Génération de transformation de modèles par application de l'ARC sur des exemples

Xavier Dolques*, Marianne Huchard*, Clémentine Nebut*

*LIRMM, Univ Montpellier 2 et CNRS
161 rue Ada 34392 MONTPELLIER CEDEX 5
{dolques, huchard, nebut}@lirmm.fr,

Résumé. Les transformations de modèles sont au cœur de l'ingénierie dirigée par les modèles. Elles sont habituellement développées par des programmeurs spécialisés et leur code doit être remis à jour lors de toute variation des besoins ou des métamodèles impliqués. Pour faciliter le développement de ces transformations, une approche possible consiste à générer des règles de transformation à partir d'exemples de transformation. L'avantage des exemples est qu'ils peuvent être d'une part écrits dans une syntaxe concrète, plus accessible pour l'utilisateur qu'un langage de transformation et d'autre part plus faciles à définir que les règles elles-mêmes. Dans cet article, nous spécifions une méthode qui utilise l'analyse relationnelle de concepts pour capturer et organiser par spécialisation les schémas récurrents qui apparaissent dans les exemples de transformation. Ces schémas récurrents peuvent ensuite être exploités pour la production de règles de transformation.

1 Introduction

L'ingénierie dirigée par les modèles place le modèle au centre du processus de développement, ainsi les différentes étapes du développement se traduisent par une succession de transformations de modèles. Ces transformations sont automatisées autant que possible par le biais de programmes de transformation, qui sont dans la plupart des cas développés dans des langages généralistes (par exemple Java) ou dédiés (par exemple ATL (Jouault et Kurtev, 2005), Kermeta, QVT (OMG, 2008)).

Beaucoup de ces transformations sont assez simples à appréhender, puisqu'elles consistent à associer un motif du modèle source à un motif du modèle cible, notamment dans le cas d'un changement de méta-modèle pour la représentation d'un même domaine (par exemple une transformation d'UML vers un modèle Entité-Relation, ou d'UML vers Java).

Mais la programmation d'une transformation de modèles nécessite la maîtrise d'un langage de transformation ainsi qu'une connaissance suffisante des méta-modèles des modèles sur lesquels s'applique la transformation. Une approche par l'exemple permet de s'affranchir d'une partie de ces connaissances, en ne manipulant que des modèles dans leur syntaxe concrète. Nous proposons dans ce papier la spécification d'une méthode de génération de transformations à partir d'exemples. Cette méthode s'appuie sur une technique de découverte d'abstractions à base de treillis.

Tout d'abord nous présentons notre problème et introduisons l'exemple sur lequel nous nous appuierons par la suite. Nous présentons ensuite la méthode de découverte d'abstractions : l'Analyse Relationnelle de Concepts, que nous appliquons sur une partie de notre exemple. Dans la partie suivante nous donnons en détail la modélisation à adopter pour appliquer l'Analyse Relationnelle de Concepts en vue de générer des règles de transformation. Nous présentons alors l'interprétation des résultats obtenus sous forme de treillis. Nous nous positionnerons enfin par rapport aux travaux connexes avant de conclure et de présenter nos perspectives de recherche.

2 Présentation du problème sur un exemple

La génération de transformations de modèles par l'exemple permet la création d'une transformation de modèles en manipulant essentiellement des éléments du modèle dans leur syntaxe concrète (niveau M1 de méta-modélisation). La connaissance en détail du méta-modèle n'est alors plus indispensable. La transformation s'effectue en créant des exemples de modèles source et cible et en faisant correspondre un élément du modèle source à un élément du modèle cible. Il n'est pas forcément nécessaire que les exemples soient nombreux, il suffit qu'ils couvrent toutes les particularités de la transformation. Ainsi dans la plupart des cas, les modèles à traiter sont de taille raisonnable.

Pour présenter au lecteur la technique que nous proposons, nous utilisons un exemple simple de transformation. L'exemple concerne la transformation de modèles UML vers des modèles de type Entité-Relation. Cette transformation est en général présentée dans la littérature comme une transformation classique, elle a notamment été utilisée lors du workshop MTIP2005 (MTIP, 2005). Les modèles de notre exemple sont élaborés à partir de méta-modèles simplifiés des langages étudiés. Ces méta-modèles sont représentés par la figure 1.

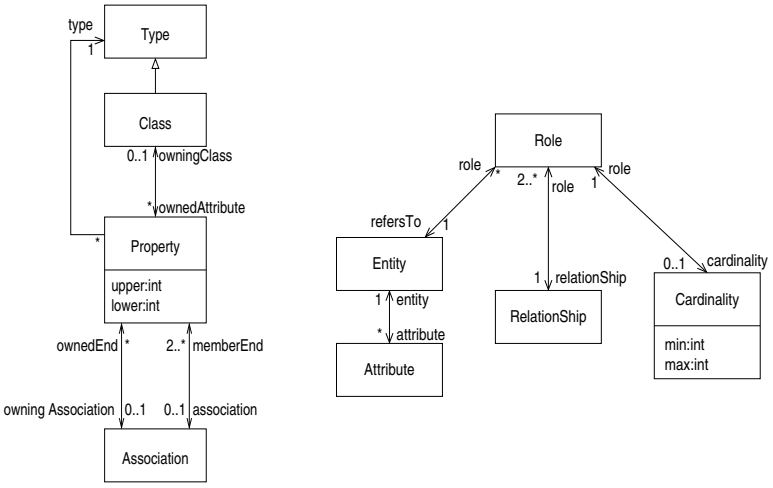

FIG. 1 – *Méta-modèles simplifiés de UML (à gauche) et Entité-Relation (à droite) utilisés dans Wimmer et al. (2007) .*

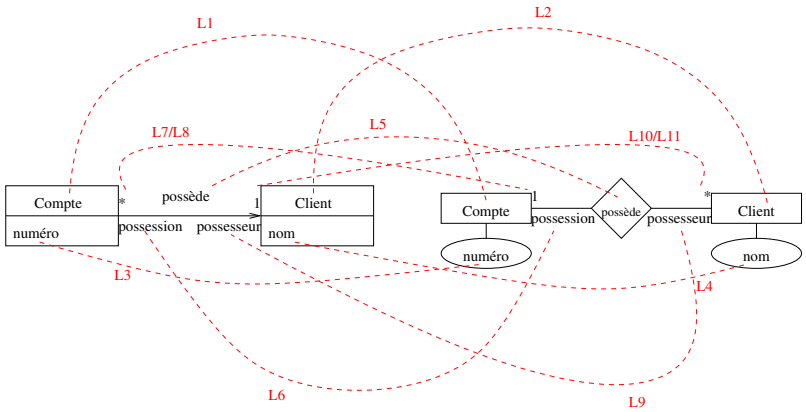

FIG. 2 – *Exemples de modèles dans leur syntaxe concrète, UML à gauche et Entité-Relation à droite.*

Notre but est de produire des règles de transformation d'un méta-modèle à l'autre à partir de modèles conformes à ces méta-modèles. Pour cela il faut que le développeur crée un ensemble de modèles sources, les modèles résultant de leur transformation et des liens de traçabilité entre éléments des modèles source et cible. Nous nous basons sur l'environnement d'un élément, avec une approche similaire au *context analysis* introduit par Varró (2006), pour générer des règles de transformation. Nous proposons à la figure 2 un exemple de modèles dans leur syntaxe concrète avec les liens de transformation indiqués par un utilisateur pour les méta-modèles présentés précédemment et la figure 3 représente les mêmes modèles dans leur syntaxe abstraite avec une notation UML.

Dans la figure 3, la partie gauche de la figure représente un modèle UML tandis que la partie droite représente un modèle Entité-Relation (ER). Les courbes en pointillés, issues des liens de transformations indiqués par l'utilisateur sur la syntaxe concrète, représentent la trace de la transformation d'un modèle à l'autre. Le passage de la syntaxe concrète à la syntaxe abstraite est un problème en soi, abordé notamment par Wimmer et al. (2007), mais qui ne rentre pas dans le cadre de cet article. Les noms attribués à ces courbes (L1, L2 ...) ne servent ici que d'identifiants. On voit sur l'exemple qu'il n'est pas possible de créer une règle de transformation en s'appuyant uniquement sur la méta-classe des éléments du modèle source, puisqu'une *Property* en UML peut être transformée en *Attribute* ou *Role* dans le modèle ER. Pour différencier ces 2 cas, nous étudions le voisinage des éléments. On voit que dans le cas d'une *property* transformée en *attribute* (par exemple la *property* numéro) la *property* est reliée uniquement à une *Class* par la relation *owningClass* ; tandis que dans le cas d'une *property* transformée en *role* (par exemple la *property* possession) la *property* est reliée à une *association* et à un *type*. D'autres caractéristiques pourraient aussi être prises en compte comme la présence ou non d'attributs ainsi que leur valeur, mais pour le moment nous nous concentrerons essentiellement sur les relations environnantes des éléments et leur type.

Il est donc nécessaire de pouvoir organiser les éléments en fonction de caractéristiques incluant des liens entre ces éléments. C'est pour cette raison que nous nous sommes tournés vers l'Analyse Relationnelle de Concept (dont les principes sont rappelés dans la section 3)

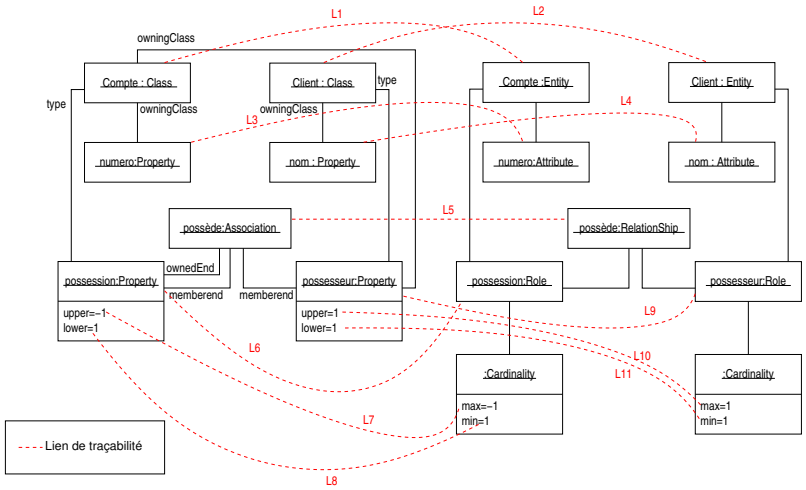

FIG. 3 – *Exemple de modèles dans leur syntaxe abstraite, avec liens de transformation, utilisés pour la génération de transformations. Inspiré par Wimmer et al. (2007). Les noms de rôles ne sont pas tous affichés pour garder la figure la plus claire possible.*

qui a pour but la création d'abstractions à partir d'éléments possédant des caractéristiques communes.

3 Analyse relationnelle de concepts

Nous avons choisi d'utiliser l'Analyse Relationnelle de Concepts (ARC), extension de l'Analyse Formelle de Concepts (AFC) qui appartient à une branche de la théorie des treillis. Leur particularité qui nous intéresse est la production de regroupements pertinents d'objets sur la base de caractéristiques communes. L'ARC propose en plus la réutilisation de regroupements déjà créés comme nouvelles caractéristiques pour la création de nouveaux regroupements.

3.1 Analyse Formelle de Concepts

L'AFC (Birkhoff (1940) ; Barbut et Monjardet (1970) ; Ganter et Wille (1999)) est une technique qui, à partir de données décrites sous forme d'un contexte montrant leurs caractéristiques, génère un treillis. Celui-ci décrit les regroupements d'objets sous forme de concepts.

Définition 1 (Contexte formel) *Un contexte $K = (O, A, I)$ est un triplet où O est un ensemble d'*objets*, A est un ensemble d'*attributs *et $I \subseteq O \times A$. $(o, a) \in I$ lorsque a est une caractéristique de o.*

À partir de l'exemple en UML de la figure 3, il est possible de créer un contexte formel dont les objets O seront des éléments du modèle et dont l'ensemble des attributs A sera partagé en deux sous-ensembles A_1 et A_2. Un couple $(o, a) \in (O, A_1)$ appartient à la relation d'incidence

si et seulement si a est la méta-classe de o. Un couple $(o, a) \in (O, A_2)$ appartient à la relation d'incidence si et seulement si a est la méta-classe d'un voisin de o. Ce contexte formel peut être présenté sous forme d'une table (Tab.1).

	type			type des voisins		
	Class	Property	Association	Class	Property	Association
Compte	X				X	
Client	X				X	
possesseur		X		X		X
possession		X		X		X
possède			X		X	
numéro		X		X		
nom		X		X		

TAB. 1 – *Exemple de contexte formel tiré du modèle de la figure 3.*

Définition 2 (Concept) *Un concept est une paire* (X, Y) *avec* $X \subseteq O$, $Y \subseteq A$ *et*
$X = \{o \in O | \forall y \in Y, (o, y) \in I\}$ *est l'extension (objets couverts),*
$Y = \{a \in A | \forall x \in X, (x, a) \in I\}$ *est l'intension (attributs partagés).*

Plus simplement un concept est le plus grand ensemble d'objets ayant en commun un même ensemble d'attributs. Cet ensemble d'attributs doit contenir tous les attributs communs à ces objets.

Le treillis de concepts généré à partir du contexte formel de notre exemple est représenté par la figure 4 sous une forme simplifiée précisée ci-après. Chaque boîte représente un concept, le compartiment du haut contient le nom du concept (généré automatiquement), celui du milieu son intension simplifiée et celui du bas contient son extension simplifiée.

Les concepts sont ordonnés par spécialisation selon l'ordre partiel suivant : c spécialise c' si l'extension de c est incluse dans celle de c' (et inversement, l'intension de c' est incluse dans celle de c). Cet ordre nous permet de représenter l'intension (resp. extension) de chaque concept de manière simplifiée : pour un concept c donné nous n'affichons que les éléments de l'intension (resp. extension) qui ne sont pas présents dans les concepts dont il est la spécialisation (resp. généralisation). Plus simplement, on peut faire un parallèle avec l'héritage des langages à objets où les propriétés d'une classe (son intension) sont héritées dans les classes qui la spécialisent, et une instance d'une classe appartient à l'extension des classes qui généralisent la classe.

L'exemple de la figure 4 nous montre une classification des éléments du modèle en fonction de leur type et de celui de leur voisin. On voit ainsi que les éléments de type *Property*, représentés par le concept $c2$, ont tous pour voisins au moins un élément de type *Class*. Mais une partie des éléments de ce concept, représentée par le concept $c4$, ont aussi pour voisin au moins une association. Nous verrons par la suite que cette classification des éléments permet l'élaboration de règles de transformation différentes selon l'appartenance d'un élément à un concept.

Comme on le voit dans l'exemple, il est possible, avec une seule relation d'incidence, de représenter plusieurs sortes de caractéristiques pour un ensemble d'objets. On peut par exemple mettre en attribut le type des voisins de l'objet ou même le type des voisins de l'objet en fonction d'un lien qui le relie à son voisin. Mais une des limitations de l'AFC est que les nouveaux concepts créés ne peuvent pas être pris en compte. C'est-à-dire qu'il n'est pas

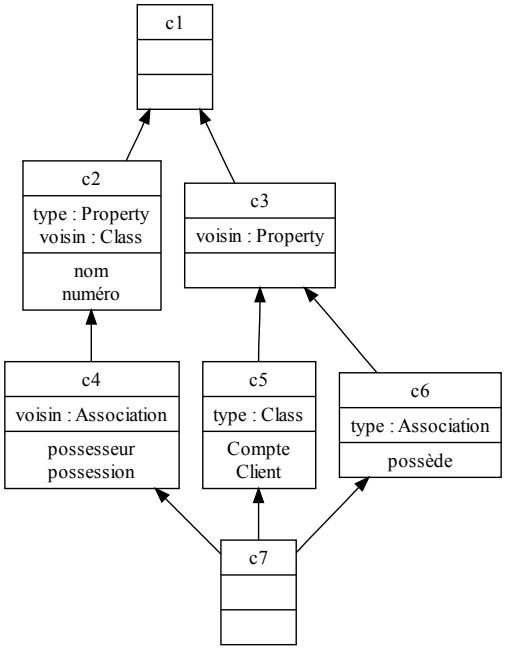

FIG. 4 – *Treillis généré par AFC à partir du contexte représenté Tab.1*

possible de créer un contexte qui associe par une relation d'incidence pour un objet donné le concept auquel appartient ses voisins. Un des intérêts de procéder ainsi est de caractériser un élément par ses voisins mais aussi par les voisins des voisins, etc. de manière à avoir une définition du concept la plus précise possible.

3.2 Analyse Relationnelle de Concepts

L'ARC (Huchard et al., 2007) est une extension de l'AFC qui permet de prendre en compte les concepts nouvellement créés pour obtenir de nouvelles abstractions.

Dans notre exemple le besoin s'en fait ressentir concernant la description du voisinage d'un élément. Avec l'AFC, en agrandissant notre contexte on pourrait décrire le voisinage des éléments d'un concept par le type des voisins et le type de lien qui les relie. Pour cela, il faudrait pour chaque type de lien possible créer autant d'attributs qu'il y a de méta-classes dans le méta-modèle. Malgré tout, on n'aurait une description du voisinage qu'à distance 1. Si à la place du type des voisins on pouvait avoir en attribut les concepts auxquels ces derniers appartiennent, on aurait non seulement une information sur le type des voisins, mais aussi des informations sur l'environnement des voisins. C'est ce que propose de faire l'ARC.

L'ARC nécessite un ou plusieurs contextes formels ainsi que des contextes relationnels qui forment une famille de contextes relationnels. Un contexte relationnel décrit une relation entre les objets de deux contextes formels (qui ne sont pas forcément différents).

Définition 3 (Famille de Contextes Relationnels (RCF)) *Une famille de contextes relationnels $\mathcal{R}$ est un couple (K, R). K est un ensemble de contextes formels $K_i = (O_i, A_i, I_i)$, R est un ensemble de contextes relationnels $R_j = (O_k, O_l, I_j)$ (O_k et O_l sont les ensembles d'objets des contextes K_k et K_l de K).*

L'ARC permet un découpage des données du problème en plusieurs ensembles d'objets. Dans notre exemple, nous allons considérer les éléments du modèle et les méta-classes du méta-modèle. Ces deux ensembles seront les objets de deux contextes formels, respectivement K_1 et K_2, représentés dans la figure 5. Nous cherchons à décrire les objets de K_1 par 2 sortes de caractéristiques : leur méta-classe et leurs voisins. La relation qui les relie à une méta-classe s'exprime par un contexte relationnel décrivant une relation de K_1 à K_2. La relation qui les relie à un voisin s'exprime par un contexte relationnel allant de K_1 à K_1. Pour différencier les différents types de liens (ou rôles en UML), nous faisons un contexte relationnel différent pour chaque type de lien (figure 7). L'ensemble des attributs de K_1 est vide. Les éléments de K_2 ne sont décrits que par eux-mêmes, chaque objet a pour attribut un identifiant unique, générant ainsi un treillis où chaque objet appartient à un concept différent des autres ou au supremum (figure 6). L'ensemble des méta-classes n'a pas été mis en attributs de K_1 pour obtenir un découpage plus clair et pour permettre de faire évoluer plus facilement la modélisation. Cela nous permet de rajouter facilement un contexte relationnel au contexte formel K_2, par exemple pour décrire la relation d'héritage entre méta-classes. Cette dernière n'apparaît pas ici pour des raisons de simplicité, et c'est d'ailleurs pour cela que la méta-classe *Type* n'apparaît pas dans K_2.

K_1	
Compte	
Client	
possesseur	
possession	
possède	
numéro	
nom	

K_2	idClass	idProperty	idAssociation
Class	X		
Property		X	
Association			X

FIG. 5 – *Contextes formels initiaux K_1 (en haut) et K_2 (en bas) pour le modèle UML.*

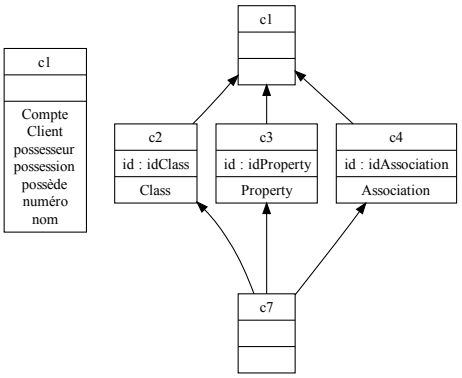

FIG. 6 – *Treillis, à l'étape d'initialisation, des contextes K_1 (à gauche) et K_2 (à droite).*

De nouvelles abstractions émergent par itération d'une étape de construction des treillis de concepts suivie d'une étape de concaténation des contextes formels avec les contextes relationnels correspondants enrichis par les concepts créés à l'étape précédente.

Etape d'initialisation. Les treillis sont construits à cette étape en utilisant l'AFC classique : pour chaque contexte formel K_i, un treillis $\mathcal{L}_i^0$ est créé. Les treillis de notre exemple sont

owningClass	Compte	Client
numéro	x	
nom		x
possesseur	x	

ownedAttribute	numéro	nom	possesseur
Compte	x		x
Client		x	

type	Compte	Client
possession	x	
possesseur		x

owningAssociation	possède
possession	x

memberEnd	possession	possesseur
possède	x	x

association	possède
possession	x
possesseur	x

ownedEnd	possession
possède	x

meta-classe	Class	Property	Association
Compte	X		
Client	X		
possesseur		X	
possession		X	
possède			X
numéro		X	
nom		X	

FIG. 7 – *Contextes relationnels. Ceux sur les deux lignes du haut expriment une relation de K_1 vers K_1. Celui du bas représente une relation de K_1 vers K_2.*

représentés par la figure 6. Nous pouvons voir que le treillis issu de K_1 ne comporte qu'un seul concept car les contextes relationnels n'ont pas encore été pris en compte. La classification en concepts apparaîtra dans les étapes suivantes.

Etape n+1. Pour chaque contexte relationnel $R_j = (O_k, O_l, I_j)$, un contexte enrichi $R_j^s = (O_k, A, I)$ est créé. A correspond aux concepts du treillis $\mathcal{L}_l^n$.

La relation d'incidence I contient l'élément (o, a) si $S(R(o), Extension(a))$ est $vrai$. La fonction S est un opérateur de *scaling*. L'opérateur de scaling que nous utiliserons dans le reste du papier est $S_\exists(R(o), Extension(a))$, qui est $vrai$ si et seulement si $\exists x \in R(o), x \in Extension(a)$. D'autres opérateurs peuvent être utilisés pour préciser la description des concepts, notamment $S_{\forall\exists}(R(o), Extension(a))$, qui est $vrai$ ssi $\forall x \in R(o), x \in Extension(a) \wedge \exists x \in R(o), x \in Extension(a)$, mais nous ne les aborderons pas ici pour des raisons de clarté et de manque de place.

À chaque étape on obtient ainsi de nouveaux treillis. Tant que les contextes enrichis diffèrent d'une étape à l'autre, c'est à dire tant que le processus fait apparaître de nouveaux concepts, nous devons passer à l'étape suivante. Lorsqu'aucun concept nouveau n'apparaît, un point fixe a été atteint et le processus s'arrête.

Le treillis issu du contexte K_1 est représenté par la figure 8. Le treillis final obtenu à partir du contexte K_2 reste le treillis de la figure 6 car il n'y a pas de contexte relationnel dont K_2 serait source.

4 Modélisation du problème sous forme de contextes

Notre modélisation du problème sous forme de contextes se divise en trois étapes :
- la classification des éléments des modèles acteurs de la transformation ;
- la classification des liens reliant les éléments des deux modèles entre eux ;
- la transformation des concepts obtenus en règles de transformation.

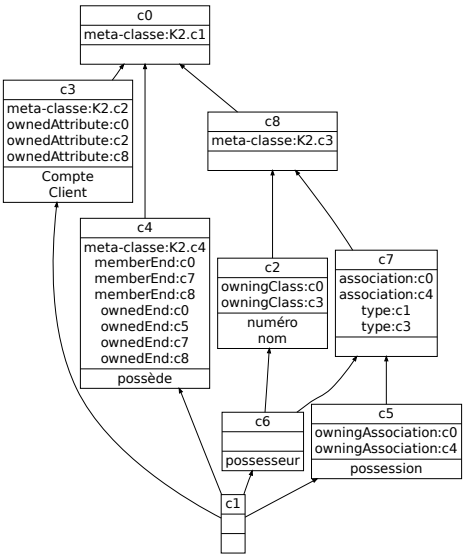

FIG. 8 – *Treillis final obtenu par ARC. Le treillis est issu du contexte formel* K_1.

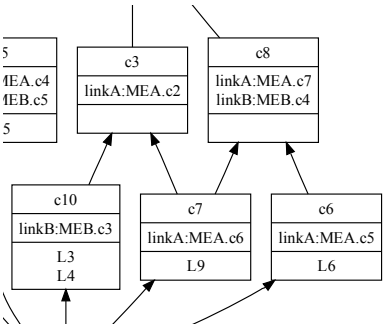

FIG. 9 – *Extrait du treillis MapLink final obtenu par ARC. MEA (resp. MEB) fait référence au treillis issu de ModelElementA (resp. ModelElementB).*

4.1 Classification des éléments d'un modèle

Les éléments d'un modèle (niveau M1) conforme à un méta-modèle appartiennent déjà à une classification explicite qui correspond aux classes du méta-modèle (niveau M2). Mais il arrive que des éléments instances d'une même méta-classe puissent avoir des significations différentes et ainsi ne pas être transformés de manière similaire. Pour différencier l'usage auquel est destiné un élément du modèle, on étudie les éléments qui l'entourent. Si on reprend notre exemple précédent une *Property* utilisée en tant que rôle sera reliée à une *association*, tandis qu'une *property* utilisée en tant qu'attribut aura une référence *owningClass* vers une classe et aucune référence vers une association. L'utilisation de l'ARC nous permet d'obtenir une classification des éléments d'un modèle selon leur environnement. Pour un modèle donné, la classification s'obtient en créant un contexte formel contenant tous les éléments du modèle nommé *ModelElement* (équivalent au modèle K_1 de la section 3). Ce contexte aura en *Objets* l'ensemble des éléments du modèle et l'ensemble vide en *Attributs*. C'est à partir de ce contexte que va être construit le treillis contenant la classification qui nous intéresse. On construit ensuite un contexte contenant les différents éléments du méta-modèle (niveau M2) nommé *MetaModelElement* (équivalent au modèle K_2 de la section précédente), ainsi qu'un contexte relationnel entre *ModelElement* et *MetaModelElement* ayant pour relation d'incidence le lien d'instanciation (équivalent au contexte *méta-classe* de la figure 7). Cela va conduire à une classification des éléments de *ModelElement* en fonction de leurs méta-classes. Pour chaque relation *R* du méta-modèle on va créer un contexte relationnel $(ModelElement \times ModelElement)$ ayant pour relation d'incidence l'existence de lien de type *R* (tableaux de la figure 7). Cela va conduire à une classification des éléments de *ModelElement* en fonction de leurs relations avec d'autres éléments.

Le résultat d'une telle modélisation est un treillis de concepts décrivant les différents éléments disponibles dans un modèle. L'utilisation de l'intension de ces concepts pour les décrire permettra la génération de règles une fois que les liens de transformations seront repérés. La figure 8 correspond au treillis de concepts décrivant les différents éléments de notre modèle UML. Par exemple le concept c_2 décrit les *Property* qui sont reliées par un lien de type *owningClass* à des éléments de type *Class*. On peut aller plus loin en disant que les éléments qui appartiennent à l'extension simplifiée de c_2 (soient *numéro* et *nom*) ne sont reliés à rien d'autre par une autre relation.

4.2 Classification des liens de transformation

Les liens de transformation donnés dans un exemple décrivent la correspondance entre plusieurs éléments de deux modèles. Les liens que nous étudions ici sont des liens reliant un élément à un autre ou un élément à plusieurs éléments. Pour modéliser notre problème nous utiliserons un contexte formel nommé *MapLinks* dont les *Objets* sont les liens et qui est vide d'*Attributs*. Dans l'exemple de la figure 3 les liens sont representés par des lignes en pointillés. Le contexte *MapLinks* est à l'origine du treillis qui nous permettra de générer des règles de transformation.

Pour les deux modèles de la transformation, que nous nommerons par la suite *A* et *B*, nous allons créer des contextes relationnels $LinkA \subseteq MapLinks \times ModelElementA$ et $LinkB \subseteq MapLinks \times ModelElementB$. Un élément de *MapLinks* est lié à un concept d'un *ModelElement* si un élément de ce concept est une extrémité du lien qui nous intéresse. Le treillis[1] nous permet alors de classer les liens en fonction de leurs extrémités.

4.3 Interprétation des concepts pour générer des règles

Un lien de transformation se caractérise par le concept auquel appartient son extrémité dans le modèle *A* et le concept auquel appartient son extrémité dans le modèle *B*. Ainsi les concepts obtenus dans le treillis *MapLinks* regroupent les liens qui ont des caractéristiques communes à leurs deux extrémités. Nous proposons d'extraire des règles de transformation à partir de ces caractéristiques. Nous considérons une règle comme une fonction prenant en paramètre un élément d'un modèle conforme au méta-modèle de *A*. Si cet élément possède les propriétés requises alors il est transformé en éléments d'un modèle conforme au méta-modèle de *B*.

Pour une règle issue d'un concept c du treillis *MapLink*, on obtient les propriétés requises en analysant le concept *ModelElementA.c'*, qui est le concept le plus spécialisé de *ModelElementA* appartenant à l'intension de c. On peut voir 3 types de propriétés :

- **Les caractéristiques obligatoires**, décrites par l'intension de *ModelElementA.c'*. Ce sont des caractéristiques qui sont communes à tous les liens exemples de c.
- **Les caractéristiques autorisées**, décrites par l'intension de concepts spécialisant *ModelElementA.c'*, à condition que ce concept ait dans son extension l'extrémité d'un lien exemple de c. Par exemple, pour le concept c_8 de *MapLink* regroupant les liens L_6 et L_9 (figure 3), *ModelElementA.c'* correspond à $K_1.c_7$: *possession* (extrémité de L_6) appar-

[1] L'ensemble des treillis concernant l'exemple de ce papier peut être obtenu à l'adresse suivante : http://www.lirmm.fr/~dolques/publications/data/lmo09

tient à $K_1.c_5$ et possesseur (extrémité de L_9) à $K_1.c_6$. Les caractéristiques de ces deux concepts sont donc autorisées.

– **Les caractéristiques interdites**, décrites par l'intension de concepts ne contenant pas dans leur extension l'extrémité d'un lien exemple de c. Ces caractéristiques sont particulièrement importantes si elles sont trouvées dans des concepts spécialisant *ModelElementA.c'*. Dans notre exemple, si on considère le concept dans *MapLink* regroupant les liens L_3 et L_4, *ModelElementA.c'* correspond à $K_1.c_2$. Il ne faut pas que la règle générée accepte les éléments de $K_1.c_6$ puisqu'aucun élément de son extension n'est extrémité de L_3 et L_4.

Les règles sont de la forme :

```
règle c10
pour toute Property p
où p est liée à une Classe par OwningClass
   et p n'est pas reliée à une Association par association
créer Attribut a
```

5 Travaux connexes

L'intérêt pour la génération automatique ou semi-automatique de transformations de modèles en tant que telle s'est développé relativement récemment, quoiqu'elle se rapproche d'autres domaines connexes, tels que l'appariement de schémas (Rahm et Bernstein, 2001; Shvaiko et Euzenat, 2005) ou la programmation par l'exemple ou par démonstration (Cypher et al., 1993; Lieberman, 2000), dont elle exploite certaines idées et certains résultats.

Une première catégorie d'approches se base sur l'alignement des métamodèles et la génération des transformations à partir des appariements ainsi constitués. Dans Roser et Bauer (2006), l'alignement des deux méta-modèles passe par leur projection sur une ontologie ; un raisonnement basé sur l'ontologie permet de construire une transformation en QVT relationnel. Plusieurs approches se basent sur le *Similarity Flooding* (Melnik et al., 2002), une méthode de propagation de similarités dans un graphe étiqueté pour déterminer un appariement. Dans Lopes et al. (2006,?), les éléments structurels étudiés sont les classes, les attributs, les associations, les types de données et les énumérations. Des études de cas sont menées sur des métamodèles Java et UML partiels. Une application aux plate-formes de services web est citée dans Lopes et al. (2005). Dans Fabro (2007); Fabro et Valduriez (2007), une architecture pour automatiser la génération de transformations de modèles est proposée, incluant un méta-modèle de tissage qui décrit les sortes de liens générés par l'outil d'appariement et des patrons de transformation. La référence Fabro (2007) présente plusieurs études de cas détaillées. Falleri et al. (2008) étudie plus précisément le comportement du *Similarity Flooding* pour plusieurs sortes d'encodage des méta-modèles à apparier. L'approche est testée sur plusieurs méta-modèles structurels connus complets.

Une deuxième catégorie de solutions explore l'utilisation d'exemples de transformation selon plusieurs méthodes. L'approche de Kessentini et al. (2008) se base sur un algorithme d'optimisation combinatoire (Particle swarm optimization) pour générer la transformation la plus cohérente à partir d'exemples de transformation. La transformation d'un élément du modèle source est vue comme une particule, qu'il faut alors placer dans l'espace des transforma-

tions possibles. Dans Robbes et Lanza (2008), l'exemple de transformation est une modification apportée à un programme et qui est enregistrée lors de son exécution. Cette modification enregistrée peut ensuite se manipuler comme une entité de première classe qui va être généralisée et paramétrée puis appliquée à une partie ou à tout le système. Deux propositions se rapprochent plus de notre travail actuel. Des règles de transformation de graphes sont dérivées semi-automatiquement à partir d'appariements entre modèles définis à la main par un utilisateur dans Varró (2006). Cette approche se base sur l'analyse de contexte des noeuds du modèle. Une technique d'analyse basée sur la logique inductive est présentée dans Varró et Balogh (2007). Wimmer et al. (2007) se basent également sur des exemples de transformations décrits avec les syntaxes concrètes des méta-modèles source et cible pour dériver des règles ATL.

Notre travail reprend le principe d'analyse de contexte à partir d'appariements définis par l'utilisateur. Nous proposons pour effectuer cette analyse l'utilisation de l'ARC. Nous offrons ainsi la possibilité d'obtenir des résultats classés sous forme de treillis qui permettra une navigation dans la hiérarchie des règles générées. Les appariements correspondent actuellement à des couples d'éléments, mais sont pour le moment effectués sur la syntaxe abstraite des modèles et non sur leur syntaxe concrète.

6 Conclusion et perspectives

Notre proposition dans ce papier est de générer une relation de transformation de modèles à partir d'exemples de modèles transformés et d'appariements entre des éléments des modèles source et cible. Grâce à l'utilisation de notre méthode, il est possible de générer une transformation de modèles sans avoir besoin de maîtriser un langage de transformation et sans passer par les méta-modèles. L'utilisation de l'ARC permet par ailleurs la classification des règles sous forme de treillis. L'utilisateur peut ainsi naviguer facilement entre les différentes règles.

La méthode présentée dans ce papier a été testée sur des exemples de taille réduite. Les treillis sont générés automatiquement grâce à l'outil Galatea (Falleri et al., 2007), les contextes sont générés et les règles sont calculées à partir du treillis manuellement. Une des perspectives sera donc l'implémentation d'un outil automatisant les phases d'ARC et de génération de règles permettant des tests à plus grande échelle. Il apparaît nécessaire aussi de prendre en compte des liens à plus de deux extrémités, l'exemple nous le montre avec des *Property* qui sont transformées en *Role* et *Cardinality*. Enfin, comme nous l'indiquons dans la section 3, l'utilisation d'opérateurs de scaling différents permettra d'enrichir les règles produites et la détection de nouveaux types de motifs.

Références

Barbut, M. et B. Monjardet (1970). *Ordre et Classification : Algèbre et Combinatoire*, Volume 2. Hachette.

Birkhoff, G. (1940). *Lattice theory*. American Math. Society, Providence, RI, 1st edition.

Cypher, A., D. C. Halbert, D. Kurlander, H. Lieberman, D. Maulsby, B. A. Myers, et A. Turransky (1993). *Watch What I Do : Programming by Demonstration*. The MIT Press.

Fabro, M. D. D. (2007). *Metadata management using model weaving and model transformation*. Ph. D. thesis, Université de Nantes.

Fabro, M. D. D. et P. Valduriez (2007). Semi-automatic model integration using matching transformations and weaving models. In *ACM SAC'07*, pp. 963–970.

Falleri, J.-R., G. Arévalo, M. Huchard, et C. Nebut (2007). Use of Model Driven Engineering in Building Generic FCA/RCA Tools. In *CLA'07, CEUR Work. Proc. 331*, pp. 225–236.

Falleri, J.-R., M. Huchard, M. Lafourcade, et C. Nebut (2008). Meta-model Matching for Automatic Model Transformation Generation. In *MODELS'08, LNCS 5301*, pp. 326–340. Springer.

Ganter, B. et R. Wille (1999). *Formal Concept Analysis, Mathematical Foundations*. Springer.

Huchard, M., M. R. Hacene, C. Roume, et P. Valtchev (2007). Relational concept discovery in structured datasets. *Ann. Math. Artif. Intell. 49*(1-4), 39–76.

Jouault, F. et I. Kurtev (2005). Transforming models with atl. In J.-M. Bruel (Ed.), *MoDELS Satellite Events*, pp. 128–138. Springer.

Kessentini, M., H. Sahraoui, et M. Boukadoum (2008). Model Transformation as an Optimization Problem. In *MODELS'08, LNCS 5301*, pp. 159–173. Springer.

Lieberman, H. (2000). *Your Wish is My Command : Giving Users the Power to Instruct their Software*. Morgan Kaufmann.

Lopes, D., S. Hammoudi, et Z. Abdelouahab (2006). Schema matching in the context of model driven engineering : From theory to practice. In *Advances in Systems, Computing Sciences and Software Eng.*, pp. 219–227. Springer.

Lopes, D., S. Hammoudi, J. Bézivin, et F. Jouault (2005). Generating transformation definition from mapping specification : Application to web service platform. In *CAiSE'05, LNCS 3520*, pp. 309–325.

Lopes, D., S. Hammoudi, J. de Souza, et A. Bontempo (2006). Metamodel matching : Experiments and comparison. In *ICSEA*, pp. 2. IEEE Computer Society.

Melnik, S., H. Garcia-Molina, et E. Rahm (2002). Similarity flooding : A versatile graph matching algorithm and its application to schema matching. In *ICDE, LNCS 2593*, pp. 117–128.

MTIP (2005). Model transformations in practice workshop. http ://sosym.dcs.kcl.ac.uk/events/mtip05/.

OMG (2008). MOF^{TM} Query / Views / Transformations. Technical report, OMG.

Rahm, E. et P. A. Bernstein (2001). A survey of approaches to automatic schema matching. *VLDB J. 10*(4), 334–350.

Robbes, R. et M. Lanza (2008). Example-based Program Transformation. In *MODELS'08, LNCS 5301*, pp. 174–188. Springer.

Roser, S. et B. Bauer (2006). An approach to automatically generated model transformations using ontology engineering space. In *Proceedings of Workshop on Semantic Web Enabled Software Engineering (SWESE)*.

Shvaiko, P. et J. Euzenat (2005). A survey of schema-based matching approaches. In *J. Data Semantics IV, Volume 3730 of LNCS*, pp. 146–171.

Varró, D. (2006). Model transformation by example. In *MODELS'06, LNCS 4199*, pp. 410–424. Springer.

Varró, D. et Z. Balogh (2007). Automating model transformation by example using inductive logic programming. In *ACM SAC'07*, pp. 978–984.

Wimmer, M., M. Strommer, H. Kargl, et G. Kramler (2007). Towards model transformation generation by-example. In *HICSS*, pp. 285. IEEE Computer Society.

Summary

In Model Driven Engineering, transformation operations are basic processes. They are usually developped by specialised programmers and for every change in the metamodels or the needs, the source code must be updated. A possible way to make transformation development easier is to generate transformation rules from transformations examples. The advantage of using examples is that they are written in their concrete syntax, which is more accessible for the user than a transformation language. In this paper, we specify a method using relational concept analysis to find and organise by specialisation recurrent patterns in transformation examples. Those recurrent patterns can then be used for transformation rules production.