACMS

Annexe B

ACMS : Création d'un modèle via l'interface graphique

Pour utiliser le prototype que nous avons développé, il faut une distribution d'Eclipse et installer les features du projet ACMS.

B.1 Création d'un nouveau modèle

La création d'un modèle doit être précédée par la création d'un projet Eclipse. En effet, le modèle doit appartenir à un projet. Ci-dessous, vous trouverez la description des différentes étapes de création d'un modèle avec ACMS :

1. Cliquer sur le menu File -> New -> Project. Sélectionner 'Project' dans le dossier 'General' et cliquer sur 'Next'. Saisir le nom du projet et cliquer sur 'Finish' :

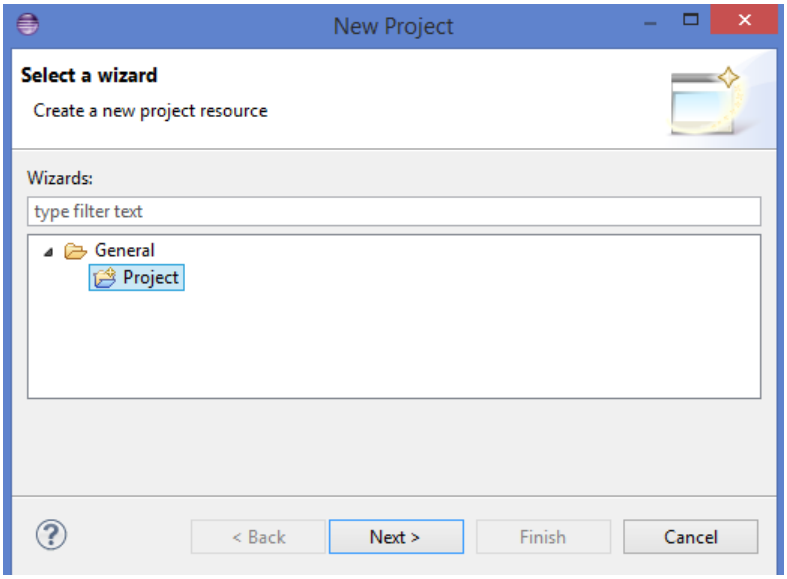

FIGURE B.1 – Création d'un projet dans Eclispe

2. Faire un clic droit dans le projet créé et aller dans New -> Other... :

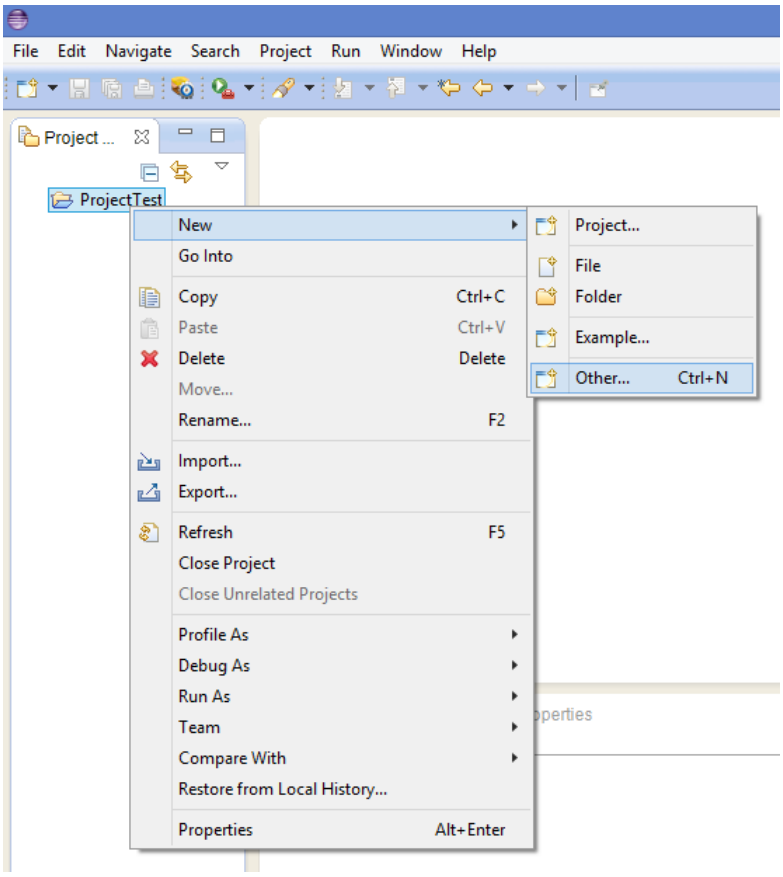

FIGURE B.2 – Création d'un modèle ACMS dans un projet (1)

3. Sélectionner dans Example 'Acms diagram' puis cliquer sur 'Next' :

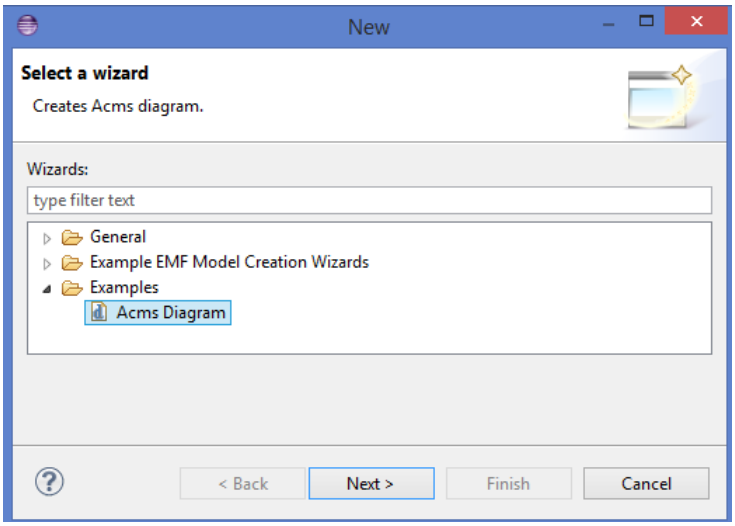

FIGURE B.3 – Création d'un modèle ACMS dans un projet (2)

4. Donner un nom au modèle et cliquer sur 'Next' puis sur 'Finish'.

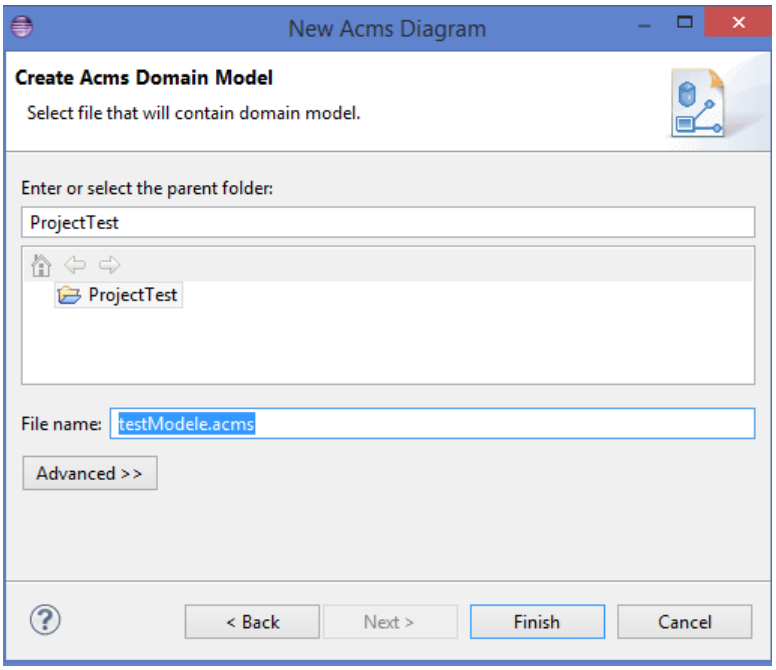

FIGURE B.4 – Création d'un modèle ACMS dans un projet (3)

5. Sélectionner le type de modèle à créer (Objet ou utilisateur), dans notre exemple nous choisissons 'modèle Objet' :

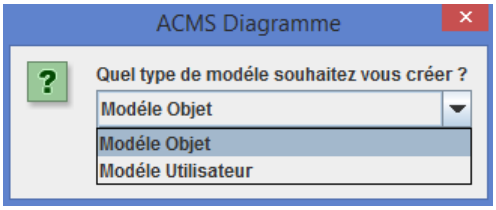

FIGURE B.5 – Création d'un modèle ACMS dans un projet (4)

6. Vous trouverez dans l'explorateur de fichier (à gauche) le modèle créé, l'espace de travail (au centre) pour la création du modèle et une palette de création (à droite) :

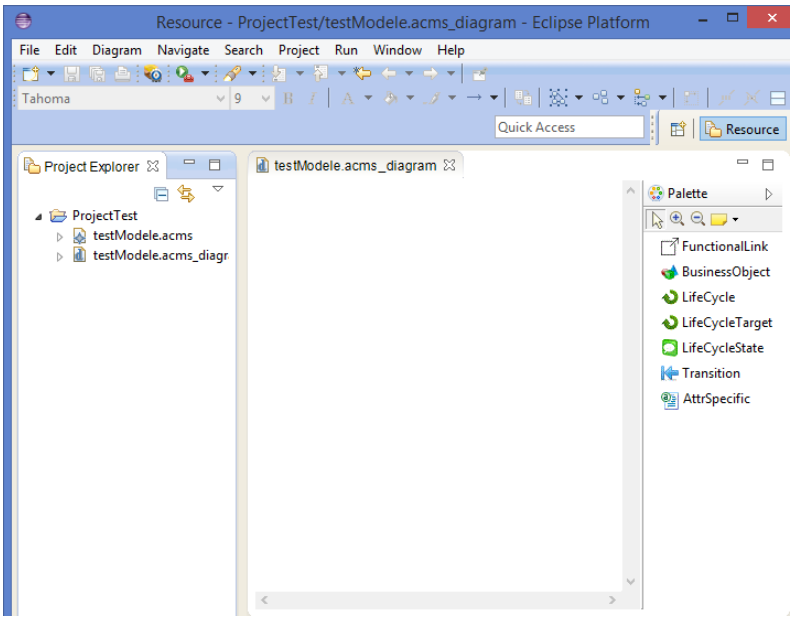

FIGURE B.6 – Interface de création d'un modèle

B.2 Création et suppression des objets

1. Pour créer un objet, vous devez sélectionner dans la palette de création l'élément 'BuinessObject' puis faire un clic sur l'espace de travail (au centre), une boîte de dialogue apparaît pour vous demander le nom de l'objet que vous souhaitez créer :

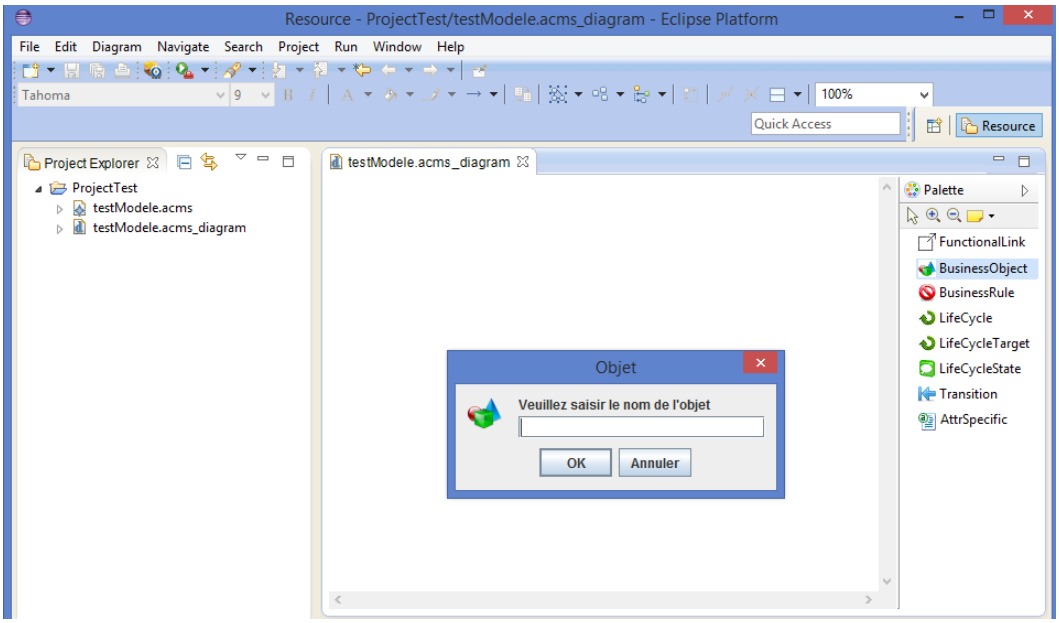

FIGURE B.7 – Création d'un 'BusinessObject' dans ACMS

2. Voici la présentation d'un modèle avec deux objets. La création d'un objet engendre

la création automatique de 10 attributs systèmes (en lecture seule) qu'il est possible de cacher ou d'afficher selon ses préférences :

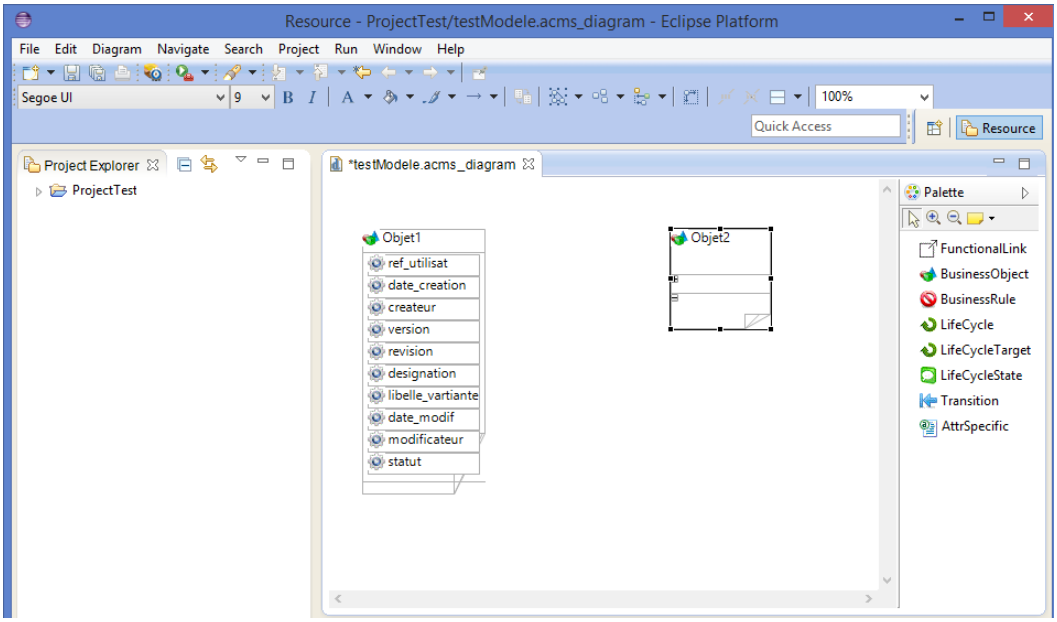

FIGURE B.8 – Présentation des 'BusinessObject' dans ACMS

3. Un contrôle sur le nom des objets permet de maîtriser l'unicité. Il est impossible de créer deux objets ayant le même identifiant (c'est à dire le même nom) :

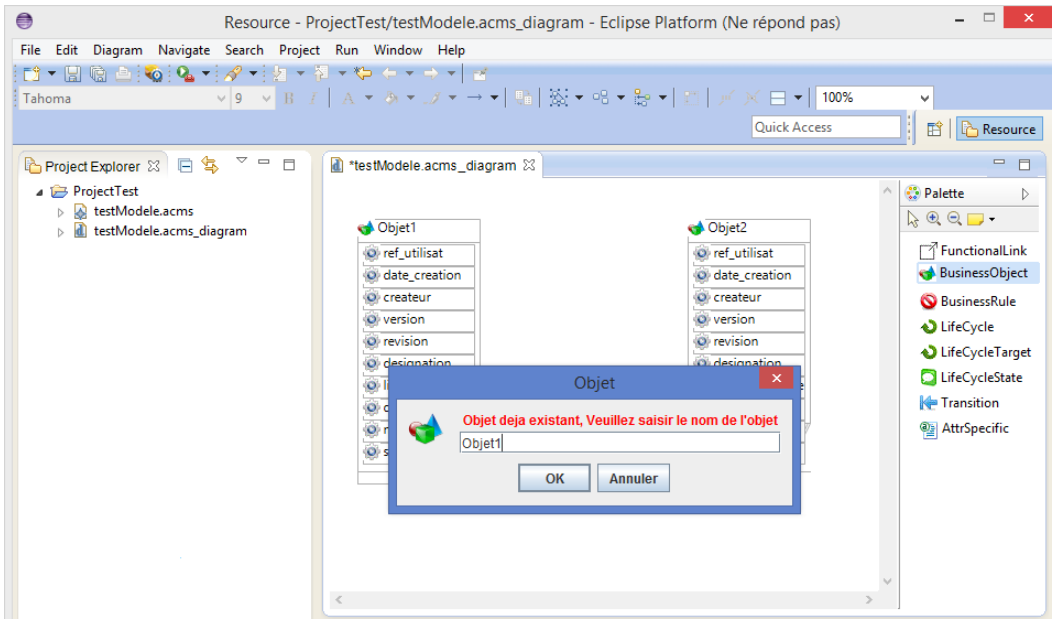

FIGURE B.9 – Contrôle de l'unicité sur les 'BusinessObject'

4. Pour ajouter des attributs spécifiques sur un objet, vous devez sélectionner l'élément 'AttrSpec' dans la palette de création et cliquer dans l'espace de travail sur l'entête

de l'objet sur lequel vous souhaitez l'ajouter. Vous pouvez afficher ou cacher les attributs spécifiques de la même manière que les attributs systèmes :

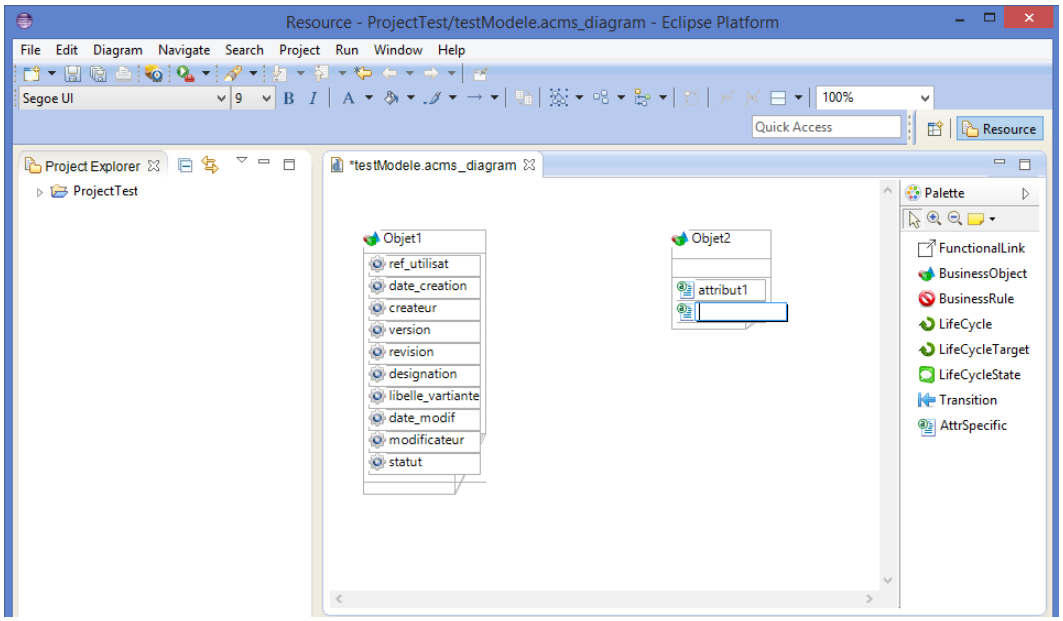

FIGURE B.10 – Ajouter des attributs spécifiques sur un objet

5. Pour modifier un attribut spécifique, il suffit de faire un clic droit sur l'attribut souhaité et choisir 'Edit' pour effectuer la modification. Une autre solution consiste à faire un double clic sur l'attribut pour effectuer la modification.
6. Pour supprimer un attribut spécifique, il suffit de faire un clic droit sur l'attribut souhaité et choisir 'Delete from Model'. Une autre solution consiste à sélectionner l'attribut avec un clic gauche et d'appuyer sur le bouton 'Suppr' du clavier :

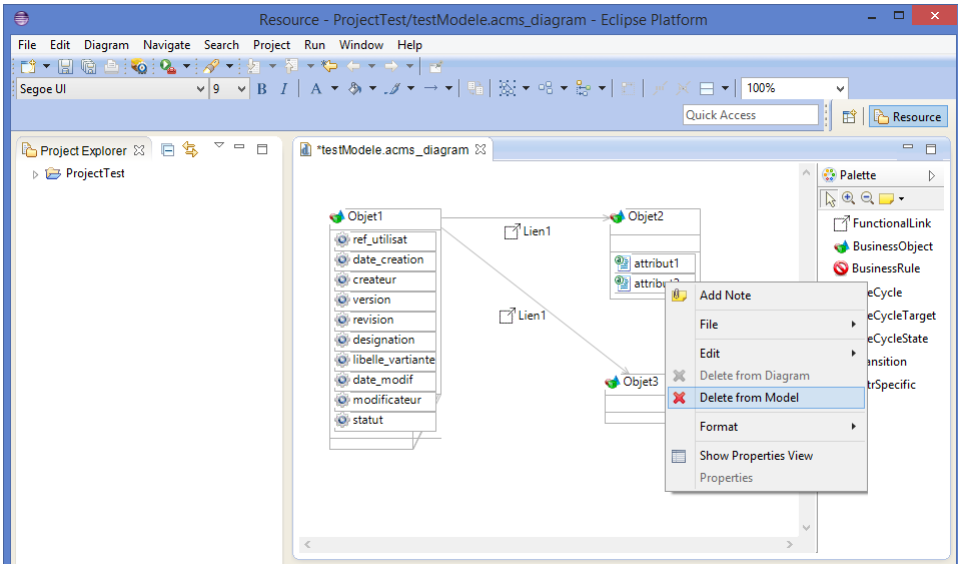

FIGURE B.11 – Supprimer des attributs spécifiques sur un objet

7. Pour supprimer un objet, vous devez faire un clic droit sur l'objet souhaité et choisir 'Delete from Model'. Une autre solution consiste à sélectionner l'objet avec un clic gauche et d'appuyer sur le bouton 'Suppr' du clavier.

B.3 Création et suppression des liens

1. Pour créer un lien entre deux objets, vous devez sélectionner l'élément 'FunctionnalLink' dans la palette de création puis cliquer dans l'espace de travail sur l'entête du premier objet (source) et vous diriger vers l'entête du deuxième objet (cible) pour créer le lien. L'application vous propose d'utiliser un lien déjà existant (utile dans le cas où des liens ont déjà été définis) ou de créer un nouveau lien :

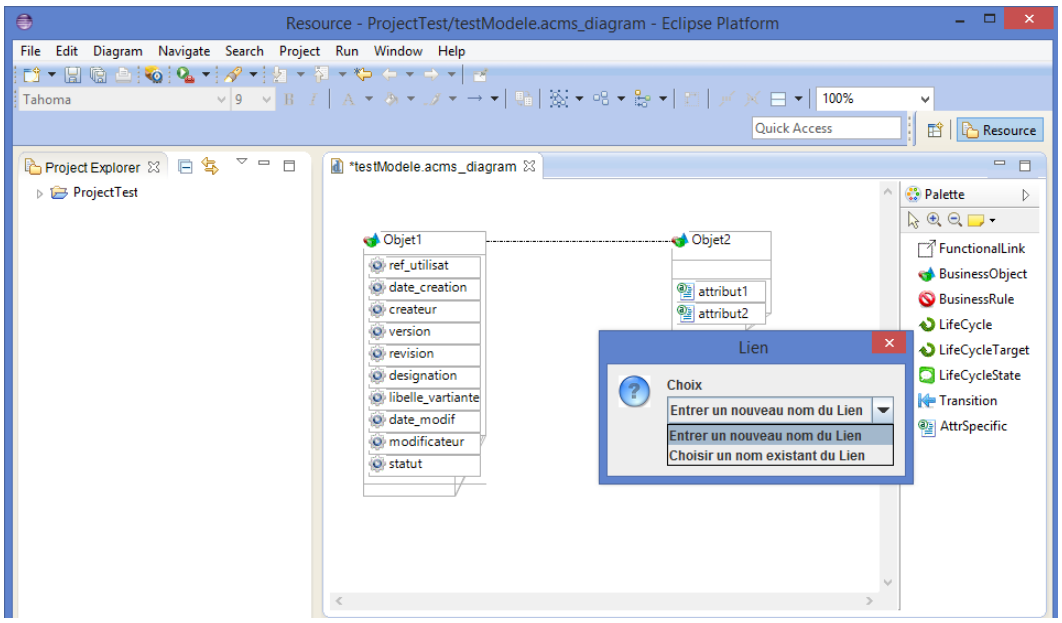

FIGURE B.12 – Création d'un 'FunctionnalLink' entre deux objets dans ACMS

2. Dans le cas d'une nouvelle création, saisir le nom du lien et valider :

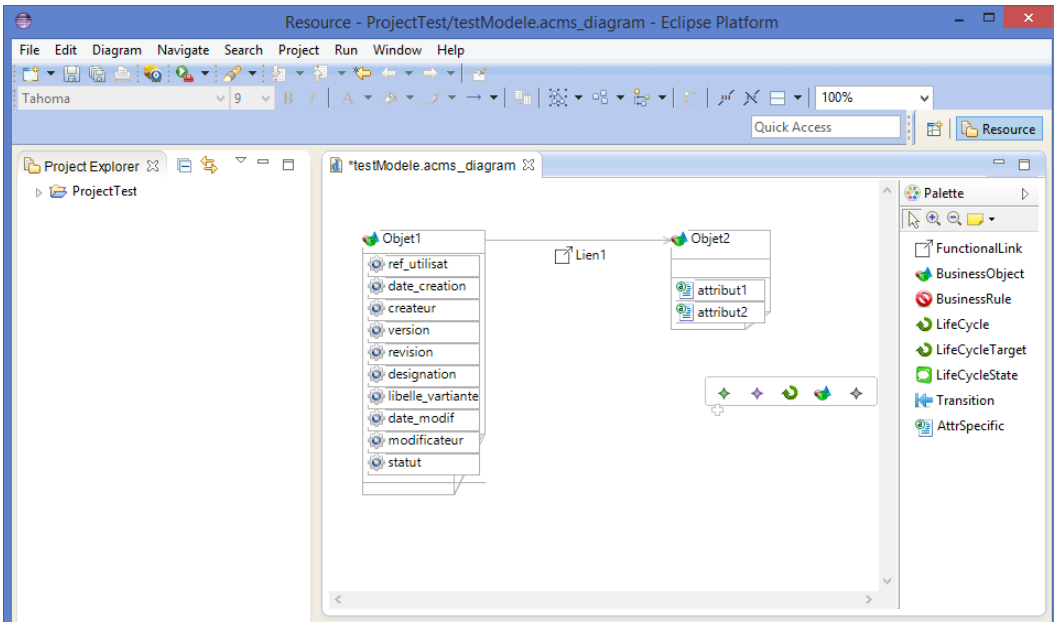

FIGURE B.13 – Présentation d'un 'FunctionnalLink' avec ACMS

3. Dans le cas d'un clic sur le choix 'Choisir un nom existant de lien', l'application vous propose de choisir parmi une liste de liens déjà créée :

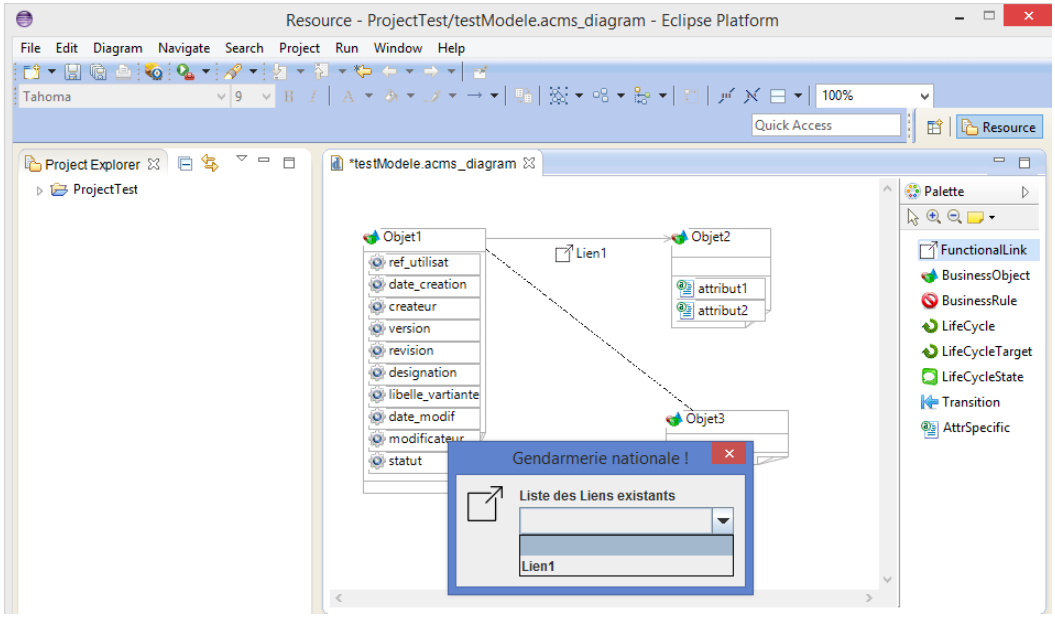

FIGURE B.14 – Création d'un 'FunctionnalLink' avec la définition d'un lien déjà existant

4. Pour ajouter, modifier ou supprimer un attribut spécifique d'un lien, vous devez faire un double clic sur le lien souhaité. Une nouvelle interface est affichée. Elle permet de gérer les opérations sur les attributs spécifiques des liens. A noter que toute création de lien engendre la création de 5 attributs systèmes (en lecture seule) :

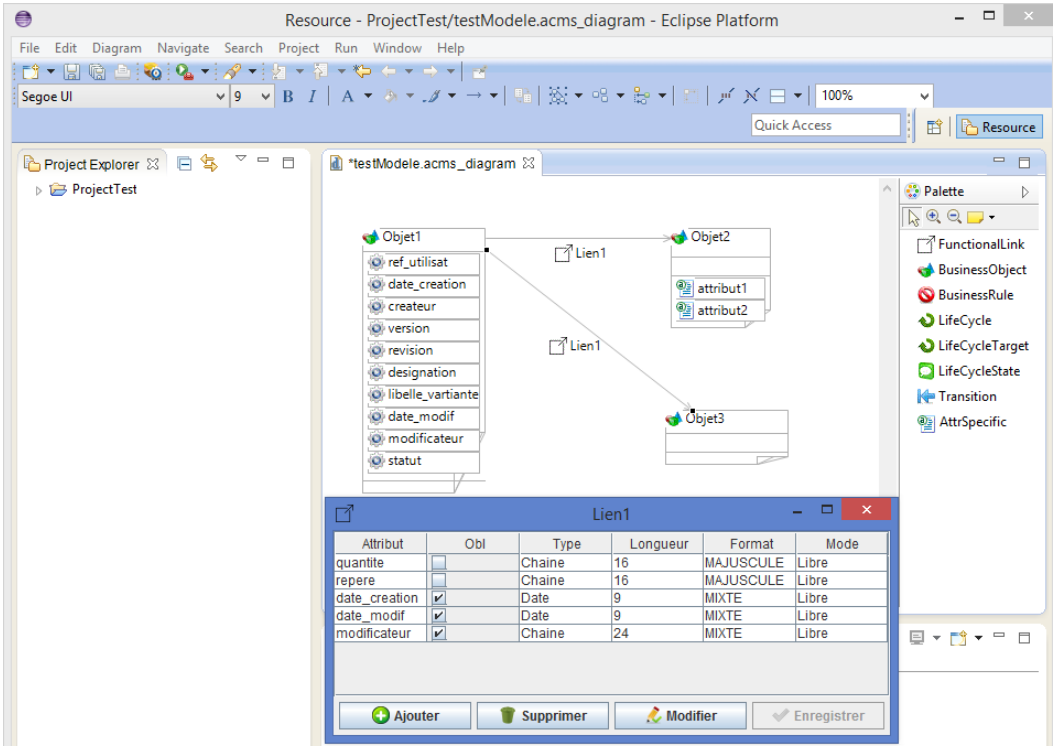

FIGURE B.15 – Gestion des attributs spécifiques des liens

5. Pour supprimer un lien, il suffit de faire un clic droit sur le lien souhaité et choisir 'Delete from Model'. Une autre solution consiste à sélectionner le lien avec un clic gauche et d'appuyer sur le bouton 'Suppr' du clavier.

B.4 Création et suppression des cycles de vie

1. Pour créer un cycle de vie, vous devez sélectionner l'élément 'LifeCycle' dans la palette de création et cliquer dans l'espace de travail de l'application. Donner un nom à votre cycle de vie :

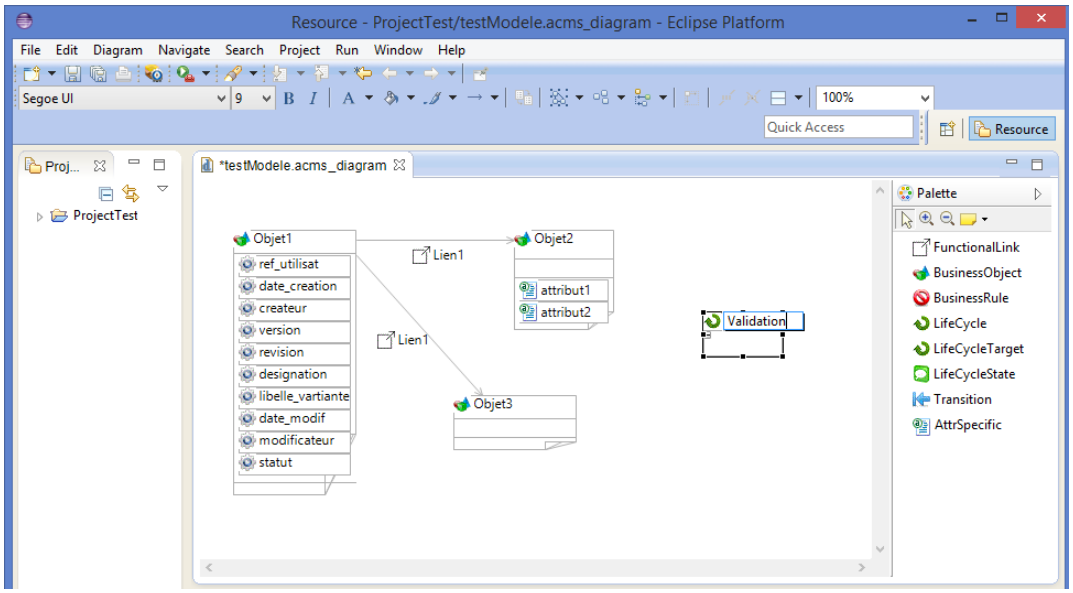

FIGURE B.16 – Création d'un cycle de vie

2. Pour ajouter un état dans le cycle de vie, il faut sélectionner l'élément 'LifeCycleState' dans la palette de création et cliquer ensuite dans l'espace du cycle de vie souhaité. Donner un nom à votre état :

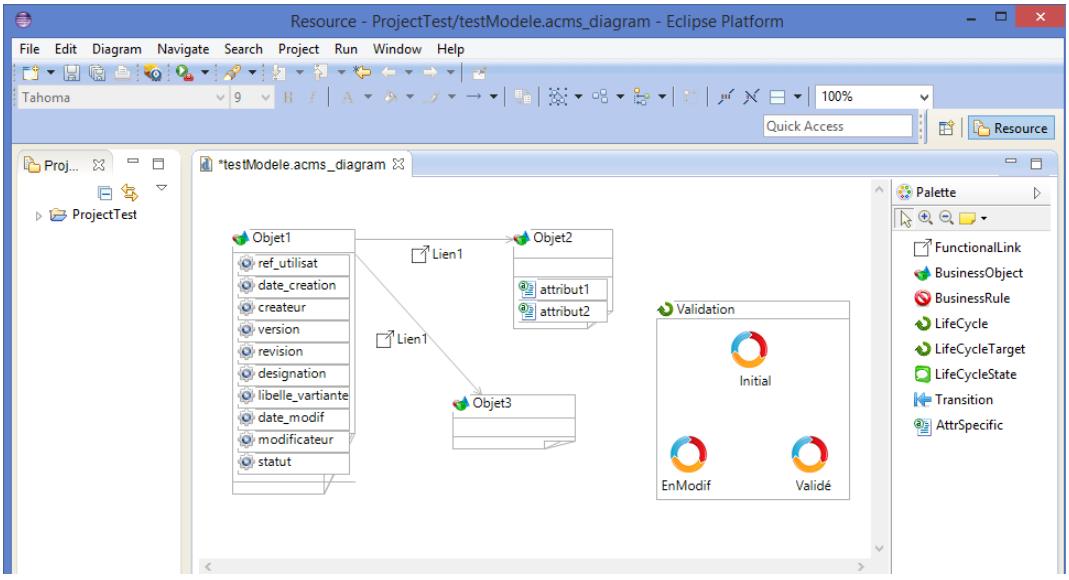

FIGURE B.17 – Création des états d'un cycle de vie

3. Pour la création d'une transition entre deux états d'un cycle de vie, il faut sélectionner l'élément 'Transition' dans la palette et partir d'un état source vers un état cible (identique à la création d'un lien entre deux objets) :

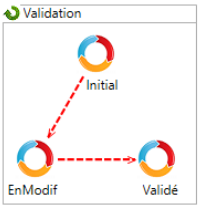

FIGURE B.18 – Création des transitions entre les états

4. Pour associer un cycle de vie à un objet, il suffit de sélectionner dans la palette de création l'élément 'LifeCycleTarget' et partir du cycle de vie vers un objet :

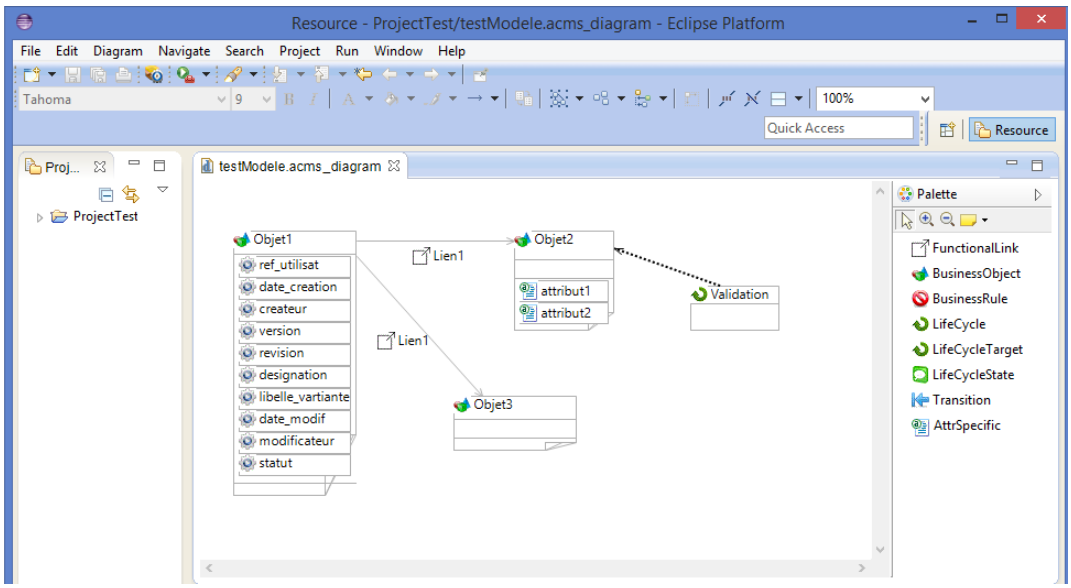

FIGURE B.19 – Association d'un cycle de vie à un objet

5. La suppression d'un cycle de vie, d'un état, d'une transition ou d'une association entre un cycle de vie et un objet se fait de la même façon que pour les objets et les liens.

B.5 Gestion des contraintes

1. Pour créer une contrainte, il suffit de sélectionner dans la palette de création l'élément 'BusinessRule' et cliquer dans l'espace de travail, une boîte de dialogue s'affiche pour gérer les contraintes sur le modèle :

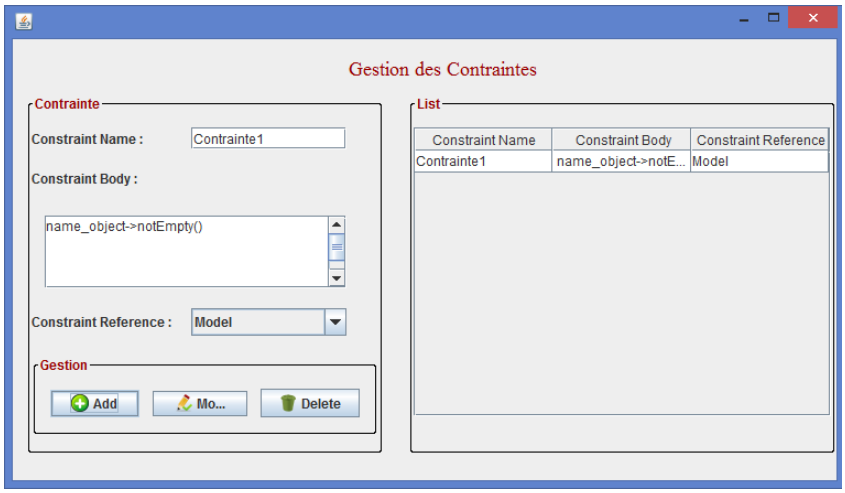

FIGURE B.20 – Création d'une contrainte avec ACMS

2. Il est nécessaire de donner un nom à la contrainte et de saisir la définition de la contrainte à l'aide du langage OCL. C'est le code OCL qui détermine la porté de la contrainte (objet, lien...) :

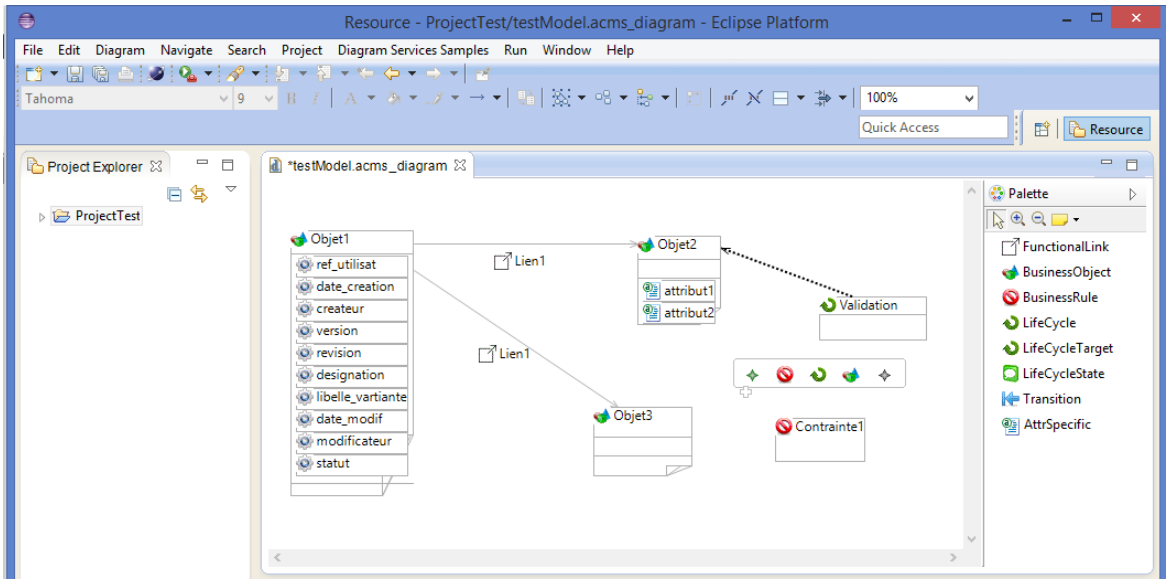

FIGURE B.21 – Présentation des contraintes dans le modèle

3. La modification et la suppression d'une contrainte se gère par cette même boîte de dialogue. Pour l'afficher vous devez double cliquer sur n'importe quel contrainte du modèle.

Annexe C

ACMS : Les règles de transformations et la persistance des données

C.1 Les transformations M2M et M2T

La transformation M2M repose sur un module ATL qui fait la correspondance entre les éléments des méta-modèles source (PLM) et cible (Audros). Le module ATL de transformation business2db.atl est situé dans le package fr.audros.acms.modelEditor.m2m.business2db.files du projet fr.audros.acms.modelEditor.m2m.business2db. Ci-dessous le code des transformations en ATL :

```
-- @path Business=http://acms/1.0
-- @path DB=http://dbmetamodel/1.0
-- @nsURI DiffModel=http://www.eclipse.org/emf/compare

module business2db;
create OUT : DB from IN : Business, IN2: DiffModel;

--Helpers
--Verifier que c'est le premiere execution de ce module
helper def : isFirstRun() : Boolean =
  if DB!Model.allInstances()->size() = 0 then
    true
  else
    false
  endif;

--Verifier que la cible des changements est une entite appartenant a l'element Classifier
helper context DiffModel!Diff def : belongsToclassifier() : Boolean =

  if self.isBusinessObject() or
     self.isFunctionnalLink()
     --or
     --self.isClassAttribute() or
     --self.isRelationAttribute()
```

```
    then

    true

  else
    false
  endif;

--Verifier que la cible des changements est un objet metier
helper context DiffModel!Diff def : isBusinessObject() : Boolean =

  if (self.value.oclIsTypeOf(Business!BusinessObject) and self.requires->size() = 0) then
    true
  else
    false
  endif;

--Verifier que la cible des changements est un lien metier
helper context DiffModel!Diff def : isFunctionalLink() : Boolean =

  if (self.value.oclIsTypeOf(Business!FunctionalLink) and self.requires->size() = 0) then
    true
  else
    false
  endif;

--Verifier que la cible des changements est un element Attribut d'un objet metier
helper context DiffModel!Diff def : isClassAttribute() : Boolean =

  if self.oclIsTypeOf(DiffModel!ReferenceChange) then

    if self.value.oclIsKindOf(Business!Attribute) then
      if not self.value.eContainer().oclIsUndefined() then
        if self.value.eContainer().oclIsTypeOf(Business!BusinessObject) then
          if self.requires->size() = 0 then
            true
          else
            false
          endif
        else
          false
        endif
      else
        false
      endif
    else
      false
```

```
    endif
  else
    false
  endif;

--Verifier que la cible des changements est un l'element Attribut d'un lien metier
helper context DiffModel!Diff def : isRelationAttribute() : Boolean =

  if  self.oclIsTypeOf(DiffModel!ReferenceChange) then

    if  self.value.oclIsKindOf(Business!Attribute) then
      if not self.value.eContainer().oclIsUndefined() then
        if  self.value.eContainer().oclIsTypeOf(Business!FunctionalLink) then
          if  self.requires->size() = 0 then
            true
          else
            false
          endif
        else
          false
        endif
      else
        false
      endif
    else
      false
    endif
  else
    false
  endif;

--Changement de nom de l'objet metier
helper context DiffModel!Diff def : isClassRename() : Boolean =

  --puisque c'est une logique binaire
  --on imbrique les conditions sinon on aura des erreurs puisque il evalue tous les
      conditions
  --meme celles pouvant produire des exceptions
  if  self.oclIsKindOf(DiffModel!AttributeChange) then
    if  self.attribute.eContainer().name = 'BusinessObject' then
      if  self.attribute.name = 'name_object' then
        true
      else
        false
      endif
    else
      false
```

```
    endif
  else
    false
  endif;

--Changement de nom de l'objet metier
helper context DiffModel!Diff def : isRelationRename() : Boolean =

  --puisque c'est une logique binaire
  --on imbrique les conditions
  if  self.oclIsKindOf(DiffModel!AttributeChange) then
    if  self.attribute.eContainer().name = 'FunctionalLink' then
        if  self.attribute.name = 'name_link' then
            true
        else
            false
        endif
    else
        false
    endif
  else
    false
  endif;
--Lazy rules
--Transformations des differences
--Transforme les changements ayant comme cible les objets metier
lazy rule BusinessObject2Class {
  from
    diff :DiffModel!Diff
  to
    c : DB!Class (
      name <- diff.value.name_object,
      Change <- diff.kind,
      attribute <- diff.value.ListObjAttr->collect(attr | thisModule.
          BusinessAttribute2Attribute(attr))
    )
}
--Transforme les changements ayant comme cible les objets metier
lazy rule BusinessRelationship2Relationship {
  from
    diff :DiffModel!Diff
  to
    r : DB!Relationship (
      name <- diff.value.name_link,
      Change <- diff.kind,
      attribute <- diff.value.ListLinkAttr->collect(attr | thisModule.
          BusinessAttribute2Attribute(attr))
```

```
    )
}

lazy rule BusinessObjectAttribut2ClassAttribut {
  from
    diff :DiffModel!Diff
  to
    r : DB!Class (
      name <- diff.value.eContainer().name_object,
      Change <- diff.kind.toString()+'ATTR',
      attribute <- thisModule.BusinessAttribute2Attribute(diff.value)
    )
}

lazy rule BusinessLinkAttribut2RelationshipAttribut {
  from
    diff :DiffModel!Diff
  to
    r : DB!Relationship (
      name <- diff.value.eContainer().name_link,
      Change <- diff.kind.toString()+'ATTR',
      attribute <- thisModule.BusinessAttribute2Attribute(diff.value)
    )
}

lazy rule BusinessObjectRename {
  from
    diff :DiffModel!Diff
  to
    r : DB!Class (
      name <- diff.value,
      Change <- 'RENAME',
      oldName <- diff.value
    )
}

lazy rule BusinessLinkRename {
  from
    diff :DiffModel!Diff
  to
    r : DB!Relationship (
      name <- diff.value,
      Change <- 'RENAME',
      oldName <- diff.value
    )
}
```

```
--Transformations standard des elements
--Regle de transformation d'un attribut du modele metier en un attribut du modele BD
lazy rule BusinessAttribute2Attribute {
  from
    ba : Business!Attribute
  to
    a : DB!Attribute (
      name <- ba.Nom,
      format <- ba.Format,
      len <- ba.Longueur,
      type <- ba.Type.toString().toLower(),
      mode <- ba.Mode,
      nullable <- ba.Obl

    )
}

--Transforamtions des elements conteneurs
lazy rule Diffs2DBClassifier {
  from
    diff :DiffModel!Diff
  to
     classifier :DB!Classifier (
       class <- DiffModel!Diff.allInstances()->select(oneDiff |oneDiff.isBusinessObject())->
           collect(oneDiff|thisModule.BusinessObject2Class(oneDiff)),
       relationship <- DiffModel!Diff.allInstances()->select(oneDiff |oneDiff.
           isFunctionalLink())->collect(oneDiff|thisModule.BusinessRelationship2Relationship
           (oneDiff)),

       class <- DiffModel!Diff.allInstances()->select(oneDiff |oneDiff.isClassAttribute())->
           collect(oneDiff|thisModule.BusinessObjectAttribut2ClassAttribut(oneDiff)),
       relationship <- DiffModel!Diff.allInstances()->select(oneDiff |oneDiff.
           isRelationAttribute())->collect(oneDiff|thisModule.
           BusinessLinkAttribut2RelationshipAttribut(oneDiff)),

       class <- DiffModel!Diff.allInstances()->select(oneDiff |oneDiff.isClassRename())->
           collect(oneDiff|thisModule.BusinessObjectRename(oneDiff)),
       relationship <- DiffModel!Diff.allInstances()->select(oneDiff |oneDiff.
           isRelationRename())->collect(oneDiff|thisModule.BusinessLinkRename(oneDiff))
    )
}


--Matching rules
--Regle de transformation de la racine des modeles
rule diffModel2BDModel {
  from
```

```
    diffs : DiffModel!Diff (thisModule.isFirstRun())
  to
    bdm : DB!Model (
      classes <- thisModule.Diffs2DBClassifier(diffs)
    )

}
```

Listing C.14 – Code ATL du fichier business2db.atl

La transformation M2T Acceleo transforme un modèle Audros en entrée en une structure XML en sortie reconnaissable par l'outil Liquibase qui la transforme à son tour en requêtes SQL. Cette transformation est effectuée à base de templates qui résident dans les fichiers .mtl (fig. C.1). A chaque élément du méta-modèle Audros, une template permet de générer la structure XML correspondante compatibles avec l'outil Liquibase pour effectuer chacune des opérations spécifiées dans la base de données.

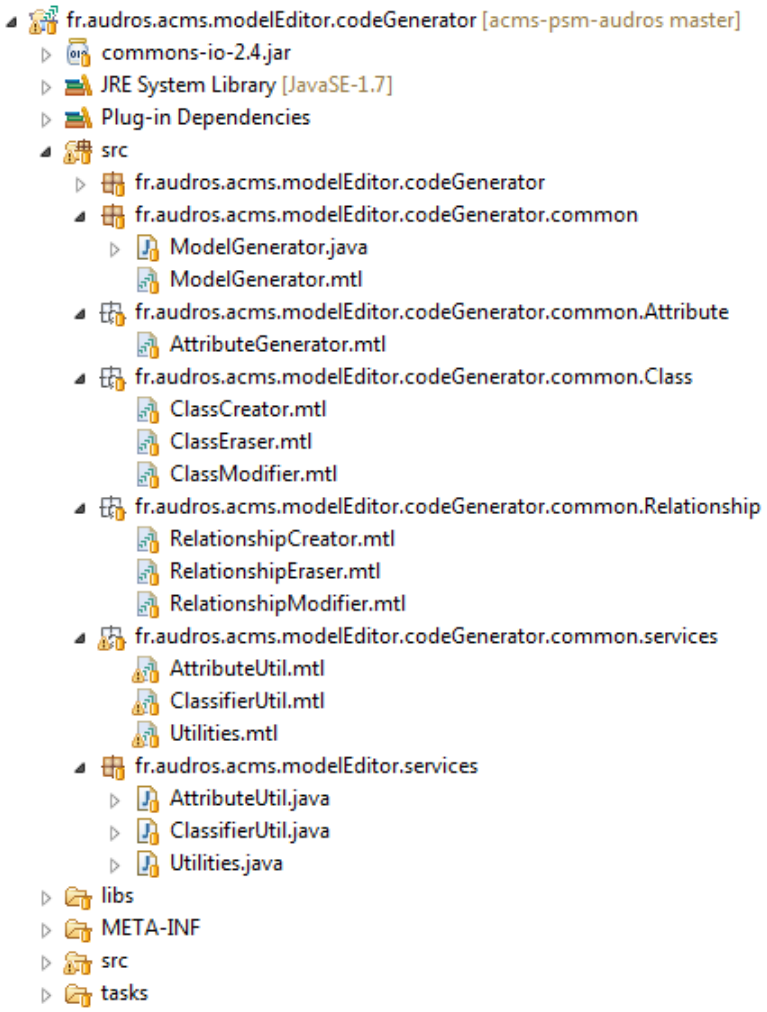

FIGURE C.1 – Projet pour la génération du code XML avec acceleo

La template principale est ModelGenerator.mtl (déclarée en tant que main). Dans cette template, on créé le fichier contenant la structure XML et son contenu qui est généré en parcourant les différents fils du nœud Model et en appelant les templates correspondantes :

```
[comment encoding = UTF-8 /]
[module ModelGenerator('http://dbmetamodel/1.0', 'http://www.eclipse.org/emf/2002/
    Ecore')]

[import fr::audros::acms::modelEditor::codeGenerator::common::Class::ClassCreator/]
[import fr::audros::acms::modelEditor::codeGenerator::common::Class::ClassEraser/]
[import fr::audros::acms::modelEditor::codeGenerator::common::Class::ClassModifier/]
[import fr::audros::acms::modelEditor::codeGenerator::common::Class::ClassAttributeChange
    /]
[import fr::audros::acms::modelEditor::codeGenerator::common::Class::ClassPropertyChange
    /]

[import fr::audros::acms::modelEditor::codeGenerator::common::Relationship::
    RelationshipCreator/]
[import fr::audros::acms::modelEditor::codeGenerator::common::Relationship::
    RelationshipEraser/]
[import fr::audros::acms::modelEditor::codeGenerator::common::Relationship::
    RelationshipModifier/]
[import fr::audros::acms::modelEditor::codeGenerator::common::Relationship::
    RelationshipAttributeChange/]
[import fr::audros::acms::modelEditor::codeGenerator::common::Relationship::
    RelationshipPropertyChange/]

[import fr::audros::acms::modelEditor::codeGenerator::common::services::Utilities /]

[template public create(aModel : Model)]
  [comment @main /]
  [ file  (getModuleID().concat('.xml'), false, 'UTF-8')]
<?xml version="1.0" encoding="UTF-8"?>

<databaseChangeLog
        xmlns="http://www.liquibase.org/xml/ns/dbchangelog"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xmlns:ext="http://www.liquibase.org/xml/ns/dbchangelog-ext"
        xsi:schemaLocation="http://www.liquibase.org/xml/ns/dbchangelog http://www.
            liquibase.org/xml/ns/dbchangelog/dbchangelog-3.1.xsd
        http://www.liquibase.org/xml/ns/dbchangelog-ext http://www.liquibase.org/xml/ns/
            dbchangelog/dbchangelog-ext.xsd">
    <!--delimiters-->
  <property name="delimiter" value="'" dbms="oracle"/>

  <!--Valeur de la colonne "a_ou_f" des tables des_imp_arb et des_cor_arb , pour
      les classes-->
  <property name="classeAF" value="F" />
```

```
<property name="relationshipAF" value="A" />


<!--Type de compteur-->
<property name="compteurTypeClasse" value="1" />
<property name="compteurTypeRelationship" value="1" />

<!--Retourne la valeur du compteur pour chaque type de BD / type de compteur
    -->
<!--Oracle-->
<property name="compteurClasse" value="(Select valeur from compteur where TYPE
    =${compteurTypeClasse})" dbms="oracle"/>
<property name="compteurRelationship" value="(Select valeur from compteur
    where TYPE=${compteurTypeRelationship})" dbms="oracle"/>


<!--Condition de mise a jour du compteur-->
<!--Update la valeur du compteur pour chaque type de BD / type de compteur-->
<!--Oracle-->
<property name="updateCompteurClasse" value="(Select valeur+1 from compteur
    where TYPE=${compteurTypeClasse})" dbms="oracle"/>
<property name="updateCompteurRelationship" value="(Select valeur+1 from
    compteur where TYPE=${compteurTypeRelationship})" dbms="oracle"/>

<!--Formatter la date par defaut-->
<property name="defaultDate" value="to_date('01/01/1980 00:00:00','mm/dd/yyyy
    hh24:mi:ss')" dbms="oracle"/>

<!--Preconditions obligatoires avant l'execution de la mise a jour-->
<preConditions>
  <!--L'utilisateur executant ce script doit etre Audros-->
      <runningAs username="audros" />
</preConditions>

<!--Chaque changeset est execute comme une seule transaction-->
<!--La recuperation et la mise a jour du compteur est donc atomique-->
<changeSet id="[getChangesetID()/]" author="audros" failOnError="true"
    runInTransaction="true" >

   <!--code de migration de la BD-->

     [for (class : Class | aModel.classes.class)]

    [if (class.Change.toString()='DELETE')]
      <!--Droping Class [class.name /] -->
      [class.drop() /]
```

```
        [elseif (class.Change.toString()='ADD')]
  <!--Adding Class [class.name /] -->
        [class.create() /]

[elseif (class.Change.toString()='RENAME')]
  <!--Renaming Class [class.name /] -->
        [class.rename() /]

[elseif (class.Change.toString()='ADDATTR')]
  <!--Adding Attributes to Class [class.name /] -->
        [class.addAttribute() /]

[elseif (class.Change.toString()='DELETEATTR')]
  <!--Droping Attributes from Class [class.name /] -->
        [class.deleteAttribute() /]

      [/if]

  [/for]

[for (relationship : Relationship | aModel.classes.relationship)]

      [if (relationship.Change.toString()='DELETE')]
<!--Droping Relation [relationship.name /] -->
  [relationship.drop() /]

      [elseif (relationship.Change.toString()='ADD')]
<!--Adding Relation [relationship.name /] -->
        [relationship.create() /]

[elseif (relationship.Change.toString()='RENAME')]
  <!--Renaming Relation [relationship.name /] -->
        [relationship.rename() /]

[elseif (relationship.Change.toString()='ADDATTR')]
  <!--Adding Attributes to Relation [relationship.name /] -->
        [relationship.addAttribute() /]
```

```
    [ elseif  (relationship.Change.toString()='DELETEATTR')]
      <!--Droping Attributes from Relation [relationship.name /] -->
          [relationship.deleteAttribute()  /]


        [/ if ]

      [/for]


  </changeSet>



</databaseChangeLog>


  [/ file ]

[/template]
```

Listing C.15 – Code du fichier ModelGenerator.mtl

Les templates correspondantes aux différents autres éléments (Class, Relationship, Attributs) sont regroupées dans des packages. Chaque package regroupe plusieurs templates pour chaque élément. Ces différentes templates correspondent aux différentes opérations qui peuvent être effective sur ces éléments (création, modification, suppression) :

- templates de l'élément class : elles permettent de créer, supprimer et de modifier les éléments de type class dans la base de données.

```
[comment encoding = UTF-8 /]
[module ClassCreator('http://dbmetamodel/1.0', 'http://www.eclipse.org/emf
    /2002/Ecore')]

[import fr::audros::acms::modelEditor::codeGenerator::common::Attribute::
    AttributeGenerator /]
[import fr::audros::acms::modelEditor::codeGenerator::common::services::ClassifierUtil
     /]
[import fr::audros::acms::modelEditor::codeGenerator::common::services::Utilities  /]

[comment Template de creation d'une classe /]
[template public create(aClass : Class)]
  <sql dbms="oracle"
    splitStatements="false"
    >
    BEGIN
    execute immediate 'drop table DT_[aClass.name /]';
```

```
  EXCEPTION
    when others then
      null;
  END;
</sql>

<createTable tableName="DT_[aClass.name /]" schemaName="AUDROS"
    tablespace="ts_audros">
  <column name="AUDROS_PROVENANCE" type="java.sql.Types.VARCHAR (14)"
      defaultValue="">
    <constraints nullable="true" />
  </column>
  <column name="SUJET" type="java.sql.Types.INTEGER" defaultValueNumeric
      ="0">
    <constraints nullable="true" />
  </column>
  <column name="AUDROS_SYSTEME" type="java.sql.Types.VARCHAR (12)"
      defaultValue="">
    <constraints nullable="true" />
  </column>
  <column name="ID_STAT" type="java.sql.Types.INTEGER"
      defaultValueNumeric="0">
    <constraints nullable="true" />
  </column>
  <column name="ID_SU_CL_OB" type="java.sql.Types.INTEGER"
      defaultValueNumeric="0">
    <constraints nullable="true" />
  </column>
  <column name="NUM_ART" type="java.sql.Types.INTEGER">
    <constraints nullable="true" />
  </column>
  [for (attr : Attribute | aClass.attribute)]
    [attr.create() /]
  [/for]
</createTable>
<rollback>
  <sql dbms="oracle">
    drop table DT_[aClass.name /];
  </sql>

</rollback>

<!--Creation des synonymes-->
<!--Affectation des droits sur la table-->
<sql dbms="oracle">
  CREATE OR REPLACE public synonym DT_[aClass.name /] for audros.DT_[
      aClass.name /] ;
```

```
  grant all on DT_[aClass.name /] to public ;
</sql>

<rollback>
  <sql dbms="oracle">
    drop synonym DT_[aClass.name /];
  </sql>
</rollback>

<sql dbms="oracle"
  splitStatements="false"
  >
  BEGIN
  execute immediate 'drop index DT_[aClass.name /]0';
  EXCEPTION
    when others then
      null;
  END;
</sql>

<!--Creation index0-->
<createIndex indexName="DT_[aClass.name /]0" tableName="DT_[aClass.name
    /]" tablespace="au_ts_index" unique="true">
         <column name="NUM_ART"/>
  </createIndex>

<rollback>
  <sql dbms="oracle">
    drop index DT_[aClass.name /]0;
  </sql>
</rollback>

<sql dbms="oracle"
  splitStatements="false"
  >
  BEGIN
  execute immediate 'drop index DT_[aClass.name /]1';
  EXCEPTION
    when others then
      null;
  END;
</sql>

<!--Creation index1-->
<createIndex indexName="DT_[aClass.name /]1" tableName="DT_[aClass.name
    /]" tablespace="au_ts_index" unique="true">
         <column name="REF_UTILISAT"/>
```

```
          <column name="version desc"/>
          <column name="revision desc"/>
          <column name="AUDROS_PROVENANCE"/>
  </createIndex>

<rollback>
  <sql dbms="oracle">
    drop index DT_[aClass.name /]1;
  </sql>
</rollback>

<!--Creation de procedure-->
<!--Procedure 1-->
<!--Oracle-->
<createProcedure
          dbms="oracle"
          encoding="utf8"
          procedureName="pin_[aClass.name /]_art">

    CREATE OR REPLACE procedure pin_[aClass.name /]_art(ref_utilisat_var
        varchar,date_creation_var
    date,num_art_var integer,createur_var varchar,version_var varchar,
        revision_var varchar,designation_var
    varchar,libelle_variante_var varchar,date_modif_var date,
        modificateur_var varchar,statut_var
    varchar,audros_provenance_var varchar,audros_systeme_var varchar,
        id_stat_var integer,id_su_cl_ob_var
    integer,sujet_var integer) as begin if audros_provenance_var = '[aClass
        .name /]' then insert into
    articles (num_int, nom_type, ref_user, date_creation, createur,
        version, revision, designation,
    libelle_variante,date_modif, statut,modificateur, id_stat, id_su_cl_ob
        ,sujet,revision2,revision3)
    values (num_art_var,'[aClass.name /]', ref_utilisat_var,
        date_creation_var, createur_var, version_var,
    revision_var, designation_var, libelle_variante_var, date_modif_var,
        statut_var, modificateur_var,
    id_stat_var, id_su_cl_ob_var, sujet_var, version_var, revision_var);
        insert into historiques(nom_classe,
    num_art1, statut1, reference1, version1, revision1, num_art2, statut2,
         reference2, version2,
    revision2, date_modif, objet_modif, code_user, code_modif,
        num_fonction) values ('[aClass.name /]',
    num_art_var, id_stat_var, ref_utilisat_var, version_var, revision_var,
         num_art_var, id_stat_var,
    ref_utilisat_var, version_var, revision_var, date_creation_var, '
        Object creation', createur_var,
```

```
    createur_var, 1); end if; end;
</createProcedure>

<rollback>
  <sql dbms="oracle">
    drop procedure pin_[aClass.name /]_art;
  </sql>
</rollback>


<!--Creation des synonymes-->
<!--Affectation des droits sur la procedure-->
<sql dbms="oracle">
  CREATE OR REPLACE public synonym pin_[aClass.name /]_art for audros.
      pin_[aClass.name /]_art ;
  grant all on pin_[aClass.name /]_art to public;
</sql>

<rollback>
  <sql dbms="oracle">
    drop public synonym pin_[aClass.name /]_art;
  </sql>
</rollback>

<!--Procedure 2-->
<!--Oracle-->
<createProcedure
         dbms="oracle"
         encoding="utf8"
         procedureName="pmo_[aClass.name /]_art">

    CREATE OR REPLACE procedure pmo_[aClass.name /]_art(ref_utilisat_var
        varchar,date_creation_var
    date,num_art_var integer,createur_var varchar,version_var varchar,
        revision_var varchar,designation_var
    varchar,libelle_variante_var varchar,date_modif_var date,
        modificateur_var varchar,statut_var
    varchar,audros_provenance_var varchar,audros_systeme_var varchar,
        id_stat_var integer,id_su_cl_ob_var
    integer,sujet_var integer) as begin if audros_provenance_var = '[aClass
        .name /]' then update historiques
    set version2=version_var, revision2=revision_var where num_art2=
        num_art_var and reference2=ref_utilisat_var;
    update articles set ref_user=ref_utilisat_var, nom_type='[aClass.name /]
        ', date_creation=date_creation_var,
    createur=createur_var, version=version_var, revision=revision_var,
        designation=designation_var,
```

```
    libelle_variante=libelle_variante_var, date_modif=date_modif_var,
        statut=statut_var, modificateur=modificateur_var,
    id_stat=id_stat_var, sujet=sujet_var, id_su_cl_ob=id_su_cl_ob_var
        where num_int=num_art_var;
    end if; end;
</createProcedure>

<rollback>
  <sql dbms="oracle">
    drop procedure pmo_[aClass.name /]_art;
  </sql>
</rollback>

<!--Creation des synonymes-->
<!--Affectation des droits sur la procedure-->
<sql dbms="oracle">
  CREATE OR REPLACE public synonym pmo_[aClass.name /]_art for audros.
      pmo_[aClass.name /]_art ;
  grant all on pmo_[aClass.name /]_art to public ;
</sql>

<rollback>
  <sql dbms="oracle">
    drop synonym pmo_[aClass.name /]_art;
  </sql>
</rollback>

<!--Procedure 3-->
<!--Oracle-->
<createProcedure
         dbms="oracle"
         encoding="utf8"
         procedureName="psu_[aClass.name /]_art">

    CREATE OR REPLACE procedure psu_[aClass.name /]_art(num_art_var
        integer , audros_provenance_var
    varchar) as begin if audros_provenance_var = '[aClass.name /]' then
        delete from articles where
    num_int=num_art_var; end if; end;
</createProcedure>

<rollback >
  <sql dbms="oracle">
    drop procedure psu_[aClass.name /]_art;
  </sql>
</rollback>
```

```
<!--Creation des synonymes-->
<!--Affectation des droits sur la procedure-->
<sql dbms="oracle">
  CREATE OR REPLACE public synonym psu_[aClass.name /]_art for audros.
      psu_[aClass.name /]_art ;
  grant all on psu_[aClass.name /]_art to public ;
</sql>

<rollback>
  <sql dbms="oracle">
    drop public synonym psu_[aClass.name /]_art;
  </sql>
</rollback>

<!--Procedure 4-->
<!--Oracle-->
<createProcedure
         dbms="oracle"
         encoding="utf8"
         procedureName="pd_dt_[aClass.name /]">

    CREATE OR REPLACE procedure pd_dt_[aClass.name /](nn_var integer ,
        na_var integer , audros_provenance_var
    varchar, ref_user_var varchar, ref_var varchar) as begin if
        audros_provenance_var = '[aClass.name /]'
    then delete from articles where num_int=nn_var; end if; end;
</createProcedure>

<rollback >
  <sql dbms="oracle">
         drop procedure pd_dt_[aClass.name /];
  </sql>
</rollback>

<!--Creation des synonymes-->
<!--Affectation des droits sur la procedure-->
<sql dbms="oracle">
  CREATE OR REPLACE public synonym pd_dt_[aClass.name /] for audros.
      pd_dt_[aClass.name /] ;
  grant all on pd_dt_[aClass.name /] to public ;
</sql>

<rollback >
  <sql dbms="oracle">
    drop public synonym pd_dt_[aClass.name /];
  </sql>
</rollback>
```

```
  <!--Creation des triggers-->
  <!--oracle-->
  <sql dbms="oracle">

CREATE OR REPLACE trigger rin_[aClass.name /]_art after insert on dt_[
    aClass.name /] for each row begin
pin_[aClass.name /]_art (ref_utilisat_var => :new.ref_utilisat,
    date_creation_var => :new.date_creation,num_art_var
=> :new.num_art,createur_var => :new.createur,version_var => :new.version,
    revision_var => :new.revision,designation_var
=> :new.designation,libelle_variante_var => :new.libelle_variante,
    date_modif_var => :new.date_modif,modificateur_var
=> :new.modificateur,statut_var => :new.statut,audros_provenance_var => :
    new.audros_provenance,audros_systeme_var
=> :new.audros_systeme,id_stat_var => :new.id_stat,id_su_cl_ob_var => :new.
    id_su_cl_ob,sujet_var
=> :new.sujet); end;

CREATE OR REPLACE trigger rmo_[aClass.name /]_art after update on dt_[
    aClass.name /] for each row begin
pmo_[aClass.name /]_art (ref_utilisat_var => :new.ref_utilisat,
    date_creation_var => :new.date_creation,num_art_var
=> :new.num_art,createur_var => :new.createur,version_var => :new.version,
    revision_var => :new.revision,designation_var
=> :new.designation,libelle_variante_var => :new.libelle_variante,
    date_modif_var => :new.date_modif,modificateur_var
=> :new.modificateur,statut_var => :new.statut,audros_provenance_var => :
    new.audros_provenance,audros_systeme_var
=> :new.audros_systeme,id_stat_var => :new.id_stat,id_su_cl_ob_var => :new.
    id_su_cl_ob,sujet_var
=> :new.sujet); end;


CREATE OR REPLACE trigger rsu_[aClass.name /]_art after delete on dt_[
    aClass.name /] for each row begin
psu_[aClass.name /]_art (num_art_var => :old.num_art,audros_provenance_var
    => :old.audros_provenance)
end;

  </sql>
  <rollback >
    <sql dbms="oracle">
      drop trigger rin_[aClass.name /]_art;
      drop trigger rmo_[aClass.name /]_art;
      drop trigger rsu_[aClass.name /]_art;
```

```
  </sql>
</rollback>

<sql dbms="oracle">
    delete from DICOTYPES where TYPE=${delimiter}[aClass.name.toUpper() /]
        ${delimiter};
</sql>
<insert tableName="DICOTYPES">
  <column name="UNICITE" value="B"/>
  <column name="MODVER" value="A"/>
  <column name="MODREV" value="1"/>
  <column name="FICHIER" value="0"/>
  <column name="HISTO" value="0"/>
  <column name="CHANGREF" valueNumeric="0"/>
  <column name="PROC_ID" valueNumeric="0"/>
  <column name="PROC_VERSION" value="Initial"/>
  <column name="PROC_REVISION" value=" "/>
  <column name="PROC_CHANG_VERSION" value=" "/>
  <column name="PROC_CHANG_REVISION" value=" "/>
  <column name="ADRESSE_ICONE" value=" "/>
  <column name="ACTION" value=" "/>
  <column name="SUJET" value="N"/>
  <!--La valeur sera changee avec un update-->
  <column name="ID_CLASSE" valueComputed="${compteurClasse}"/>
  <column name="CAD_TYPE" value=" "/>
  <column name="COMMENTAIRE" value=" "/>
  <column name="PROPA_VIEWS" valueNumeric="0"/>
  <column name="PROPA_PERMISSIONS" valueNumeric="0"/>
  <column name="TYPE" value="[aClass.name.toUpper() /]"/>
</insert>
<rollback>
  <sql dbms="oracle">
    delete from DICOTYPES where TYPE='[aClass.name.toUpper() /]';
  </sql>
</rollback>

<!--Mise a jour du compteur-->
<update tableName="compteur">
  <column name="valeur" valueComputed="${updateCompteurClasse}" />
  <where>TYPE=${compteurTypeClasse}</where>
</update>
<rollback />

<sql dbms="oracle">
    delete from DESCTYPE where NOM_TYPE=${delimiter}[aClass.name.toUpper()
        /]${delimiter};
```

```
</sql>

<!--Enregistrement des attributs dans la table Desctype -->
[for (attr : Attribute | aClass.attribute)]
<insert tableName="DESCTYPE">
  <column name="REEL6" valueNumeric="0.0"/>
  <column name="MMIN" valueNumeric="0.0"/>
  <column name="NOM_RUB_HERIT" value=" "/>
  <column name="RMIN" valueNumeric="0.0"/>
  <column name="MESURE" value=" "/>
  <column name="ENT5" valueNumeric="0"/>
  <column name="THESAURUS" value=" "/>
  <column name="POINTEUR_RUB" value=" "/>
  <column name="EMIN" valueNumeric="0"/>
  <column name="DMIN" valueDate="${defaultDate}"/>
  <column name="ACTION" value=" "/>
  <column name="DMAX" valueDate="${defaultDate}"/>
  <column name="HERITE" valueNumeric="0"/>
  <column name="DATE4" valueDate="${defaultDate}"/>
  <column name="EMAX" valueNumeric="0"/>
  <column name="NOM_ACTION" value=" "/>
  <column name="TEXTE7" value=" "/>
  <column name="POINTEUR" value=" "/>
  <column name="NUM_RUB" valueNumeric="[getNextNumber(attr.name) /]"/>
  <column name="MODE_RUB" value="LI"/>
  <column name="TEMPMAS" value=" "/>
  <column name="MMAX" valueNumeric="0.0"/>
  <column name="NOM_DICO" value=" "/>
  [comment ENT1 ? /]
  <column name="ENT1" valueNumeric="[attr.len /]"/>
  <column name="DEFAUT" value=" "/>
  <column name="ENT3" valueNumeric="0"/>
  <column name="GESTION" value="[getManagementMode(attr.type.toString())
      /]"/>
  <column name="HERITEE_DE" value=" "/>
  <column name="TYPE_RUB" value="[getRubricType(attr.type.toString())
      /]"/>
  <column name="RMAX" valueNumeric="0.0"/>
  <column name="AIDE_RUB" value="[aClass.name.toUpper() /].[attr.name
      /]"/>
  <column name="DICO" value=" "/>
  <column name="CHAI2" value=" "/>
  <column name="INPUTMAS" valueNumeric="0"/>
  <column name="FORMAT" value="[getFormat(attr.type.toString(), attr.name
      ) /]"/>
  <column name="NOM_RUB" value="[attr.name /]"/>
  <column name="NOM_TYPE" value="[aClass.name.toUpper()/]"/>
```

```
</insert>


[/for]

<!--attributs BD-->
<insert tableName="DESCTYPE">
  <column name="REEL6" valueNumeric="0.0"/>
  <column name="MMIN" valueNumeric="0.0"/>
  <column name="NOM_RUB_HERIT" value=" "/>
  <column name="RMIN" valueNumeric="0.0"/>
  <column name="MESURE" value=" "/>
  <column name="ENT5" valueNumeric="0"/>
  <column name="THESAURUS" value=" "/>
  <column name="POINTEUR_RUB" value=" "/>
  <column name="EMIN" valueNumeric="0"/>
  <column name="DMIN" valueDate="${defaultDate}"/>
  <column name="ACTION" value=" "/>
  <column name="DMAX" valueDate="${defaultDate}"/>
  <column name="HERITE" valueNumeric="0"/>
  <column name="DATE4" valueDate="${defaultDate}"/>
  <column name="EMAX" valueNumeric="0"/>
  <column name="NOM_ACTION" value=" "/>
  <column name="TEXTE7" value=" "/>
  <column name="POINTEUR" value=" "/>
  <column name="NUM_RUB" valueNumeric="[getNextNumber('
     AUDROS_PROVENANCE') /]"/>
  <column name="MODE_RUB" value="LI"/>
  <column name="TEMPMAS" value=" "/>
  <column name="MMAX" valueNumeric="0.0"/>
  <column name="NOM_DICO" value=" "/>
  [comment ENT1 ? /]
  <column name="ENT1" valueNumeric="14"/>
  <column name="DEFAUT" value=" "/>
  <column name="ENT3" valueNumeric="0"/>
  <column name="GESTION" value="[getManagementMode('String') /]"/>
  <column name="HERITEE_DE" value=" "/>
  <column name="TYPE_RUB" value="[getRubricType('String') /]"/>
  <column name="RMAX" valueNumeric="0.0"/>
  <column name="AIDE_RUB" value="[aClass.name.toUpper().concat('String')
     /]"/>
  <column name="DICO" value=" "/>
  <column name="CHAI2" value=" "/>
  <column name="INPUTMAS" valueNumeric="0"/>
  <column name="FORMAT" value="[getFormat('String','
     AUDROS_PROVENANCE') /]"/>
  <column name="NOM_RUB" value="AUDROS_PROVENANCE"/>
```

```
  <column name="NOM_TYPE" value="[aClass.name.toUpper()/]"/>
</insert>
<insert tableName="DESCTYPE">
  <column name="REEL6" valueNumeric="0.0"/>
  <column name="MMIN" valueNumeric="0.0"/>
  <column name="NOM_RUB_HERIT" value=" "/>
  <column name="RMIN" valueNumeric="0.0"/>
  <column name="MESURE" value=" "/>
  <column name="ENT5" valueNumeric="0"/>
  <column name="THESAURUS" value=" "/>
  <column name="POINTEUR_RUB" value=" "/>
  <column name="EMIN" valueNumeric="0"/>
  <column name="DMIN" valueDate="${defaultDate}"/>
  <column name="ACTION" value=" "/>
  <column name="DMAX" valueDate="${defaultDate}"/>
  <column name="HERITE" valueNumeric="0"/>
  <column name="DATE4" valueDate="${defaultDate}"/>
  <column name="EMAX" valueNumeric="0"/>
  <column name="NOM_ACTION" value=" "/>
  <column name="TEXTE7" value=" "/>
  <column name="POINTEUR" value=" "/>
  <column name="NUM_RUB" valueNumeric="[getNextNumber('SUJET') /]"/>
  <column name="MODE_RUB" value="LI"/>
  <column name="TEMPMAS" value=" "/>
  <column name="MMAX" valueNumeric="0.0"/>
  <column name="NOM_DICO" value=" "/>
  [comment ENT1 ? /]
  <column name="ENT1" valueNumeric="22"/>
  <column name="DEFAUT" value=" "/>
  <column name="ENT3" valueNumeric="0"/>
  <column name="GESTION" value="[getManagementMode('Integer') /]"/>
  <column name="HERITEE_DE" value=" "/>
  <column name="TYPE_RUB" value="[getRubricType('Integer') /]"/>
  <column name="RMAX" valueNumeric="0.0"/>
  <column name="AIDE_RUB" value="[aClass.name.toUpper().concat('Integer')
      /]"/>
  <column name="DICO" value=" "/>
  <column name="CHAI2" value=" "/>
  <column name="INPUTMAS" valueNumeric="0"/>
  <column name="FORMAT" value="[getFormat('Integer','SUJET') /]"/>
  <column name="NOM_RUB" value="SUJET"/>
  <column name="NOM_TYPE" value="[aClass.name.toUpper()/]"/>
</insert>
  <insert tableName="DESCTYPE">
  <column name="REEL6" valueNumeric="0.0"/>
  <column name="MMIN" valueNumeric="0.0"/>
  <column name="NOM_RUB_HERIT" value=" "/>
```

```
  <column name="RMIN" valueNumeric="0.0"/>
  <column name="MESURE" value=" "/>
  <column name="ENT5" valueNumeric="0"/>
  <column name="THESAURUS" value=" "/>
  <column name="POINTEUR_RUB" value=" "/>
  <column name="EMIN" valueNumeric="0"/>
  <column name="DMIN" valueDate="${defaultDate}"/>
  <column name="ACTION" value=" "/>
  <column name="DMAX" valueDate="${defaultDate}"/>
  <column name="HERITE" valueNumeric="0"/>
  <column name="DATE4" valueDate="${defaultDate}"/>
  <column name="EMAX" valueNumeric="0"/>
  <column name="NOM_ACTION" value=" "/>
  <column name="TEXTE7" value=" "/>
  <column name="POINTEUR" value=" "/>
  <column name="NUM_RUB" valueNumeric="[getNextNumber('ID_STAT') /]"/>
  <column name="MODE_RUB" value="LI"/>
  <column name="TEMPMAS" value=" "/>
  <column name="MMAX" valueNumeric="0.0"/>
  <column name="NOM_DICO" value=" "/>
  [comment ENT1 ? /]
  <column name="ENT1" valueNumeric="22"/>
  <column name="DEFAUT" value=" "/>
  <column name="ENT3" valueNumeric="0"/>
  <column name="GESTION" value="[getManagementMode('Integer') /]"/>
  <column name="HERITEE_DE" value=" "/>
  <column name="TYPE_RUB" value="[getRubricType('Integer') /]"/>
  <column name="RMAX" valueNumeric="0.0"/>
  <column name="AIDE_RUB" value="[aClass.name.toUpper().concat('Integer')
     /]"/>
  <column name="DICO" value=" "/>
  <column name="CHAI2" value=" "/>
  <column name="INPUTMAS" valueNumeric="0"/>
  <column name="FORMAT" value="[getFormat('Integer','ID_STAT') /]"/>
  <column name="NOM_RUB" value="ID_STAT"/>
  <column name="NOM_TYPE" value="[aClass.name.toUpper()/]"/>
</insert>
  <insert tableName="DESCTYPE">
  <column name="REEL6" valueNumeric="0.0"/>
  <column name="MMIN" valueNumeric="0.0"/>
  <column name="NOM_RUB_HERIT" value=" "/>
  <column name="RMIN" valueNumeric="0.0"/>
  <column name="MESURE" value=" "/>
  <column name="ENT5" valueNumeric="0"/>
  <column name="THESAURUS" value=" "/>
  <column name="POINTEUR_RUB" value=" "/>
  <column name="EMIN" valueNumeric="0"/>
```

```
  <column name="DMIN" valueDate="${defaultDate}"/>
  <column name="ACTION" value=" "/>
  <column name="DMAX" valueDate="${defaultDate}"/>
  <column name="HERITE" valueNumeric="0"/>
  <column name="DATE4" valueDate="${defaultDate}"/>
  <column name="EMAX" valueNumeric="0"/>
  <column name="NOM_ACTION" value=" "/>
  <column name="TEXTE7" value=" "/>
  <column name="POINTEUR" value=" "/>
  <column name="NUM_RUB" valueNumeric="[getNextNumber('ID_SU_CL_OB')
      /]"/>
  <column name="MODE_RUB" value="LI"/>
  <column name="TEMPMAS" value=" "/>
  <column name="MMAX" valueNumeric="0.0"/>
  <column name="NOM_DICO" value=" "/>
  [comment ENT1 ? /]
  <column name="ENT1" valueNumeric="22"/>
  <column name="DEFAUT" value=" "/>
  <column name="ENT3" valueNumeric="0"/>
  <column name="GESTION" value="[getManagementMode('Integer') /]"/>
  <column name="HERITEE_DE" value=" "/>
  <column name="TYPE_RUB" value="[getRubricType('Integer') /]"/>
  <column name="RMAX" valueNumeric="0.0"/>
  <column name="AIDE_RUB" value="[aClass.name.toUpper().concat('Integer')
      /]"/>
  <column name="DICO" value=" "/>
  <column name="CHAI2" value=" "/>
  <column name="INPUTMAS" valueNumeric="0"/>
  <column name="FORMAT" value="[getFormat('Integer','ID_SU_CL_OB')
      /]"/>
  <column name="NOM_RUB" value="ID_SU_CL_OB"/>
  <column name="NOM_TYPE" value="[aClass.name.toUpper()/]"/>
</insert>
<insert tableName="DESCTYPE">
  <column name="REEL6" valueNumeric="0.0"/>
  <column name="MMIN" valueNumeric="0.0"/>
  <column name="NOM_RUB_HERIT" value=" "/>
  <column name="RMIN" valueNumeric="0.0"/>
  <column name="MESURE" value=" "/>
  <column name="ENT5" valueNumeric="0"/>
  <column name="THESAURUS" value=" "/>
  <column name="POINTEUR_RUB" value=" "/>
  <column name="EMIN" valueNumeric="0"/>
  <column name="DMIN" valueDate="${defaultDate}"/>
  <column name="ACTION" value=" "/>
  <column name="DMAX" valueDate="${defaultDate}"/>
  <column name="HERITE" valueNumeric="0"/>
```

```
    <column name="DATE4" valueDate="${defaultDate}"/>
    <column name="EMAX" valueNumeric="0"/>
    <column name="NOM_ACTION" value=" "/>
    <column name="TEXTE7" value=" "/>
    <column name="POINTEUR" value=" "/>
    <column name="NUM_RUB" valueNumeric="[getNextNumber('
        AUDROS_SYSTEME') /]"/>
    <column name="MODE_RUB" value="LI"/>
    <column name="TEMPMAS" value=" "/>
    <column name="MMAX" valueNumeric="0.0"/>
    <column name="NOM_DICO" value=" "/>
    [comment ENT1 ? /]
    <column name="ENT1" valueNumeric="22"/>
    <column name="DEFAUT" value=" "/>
    <column name="ENT3" valueNumeric="0"/>
    <column name="GESTION" value="[getManagementMode('String') /]"/>
    <column name="HERITEE_DE" value=" "/>
    <column name="TYPE_RUB" value="[getRubricType('String') /]"/>
    <column name="RMAX" valueNumeric="0.0"/>
    <column name="AIDE_RUB" value="[aClass.name.toUpper().concat('String')
        /]"/>
    <column name="DICO" value=" "/>
    <column name="CHAI2" value=" "/>
    <column name="INPUTMAS" valueNumeric="0"/>
    <column name="FORMAT" value="[getFormat('String','AUDROS_SYSTEME')
        /]"/>
    <column name="NOM_RUB" value="AUDROS_SYSTEME"/>
    <column name="NOM_TYPE" value="[aClass.name.toUpper()/]"/>
  </insert>

  <rollback>
    <sql dbms="oracle">
      delete from DESCTYPE where NOM_TYPE='[aClass.name.toUpper() /]';
    </sql>
  </rollback>

  [comment reinitialiser le compteur /]
  [initCounter(aClass.name) /]
[/template]
```

Listing C.16 – Code du template classCreator.mtl

```
[comment encoding = UTF-8 /]
[module ClassEraser('http://dbmetamodel/1.0', 'http://www.eclipse.org/emf
    /2002/Ecore')]

[import fr::audros::acms::modelEditor::codeGenerator::common::Attribute::
    AttributeGenerator /]
```

```
[comment Template de suppression d'une classe /]
[template public drop(aClass : Class)]

 <delete tableName="heritages">
        <where>famille_pere=${delimiter}[aClass.name/]${delimiter}</where>
 </delete>

 <update tableName="desctype">
   <column name="herite" valueNumeric="0" />
   <column name="heritee_de" value=" " />
   <column name="nom_rub_herit" value=" " />
   <where>heritee_de=${delimiter}[aClass.name/]${delimiter}</where>
 </update>

 <delete tableName="des_imp_arb">
        <where>arbo_fami=${delimiter}[aClass.name/]${delimiter} and a_ou_f
            =${delimiter}${classeAF}${delimiter}</where>
 </delete>

 <delete tableName="des_cor_arb">
        <where>arbo_fami=${delimiter}[aClass.name/]${delimiter} and a_ou_f
            =${delimiter}${classeAF}${delimiter}</where>
 </delete>

 <delete tableName="acces_role">
        <where>class_name=${delimiter}[aClass.name/]${delimiter}</where>
 </delete>

 <delete tableName="acces_role2">
        <where>parent_class=${delimiter}[aClass.name/]${delimiter} or
           child_class = ${delimiter}[aClass.name/]${delimiter} </where>
 </delete>

 <delete tableName="grp_niv_ecran">
        <where>id_ecran IN (select id_ecran from definition_ecran where
           classe = ${delimiter}[aClass.name/]${delimiter})</where>
 </delete>

 <delete tableName="definition_ecran">
        <where>classe = ${delimiter}[aClass.name/]${delimiter} and
           lien_objet = 1</where>
 </delete>

 <delete tableName="definition_ecran_xml">
        <where>classe = ${delimiter}[aClass.name/]${delimiter} and
           lien_objet = 1</where>
```

```
</delete>

<delete tableName="table_requete">
       <where>famille = ${delimiter}[aClass.name/]${delimiter}</where>
</delete>

<delete tableName="au_status_sequence">
       <where>class_name = ${delimiter}[aClass.name/]${delimiter}</where>
</delete>

<delete tableName="dicotypes">
       <where>type = ${delimiter}[aClass.name/]${delimiter}</where>
</delete>

<delete tableName="desctype">
       <where>nom_type = ${delimiter}[aClass.name/]${delimiter}</where>
</delete>

<delete tableName="variantype">
       <where>nom_type = ${delimiter}[aClass.name/]${delimiter}</where>
</delete>

<delete tableName="act_type">
       <where>nom_type = ${delimiter}[aClass.name/]${delimiter}</where>
</delete>

<delete tableName="act_rub_type">
       <where>nom_type = ${delimiter}[aClass.name/]${delimiter}</where>
</delete>

<delete tableName="au_lifecycle">
       <where>lc_class_name = ${delimiter}[aClass.name/]${delimiter}</
          where>
</delete>

<delete tableName="au_lifecycle_req">
       <where>lcr_class_name = ${delimiter}[aClass.name/]${delimiter}</
          where>
</delete>

<delete tableName="au_naming">
       <where>nm_class_name = ${delimiter}[aClass.name/]${delimiter}</
          where>
</delete>

<delete tableName="associat">
       <where>type_pere = ${delimiter}[aClass.name/]${delimiter} or
```

```
            type_fils = ${delimiter}[aClass.name/]${delimiter}</where>
</delete>

<delete tableName="heritages">
       <where>famille_fils = ${delimiter}[aClass.name/]${delimiter}</
           where>
</delete>

<delete tableName="sujet">
       <where>nom_classe = ${delimiter}[aClass.name/]${delimiter} and
           type = ${delimiter}SYCL${delimiter} </where>
</delete>

<delete tableName="restriction">
       <where>nom_table = ${delimiter}[aClass.name/]${delimiter} and type
            = ${delimiter}${classeAF}${delimiter} </where>
</delete>

<delete tableName="dt_[aClass.name /]">
</delete>

<delete tableName="asso_formu">
       <where>num_int IN ( SELECT num_art FROM dt_[aClass.name/] ) OR
           num_int_objet IN ( SELECT num_art FROM dt_[aClass.name/] ) </
           where>
</delete>

<delete tableName="champs">
       <where>num_form IN ( SELECT num_form FROM formulaires WHERE
           famille_formulaire = ${delimiter}[aClass.name/]${delimiter} OR
           classe_entete = ${delimiter}[aClass.name/]${delimiter})</where>
</delete>

<delete tableName="formulaires">
       <where> famille_formulaire = ${delimiter}[aClass.name/]${delimiter
           } OR classe_entete = ${delimiter}[aClass.name/]${delimiter}</
           where>
</delete>

<!--Supression des triggers-->
<!--Oracle-->
<sql dbms="oracle">
  drop trigger rsu_[aClass.name /]_art;
  drop trigger rin_[aClass.name /]_art;
  drop trigger rmo_[aClass.name /]_art;
</sql>
```

```
  <!--Suppression des procedures-->
  <dropProcedure procedureName="pin_[aClass.name /]_art" />
  <dropProcedure procedureName="psu_[aClass.name /]_art" />
  <dropProcedure procedureName="pmo_[aClass.name /]_art" />
  <dropProcedure procedureName="pd_dt_[aClass.name /]" />

  <!--Suppression des synonymes des procedures-->
  <!--oracle-->
  <sql dbms="oracle">
    drop public synonym pin_[aClass.name /]_art ;
    drop public synonym psu_[aClass.name /]_art ;
    drop public synonym pmo_[aClass.name /]_art ;
    drop public synonym pd_dt_[aClass.name /] ;
  </sql>

  <!--Suppression de la table de la classe-->
  <dropTable tableName="dt_[aClass.name /]"/>

  <!--Suppression du synonyme de la table-->
  <!--oracle-->
  <sql dbms="oracle">
    drop public synonym dt_[aClass.name /];
  </sql>
[/template]
```

Listing C.17 – Code du template classEraser.mtl

```
[comment encoding = UTF-8 /]
[module ClassModifier('http://dbmetamodel/1.0', 'http://www.eclipse.org/emf
    /2002/Ecore')]

[import fr::audros::acms::modelEditor::codeGenerator::common::Attribute::
    AttributeGenerator /]
[import fr::audros::acms::modelEditor::codeGenerator::common::services::ClassifierUtil
    /]
[import fr::audros::acms::modelEditor::codeGenerator::common::services::Utilities /]

[comment Template de modification d'une classe /]
[template public modify(aClass : Class)]

  <delete tableName="heritages">
         <where>famille_fils=${delimiter}[aClass.name/]${delimiter}</where>
  </delete>

  <update tableName="desctype">
    <column name="herite" valueNumeric="0" />
    <column name="heritee_de" value=" " />
    <column name="nom_rub_herit" value=" " />
```

```
  <where>heritee_de=${delimiter}[aClass.name/]${delimiter}</where>
</update>

<delete tableName="act_type">
       <where>nom_type = ${delimiter}[aClass.name/]${delimiter}</where>
</delete>

<delete tableName="desctype">
       <where>nom_type = ${delimiter}[aClass.name/]${delimiter}</where>
</delete>

<!--Enregistrement des attributs dans la table Desctype -->
[for (attr : Attribute | aClass.attribute)]
<insert tableName="DESCTYPE">
  <column name="REEL6" valueNumeric="0.0"/>
  <column name="MMIN" valueNumeric="0.0"/>
  <column name="NOM_RUB_HERIT" value=" "/>
  <column name="RMIN" valueNumeric="0.0"/>
  <column name="MESURE" value=" "/>
  <column name="ENT5" valueNumeric="0"/>
  <column name="THESAURUS" value=" "/>
  <column name="POINTEUR_RUB" value=" "/>
  <column name="EMIN" valueNumeric="0"/>
  <column name="DMIN" valueDate="${defaultDate}"/>
  <column name="ACTION" value=" "/>
  <column name="DMAX" valueDate="${defaultDate}"/>
  <column name="HERITE" valueNumeric="0"/>
  <column name="DATE4" valueDate="${defaultDate}"/>
  <column name="EMAX" valueNumeric="0"/>
  <column name="NOM_ACTION" value=" "/>
  <column name="TEXTE7" value=" "/>
  <column name="POINTEUR" value=" "/>
  <column name="NUM_RUB" valueNumeric="[getNextNumber(attr.name) /]"/>
  <column name="MODE_RUB" value="LI"/>
  <column name="TEMPMAS" value=" "/>
  <column name="MMAX" valueNumeric="0.0"/>
  <column name="NOM_DICO" value=" "/>
  [comment ENT1 ? /]
  <column name="ENT1" valueNumeric="[attr.len /]"/>
  <column name="DEFAUT" value=" "/>
  <column name="ENT3" valueNumeric="0"/>
  <column name="GESTION" value="[getManagementMode(attr.type.toString())
      /]"/>
  <column name="HERITEE_DE" value=" "/>
  <column name="TYPE_RUB" value="[getRubricType(attr.type.toString())
      /]"/>
  <column name="RMAX" valueNumeric="0.0"/>
```

```
  <column name="AIDE_RUB" value="[aClass.name.toUpper() /].[attr.name
     /]"/>
  <column name="DICO" value=" "/>
  <column name="CHAI2" value=" "/>
  <column name="INPUTMAS" valueNumeric="0"/>
  <column name="FORMAT" value="[getFormat(attr.type.toString(), attr.name
     ) /]"/>
  <column name="NOM_RUB" value="[attr.name /]"/>
  <column name="NOM_TYPE" value="[aClass.name.toUpper()/]"/>
</insert>

[/for]

[comment reinitialiser le compteur /]
[initCounter(aClass.name) /]

<delete tableName="act_rub_type">
       <where>nom_type = ${delimiter}[aClass.name/]${delimiter} and
           nom_rub NOT IN (SELECT nom_rub FROM desctype where nom_type=${
           delimiter}[aClass.name/]${delimiter} AND action=${delimiter}
           ACT${delimiter})</where>
</delete>

<delete tableName="variantype">
       <where>nom_type = ${delimiter}[aClass.name/]${delimiter} and
           nom_rub NOT IN (SELECT nom_rub FROM desctype where nom_type=${
           delimiter}[aClass.name/]${delimiter} AND mode_rub =${delimiter}
           VO${delimiter} or mode_rub =${delimiter}VE${delimiter} or
           mode_rub =${delimiter}VP${delimiter})</where>
</delete>

<!--Supression des triggers-->
<!--Oracle-->
<sql dbms="oracle">
  drop trigger rsu_[aClass.name /]_art;
  drop trigger rin_[aClass.name /]_art;
  drop trigger rmo_[aClass.name /]_art;
</sql>

<!--Suppression des procedures-->
<dropProcedure procedureName="pin_[aClass.name /]_art" />
<dropProcedure procedureName="psu_[aClass.name /]_art" />
<dropProcedure procedureName="pmo_[aClass.name /]_art" />
<dropProcedure procedureName="pd_dt_[aClass.name /]" />

<!--Suppression des synonymes des procedures-->
<!--oracle-->
```

```
<sql dbms="oracle">
  drop public synonym pin_[aClass.name /]_art ;
  drop public synonym psu_[aClass.name /]_art ;
  drop public synonym pmo_[aClass.name /]_art ;
  drop public synonym pd_dt_[aClass.name /] ;
</sql>

<!--Suppression des index sur la table de la classe -->
<dropIndex indexName="DT_[aClass.name /]0" tableName="DT_[aClass.name /]"
     />
<dropIndex indexName="DT_[aClass.name /]1" tableName="DT_[aClass.name /]"
     />

<!--Ajout des nouveaux colonnes (S'ils en existent) e la table-->
[for (attr : Attribute | aClass.attribute)]
  [comment Test si l'attribut est nouveau. La propriete dirty = true /]
  [if (attr.dirty)]
    [attr.add(aClass) /]
  [/if]

[/for]


<!--Creation index0-->
<createIndex indexName="DT_[aClass.name /]0" tableName="DT_[aClass.name
    /]" tablespace="au_ts_index" unique="true">
         <column name="NUM_ART"/>
  </createIndex>

<!--Creation index1-->
<createIndex indexName="DT_[aClass.name /]1" tableName="DT_[aClass.name
    /]" tablespace="au_ts_index" unique="true">
         <column name="REF_UTILISAT"/>
         <column name="version desc"/>
         <column name="revision desc"/>
         <column name="AUDROS_PROVENANCE"/>
  </createIndex>

<!--Creation de procedure-->
<!--Procedure 1-->
<!--Oracle-->
<createProcedure
         dbms="oracle"
         encoding="utf8"
         procedureName="pin_[aClass.name /]_art">

    CREATE OR REPLACE procedure pin_[aClass.name /]_art(ref_utilisat_var
```

```
        varchar,date_creation_var
    date,num_art_var integer,createur_var varchar,version_var varchar,
        revision_var varchar,designation_var
    varchar,libelle_variante_var varchar,date_modif_var date,
        modificateur_var varchar,statut_var
    varchar,audros_provenance_var varchar,audros_systeme_var varchar,
        id_stat_var integer,id_su_cl_ob_var
    integer,sujet_var integer) as begin if audros_provenance_var = '[aClass
        .name /]' then insert into
    articles (num_int, nom_type, ref_user, date_creation, createur,
        version, revision, designation,
    libelle_variante,date_modif, statut,modificateur, id_stat, id_su_cl_ob
        ,sujet,revision2,revision3)
    values (num_art_var,'[aClass.name /]', ref_utilisat_var,
        date_creation_var, createur_var, version_var,
    revision_var, designation_var, libelle_variante_var, date_modif_var,
        statut_var, modificateur_var,
    id_stat_var, id_su_cl_ob_var, sujet_var, version_var, revision_var);
        insert into historiques(nom_classe,
    num_art1, statut1, reference1, version1, revision1, num_art2, statut2,
         reference2, version2,
    revision2, date_modif, objet_modif, code_user, code_modif,
        num_fonction) values ('[aClass.name /]',
    num_art_var, id_stat_var, ref_utilisat_var, version_var, revision_var,
         num_art_var, id_stat_var,
    ref_utilisat_var, version_var, revision_var, date_creation_var, '
        Object creation', createur_var,
    createur_var, 1); end if; end;
</createProcedure>

<!--Creation des synonymes-->
<!--Affectation des droits sur la procedure-->
<sql dbms="oracle">
  CREATE OR REPLACE public synonym pin_[aClass.name /]_art for audros.
      pin_[aClass.name /]_art ;
  grant all on pin_[aClass.name /]_art to public;
</sql>

<!--Procedure 2-->
<!--Oracle-->
<createProcedure
         dbms="oracle"
         encoding="utf8"
         procedureName="pmo_[aClass.name /]_art">

    CREATE OR REPLACE procedure pmo_[aClass.name /]_art(ref_utilisat_var
        varchar,date_creation_var
```

```
    date,num_art_var integer,createur_var varchar,version_var varchar,
        revision_var varchar,designation_var
    varchar,libelle_variante_var varchar,date_modif_var date,
        modificateur_var varchar,statut_var
    varchar,audros_provenance_var varchar,audros_systeme_var varchar,
        id_stat_var integer,id_su_cl_ob_var
    integer,sujet_var integer) as begin if audros_provenance_var = '[aClass
        .name /]' then update historiques
    set version2=version_var, revision2=revision_var where num_art2=
        num_art_var and reference2=ref_utilisat_var;
    update articles set ref_user=ref_utilisat_var, nom_type='[aClass.name /]
        ', date_creation=date_creation_var,
    createur=createur_var, version=version_var, revision=revision_var,
        designation=designation_var,
    libelle_variante=libelle_variante_var, date_modif=date_modif_var,
        statut=statut_var, modificateur=modificateur_var,
    id_stat=id_stat_var, sujet=sujet_var, id_su_cl_ob=id_su_cl_ob_var
        where num_int=num_art_var;
    end if; end;
</createProcedure>

<!--Creation des synonymes-->
<!--Affectation des droits sur la procedure-->
<sql dbms="oracle">
  CREATE OR REPLACE public synonym pmo_[aClass.name /]_art for audros.
      pmo_[aClass.name /]_art ;
  grant all on pmo_[aClass.name /]_art to public ;
</sql>

<!--Procedure 3-->
<!--Oracle-->
<createProcedure
         dbms="oracle"
         encoding="utf8"
         procedureName="psu_[aClass.name /]_art">

    CREATE OR REPLACE procedure psu_[aClass.name /]_art(num_art_var
        integer , audros_provenance_var
    varchar) as begin if audros_provenance_var = '[aClass.name /]' then
        delete from articles where
    num_int=num_art_var; end if; end;
</createProcedure>

<!--Creation des synonymes-->
<!--Affectation des droits sur la procedure-->
<sql dbms="oracle">
  CREATE OR REPLACE public synonym psu_[aClass.name /]_art for audros.
```

```
      psu_[aClass.name /]_art ;
    grant all on psu_[aClass.name /]_art to public ;
  </sql>

  <!--Procedure 4-->
  <!--Oracle-->
  <createProcedure
          dbms="oracle"
          encoding="utf8"
          procedureName="pd_dt_[aClass.name /]">

     CREATE OR REPLACE procedure pd_dt_[aClass.name /](nn_var integer ,
        na_var integer , audros_provenance_var
     varchar, ref_user_var varchar, ref_var varchar) as begin if
        audros_provenance_var = '[aClass.name /]'
     then delete from articles where num_int=nn_var; end if; end;
  </createProcedure>

  <!--Creation des synonymes-->
  <!--Affectation des droits sur la procedure-->
  <sql dbms="oracle">
    CREATE OR REPLACE public synonym pd_dt_[aClass.name /] for audros.
       pd_dt_[aClass.name /] ;
    grant all on pd_dt_[aClass.name /] to public ;
  </sql>

  <!--Creation des triggers-->
  <!--oracle-->
  <sql dbms="oracle">

CREATE OR REPLACE trigger rin_[aClass.name /]_art after insert on dt_[
    aClass.name /] for each row begin
pin_[aClass.name /]_art (ref_utilisat_var => :new.ref_utilisat,
    date_creation_var => :new.date_creation,num_art_var
=> :new.num_art,createur_var => :new.createur,version_var => :new.version,
    revision_var => :new.revision,designation_var
=> :new.designation,libelle_variante_var => :new.libelle_variante,
    date_modif_var => :new.date_modif,modificateur_var
=> :new.modificateur,statut_var => :new.statut,audros_provenance_var => :
    new.audros_provenance,audros_systeme_var
=> :new.audros_systeme,id_stat_var => :new.id_stat,id_su_cl_ob_var => :new.
    id_su_cl_ob,sujet_var
=> :new.sujet); end;

CREATE OR REPLACE trigger rmo_[aClass.name /]_art after update on dt_[
    aClass.name /] for each row begin
pmo_[aClass.name /]_art (ref_utilisat_var => :new.ref_utilisat,
```

```
    date_creation_var => :new.date_creation,num_art_var
=> :new.num_art,createur_var => :new.createur,version_var => :new.version,
    revision_var => :new.revision,designation_var
=> :new.designation,libelle_variante_var => :new.libelle_variante,
    date_modif_var => :new.date_modif,modificateur_var
=> :new.modificateur,statut_var => :new.statut,audros_provenance_var => :
    new.audros_provenance,audros_systeme_var
=> :new.audros_systeme,id_stat_var => :new.id_stat,id_su_cl_ob_var => :new.
    id_su_cl_ob,sujet_var
=> :new.sujet); end;


CREATE OR REPLACE trigger rsu_[aClass.name /]_art after delete on dt_[
    aClass.name /] for each row begin
psu_[aClass.name /]_art (num_art_var => :old.num_art,audros_provenance_var
    => :old.audros_provenance)
end;

  </sql>
[/template]
```

Listing C.18 – Code du template classModifier.mtl

- templates de l'élément Relationship : de la même façon que pour le type Class, les différentes templates associées au type Relationship permettent de créer, de modifier et de supprimer les éléments de type Relationship dans la BD.
- template de l'élément Attribut : l'élément Attribut dispose de plusieurs templates regroupées dans le fichier AttributeGenerator.mtl. Certaines templates existent en 2 versions (sur-chargement), une appliquée sur les éléments de type Class et l'autre sur les Relationship.

```
[comment encoding = UTF-8 /]
[module AttributeGenerator('http://dbmetamodel/1.0', 'http://www.eclipse.org/
    emf/2002/Ecore')]
[import fr::audros::acms::modelEditor::codeGenerator::common::services::AttributeUtil
    /]


[template public create(anAttribute : Attribute)]
  <column name="[anAttribute.name.toUpper()/]" type="[convertType(anAttribute.
      type.toString()) /]([convertLength(anAttribute.type.toString(),anAttribute.len.
      toString()) /])" [convertDefault(anAttribute.type.toString()) /]="[
      convertDefaultValue(anAttribute.type.toString(), anAttribute.default.toString())
      /]">
    <constraints nullable="[anAttribute.nullable/]" />
  </column>
[/template]

[comment Ajouter un attribut a une classe /]
```

```
[template public add(anAttribute : Attribute,aClass:Class)]

  <sql dbms="oracle"
    splitStatements="false"
    >
    BEGIN
    execute immediate 'alter table DT_[aClass.name /] drop column [
        anAttribute.name.toUpper()/] ';
    EXCEPTION
      when others then
        null;
    END;
  </sql>

  <addColumn tableName="DT_[aClass.name /]">
                  [self.create() /]
  </addColumn>
[/template]

[comment Ajouter un attribut a une relation /]
[template public add(anAttribute : Attribute,aRelationship:Relationship)]

  <sql dbms="oracle"
    splitStatements="false"
    >
    BEGIN
    execute immediate 'alter table DS_[aRelationship.name /] drop column [
        anAttribute.name.toUpper()/] ';
    EXCEPTION
      when others then
        null;
    END;
  </sql>

  <addColumn tableName="DS_[aRelationship.name /]">
                  [self.create() /]
  </addColumn>
[/template]

[comment Supprimer un attribut d'une classe /]
[template public drop(anAttribute : Attribute,aClass:Class)]
  <dropColumn columnName="[self.name /]" tableName="DT_[aClass.name /]" />

[/template]

[comment Supprimer un attribut d'une classe /]
[template public drop(anAttribute : Attribute,aRelationship:Relationship)]
```

```
  <dropColumn columnName="[self.name /]" tableName="DS_[aRelationship.name
      /]" />
[/template]

[comment Renommer un attribut d'une classe /]
[template public rename(anAttribute : Attribute,aClass:Class)]
  <renameColumn
          columnDataType="[convertType(self.type.toString()) /]([self.len
              /])"
          newColumnName="[self.name /]"
          oldColumnName="[self.oldName/]"
          tableName="DT_[aClass.name /]"/>
[/template]

[comment Renommer un attribut d'une relation /]
[template public rename(anAttribute : Attribute,aRelationship:Relationship)]
  <renameColumn
          columnDataType="[convertType(self.type.toString()) /]([self.len  /])" [
              comment necessaire pour MySQL /]
          newColumnName="[self.name /]"
          oldColumnName="[self.oldName/]"
          tableName="DS_[aRelationship.name /]"/>
[/template]

[comment changer le type d'un attribut d'une classe /]
[template public modifyType(anAttribute : Attribute,aClass:Class)]
  <modifyDataType
            columnName="[self.name /]"
            newDataType="[convertType(self.type.toString()) /]([self.len  /])" [
                comment necessaire pour MySQL /]
            tableName="DT_[aClass.name /]"/>
[/template]

[comment changer le type d'un attribut d'une relation /]
[template public modifyType(anAttribute : Attribute,aRelationship:Relationship)]
  <modifyDataType
            columnName="[self.name /]"
            newDataType="[convertType(self.type.toString()) /]([self.len  /])"
            tableName="DS_[aRelationship.name /]"/>
[/template]
```

Listing C.19 – Code du template AttributeGenerator.mtl

Les services sont un moyen pour étendre les fonctionnalités offertes dans les templates Acceleo. A l'origine, les fonctionnalités disponibles par défaut sont limitées : accéder aux propriétés de chaque élément, ses fils... Afin de pouvoir récupérer par exemple le Timestamp courant, il faut créer un service qui n'est que du code Java encapsulé dans des templates Acceleo. Dans notre cas, les classes java qui définissent les différentes méthodes sont

situées dans le package fr.audros.acms.modelEditor.services et les templates les encapsulant dans le package fr.audros.acms.modelEditor.codeGenerator.common.services. Les classes java sont définies de façon ordinaire.

– la classe AttributeUtil en java et son encapsulation par une template

```
package fr.audros.acms.modelEditor.services;

public class AttributeUtil {

  private static final String DEFAULT_STRING_LENGTH = "64";
  private static final String DEFAULT_MEMO_LENGTH = "0";
  private static final String DEFAULT_DATE_LENGTH = "0";
  private static final String DEFAULT_INTEGER_LENGTH = "0";
  private static final String DEFAULT_DOUBLE_LENGTH = "0";

  //Convertir le type decrit dans le metamodele en un type SQL
  public String convertType(String typeName) {
    String newType="";
    if(typeName.equalsIgnoreCase("Chaine")) {
      newType= "java.sql.Types.VARCHAR";
    }
    else if(typeName.equalsIgnoreCase("memo")) {
      newType= "java.sql.Types.NVARCHAR";
    }

    else if(typeName.equalsIgnoreCase("Entier")) {
      newType= "java.sql.Types.INTEGER";
    }

    else if(typeName.equalsIgnoreCase("Reel")) {
      newType= "java.sql.Types.DOUBLE";
    }
    else if(typeName.equalsIgnoreCase("Date")) {
      newType= "java.sql.Types.DATE";
    }

    return newType;
  }

  //Retourne les longueurs par defaut des types si non specifiee
  public String convertLength(String typeName,String length) {
    if(!length.equals("")) {
      return length;
    }
    else {
      String len="";
      if(typeName.equalsIgnoreCase("Chaine")) {
        len=DEFAULT_STRING_LENGTH;
```

```
        }
        else if(typeName.equalsIgnoreCase("memo")) {
          len=DEFAULT_MEMO_LENGTH;
        }

        else if(typeName.equalsIgnoreCase("Entier")) {
          len=DEFAULT_INTEGER_LENGTH;
        }

        else if(typeName.equalsIgnoreCase("Reel")) {
          len=DEFAULT_DOUBLE_LENGTH;
        }
        else if(typeName.equalsIgnoreCase("Date")) {
          len=DEFAULT_DATE_LENGTH;
        }

        return len;
      }

    }

    //Retourne l'expression "default" suivant le type de la donnees par defaut (Imposee
        par LiquiBase)
    public String convertDefault(String typeName) {
      String defaultType="";
      if(typeName.equalsIgnoreCase("Chaine")) {
        defaultType= "defaultValue";
      }
      else if(typeName.equalsIgnoreCase("memo")) {
        defaultType= "defaultValue";
      }

      else if(typeName.equalsIgnoreCase("Entier")) {
        defaultType= "defaultValueNumeric";
      }

      else if(typeName.equalsIgnoreCase("Reel")) {
        defaultType= "defaultValueNumeric";
      }
      else if(typeName.equalsIgnoreCase("Date")) {
        defaultType= "defaultValueDate";
      }

      return defaultType;
    }

    //Retourne l'expression "default" suivant le type de la donnees par defaut (Imposee
```

```
        par LiquiBase)
    public String convertDefaultValue(String typeName,String defaultValue) {

      if(defaultValue.equalsIgnoreCase("")) {
        String defValue="NULL";
        if(typeName.equalsIgnoreCase("Chaine")) {
          defValue= " ";
        }
        else if(typeName.equalsIgnoreCase("memo")) {
          defValue= " ";
        }

        else if(typeName.equalsIgnoreCase("Entier")) {
          defValue= "0";
        }

        else if(typeName.equalsIgnoreCase("Reel")) {
          defValue= "0.0";
        }
        else if(typeName.equalsIgnoreCase("Date")) {
          defValue= "NULL";
        }

        return defValue;
      }
      else {
        return defaultValue;
      }


    }

}
```

Listing C.20 – Code java de la classe AttributeUtil

```
[comment encoding = Cp1252 /]
[module AttributeUtil('http://dbmetamodel/1.0')/]

[query public convertType(arg0 : String) : String
  = invoke('fr.audros.acms.modelEditor.services.AttributeUtil', 'convertType(
      java.lang.String)', Sequence{arg0}) /]

[query public convertDefaultValue(arg0 : String, arg1 : String) : String
  = invoke('fr.audros.acms.modelEditor.services.AttributeUtil', '
      convertDefaultValue(java.lang.String, java.lang.String)', Sequence{arg0,
       arg1}) /]
```

```
[query public convertDefault(arg0 : String) : String
  = invoke('fr.audros.acms.modelEditor.services.AttributeUtil', '
      convertDefault(java.lang.String)', Sequence{arg0}) /]

[query public convertLength(arg0 : String, arg1 : String) : String
  = invoke('fr.audros.acms.modelEditor.services.AttributeUtil', '
      convertLength(java.lang.String, java.lang.String)', Sequence{arg0, arg1})
       /]
```

Listing C.21 – encapsulation de la classe AttributeUtil

– la classe ClassifierUtil en java et son encapsulation par une template

```
package fr.audros.acms.modelEditor.services;

public class ClassifierUtil {

  //Retourne la valeur de la colonne Gestion de la table DESCTYPE
  public String getManagementMode(String columnName) {
    if(columnName.equals("designation")) {
      return " ";
    }
    else {
      return "OB";
    }

  }

  //Retourne la valeur de la colonne TypeRubrique de la table DESCTYPE
  public String getRubricType(String columnType) {
    if(columnType.equalsIgnoreCase("int")) {
      return "EN";
    }
    if(columnType.equalsIgnoreCase("date")) {
      return "DA";
    }
    if(columnType.equalsIgnoreCase("String")) {
      return "CH";
    }
    if(columnType.equalsIgnoreCase("memo")) {
      return "CH";
    }
    else {
      return " ";
    }

  }

  //Retourne la valeur de la colonne Format de la table DESCTYPE
```

```
  public String getFormat(String columnType,String columnName) {
    if(columnName.equalsIgnoreCase("ref_utilisat")) {
      return "MAJ";
    }
    else {
      if(columnType.equalsIgnoreCase("String")) {
        return "MIX";
      }
      if(columnType.equalsIgnoreCase("memo")) {
        return "MIX";
      }
      else {
        return " ";
      }
    }
  }

}
```

Listing C.22 – Code java de la classe ClassifierUtil

```
[comment encoding = Cp1252 /]
[module ClassifierUtil('http://dbmetamodel/1.0')/]

[query public getFormat(arg0 : String, arg1 : String) : String
  = invoke('fr.audros.acms.modelEditor.services.ClassifierUtil', 'getFormat(
      java.lang.String, java.lang.String)', Sequence{arg0, arg1}) /]

[query public getManagementMode(arg0 : String) : String
  = invoke('fr.audros.acms.modelEditor.services.ClassifierUtil', '
      getManagementMode(java.lang.String)', Sequence{arg0}) /]

[query public getRubricType(arg0 : String) : String
  = invoke('fr.audros.acms.modelEditor.services.ClassifierUtil', '
      getRubricType(java.lang.String)', Sequence{arg0}) /]
```

Listing C.23 – encapsulation de la classe ClassifierUtil

– la classe Utilities en java et son encapsulation par une template

```
package fr.audros.acms.modelEditor.services;

import java.io.File;
import java.io.IOException;
import java.nio.charset.Charset;
import java.sql.Timestamp;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.List;
```

```
import java.util.Random;

import org.apache.commons.io.FileUtils;

public class Utilities {

  //Valeur du compteur par defaut
  private static final int DEFAULT_COUNTER=1;

  //Chemin par defaut du fichier du compteur
  private static final String COUNTER_DIR="DBMigration";
  private static final String COUNTER_FILENAME=COUNTER_DIR+File.
      separator+"counter";
  private static final String COUNTER_CHANGESET_FILENAME=
      COUNTER_DIR+File.separator+"changeSetCounter";
  //format des dates adopte pour nommer les logs de migration de la base de donnees
  private SimpleDateFormat dateFormat=new SimpleDateFormat("yyyy-MM-dd-HH-mm
      -ss");

  //Retourne la date courante formattee afin de l'utiliser comme nom des changelogs
      Liquibase
  //@param : -
  //@return : la date courante formattee
  public String getModuleID() {
    return dateFormat.format(new Date());

  }

  //Retourne un timestamp afin de l'utiliser comme ID des changeset Liquibase
  //@param : -
  //@return : Timestamp
  public String getChangesetID() {
    return new Timestamp(new Date().getTime()).toString();

  }

  //Retourne la valeur suivante du compteur - A utiliser lors de la saisie des
      attributs dans DescType et DescArbo
  //@param : - Une valeur aleatoire pour forcer Acceleo a ne pas utiliser le cache de
      cette methode (Nouveau Appel a la methode)
  //@return : La valeur suivante du compteur
  public static int getNextNumber(String unused){
    try {
      //Recuperation des differents lignes contenu dans le fichier du compteur
      List<String> entries=FileUtils.readLines(new File(COUNTER_FILENAME),
          Charset.defaultCharset());
      try{
```

```
        //Cast
        int counter=Integer.parseInt(entries.get(0));
        FileUtils.write(new File(COUNTER_FILENAME),String.valueOf(counter+1)
            );
        //renvoi de la valeur suivante
        return counter;
      }catch(Exception e) {
        //renvoie de la valeur par defaut du compteur en cas d'erreur
        return DEFAULT_COUNTER;
      }
    } catch (IOException e) {
      //renvoie de la valeur par defaut du compteur en cas d'erreur
      try {
        FileUtils.write(new File(COUNTER_FILENAME),String.valueOf(
            DEFAULT_COUNTER+1));
      } catch (IOException e1) {
        // TODO Auto-generated catch block
        e1.printStackTrace();
      }
      return DEFAULT_COUNTER;
    }

  }

  public static int getNextChangeSetNumber(String unused){
    try {
      //Recuperation des differents lignes contenu dans le fichier du compteur
      List<String> entries=FileUtils.readLines(new File(
          COUNTER_CHANGESET_FILENAME), Charset.defaultCharset());
      try{
        //Cast
        int counter=Integer.parseInt(entries.get(0));
        FileUtils.write(new File(COUNTER_CHANGESET_FILENAME),String.
            valueOf(counter+1));
        //renvoi de la valeur suivante
        return counter;
      }catch(Exception e) {
        //renvoie de la valeur par defaut du compteur en cas d'erreur
        return DEFAULT_COUNTER;
      }
    } catch (IOException e) {
      //renvoie de la valeur par defaut du compteur en cas d'erreur
      return DEFAULT_COUNTER;
    }

  }
```

```
  public int getRandomNumber() {
    Random ran=new Random();
    return (int) (ran.nextDouble()*10000);
  }
  // Reinitialiser le compteur
  //@param : - Une valeur aleatoire pour forcer Acceleo a ne pas utiliser le cache de
      cette methode (Nouveau Appel a la methode)
  //@return : -
  public static void initCounter(String unused){
    try {
      //Reecriture de la valeur par defaut du compteur dans le fichier
      FileUtils.write(new File(COUNTER_FILENAME),String.valueOf(
          DEFAULT_COUNTER));
    } catch (IOException e) {
      // TODO Auto-generated catch block
      e.printStackTrace();
    }
  }

}
```

Listing C.24 – Code java de la classe Utilities

```
[comment encoding = Cp1252 /]
[module Utilities('http://dbmetamodel/1.0')/]

[query public getNextChangeSetNumber(arg0 : String) : Integer
  = invoke('fr.audros.acms.modelEditor.services.Utilities', '
      getNextChangeSetNumber(java.lang.String)', Sequence{arg0}) /]

[query public getModuleID() : String
  = invoke('fr.audros.acms.modelEditor.services.Utilities', 'getModuleID()',
      Sequence{}) /]

[query public getChangesetID() : String
  = invoke('fr.audros.acms.modelEditor.services.Utilities', 'getChangesetID()
      ', Sequence{}) /]

[query public getRandomNumber() : Integer
  = invoke('fr.audros.acms.modelEditor.services.Utilities', 'getRandomNumber
      ()', Sequence{}) /]

[query public initCounter(arg0 : String) : OclVoid
  = invoke('fr.audros.acms.modelEditor.services.Utilities', 'initCounter(java
      .lang.String)', Sequence{arg0}) /]

[query public getNextNumber(arg0 : String) : Integer
  = invoke('fr.audros.acms.modelEditor.services.Utilities', 'getNextNumber(
```

```
    java.lang.String)', Sequence{arg0}) /]
```

Listing C.25 – encapsulation de la classe Utilities

C.2 La comparaison de modèles

Lors de l'injection d'un nouveau modèle métier dans une plateforme définie, il faut garder les éléments qui n'ont pas subis de modifications. Le processus de comparaison nous permet de détecter les différences entre les différentes versions du modèle métier et d'implémenter seulement les différences. Le chargement du modèle métier courant et de l'ancienne version (oldVersion) est la responsabilité de la classe ModelLoader du plugin de comparaison (fig. C.1).

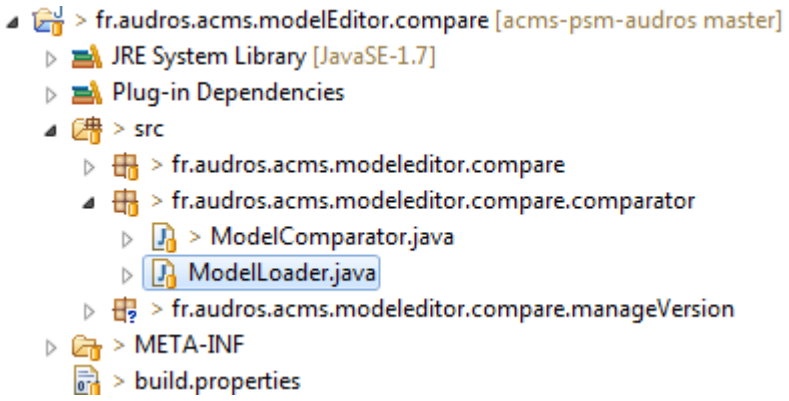

FIGURE C.2 – Description du plugin de comparaison de modèles

Le chargement est effectué dans la méthode compare de la classe ModelComparator :

```
package fr.audros.acms.modeleditor.compare.comparator;

import java.io.IOException;
import java.util.Collections;
import java.util.Iterator;
import java.util.List;

import org.eclipse.core.resources.IFile;
import org.eclipse.emf.common.util.URI;
import org.eclipse.emf.compare.Comparison;
import org.eclipse.emf.compare.Diff;
import org.eclipse.emf.compare.EMFCompare;
import org.eclipse.emf.compare.match.DefaultComparisonFactory;
import org.eclipse.emf.compare.match.DefaultEqualityHelperFactory;
import org.eclipse.emf.compare.match.DefaultMatchEngine;
import org.eclipse.emf.compare.match.IComparisonFactory;
import org.eclipse.emf.compare.match.IMatchEngine;
import org.eclipse.emf.compare.match.eobject.IEObjectMatcher;
```

```
import org.eclipse.emf.compare.match.impl.MatchEngineFactoryImpl;
import org.eclipse.emf.compare.match.impl.MatchEngineFactoryRegistryImpl;
import org.eclipse.emf.compare.scope.IComparisonScope;
import org.eclipse.emf.compare.utils. UseIdentifiers ;
import org.eclipse.emf.ecore.resource.Resource;
import org.eclipse.emf.ecore.resource.ResourceSet;
import org.eclipse.emf.ecore.resource.impl.ResourceSetImpl;

public class ModelComparator {
  private static String leftModel,rightModel;

  public void LaunchComparison(String leftModelPath,String rightModelPath, IFile model) {
    System.out.println("starting comparison");
    Comparison compRes=compare(leftModelPath,rightModelPath);
    System.out.println("starting differences");

    ResourceSet ressourceSet=new ResourceSetImpl();
    Resource res=ressourceSet.createResource(URI.createURI("/"+model.getProject().
        getName()+"/diffs.emfdiff"));

    leftModel=leftModelPath;
    rightModel=rightModelPath;

    List<Diff> diff=compRes.getDifferences();
    for (Iterator<Diff> iterator = diff.iterator(); iterator.hasNext();) {
      Diff oneDiff = (Diff) iterator.next();

      //System.out.println("kind: "+oneDiff.getKind().getName());
      //System.out.println("State: "+oneDiff.getState().getName());

      res.getContents().add(oneDiff);

    }

    try {

      res.save(Collections.EMPTY_MAP);
    } catch (IOException e) {
      e.printStackTrace();
    }

  }
  public Comparison compare(String leftModelPath,String rightModelPath) {
    // Load the two input models
    ResourceSet leftResourceSet = new ResourceSetImpl();
    ResourceSet rightResourceSet = new ResourceSetImpl();
    ModelLoader.load(leftModelPath, leftResourceSet);
```

```
    ModelLoader.load(rightModelPath,rightResourceSet);

    // Configure EMF Compare
    IEObjectMatcher matcher = DefaultMatchEngine.createDefaultEObjectMatcher(
        UseIdentifiers.NEVER);
    IComparisonFactory comparisonFactory = new DefaultComparisonFactory(new
        DefaultEqualityHelperFactory());
    IMatchEngine.Factory matchEngineFactory = new MatchEngineFactoryImpl(matcher,
        comparisonFactory);
          matchEngineFactory.setRanking(20);
          IMatchEngine.Factory.Registry matchEngineRegistry = new
              MatchEngineFactoryRegistryImpl();
          matchEngineRegistry.add(matchEngineFactory);
    EMFCompare comparator = EMFCompare.builder().setMatchEngineFactoryRegistry(
        matchEngineRegistry).build();


    // Compare the two models
    IComparisonScope scope = EMFCompare.createDefaultScope(leftResourceSet,
        rightResourceSet);
    return comparator.compare(scope);
  }
  public static String getLeftModel() {
    return leftModel;
  }
  public static String getRightModel() {
    return rightModel;
  }

}
```

Listing C.26 – Code java du fichier ModelComparator.java

Au lancement de la comparaison, le fichier représentant l'ancienne version du modèle métier implémenté est récupéré (oldVersion). Si le fichier n'existe pas dans le workspace, il sera créé à partir d'un fichier standard dans le répertoire ModelComparator accompagnant la distribution Eclipse. Le fichier diffs.diffModel est un modèle résultant de la comparaison des différentes versions du modèle métier et regroupant toutes les différences détectées. Il est créé automatiquement dans le workspace après chaque comparaison.

Les règles de transformation des différences détectées entre les versions des modèles métiers sont définies par le langage ATL dans le modèle business2db.atl du plugin fr.audros.acms.modelE
Il faut inclure le méta-modèle de l'EMF Compare avec ceux du méta-modèle PLM et d'Audros. Le modèle de transformation prend en entrée le modèle des différences (diffModel) et le modèle PLM courant et produit en sortie un modèle Audros. La transformation d'une différence en un élément du méta-modèle Audros est réalisée par des Lazy Rules, c'est-à-dire qu'une différence au niveau d'un BusinessObject est transformée en une Class :

```
-- @path Business=http://acms/1.0
```

```
-- @path DB=http://dbmetamodel/1.0
-- @nsURI DiffModel=http://www.eclipse.org/emf/compare

module business2db;
create OUT : DB from IN : Business, IN2: DiffModel;

--Helpers
--Verifier que c'est le premiere execution de ce module
helper def : isFirstRun() : Boolean =
  if DB!Model.allInstances()->size() = 0 then
    true
  else
    false
  endif;

--Verifier que la cible des changements est une entite appartenant a l'element Classifier
helper context DiffModel!Diff def : belongsToclassifier() : Boolean =

  if self.isBusinessObject() or
     self.isFunctionnalLink()
     --or
     --self.isClassAttribute() or
     --self.isRelationAttribute()
     then

     true

  else
     false
  endif;


-- Verifier que la cible des changements est un objet metier
helper context DiffModel!Diff def : isBusinessObject() : Boolean =

  if (self.value.oclIsTypeOf(Business!BusinessObject) and self.requires->size() = 0) then
    true
  else
    false
  endif;

-- Verifier que la cible des changements est un lien metier
helper context DiffModel!Diff def : isFunctionalLink() : Boolean =

  if (self.value.oclIsTypeOf(Business!FunctionalLink) and self.requires->size() = 0) then
    true
  else
```

```
    false
  endif;

-- Verifier que la cible des changements est un element Attribut d'un objet metier
helper context DiffModel!Diff def : isClassAttribute() : Boolean =

  if self.oclIsTypeOf(DiffModel!ReferenceChange) then

    if self.value.oclIsKindOf(Business!Attribute) then
      if not self.value.eContainer().oclIsUndefined() then
        if self.value.eContainer().oclIsTypeOf(Business!BusinessObject) then
          if self.requires->size() = 0 then
            true
          else
            false
          endif
        else
          false
        endif
      else
        false
      endif
    else
      false
    endif
  else
    false
  endif;

-- Verifier que la cible des changements est un l'element Attribut d'un lien metier
helper context DiffModel!Diff def : isRelationAttribute() : Boolean =

  if self.oclIsTypeOf(DiffModel!ReferenceChange) then

    if self.value.oclIsKindOf(Business!Attribute) then
      if not self.value.eContainer().oclIsUndefined() then
        if self.value.eContainer().oclIsTypeOf(Business!FunctionalLink) then
          if self.requires->size() = 0 then
            true
          else
            false
          endif
        else
          false
        endif
      else
        false
```

```
      endif
    else
      false
    endif
  else
    false
  endif;

-- Changement de nom de l'objet metier
helper context DiffModel!Diff def : isClassRename() : Boolean =

  --puisque c'est une logique binaire
  -- on imbrique les conditions sinon on aura des erreurs puisque il evalue tous les
      conditions
  -- meme celles pouvant produire des exceptions
  if  self.oclIsKindOf(DiffModel!AttributeChange) then
    if  self.attribute.eContainer().name = 'BusinessObject' then
      if  self.attribute.name = 'name_object' then
       true
      else
        false
      endif
    else
      false
    endif
  else
    false
  endif;

-- Changement de nom de l'objet metier
helper context DiffModel!Diff def : isRelationRename() : Boolean =

  --puisque c'est une logique binaire
  -- on imbrique les conditions
  if  self.oclIsKindOf(DiffModel!AttributeChange) then
    if  self.attribute.eContainer().name = 'FunctionalLink' then
      if  self.attribute.name = 'name_link' then
        true
      else
        false
      endif
    else
      false
    endif
  else
    false
  endif;
```

```
--Lazy rules
--Transformations des differences
--Transforme les changements ayant comme cible les objets metier
lazy rule BusinessObject2Class {
  from
    diff :DiffModel!Diff
  to
    c : DB!Class (
      name <- diff.value.name_object,
      Change <- diff.kind,
      attribute <- diff.value.ListObjAttr->collect(attr | thisModule.
          BusinessAttribute2Attribute(attr))
    )
}
--Transforme les changements ayant comme cible les objets metier
lazy rule BusinessRelationship2Relationship {
  from
    diff :DiffModel!Diff
  to
    r : DB!Relationship (
      name <- diff.value.name_link,
      Change <- diff.kind,
      attribute <- diff.value.ListLinkAttr->collect(attr | thisModule.
          BusinessAttribute2Attribute(attr))
    )
}

lazy rule BusinessObjectAttribut2ClassAttribut {
  from
    diff :DiffModel!Diff
  to
    r : DB!Class (
      name <- diff.value.eContainer().name_object,
      Change <- diff.kind.toString()+'ATTR',
      attribute <- thisModule.BusinessAttribute2Attribute(diff.value)
    )
}

lazy rule BusinessLinkAttribut2RelationshipAttribut {
  from
    diff :DiffModel!Diff
  to
    r : DB!Relationship (
      name <- diff.value.eContainer().name_link,
      Change <- diff.kind.toString()+'ATTR',
      attribute <- thisModule.BusinessAttribute2Attribute(diff.value)
    )
```

```
}

lazy rule BusinessObjectRename {
  from
    diff :DiffModel!Diff
  to
    r : DB!Class (
      name <- diff.value,
      Change <- 'RENAME',
      oldName <- diff.value
    )
}

lazy rule BusinessLinkRename {
  from
    diff :DiffModel!Diff
  to
    r : DB!Relationship (
      name <- diff.value,
      Change <- 'RENAME',
      oldName <- diff.value
    )
}

--Transformations standard des elements
-- Regle de transformation d'un attribut du modele metier en un attribut du modele BD
lazy rule BusinessAttribute2Attribute {
  from
    ba : Business!Attribute
  to
    a : DB!Attribute (
      name <- ba.Nom,
      format <- ba.Format,
      len <- ba.Longueur,
      type <- ba.Type.toString().toLower(),
      mode <- ba.Mode,
      nullable <- ba.Obl

    )
}

-- Transforamtions des elements conteneurs
lazy rule Diffs2DBClassifier {
  from
    diff :DiffModel!Diff
  to
    classifier :DB!Classifier (
```

```
        class <- DiffModel!Diff.allInstances()->select(oneDiff |oneDiff.isBusinessObject())->
            collect(oneDiff|thisModule.BusinessObject2Class(oneDiff)),
        relationship <- DiffModel!Diff.allInstances()->select(oneDiff |oneDiff.
            isFunctionalLink())->collect(oneDiff|thisModule.BusinessRelationship2Relationship
            (oneDiff)),

        class <- DiffModel!Diff.allInstances()->select(oneDiff |oneDiff.isClassAttribute())->
            collect(oneDiff|thisModule.BusinessObjectAttribut2ClassAttribut(oneDiff)),
        relationship <- DiffModel!Diff.allInstances()->select(oneDiff |oneDiff.
            isRelationAttribute())->collect(oneDiff|thisModule.
            BusinessLinkAttribut2RelationshipAttribut(oneDiff)),

        class <- DiffModel!Diff.allInstances()->select(oneDiff |oneDiff.isClassRename())->
            collect(oneDiff|thisModule.BusinessObjectRename(oneDiff)),
        relationship <- DiffModel!Diff.allInstances()->select(oneDiff |oneDiff.
            isRelationRename())->collect(oneDiff|thisModule.BusinessLinkRename(oneDiff))
    )
}


--Matching rules
-- Regle de transformation de la racine des modeles
rule diffModel2BDModel {
  from
    diffs : DiffModel!Diff (thisModule.isFirstRun())
  to
    bdm : DB!Model (
      classes <- thisModule.Diffs2DBClassifier(diffs)
    )
}
```

Listing C.27 – Code ATL pour la comparaison de modèle

Après une exécution réussie des requêtes SQL de migration, le modèle métier courant devient l'ancienne version (oldVersion).

Résumé

Le déploiement, la maintenance et l'évolution des systèmes d'informations techniques qui accompagnent les processus de création de produits (PLM) constituent des tâches complexes et parfois onéreuses pour des structures de type PME ou micro entreprise innovantes. Si l'appui sur un progiciel développé et maintenu par un éditeur permet aujourd'hui des solutions pérennes, la question de l'évolution conjointe des processus métiers de l'entreprise et du progiciel suite aux évolutions techniques de l'ingénierie numérique pose aux chercheurs la problématique des méthodologies à mettre en œuvre pour faciliter ce double axe d'évolution. Dans le cadre de ses démarches d'innovation, l'éditeur progiciel souhaite rendre accessibles à ses clients PME les capacités fonctionnelles de sa solution PLM en développant un atelier de modélisation pour la création de modèles métier au sein des systèmes PLM et la gestion de leur cohérence au cours du temps. Ce projet, réalisé dans le cadre d'une thèse CIFRE avec la société AUDROS, a pour but de fournir les concepts et les outils qui simplifient la synchronisation des différents outils métiers au sein du système d'information dans le but de gérer l'entreprise de façon la plus étendue et la plus homogène possible.

Mots-clés: PLM (Product Lifecycle Management), évolution, contraintes, transformations, IDM (Ingénierie dirigée par les modèles)

Abstract

The specification, the deployment, the maintenance and the evolution of technical information systems which support the processes of products developement (PLM) constitute complex tasks for organisations like SME or innovative companies. If today the support on a software package developed and maintained by an editor allows long-lasting solutions, researchers face the issue of the convergent evolutions of the business processes and the software package. As the technical evolutions embedded in digital engineering, this thesis proposes methodologies to be implemented to facilitate this double axis of evolution. Within his innovation framework, the software editor wishes to provide his customers with PLM systems evolution tools. This project, granted as an industrial thesis with AUDROS company, aims at supplying the concepts and the tools which simplify the synchronization of the various business tools within the information system. The company can thus be dynamically supported in a context of extended enterprise.

Keywords: PLM (Product Lifecycle Management), evolution, constraints, tranformations, MDE (Model Driven Engineering