Transformation et vérification de cohérence entre modèles du Génie Logiciel et modèles de l'Interface Homme-Machine

Jorge Luis PEREZ-MEDINA*, - Stéphanie MARSAL-LAYAT***

**LIG, équipe SIGMA – Université de Grenoble*
BP 72 38402 Saint Martin d'Hères Cedex, France
{Jorge-Luis.Perez-Medina, Stéphanie.Marsal-Layat}@imag.fr

***LIG, équipe IIHM – Université de Grenoble*
B.P. 53 38041 Grenoble Cedex 9
Jorge-Luis.Perez-Medina@imag.fr

Catégorie : Jeune Chercheur

RÉSUMÉ. *Dans le cadre de la conception collaborative, il est nécessaire que les spécialistes de divers domaines puissent gérer, de manière automatisée, les liens entre des modèles différents qui sont proches sémantiquement. Basé sur un exemple concret de transformation d'un diagramme de cas d'utilisation (Génie Logiciel) en un arbre de tâches (IHM), des langages proposés par l'Ingénierie dirigée par les modèles (IDM) sont évalués. Le but est alors de trouver un langage adéquat à notre transformation de modèles. Une comparaison de certains outils d'IDM est ensuite réalisée par rapport à des critères établis suivant nos besoins en conception collaborative. Les outils correspondant le mieux à nos attentes sont alors plus précisément testés à partir de l'exemple concret de la transformation.*

ABSTRACT. *Within the framework of collaborative design, it is necessary that specialists of any domain can automatically manage links between different models that are semantically close. Based on a concrete transformation example from a use case diagram (Software Engineering) to a task tree (HCI), languages proposed by model driven engineering (MDE) are evaluated. Our goal is to find an adequate language for our model transformation. Among the multitude of MDE tools, a comparison is realised with respect to criteria according to our collaborative design needs. The tools that best correspond to our expectations are then tested more thoroughly on the concrete example of the transformation.*

MOTS-CLÉS : *Conception collaborative, IDM, transformation de modèles, langages de transformation, outils pour l'IDM*

KEYWORDS : *Collaborative design, MDE, model transformation, transformation languages, MDE Tools*

1. Introduction

Généralement, le développement d'un système d'information s'appuie sur des méthodes qui proposent des modèles et des démarches pour produire des vues cohérentes, complémentaires et de différents niveaux d'abstraction du système d'information à implanter. Deux points de vue sont en particulier abordés : l'étude des fonctionnalités du futur système par les spécialistes du génie logiciel (GL) et son utilisabilité par les ergonomes et les spécialistes de l'interaction homme-machine (IHM). Chacun de ces acteurs utilise des modèles qui lui sont propres. Les fonctionnalités sont généralement représentées par des diagrammes UML (UML, 2006) alors que les spécialistes de l'IHM se basent sur des modèles spécifiques comme les arbres de tâches (Paterno, 1997). Chaque spécialiste souhaite conserver ses propres modèles et les outils adaptés. Le but de nos recherches vise à faciliter la collaboration entre concepteurs de différents domaines tout en laissant à chacun ses propres pratiques d'ingénierie.

A l'intérieur de chaque point de vue (GL ou IHM), il est depuis longtemps reconnu qu'il est nécessaire de vérifier la cohérence inter-modèles, et de gérer le raffinement et la traçabilité des modèles. Par exemple, dans des méthodes de développement telles que Symphony développée par la société Umanis en collaboration avec l'équipe SIGMA du laboratoire LSR (Hassine et al., 2005), les diagrammes de séquence sont raffinés tout au long de la conception et doivent être cohérents avec le diagramme de classe.

Dans le cadre d'une collaboration entre des spécialistes de différents domaines, appelée conception collaborative ou co-conception (Juras, 2006), il est désormais nécessaire de gérer d'une part des modèles consensuels communs aux deux domaines (les scénarios décrits en langue naturelle sont par exemple utilisés aussi bien pour décrire les fonctionnalités attendues du système que pour évaluer son utilisabilité) et d'autre part, des modèles différents entre lesquels existent des cohérences sémantiques souvent fortes (entre arbres de tâches et diagrammes de séquence par exemple).

Une conception collaborative implique donc de prendre en compte des modèles qui ne sont pas nécessairement exprimés en UML et qui peuvent avoir des aspects similaires ou complémentaires.

Dans ce contexte, il nous parait intéressant d'étudier l'apport que pourrait avoir l'ingénierie dirigée par les modèles (IDM) (Favre, 2004) en particulier pour gérer les liens entre les modèles des différents domaines. L'IDM met à disposition des concepts, des langages et des outils pour créer et transformer des modèles en se basant sur leur méta-modèle. L'objet de notre étude est de savoir dans quelle mesure ces concepts et ces outils répondent à nos besoins en terme de conception collaborative.

Dans la suite de cet article, nous nous basons sur un exemple simple visant à gérer le lien entre un diagramme de cas d'utilisation utilisé pour représenter les

fonctionnalités attendues du système en génie logiciel et un arbre de tâches utilisé pour décrire les tâches en terme d'IHM. A partir de ces deux modèles ayant des méta-modèles différents, nous exprimons des règles de cohérence ou des transformations inter-modèles. Puis nous évaluons les outils disponibles en IDM pour répondre aux problèmes de conception collaborative.

Le plan de cet article est le suivant. La section 2 détaille le cas d'étude afin de montrer en quoi un diagramme de cas d'utilisation et un diagramme de tâches doivent être cohérents. La section 3 présente différents langages de vérification de cohérence et de transformation entre modèles introduits par l'IDM et illustre chacun de ces langages sur le cas d'étude. La section 4 évalue différents outils de l'IDM par rapport à notre besoin en terme de conception collaborative. Enfin la conclusion et quelques perspectives sont présentées en section 5.

2. Cas d'étude

Nous nous appuyons sur un exemple simple de gestion de conférences. Cet exemple permet de montrer comment le lien entre des modèles du GL (ici un diagramme de cas d'utilisation) et des modèles de l'IHM (ici un arbre de tâches) doit être géré. Il s'agit, dans notre exemple, de mettre en place des supports informatiques pour faciliter la gestion d'une conférence. Une personne, appelée Participant, peut soit assister aux présentations, c'est-à-dire rechercher une présentation qui l'intéresse et en obtenir des détails, soit vivre une conférence, à savoir, gérer ses centres d'intérêts et communiquer.

La modélisation de cet exemple ne sera pas la même en fonction des domaines. La figure 1 montre un extrait d'un diagramme de cas d'utilisation réalisé par un spécialiste du GL pour représenter les fonctionnalités attendues du système informatisé de gestion de conférences. D'autres modèles existent définissant les fonctionnalités d'un système pour lesquels la problématique de correspondance avec les arbres de tâches est la même. Dans un but de simplicité, nous nous limitons à la correspondance entre un diagramme de cas d'utilisation et un arbre de tâches. Ici, le participant veut assister à des présentations. Il est donc associé à un cas d'utilisation « Assister aux présentations ». Ce cas d'utilisation inclut deux autres cas d'utilisation (« Rechercher une présentation » et « Détailler une présentation »).

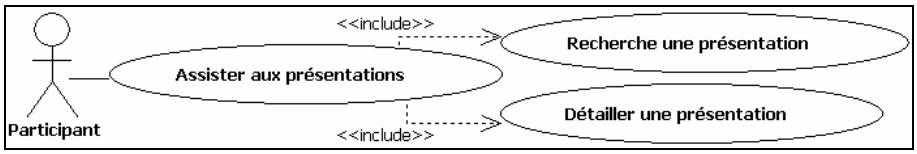

Figure 1. *Extrait d'un diagramme de cas d'utilisation.*

En IHM, les spécialistes modélisent les tâches du système et/ou de l'utilisateur dans un arbre de tâches dont un extrait est donné ci-dessous pour la gestion de conférences (Figure 2). Dans ce modèle, un participant peut assister autant de fois qu'il le souhaite à des présentations (tâche itérative notée par *). Pour ce faire, il doit, dans un premier temps, rechercher une présentation. Cette recherche se décompose en d'autres sous-tâches qui ne sont pas représentées ici. Après avoir trouvé une conférence qui lui semble intéressante, le participant peut, s'il le souhaite, obtenir des détails sur la présentation qu'il a choisie et ceci autant de fois qu'il le désire. Cette tâche se réalise en interaction entre l'utilisateur et le système (tâche « interaction »). Enfin, lorsque tout semble convenir au participant, ce dernier peut assister à la conférence choisie. Cette tâche est uniquement effectuée par l'utilisateur (tâche « utilisateur »), sans intervention du système informatique.

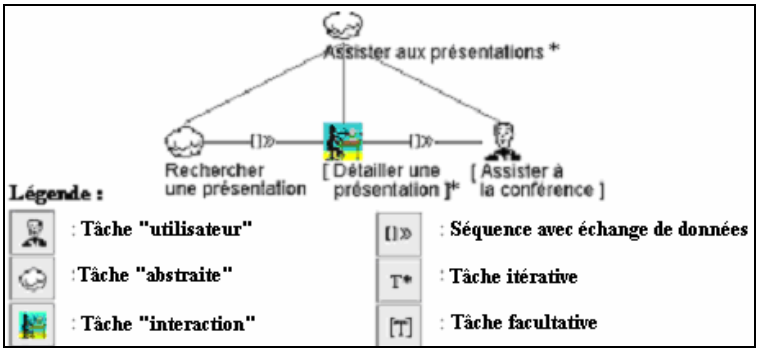

Figure 2. *Extrait d'un arbre de tâche.*

La description de ces deux modèles montre que certaines informations présentes dans l'arbre des tâches n'existent pas dans le diagramme de cas d'utilisation. Par exemple, la tâche « Assister à une conférence » qui est une tâche « utilisateur » ne figure pas dans le diagramme de cas d'utilisation qui ne représente que les aspects informatiques du système. Il n'y a donc pas d'équivalence sémantique complète entre une tâche et un cas d'utilisation, un cas d'utilisation ne correspondant pas à une tâche utilisateur.

Pour être sémantiquement correct, chacun de ces deux modèles doit être conforme à un méta-modèle. Chacun des modèles a plusieurs méta-modèles possibles. Nous avons choisi pour notre travail de nous baser sur des méta-modèles existants dans chacune des communautés : le méta-modèle proposé par l'OMG pour le diagramme de cas d'utilisation (UML, 2006) et celui proposé par (Sottet et al., 2005) pour l'arbre de tâches. La figure 3 présente un extrait de ces méta-modèles, décrits dans le standard de l'OMG pour la définition de méta-modèles, le MOF (MOF, 2005). Le but n'est pas d'utiliser toutes les subtilités de ces méta-modèles mais une version simple pour juger l'adéquation de l'IDM et des outils pour la gestion des modèles du GL et de l'IHM.

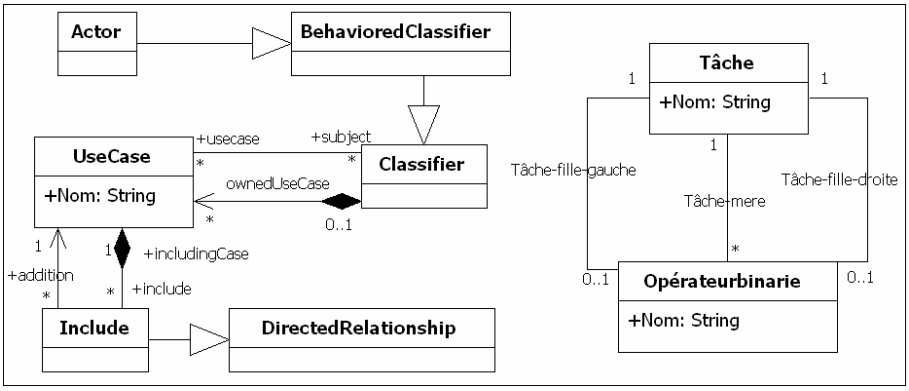

Figure 3. *Extrait des méta-modèles d'un diagramme de cas d'utilisation (à gauche) et de l'arbre de tâches (à droite).*

Ces méta-modèles nous permettent d'établir des liens entre les concepts des deux modèles pour la gestion de conférences (figures 1 et 2). Par exemple, un cas d'utilisation est équivalent à une tâche qui n'est pas une tâche utilisateur et une inclusion de ce cas d'utilisation correspond à une décomposition en sous tâches. Ce sont ces règles que nous allons exprimer par des transformations afin d'évaluer en quoi les concepts introduits par l'IDM répondent à nos besoins de conception collaborative. La première règle à définir spécifie le fait qu'à un cas d'utilisation correspond une tâche. Par exemple, le cas d'utilisation « Rechercher une présentation » dans le diagramme correspond à une tâche de même nom dans l'arbre. La seconde règle permet d'obtenir une sous tâche dans un arbre de tâches à partir d'un « include » d'un diagramme de cas d'utilisation. Ainsi le cas d'utilisation « Rechercher une présentation » inclus dans le cas d'utilisation « Assister aux présentations » doit être transformé en une sous tâche portant le même nom dont la tâche mère est la tâche « Assister aux présentations ».

Les règles données ci-dessus sont définies suivant des langages de l'IDM que nous décrivons dans la section suivante.

3. Evaluation des langages de l'IDM

L'IDM est un paradigme relativement récent selon lequel le code source n'est pas considéré comme l'élément central d'un logiciel, mais comme un élément dérivé de la fusion et du « tissage » d'éléments de modélisation. Un des aspects essentiels de l'IDM consiste à automatiser la transformation de modèles grâce à l'utilisation de langages de transformation.

3.1. Les transformations et le standard QVT

Les transformations de modèles se décomposent en deux catégories : la transformation d'un modèle vers du texte et la transformation d'un modèle en un autre modèle.

La transformation d'un modèle vers du texte est une opération couramment réalisée. Elle permet, une fois le modèle saisi, de l'utiliser directement après l'avoir transformé sous un format textuel. Plusieurs types de sortie textuelle sont possibles. A partir d'un modèle, il est possible de générer du code exécutable et/ou de la documentation sur ce modèle. Ce type de transformation ne correspond pas à nos besoins. En effet, nous souhaitons faire une transformation d'un modèle (un diagramme de cas d'utilisation) vers un autre modèle (un arbre de tâches).

Au niveau des transformations de modèle vers modèle, il existe deux types de transformations. La première est une transformation qui considère en entrée un modèle UML et qui fournit en sortie un autre modèle UML. La deuxième permet de faire des transformations à partir de et vers des modèles qui ne sont pas forcément exprimés en UML. Par rapport à notre exemple, le modèle d'entrée est un diagramme de cas d'utilisation, et le modèle de sortie est un arbre de tâches (modèle différent d'un modèle UML). C'est donc le deuxième type de transformation qui nous intéresse.

Une transformation de modèles nécessite un langage de transformation. Pour transformer un modèle en un autre, il existe plusieurs types de langages :

- les langages graphiques comme TrML (TrML, 2007),
- les langages basés sur XML XSLT (W3C, 2007),
- les langages basés sur le langage de programmation Java JMI (JMI, 2002),
- les langages ad-hoc comme MOLA (Kalnins et al., 2004), MTL (Vojtisek et al., 2004)
- les langages basés sur le standard QVT.

Notre choix pour un langage de transformation s'oriente vers un langage basé sur QVT. QVT (Query View Transformation) (QVT, 2005) est une spécification de l'OMG pour les langages de transformation et de manipulation de modèles. La partie Query permet de sélectionner des éléments sur un modèle grâce au langage OCL[1]. Elle peut être définie via un Query. Une vue est un modèle à part, avec éventuellement un méta-modèle restreint spécifique à cette vue. La dernière partie, la Transformation, permet de transformer un modèle en un autre. Les principes de QVT ont été implémentés dans de nombreux langages comme ATL (ATLAS Transformation Language) (Allilaire et al., 2004) qui est le plus courant et déjà utilisé dans plusieurs projets.

[1] OCL est un langage de contraintes basé sur la logique de premier ordre

3.2. Transformation en ATL

Les transformations en ATL sont caractérisées par :

- un environnement supposant que les méta-modèles soient représentés en MOF.
- des modèles sources et cibles conformes aux méta-modèles sources et cibles.
- des règles de transformation ATL construites à partir des éléments du méta-modèle.

La syntaxe d'ATL est textuelle. Elle est basée sur le langage OCL. Un programme ATL est formé d'une collection de règles et référence un méta-modèle source et un méta-modèle cible.

L'exemple de transformation d'un diagramme de cas d'utilisation en un arbre de tâches (cf. section 2) a été implémenté en ATL (Figure 5). La première règle spécifie qu'à un cas d'utilisation correspond une tâche. Dans la transformation ATL, nous pouvons voir que nous partons d'un cas d'utilisation (partie 1) pour obtenir une tâche (partie 2). Pour créer la tâche (partie 2), le nom du cas d'utilisation (u.nom) est récupéré et utilisé pour donner un nom (nom) à la tâche créée.

```
rule useCase2Tache{
from  u: MM1_UseCase!UseCase                                   } 1
to    t:MM1_Tache!Tache(
            nom <-u.nom,
            opBinaire <- 'a determiner',                       } 2
            tacheFilleGauche <- u.include,
            tacheFilleDroite <- u.include),

OpBinaire:MM1_Tache!Operateur_binaire (                        } 3
nomOp <- 'a determiner',
OPtacheMere <- u.includingCase,
OPtacheFilleGauche <- u.includingCase,
OPtacheFilleDroite <- u.includingCase)   }
```

Figure 5. *Exemple de règle de transformation en ATL.*

La deuxième règle exprime le fait qu'un « include » d'un diagramme de cas d'utilisation donne lieu à une sous-tâche dans un arbre de tâches. Ainsi un « include » entre deux cas d'utilisation donne lieu à une sous tâche (tâche fille de la figure 3) représentée soit par le lien « tacheFilleGauche », soit par le lien « tacheFilleDroite » associé à la création d'une tâche (partie 2). Cependant, l'ordre entre les sous tâches qui détermine les filles droite et gauche ne peut pas être mis en place car dans un diagramme de cas d'utilisation, les cas d'utilisation ne sont pas ordonnés chronologiquement. De plus, l'association entre une tâche et un opérateur binaire (opBinaire) reste à déterminer du fait qu'un diagramme d'utilisation ne comporte pas cette information. Pour être conforme au méta-modèle de la figure 3, nous sommes obligés de créer artificiellement un opérateur binaire et des filles droite et gauche dénuées de sens. En ce qui concerne l'opérateur binaire (partie 3), aucune information à son sujet ne peut être complétée du fait du manque d'information fourni par le diagramme de cas d'utilisation.

3.3. *Discussion*

Tout d'abord, il faut noter que comme pour toute transformation, c'est au concepteur de ces règles de s'assurer de la préservation de la sémantique du modèle traduit.

La règle de la figure 3 montre ensuite qu'ATL est un langage de transformation simple et facilement assimilable. De plus, elle montre qu'il n'est pas possible d'obtenir un arbre de tâches complet à partir d'un diagramme de cas d'utilisation du fait de l'absence de certaines informations dans le diagramme de cas d'utilisation par rapport à l'arbre de tâches. Cependant, une partie des problèmes est liée au méta-modèle. En terme d'IDM, la transformation du diagramme de cas d'utilisation produit un arbre de tâches dont une partie des informations ne sont pas pertinentes. Ainsi, une solution pour remédier à ce problème serait de redéfinir un autre méta-modèle pour l'arbre de tâches (Figure 6) qui répondrait à nos attentes.

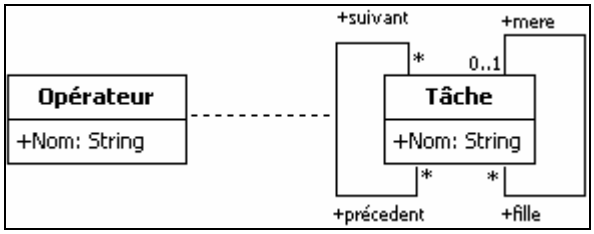

Figure 6. *Nouveau méta-modèle pour les arbres de tâches.*

Dans ce nouveau méta-modèle, deux associations sont représentées. La première, l'association mère/fille, permet d'exprimer la décomposition. Une tâche est reliée par le lien mère-fille (s'il existe) sans passer par l'intermédiaire d'un opérateur binaire. La deuxième, l'association suivant/précédent, est utilisée pour représenter l'ordre chronologique avec l'opérateur de séquencement adéquat. Ceci permet de palier le fait qu'il n'est pas possible de déterminer les filles gauche et droite d'une tâche à partir d'un diagramme de cas d'utilisation. Dans cette nouvelle règle de transformation (Figure 7), les sous-tâches seront juste classées en tant que tâches filles d'une tâche donnée. Ainsi, une inclusion de cas d'utilisation donne lieu à un lien mère/fille (partie 1). Par contre, rien n'est spécifié pour le séquencement (partie 2). Nous obtenons un méta-modèle qui permet d'avoir un modèle conforme à son méta-modèle. Ce modèle reste néanmoins incomplet. Le concepteur en IHM devra donc le compléter.

Par rapport à la remarque ci-dessus, une constatation peut être faite : en fonction des exemples étudiés, il sera nécessaire de redéfinir des méta-modèles afin d'obtenir le modèle de sortie le plus proche de celui souhaité. L'expression des règles de transformation nécessite donc de spécifier des méta-modèles adéquats, ce qui implique une « gestion » lourde des méta-modèles.

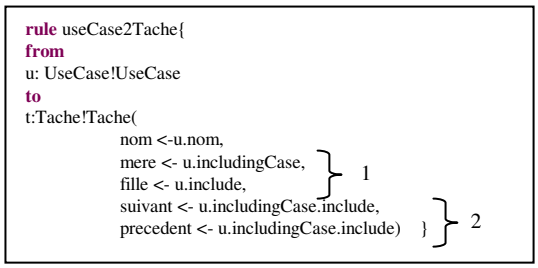

Figure 7. *Nouvelle règle de transformation en ATL.*

D'autre part, l'implémentation de la règle ATL (Figure 5) pour la transformation du diagramme de cas d'utilisation en arbre de tâches a posé quelques problèmes. Au premier abord, nous avions pu penser qu'il existait entre ces deux modèles une correspondance sémantique suffisante pour que les transformations soient intéressantes. Cependant, comme cela a été mentionné dans les commentaires des règles, le diagramme de cas d'utilisation manque d'informations par rapport à l'arbre de tâches.

Les transformations ne permettent donc d'obtenir que des modèles très partiels. L'obtention de modèles complets nécessiterait d'introduire des connaissances spécifiques au domaine (comme par exemple, le séquencement entre « Chercher » puis « Détailler une présentation »). Nous ne pourrions donc plus nous contenter de transformations au niveau des méta-modèles, il faudrait introduire des transformations sur les modèles. Par exemple, nous pourrions transformer directement le diagramme de cas d'utilisation en ajoutant les informations de séquencement afin d'obtenir sur l'exemple un arbre de tâches complet. Ceci a déjà été mis en place avec la notion de correspondance partielle définie dans le projet AMW (AMW, 2007). Ces travaux permettent à partir de deux modèles qui se complètent d'obtenir un troisième modèle, résultat du tissage des deux premiers. Ce projet peut être une solution à étudier. Dans notre cas, il est plus utile, de laisser chaque spécialiste (GL et IHM) faire ses modèles puis de vérifier leur cohérence. Cette vérification de cohérence peut être faite avec le langage OCL. La règle de cohérence en OCL suivante permet de dire qu'à un cas d'utilisation d'un diagramme de cas d'utilisation correspond une tâche d'un arbre de tâches :

Context Use Case Tache.allInstances -> exist(t, t.nom= self.nom)

4. Evaluation des outils pour l'IDM

Forts de ces constatations, nous présentons dans cette section les principaux outils utilisés par l'IDM, ensuite, nous présentons les deux outils que nous avons retenus pour expérimenter la transformation de notre cas d'étude.

4.1. Multitude d'outils existants

Au niveau commercial et au niveau de la recherche, de nombreux outils pour l'IDM sont disponibles ou en cours de développement. Ces outils sont conçus soit comme des frameworks (Amelunxen, 2006), soit comme des plugins (Allilaire et al., 2004). Plusieurs travaux de classification (Eclipse, 2007)(Planet, 2007) et de comparaison (García, 2004)(Tariq, 2005)(Meylan, 2006) d'outils ont été proposés. Cependant, aucune classification n'évalue les critères suivants que nous avons définis au regard de nos besoins en matière de conception collaborative, en particulier nos besoins en termes de modèles non standards :

- support à la méta-modélisation (méta-modèle et modèle, partie « Editeur »),
- support à la transformation de modèles, d'un espace de modélisation ou d'un niveau d'abstraction vers un autre (outil de transformation, partie « Type d'outil »),
- expression de modèles autres qu'UML (outil de transformation, partie « Type d'outil »),
- expression textuelle et graphique de modèles et méta-modèles (« Editeur »),
- expression de manière textuelle et graphique de règles de transformations (transformation, partie « Editeur »),
- support à la vérification de la cohérence inter-modèles (cohérence, « Editeur »),
- interopérabilité des différents éditeurs de modèles (partie « Repository »).

Notre étude (cf. figure 8) a été réalisée sur la documentation existante. La principale difficulté pour classer les outils conformément à nos besoins est le fait que nombre d'entre eux sont à un stade de développement intermédiaire. De plus, ces outils souffrent d'une documentation peu détaillée et généralement indiquent le support aux technologies par l'IDM mais ne détaillent pas l'aspect fonctionnel.

Le tableau de la figure 8 montre que les outils ADT, QVT Partners et SmartQVT correspondent à nos besoins. Parmi ceux-ci, nous avons choisi d'en expérimenter deux, ADT et SmartQVT sur la base de notre transformation des cas d'utilisation vers les arbres de tâches. Ces deux outils sont supposés offrir des fonctionnalités avancées pour gérer la transformation de modèle vers modèle. Cependant, notre intérêt est de savoir jusqu'à quel point la transformation est possible par rapport à notre exemple, et si l'outil répond à nos besoins de conception collaborative.

4.2. Outils testés

Dans l'expérimentation des outils présentés ci-dessous, nous avons saisi les extraits des méta-modèles du diagramme de cas d'utilisation et de l'arbre de tâches Les règles de transformation sont aussi saisies du fait que nous traitons de modèles particuliers (GL et IHM) pour lesquels aucune librairie prédéfinie ne correspond.

Outils	Editeur+(technologie utilisée) + Expression (Graphique(G) / Textuel(T))				Type d'outil + (format fichier généré)				Repository		
	Méta Modèle	Modèle	Transf.	Règles cohérence	Cohérence	Outil de Transf.	Validateur de modèle	Générateur de code	Méta Modèle / Modèle	Transf.	Règles cohérence
ACCELEO (GPL- Open source)	MOF, DSL, MDR (G, T)	Conforme à MOF, DSL, MDR (G, T)	QVT, JMI (T)	OCL	-	Modèle vers texte (JMI)	Usage de : OCL	Java, C++, VB.Net, Cobol, C, Fortran, Smalltalk, XML, Text	XMI	-	XMI
AndroMDA (Open source)	MOF (G, T)	Conforme à MOF, (G, T)	QVT, JMI (T)	OCL	-	Modèle vers texte (JMI)	Usage de : OCL, JUnit	Axis, Java, Spring, NET, Struts, Hibernate, Text	XMI	-	XMI
ADT - ATL (Open source)	MOF, MDR, KM3 (G, T)	Conforme à MOF, DSL, KM3 (G, T)	QVT, ATL (T)	ATL / OCL	ATL / OCL	Modèle vers modèle (ATL)	Usage de : OCL, JUnit	XMI, Texte	XMI	Texte / ATL	XMI
AToM3 (Open source)	Multi-formalisme à travers d'une notation graphique			-	-	Modèle vers texte (python)	-	Python	Grammaires de graphique sous Python		
ModFact (GPL- Open source)	MOF (G)	Conforme à MOF (G)	QVT (T)	-	-		Usage de : OCL	XMI, Texte	XMI	XMI	XMI
Merlin (Open source)	MOF 1.2 (G, T)	Conforme à MOF (G, T)	QVT (T)	OCL	-	Modèle vers modèle (QVT) et Modèle vers texte (JMI)	Usage de : OCL	Java, XMI, Texte	XMI	Texte / QVT	XMI
MDA Workbench	MOF (G, T)	Conforme à MOF, (G, T)	-	OCL	-	Modèle vers texte (JMI)	Usage de : OCL	Java, Texte	XMI	XMI	XMI
MOFLON (GPL- Open source)	MOF (G, T)	Conforme à MOF, (G, T)	JMI (G)	OCL	-	Modèle vers texte (JMI)	-	Java	XMI	-	XMI
OptimalJ	CWM, MOF (G)	Conforme à MOF, CWM (G)	-	OCL	-	Modèle vers texte (JMI)	Usage de : OCL	J2EE, EJB, text	XMI	XMI	XMI
QVT Partners	MOF (G, T)	Conforme à MOF, (G, T)	QVT (T)	OCL	-	Modèle vers modèle (QVT)	Usage de : OCL	XMI, Texte	XMI	Texte / QVT	XMI
SmartQVT (Open source)	MOF (G, T)	Conforme à MOF, (G, T)	QVT (T)	OCL	OCL	Modèle vers modèle (QVT)	Usage de : OCL	XMI, Texte	XMI	Texte / QVT	XMI
UMLX	MOF (G, T)	Conforme à MOF, (G, T)	XSLT (T)	OCL	-	Modèle vers modèle (XSLT)	Usage de : OCL	-	XSLT	XSLT	XSLT

Figure 8. *Synthèse d'outils IDM pour la gestion des modèles GL et IHM.*

4.2.1. ADT (ATL Development Tools)

ADT est un outil de transformation de modèles lié au langage de transformation ATL. Il est intégré sous forme d'un plugin pour l'environnement de développement Eclipse (Eclipse, 2007). Pour notre exemple nous avons utilisé la version 20060630. La figure montre le principe de fonctionnement de ADT.

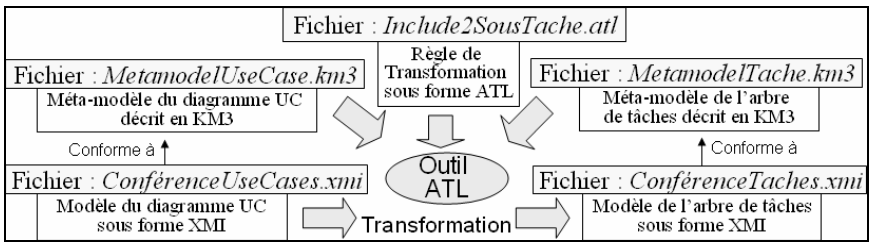

Figure 9. *Le principe de transformation modèle vers modèle avec ADT.*

Le premier pas à faire pour effectuer une transformation avec ATL consiste à écrire les méta-modèles sous forme de fichiers KM3. ADT met à disposition le langage KM3 (Jouault et al, 2006), une forme textuelle et simplifiée d'EMOF[2] qui permet de décrire des méta-modèles et des modèles. La figure 10 montre à gauche le contenu du fichier correspondant au diagramme de cas d'utilisation dans le format KM3 et à droite celui correspondant à l'arbre de tâches.

```
package MetamodelUseCase {
class UseCase {
   attribute nom: String;
   reference include [*]: Include;  }
class Include extends UseCase {
   reference includingCase :UseCase;  }
}
package PrimitiveTypes {datatype String;  }
```

```
package MetamodelTache {
class Tache { attribute nom: String;
   reference mere [0-1]: Tache oppositeOf fille;
   reference fille [0-*]: Tache oppositeOf mere;
   reference suivant [0-*]: Tache oppositeOf precedent;
   reference precedent [0-*]: Tache oppositeOf suivant;
   reference op_binaire [0-1]:Operateur;  }
class Operateur{attribute nomOp: String; }
}
package PrimitiveTypes { datatype String; }
```

Figure 10. *Méta-modèles sous forme KM3.*

Un modèle de transformation ATL se base sur des définitions de méta-modèles au format XMI. Par conséquent, ADT inclut un utilitaire qui permet de générer à partir des méta-modèles sous la forme KM3, leur équivalent au format XMI ECORE (Budinsky et al., 2003), format correspondant dans l'environnement Eclipse au modèle du standard MOF. L'étape suivante consiste à définir le modèle de transformation. Dans ADT, ce processus est défini à partir des méta-modèles source et cible exprimés en MOF. Les deux fichiers doivent donc être disponibles au moment de la transformation. Nous prenons comme référence la règle de transformation présentée dans la section 3.2 pour créer notre modèle de transformation. Une fois le modèle de transformation conçu, il faut générer l'instance du modèle source dans le format XMI. Nous avons créé une instance du modèle correspondant au diagramme de cas d'utilisation de la figure 1. L'exécution de la transformation génère une instance du méta-modèle cible sous forme XMI (cf. figure 11). Le modèle cible obtenu (l'arbre de tâches) pourra alors être manipulé par n'importe quel outil supportant XMI.

[2] EMOF (Essential Meta-Object Facilities) est une simplification de MOF 2.0 (MOF, 2005)

```
<xmi:XMI xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI" xmlns="MetamodelTache">
    <Tache nom="Assister aux presentations" fille="/1 /2"/>
    <Tache nom="Détailler une présentation" mere="/0" suivant="/2 /1" precedent="/1 /2"/>
    <Tache nom="Rechercher une présentation" mere="/0" suivant="/1 /2" precedent="/1 /2"/>
</xmi:XMI>
```

Figure 11. *Instance du méta-modèle cible sous forme XMI.*

Suite à la mise en place de notre exemple avec ADT, nous pouvons dire que cet outil est facile d'utilisation et rapidement utilisable. Il suffit de saisir les modèles, les méta-modèles, la règle de transformation et l'instance du modèle sans se préoccuper de lier ces éléments entre eux. De plus, ADT répond quasiment à l'ensemble de nos besoins, puisqu'il gère aussi la cohérence entre modèles. Il lui manque néanmoins une partie graphique qui permettrait de saisir et manipuler les modèles plus facilement.

4.3.2. SmartQVT

Le deuxième outil qui correspond a priori le mieux à nos besoins est SmartQVT (SmartQVT, 2007). Cet outil est une implémentation du langage QVT qui permet de faire des transformations de modèles. Il est intégré sous forme d'un plugin pour l'environnement Eclipse. La figure 12 montre le principe de fonctionnement de SmartQVT. Dans cet environnement, une transformation d'un modèle vers un autre modèle s'articule autour de trois éléments : des méta-modèles, des plugins et une règle de transformation.

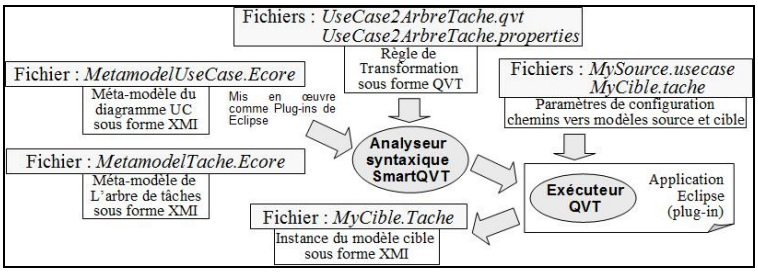

Figure 12. *Le principe de transformation modèle vers modèle avec SmartQVT.*

Dans un premier temps, il faut décrire la structure des deux méta-modèles (source et cible) sous le format ECORE. La deuxième étape consiste à créer, à partir de deux fichiers précédents, des plugins correspondant à notre exemple. Ces plugins contiennent chacun les méta-modèles convertis sous le format EMOF (simplification de la spécification MOF), un éditeur, un parser et un compilateur pour chacun des méta-modèles. Une fois ce plugin généré, il faut le placer dans notre environnement de travail pour qu'il soit utilisé lors de l'exécution de la règle de transformation. Ensuite, la règle de transformation est écrite en QVT (Figure 13). L'étape suivante

consiste à exécuter avec l'analyseur syntaxique les plugins et la règle. Un autre plugin est crée. Ce dernier plugin est alors exécuté avec l'instance du méta-modèle source (MySource.usecase) que nous aurons préalablement créée. Nous obtenons alors une instance du méta-modèle cible (l'arbre de tâches MyCible.Tache) sous format XMI.

```
transformation UseCase2ArbreTache(in UseCaseModel:USECASE,out
ArbreTacheModel:ARBRETACHE);
main() { UseCaseModel.objects()[UseCase] ->map usecase_to_arbretache(); }
mapping UseCase::usecase_to_arbretache () : Tache {
  nom := self.nom;
  mere := self.useCaseSource;
  fille := self.useCaseFils;
  suivant := self.useCaseSource.useCaseFils;
  precedent := self.useCaseSource.useCaseFils; }
```

Figure 13. *Règle de transformation en QVT.*

En conclusion sur cet outil, nous pouvons dire que nos principaux besoins de transformations de modèles (transformations entre modèles autres que UML) sont satisfaits avec SmartQVT. Cependant, à cause du manque de documentation claire, cet outil est difficile à utiliser et à manipuler. De plus, tout comme ADT, la partie graphique qui permettrait saisir les modèles, les méta-modèles et les règles de transformation plus facilement, n'est pas présente.

5. Conclusion et perspectives

Nous avons présenté, à travers cet article, comment les outils développés dans le cadre de l'IDM peuvent apporter des solutions partielles à la conception collaborative. Dans cette perspective, nous avons utilisé l'approche IDM pour créer des transformations entre un modèle de GL, le diagramme de cas d'utilisation, et un modèle de l'IHM, l'arbre de tâches. Cet exemple nous a montré que nous ne pouvions obtenir qu'un arbre de tâches partiel qui devrait ensuite être complété par un spécialiste. Dans notre optique de conception collaborative, ce genre de transformations ne semble donc applicable que dans des cas très restreints. Dans la plupart des cas, une vérification de la cohérence entre les modèles est préférable. Nos perspectives en termes de langages consistent à approfondir la gestion de cohérence avec OCL. Il sera nécessaire de définir les règles de cohérence ou de transformations entre modèles au sein de chaque phase, tout au long du cycle de développement.

Les transformations et la gestion de la cohérence font toutes deux partie de l'approche IDM et certains des outils proposés dans ce cadre apportent un support pour les deux. L'enjeu à venir va être de proposer des outils plus complets de conception collaborative. D'une part, ils devront prendre en compte les étapes de la

conception et ne proposer que les règles correspondantes afin de faciliter le travail des concepteurs. D'autre part, ils devront permettre de manipuler directement des modèles graphiques et pas simplement leur représentation XML. Les outils devront pouvoir être manipulés par tout utilisateur. Il est donc nécessaire de rendre intuitive et synthétique la manipulation des modèles grâce à leur gestion graphique. L'une de nos perspectives à long terme consiste donc à développer un nouvel outil qui s'appuierait sur un outil IDM existant afin de l'étendre pour permettre la gestion de modèles graphiques, mais aussi pour prendre en compte les étapes de la démarche de développement pour les Systèmes Mixtes que nous proposons (Juras, 2006).

Remerciements

Les auteurs remercient l'INPG, la fédération IMAG, la Fondation « Gran Mariscal de Ayacucho, l'Université UCLA-Venezuela » pour leur soutien financier.

6. Bibliographie

Allilaire, F., Idrissi T., *« ADT: Eclipse development tools for ATL »*, In: Proceedings of the Second European Workshop on Model Driven Architecture (MDA) with an emphasis on Methodologies and Transformations (EWMDA-2). Computing Laboratory, University of Kent, Canterbury, UK. England 2004, p. 171-178.

Amelunxen C., Königs A., Rötschke T., Schürr A., *« MOFLON : A Standard-Compliant Metamodeling Framework with Graph Transformations »*, in : Rensink A., Warmer J., (eds), Model Driven Architecture – Foundations and Applications : Second European Conference, Heidelberg: Springer Verlag, 2006; Lecture Notes in Computer Science (LNCS), Vol. 4066, Springer Verlag 2006, p. 361-375.

AMW ATLAS Model Weaver site officiel du projet http://www.eclipse.org/gmt/amw consultation avril 2007.

Budinsky F., Steinberg D., Merks E., Ellersick R., Grose T., *« Eclipse Modeling Framework »*, Ed. Addison Wesley, Août 2003. 720 pages.

Eclipse Modeling Project, site officiel dédié aux projets de modélisation du framework Eclipse. http://www.eclipse.org/modeling/ consultation février 2007.

Favre J.-M., *« Towards a basic theory to model-driven engineering »*, 3rd Workshop in Software Model Engineering, WiSME 2004, joint event with UML2004, Lisboa, Portugal, October 2004. p. 9-17.

García J., Rodríguez J., Menárguez M., Ortín M., Sánchez J., *« Un estudio comparativo de dos herramientas MDA : OptimalJ y ArcStyler »*, Taller de Desarrollo de Software Dirigido por Modelos, DSDM'04 en JISBD'2004, November 2004. p. 88-98.

Hassine I., Jausseran E., Rieu D., Front A., « Objets métier et composants techniques dans la méthode Symphony », Revue ISI-RSTI série ISI, numéro spécial "Méthodes avancées de développement des systèmes d'information", volume 10 - n°6/2005. p. 31-57.

JMI, Java Metadata Interface (JMI), Specifications Java Community Process. Version 1.0. Juin 2002.

Jouault F., Bézivin J., *« KM3 : a DSL for Metamodel Specification »*, In: Proceedings of 8th IFIP International Conference on Formal Methods for Open Object-Based Distributed Systems, Bologna, Italy, June 2006. p. 171-185.

Juras D., Dupuy-Chessa S., Rieu D., *« Vers une méthode de développement pour les Systèmes Mixtes », Revue Génie Logiciel*, n° 77, GL-IS. Juin 2006. p. 31-36.

Kalnins A., Barzdins J., Celms E.. *«Model Transformation Language MOLA»*. Proceedings of MDAFA 2004 (Model-Driven Architecture: Foundations and Applications 2004), Linkoeping, Sweden, June 10-11, 2004. p.14-28.

King's College London *« An Evaluation of Compuware OptimalJ Professional Edition as an MDA Tool »*, University of York, 2003. 27 pages.

Meylan S. 06, Rapport de projet Master 2 Informatique *« Etude et comparaison d'outils de transformation de modèles »*, Université de Franche-Comte, Janvier 2006. 53 pages.

Meta-Object Facility (MOF) Core Specification, OMG Available Specification, Version 2.0, January 2006.

MOF 2.0 Query/View/Transformation, OMG Final Adopted Specification, Version 2.0, November 2005.

Object Constraint Language, OMG Available Specification, Version 2.0, May 2006.

Paterno, F., Mancini, M., *« ConcurTask Tree : a diagrammatic notation for specifying task models »* Proc of Interact'97. (1997) p. 362-369.

Planet MDE, site officiel dédié aux projets MDE (Model Driven Engineering). www.planetmde.org consultation février 2007.

SmartQVT - A QVT implementation, site officiel dédié au projet SmartQVT. http://smartqvt.elibel.tm.fr/ consultation février 2007.

Sottet JS. , Calvary G., Favre JM., Ingénierie de l'Interaction Homme-Machine Dirigée par les Modèles. Premières Journées sur l'Ingénierie Dirigée par les Modèles., Meilleur papier.30 Juin - 1 Juillet 2005. p. 67-82

Tariq N., Akhter N., *« Comparison of Model Driven Architecture (MDA) based tools »* Karolinska University Hospital, A Thesis Document, Sockholm, Sweden. June 2005. 74 pages.

TrML Transformation modeling language site officiel du projet http://www2.lifl.fr/west/trml/ consultation février 2007.

Vojtisek D., Jzquel. JM, « *MTL and Umlaut NG - Engine and framework for model transformation* ». *ERCIM News 58*, 2004.

Unified Modelling Language: Superstructure, (OMG) Version 2.1, April 2006.

W3C World Wide Web Consortium, site official http://www.w3.org/TR/2007/REC-xslt20-20070123/. Recommendation 23 January 2007 consultation février 2007.