Méta-modélisation de la transformation de modèles par l'exemple : approche par méta-heuristiques

Marouane Kessentini*, Houari Sahraoui*, Mounir Boukadoum**

* Département d'informatique et de recherche opérationnelle, Université de Montréal
CP 6128 succ. Centre Ville, Montréal, Québec, H3C 3J7, Canada,
{kessentm, sahraouh}@iro.umontreal.ca
** Département d'informatique, Université du Québec à Montréal
CP 8888, succursale Centre-ville, Montréal QC H3C 3P, Canada
mounir.boukadoum@uqam.ca

Résumé. La plupart des contributions en transformation de modèles sont concernées par la définition de langages pour exprimer des règles de transformation. La définition de ces règles est une tâche difficile, car de nombreux problèmes, liés à l'écriture/génération des règles, doivent être anticipés surtout dans le cas des formalismes source/cible qui ne sont pas largement utilisés. Dans cet article, nous proposons de considérer le problème de transformation comme un problème d'optimisation combinatoire où un modèle cible peut être automatiquement généré à partir d'un nombre réduit d'exemples de transformations. Nous proposons en particulier, une méta-modélisation de notre approche MOTOE, basée sur la méta-heuristique recuit simulé, qui combine un ensemble de solutions de transformation pour converger vers une solution optimale tenant compte de la cohérence entre les transformations. Les résultats de la validation sur des données industrielles montrent que les modèles obtenus sont comparables à ceux proposés par les experts de notre partenaire industriel.

1 Introduction

Dans le cadre de l'ingénierie du logiciel dirigée par les modèles, la transformation de modèles est une activité qui prend une place de plus en plus importante dans le cycle de développement : génération de code, maintenance, optimisation de code, composition d'aspects, rétro-ingénierie, etc.. Ainsi, les langages de transformation de modèles représentent des composantes de premier ordre d'un environnement de développement. L'objet de base de ces langages est le modèle qui requiert la définition de nouveaux opérateurs, entre autres, de construction, navigation, composition, comparaison et évaluation.

Malgré de nombreuses contributions importantes telles que celles décrites dans (OMG, 2003), le problème de la transformation de modèles est loin d'être résolu. La principale difficulté est de définir ou d'exprimer des règles de transformation, en particulier pour les formalismes qui ne sont pas largement utilisés. La plupart des contributions en transformation de modèles sont concernées par la définition de formalismes pour exprimer des règles de transformation. Ces règles pourraient être mises en œuvre en utilisant différents types de lan-

gages : de programmation classique tels que Java ou C#; de transformations par graphes comme AGG (Taenzer, 2003) et VIATRA (Varro et Pataricza, 2004), ou spécifiques tels que ATL (Jouault et Kurter, 2005) et KERMETA (Muller et al., 2005).

La définition des règles de transformation est une tâche difficile (Behrens et al., 1996). En effet, il est généralement difficile de trouver des experts qui maîtrisent à la fois les formalismes source et cible de la transformation pour pouvoir l'exprimer (Varro, 2006). D'autre part, il est connu que les experts peuvent plus facilement donner des exemples de transformation plutôt que des règles cohérentes et complètes (Egyed, 2002). Ceci est particulièrement vrai au sein d'organisations industrielles où une mémoire des projets réalisés (exemples de transformations) peut être trouvée.

Cette hypothèse est la principale motivation pour les approches de transformations par l'exemple telles que celle proposée dans (Varro et Balogh, 2007; Varro, 2006). Le principe de cette approche est de générer semi-automatiquement des règles de transformation, en utilisant la programmation logique inductive (PLI), à partir d'un ensemble d'exemples de transformation. En pratique, cependant, un grand nombre d'exemples de transformation est nécessaire pour garantir la qualité des règles obtenues.

Dans ce contexte, nous proposons de considérer le problème de transformation comme un problème d'optimisation combinatoire où un modèle cible peut être automatiquement généré à partir d'un nombre réduit d'exemples. Dans ce sens, nous avons proposé, récemment, une nouvelle approche appelée MOTOE (MOdel Transformation as an Optimization by Example) (Kessentini et al., 2008). L'approche MOTOE considère la transformation d'un modèle source comme un processus de recherche dans un espace multidimensionnel où chaque dimension représente un élément du modèle source (voir la section 3). L'approche MOTOE que nous avons proposée dans (Kessentini et al., 2008) utilise la méta-heuristique "Optimisation par Essaims Particulaires" (Kennedy et al., 1995) appelée *PSO* (pour *Particle Swarm Optimization* en anglais). Notre approche combine un ensemble de solutions de transformation pour converger vers une solution optimale en optimisant l'adéquation, la cohérence interne et externe de la combinaison des transformations individuelles des éléments du modèle. Dans cet article, nous proposerons une méta-modélisation de l'approche MOTOE et une validation plus large basée sur des données industrielles comparativement à celle décrite dans (kessentini et al., 2008). D'autre part, nous évaluons une autre méta-heuristique, le recuit simulé (*SA* ou *Simultaed Annealing* en anglais), qui contrairement à PSO, consiste à modifier itérativement une solution de transformation initiale (Kirkpatrick et al.,1983).

Le contenu de cet article s'articule comme suit. La section 2 détaille les travaux existants en transformation de modèles par l'exemple. La section 3 présente la méta-modélisation sous-jacente à notre approche. La section 4 décrit l'adaptation de la méta-heuristique SA pour le problème de la transformation de modèle par l'exemple. La section 5 évalue les résultats de la validation effectuée sur des projets industriels. Enfin, la conclusion et quelques perspectives sont présentées en section 6.

2 Transformation de modèles par l'exemple

(Czarnecki et Helsen, 2003) classifient les approches existantes en deux principales catégories: modèle à modèle et modèle à code. Ils décrivent cinq catégories d'approches: approches de transformation par graphes, relationnelles, dirigées par les structures, de manipulation directe et hybrides. Différents critères sont utilisés pour les analyser, dont l'usage ou

non de l'approche MDA (Architecture Dirigée par les Modèles) en tant que base de la transformation, la complexité du mécanisme de transformation, l'usage ou non d'annotations, le niveau d'automatisation, et les langages et techniques utilisés. Par ailleurs, ces approches sont souvent basées sur des règles obtenues empiriquement (Egyed, 2002).

De nombreuses limitations sont liées à la définition des règles de transformation (Behrens et al., 1996). De plus, il est difficile d'assurer ces propriétés dans des domaines où peu de connaissances existent (Egyed, 2002). Les difficultés sont amplifiées par la maintenance de ces ensembles de règles. Une autre limitation est liée à la nature subjective de certaines transformations. En effet, les experts peuvent avoir des opinions divergentes sur la transformation de certains éléments (Varro et Balogh, 2007; Varro, 2006). Par exemple, dans le cadre de la transformation diagramme de classes (CLD) à schéma relationnel (SR), certains experts transforment un lien de généralisation entre deux classes en deux tables liées par une clé étrangère, tandis que d'autres suggèrent la création d'une unique table regroupant les informations provenant des deux classes. D'autre part, il est souvent difficile de trouver des experts qui maîtrisent à la fois les deux formalismes source et cible (Varro, 2006). Par exemple, certains sont experts en UML, mais pas en base de données ou bien le contraire. Par la suite, nous pensons que l'utilisation de l'approche "par l'exemple" permet de remédier à ces limites.

En effet, l'approche proposée dans cet article est basée sur l'utilisation de l'historique des projets déjà réalisés (exemples de transformation). En littérature, plusieurs approches "par l'exemple" ont été proposées dans plusieurs domaines. Les plus connues sont la programmation par l'exemple (Cypher, 1993) et l'interrogation par l'exemple (en anglais Query By Example) (Krishnamurthy et al., 1983). Pour la transformation de modèles, (Varro et Balogh, 2007) présentent une approche semi-automatique de transformation de modèles par l'exemple (MTBE) basée sur la "Programmation Logique Inductive" (PLI). Le principe de cette approche est de générer semi-automatiquement des règles de transformation à partir d'un ensemble d'exemples. (Wimmer, et al., 2007) proposent un autre travail similaire qui dérive des règles ATL de transformation à partir d'exemples de transformation. Ces deux approches utilisent la correspondance sémantique entre les modèles source et cible pour générer des règles. La différence entre les deux est que (Wimmer, et al., 2007) s'appuie sur une technique ad hoc, tandis que (Varro et Balogh, 2007) génèrent des règles de transformation par PLI et en interaction avec l'expert. Les deux approches sont fortement dépendantes des modèles source et cible.

Les algorithmes d'apprentissage comme PLI sont efficaces lorsque les exemples couvrent l'ensemble des éléments à transformer. Dans le cas des formalismes complexes, l'ensemble des possibilités de transformation des éléments peut être très important. Ainsi pour que les règles de transformation générées soient correctes, un grand nombre d'exemples de transformation est nécessaire. En l'absence de règles ou d'un nombre important d'exemples, une autre solution consiste à générer un modèle cible (parfois, partiel) en exploitant au mieux les exemples disponibles. C'est dans cette optique que nous proposons une nouvelle approche appelée MOTOE (MOdel Transformation as an Optimization Problem). Notre travail se situe essentiellement dans la catégorie de transformations « modèle à modèle », mais n'appartient à aucune des cinq catégories proposées dans (Czarnecki et Helsen, 2003). Par ailleurs, ce que nous proposons est également différent des approches de transformation de modèles par l'exemple décrites plus haut.

3 Méta-modélisation de la transformation de modèle par l'exemple

Dans cette section, nous présentons une méta-modélisation de notre approche. Le méta-modèle permet de décrire notre approche indépendamment des modèles source et cible utilisés. Ensuite, nous illustrons notre méta-modèle avec le cas de la transformation CLD vers SR.

3.1 Méta-modélisation de MOTOE

L'approche MOTOE a pour objectif de transformer un modèle donné, exprimé dans un formalisme en un autre modèle exprimé dans un autre formalisme en s'inspirant d'une base d'exemples de transformation. Le méta-modèle de la figure 1 décrit les composantes de base de MOTOE :

- **Modèle Source et cible** : l'entrée du processus de transformation est un modèle à transformer appelé *modèle source*. La sortie, qui représente le modèle généré, est appelée *modèle cible*.
- **Élément et sous-élément** : un modèle est constitué d'un ensemble d'*éléments*. Un élément peut être composé d'un ensemble de sous-éléments.
- **Trace** : une trace est définie comme un ensemble de liens entre un ou plusieurs éléments du modèle source et leurs équivalents dans le modèle cible.
- **Bloc** : l'ensemble de trace des transformations d'un sous-ensemble des éléments du modèle source vers le modèle cible définit *bloc*. La transformation entière d'un modèle est réalisée sous la forme d'un ensemble de blocs.
- **Exemple de transformation** : le triplet (modèle-source, modèle-cible, blocs) est appelé *exemple de transformation*. Il définit la transformation d'un modèle source en un modèle cible par un ensemble de blocs.
- **Base d'exemples** : L'ensemble des exemples de transformation disponibles à un moment donné est appelé *base d'exemples*.

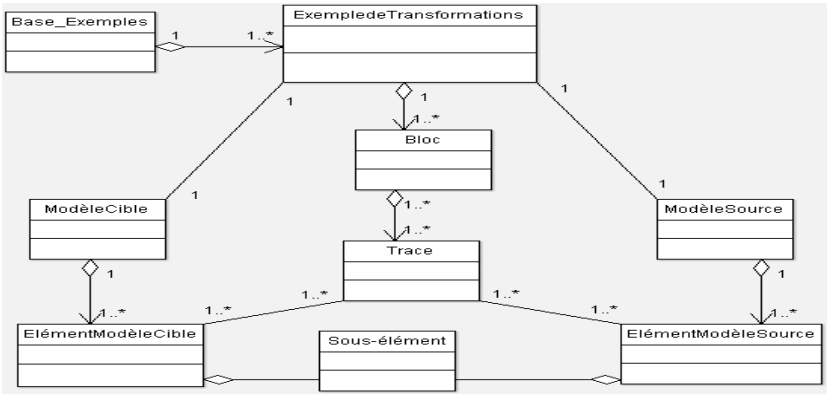


FIG. 1 – *Méta-modèle de MOTOE.*

La figure 2 illustre les différentes étapes de fonctionnement de notre approche. Le modèle source contient un ensemble d'éléments à transformer. Nous affectons aléatoirement une

possibilité de transformation (bloc) pour chacun des éléments du modèle source. Les possibilités de transformations (blocs) sont extraites de la base d'exemples. Cette étape peut générer une solution (première méthode qui correspond à SA) ou une population de solutions (deuxième méthode qui correspond à PSO). Nous évaluons, via une fonction objectif, la qualité de la ou les transformations proposées. Enfin, la dernière étape consiste à raffiner la solution ou les solutions proposées et itérer les différentes étapes jusqu'à converger vers une solution acceptable (modèle cible de bonne qualité).

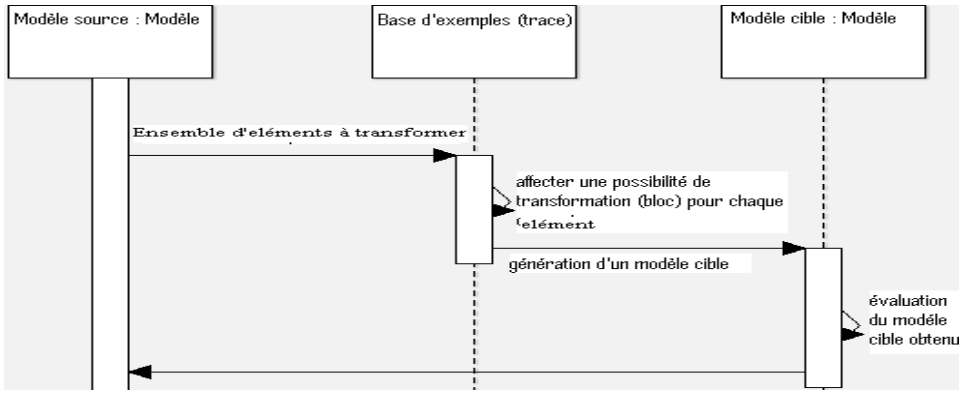
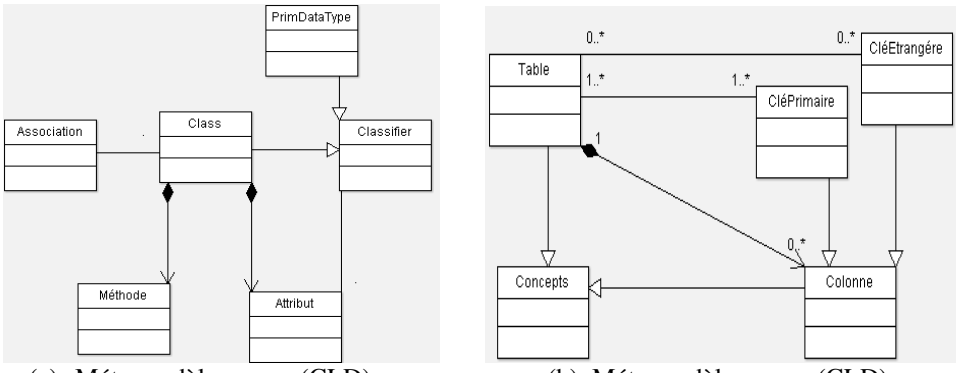


FIG. 2 – *Diagramme de séquence : fonctionnement de MOTOE.*

3.2 Instanciation du méta-modèle : CLD vers SR

Dans cette sous-section, nous présentons une instanciation du méta-modèle présenté précédemment, pour le cas de la transformation de diagrammes de classes vers des schémas relationnels. Cette instanciation permettra d'illustrer notre contribution.

La méta-modélisation simplifiée du CLD (figure 3(a)) montre qu'il est constitué principalement de 4 éléments : classe, attribut, association et méthode. Tandis que le modèle cible SR (figure 3(b)) est constitué principalement de 4 éléments : table, clé étrangère, clé primaire, colonne.

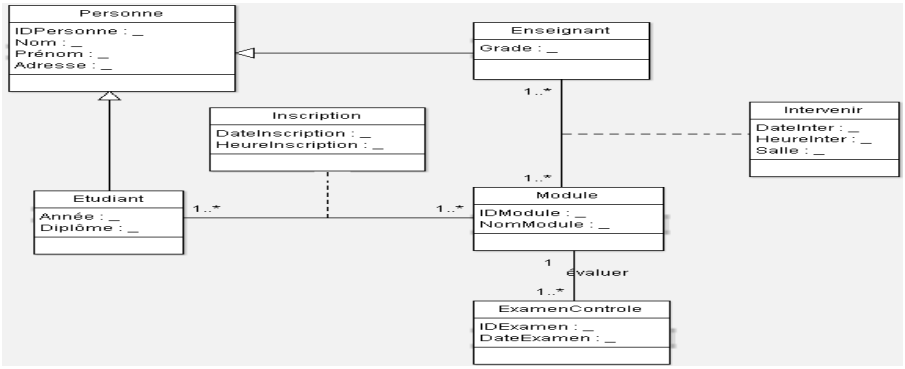


(a) Méta-modèle source (CLD) (b) Méta-modèle source (CLD)

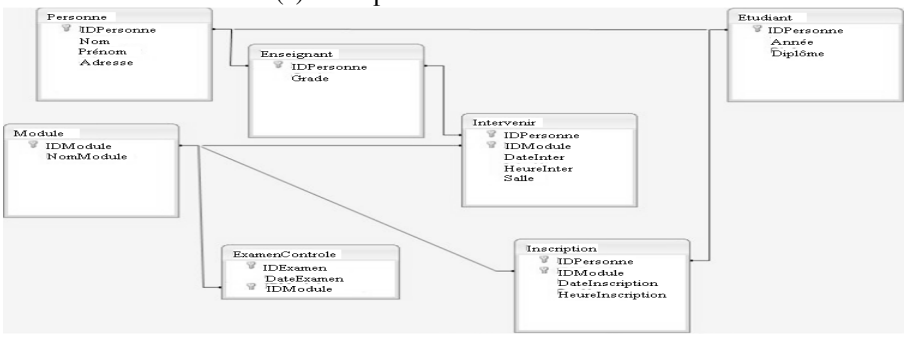
FIG. 3 – *Méta-modèles.*

La figure 4 illustre un exemple de transformation d'un diagramme de classes CLD vers un schéma relationnel SR. Le CLD est le modèle source (a) et le SR est le modèle cible (b). Le CLD contient 12 éléments qui représentent 7 classes, 3 associations et 2 liens de généralisation. Les attributs de l'élément classe sont des sous-éléments. Cinq classes sont transformées en des tables et les attributs sont transformés en colonnes de ces tables. Les associations entre les classes *Etudiant* et *Module*, et entre *Enseignant* et *Module*, sont respectivement transformées en tables *Inscription* et *Intervenir*. L'association *Evaluer* est devenue une clé étrangère dans la table *ExamenControle*. Enfin, le lien de généralisation est transformé en clé étrangère.

Les décisions prises pour ces transformations ne sont pas uniques. En effet, un lien de généralisation, par exemple, peut se transformer de plusieurs manières (plusieurs opinions des experts). Par exemple, la classe *Personne* de la figure 4 peut se transformer en une table ou bien en dupliquant ses attributs dans les tables *Etudiant* et *Enseignant*.



(a) Exemple de modèle source A.



(b) Modèle cible B généré à partir de A.

FIG. 4 – *Exemple de modèles source et cible d'une transformation.*

Un exemple de transformation est un triplet <MDS, MDC, TR>, où MDS et MDC sont, respectivement, les descriptions (en logique des prédicats) des modèles source et cible. TR est un ensemble de blocs qui relient des sous-ensembles de MDS à leurs équivalents en MDC. La base des exemples contient l'ensemble des exemples de transformation (E.T).

Comme indiqué dans la sous-section précédente, un bloc est défini comme l'ensemble minimal d'éléments du modèle source dont les transformations sont indépendantes du reste des éléments. Par exemple, l'association *évaluer* dans la figure 4(a) ne peut pas être trans-

formée de façon indépendante des classes *Module* et *ExamenControle*. La figure 5(b) présente un exemple de partitionnement d'un modèle source en cinq sous-ensembles d'éléments correspondant à cinq blocs de transformation. Le concept de bloc représente l'une des différences entre notre approche MOTOE et le raisonnement à base de cas où le niveau de granularité doit être l'exemple entier (Aamodt et Plaza, 1994). Dans notre cas, il ne s'agit pas de sélectionner l'exemple le plus proche et d'adapter sa transformation, mais plutôt de combiner plusieurs parties (blocs) des exemples.

Chaque élément du modèle est représenté par un ou plusieurs prédicats. Par exemple, la classe *Enseignant* de la figure 4 (a) est décrite comme suit :

Classe (Enseignant).
Attribut (Grade, Enseignant, _)
.

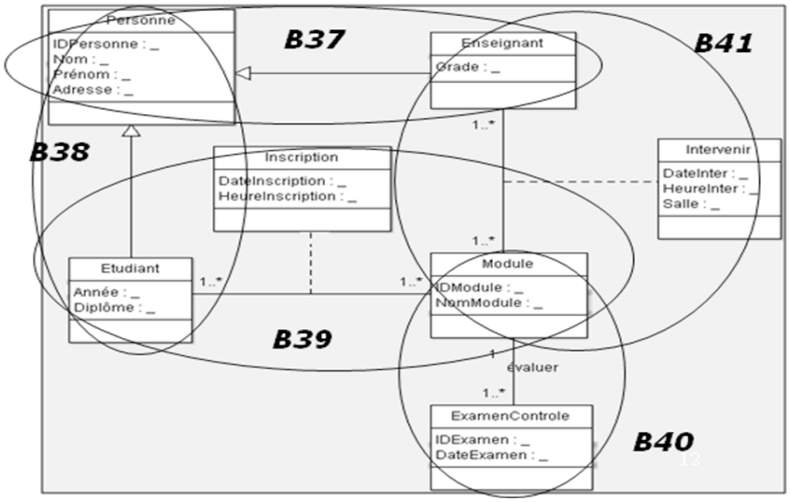


FIG. 5 – *Génération des blocs pour l'exemple de transformation n°4 (ET4).*

Le premier prédicat indique que *Enseignant* est une classe. Le second indique que *Grade* est un attribut de la classe *Enseignant* et que sa valeur n'est pas unique ("_"). Le bloc contient la trace TR des transformations des éléments d'un sous-composant du modèle source en leurs équivalents dans le modèle cible. Dans la figure 5 par exemple, le bloc B37 qui contient un lien de généralisation et les deux classes *Enseignant* et *Personne* est décrit comme suit :

Début B37
Classe (Personne): Tableau (Personne).
Attribut (IDPersonne, Personne, unique): Colonne (IDPerson, Personne, pk).
Attribut (Nom, Personne, _): Colonne (Nom, Personne, _).
Attribut (Prénom, Personne, _): Colonne (Prénom, Personne, _).
Attribut (Adresse, Personne, _): Colonne (Adresse, Personne, _).
Classe (Enseignant): Tableau (Enseignant).
Attribut (Grade Enseignant, _): Colonne (Grade, Enseignant, _).
Généralisation (Personne, Enseignant): Colonne (IDPersonne, Enseignant, pfk).
Fin B37

Les classes *Personne* et *Enseignant* sont transformées en tables avec les mêmes noms. L'attribut *IDPersonne*, qui est de type *unique*, est transformé en une clé primaire (*pk*) *IDPer-*

sonne. Le lien de généralisation génère une clé primaire-étrangère (*pfk*) *IDPerson* dans la table *Enseignant*.

En l'absence des règles de transformation, un élément du modèle peut être transformé de plusieurs manières. Chaque possibilité de transformation possède un degré de pertinence qui dépend, principalement, de trois facteurs qui sont : l'adéquation de la transformation, la cohérence interne (au niveau du bloc) et la cohérence externe (entre les blocs).

Par exemple, considérons deux classes *Chat* et *Animal* qui sont reliées par un lien de généralisation G. G pourrait être transformé de plusieurs façons en : table, colonne, clé étrangère, etc. Dans le cas où le bloc B37 a été affecté à l'élément G, l'adéquation est vérifiée puisque le prédicat généralisation existe dans B37. La cohérence interne est assurée parce les deux prédicats ont les mêmes types de paramètres (deux classes). Enfin, la cohérence externe est vérifiée si les deux éléments reliés *Chat* et *Animal* seront transformés par leurs blocs associés de la même manière que *Personne* et *Enseignant* (B37), c.-à-d, en deux tables.

La qualité de transformation d'un modèle de source est la somme des qualités de la transformation de ses éléments en tenant compte de la cohérence externe (contexte). Par conséquent, la recherche d'une bonne transformation est équivalente à la recherche de la meilleure combinaison de transformations des éléments (à partir de la base des exemples) qui maximise la qualité globale. Comme le nombre de combinaisons peut être très important, il est impossible de les évaluer de manière exhaustive. La recherche heuristique offre une bonne alternative dans ces cas. Une solution consiste à affecter un bloc (possibilité de transformation) pour chaque élément. La qualité de la transformation pour chaque solution est évaluée par la fonction objectif. La transformation d'un modèle M contenant n éléments, utilisant une base d'exemple contenant m possibilités, consiste à parcourir une combinaison de m^n possibilités avec n est grand. Pour explorer l'espace de recherche, nous proposons d'utiliser une méta-heuristique, SA (Kirkpatrick et al.,1983). Par la suite, PSO (Kennedy et al., 1995) est exécutée pour obtenir une solution initiale. Cette solution est ensuite affinée en utilisant l'algorithme SA. La modification d'une solution consiste à explorer des solutions voisines en modifiant les blocs affectés pour quelques éléments.

4 Approche méta-heuristique pour la transformation de modèles par l'exemple : le recuit simulé

Dans cette section, nous exposerons notre adaptation du SA pour l'automatisation de la transformation de modèles par l'exemple. En particulier, nous détaillerons les trois phases d'adaptation : le codage (représentation des solutions), la définition de la fonction de voisinage pour guider le mouvement dans l'espace de recherche, et la fonction objectif.

4.1 Recuit simulé (Simulated Annealing)

4.1.1 Principes de SA

Dans cet article, nous proposons de combiner PSO [15] avec une autre méta-heuristique, le recuit simulé (SA). En effet, une exécution très rapide de la méta-heuristique PSO (quelques itérations) peut être améliorée en utilisant SA. SA [17] est une méta-heuristique inspirée d'un processus utilisé en métallurgie. Ce processus alterne des cycles de refroidissement lent et de réchauffage (recuit) qui tendent à minimiser l'énergie du matériau. À chaque

itération de l'algorithme du SA une modification élémentaire d'une solution est effectuée. L'algorithme 1 montre les étapes de fonctionnement de SA.

Algorithme SA

1: solution <- solution_initiale
2: coût <- évaluation (solution)
3: T <- $T_{initiale}$
4: ***tantque*** *(T >* T_{final} *)* ***faire***
5: ***pour*** *i=1 à itérations (T)* ***faire***
6: *nouvelle_solution <- modifier(solution)*
7: *nouveau_coût <- évaluation(solution)*
8: *Δcost <- nouveau_coût – coût*
9: ***si****(Δcost≤0 OU random()<*$e^{-\Delta cost/<T}$*)*
10: *solution <- nouvelle_solution*
11: *coût <- nouveau_coût*
12: ***end if***
13: ***fin pour***
14: *T<- nouvelle_temp(T)*
15: ***fin tantque***

Algorithm1. *Algorithme SA.*

4.1.2 Adaptation du SA pour le problème de transformation

L'adaptation d'une méta-heuristique à un problème consiste en trois étapes à définir : 1) une représentation pour la solution, 2) la fonction objectif 3) la fonction de voisinage. Dans notre cas, la solution initiale pour l'algorithme de SA est générée via une exécution rapide de l'algorithme de PSO (nombre réduit d'itérations). Comme le cas de PSO, la fonction objectif (section 4.4) mesure la qualité de la solution modifiée à la fin de chaque itération. Une solution voisine est générée, par la fonction de voisinage (section 4.3).

Pour diminuer la température, nous utilisons la formule suivante [10] :

$$T_{i+1} = \alpha * T_i \quad (5)$$

Tel que α est une constante inférieure à 1. Dans ce qui suit, nous présentons en détail les trois étapes d'adaptation.

4.2 Modélisation d'une solution

Une des principales questions lors de l'application d'une méta-heuristique à notre problème est de trouver comment encoder une transformation d'un modèle source vers un modèle cible comme une solution dans l'espace de recherche. Pour cela, comme indiqué précédemment dans la section 3, nous définissons un espace de recherche à n dimensions où chaque dimension correspond à un élément du modèle à transformer. Le domaine de chaque dimension est un nombre fini de valeurs b = {i | 1 ≤ i ≤ m}, où m est le nombre de blocs (possibilités) extraits de l'ensemble des exemples de transformation (base d'exemples). L'ensemble des blocs est une entrée pour le mécanisme de transformation. Par exemple, la transformation du modèle figurant dans la figure 6 génère un espace de recherche à 7 dimensions correspondant aux quatre classes et les trois associations.

Une solution consiste à affecter pour chaque dimension (élément du modèle à transformer) un numéro de bloc proposant une possibilité de transformation. Le vecteur représentant une solution dans l'espace de recherche correspond à l'ensemble des dimensions et leurs blocs associés. Par exemple, la table 1 présente une solution qui transforme l'élément 1 selon le bloc 28, l'élément 2 selon le bloc 3, etc. où chaque bloc représente une possibilité de transformation pour un sous-composant du modèle source.

L'affectation d'un bloc à un élément du modèle à transformer ne signifie pas nécessairement que la transformation est possible, c'est-à-dire, qu'un élément similaire existe dans le bloc affecté. La fonction objectif (décrite dans la section 4.3) permet de pénaliser certaines approximations.

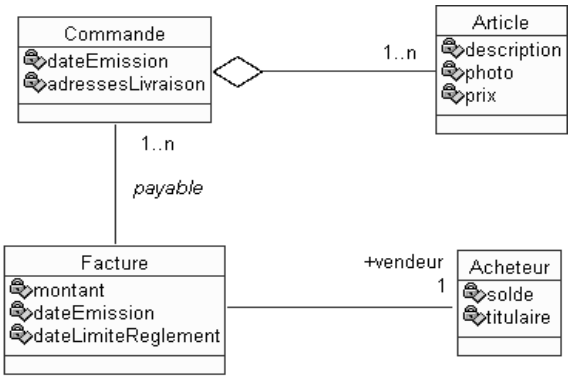


FIG.6 – *Exemple de modèle source (CLD).*

Dimension	Élément	Numéro du bloc
1	Classe(Commande)	28
2	Classe(Facture)	3
3	Classe(Article)	21
4	Classe(Acheteur)	13
5	Composition	9
6	Association(Payable)	42
7	Association(Vendeur)	5

TAB. 1 – *Codage : représentation de la solution.*

4.3 Fonction de voisinage

Un opérateur de changement permet de modifier une solution afin de générer une nouvelle solution voisine. Dans le cadre de notre problème, une solution de voisinage consiste à affecter un nouveau numéro de bloc à une ou plusieurs dimensions. En d'autres termes, l'application de l'opérateur de changement propose une nouvelle alternative pour la transformation de quelques éléments dans le modèle à transformer. Par exemple, la figure 7 présente deux solutions de transformation pour les 7 éléments du modèle source. Après l'application de l'opérateur, l'élément 1 est transformé selon le bloc B31 au lieu du bloc B28, et l'élément 3 par le bloc B5 au lieu du bloc B21. Les autres éléments du modèle gardent la même possibilité de transformation.

Le nombre de dimensions à changer dépend de la nature de la méta-heuristique à utiliser. Par exemple, pour le recuit simulé deux dimensions sont choisies au hasard.

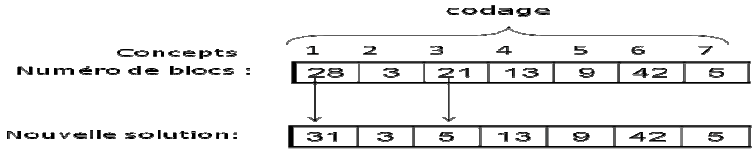


FIG. 7 – *Opérateur*

4.4 Fonction objectif

La fonction objectif (appelée aussi fonction coût ou d'évaluation) permet d'évaluer la qualité d'une solution de transformation. Comme mentionné dans la section 3, cette fonction d'évaluation doit prendre en considération les critères suivants : l'adéquation de la transformation, la cohérence interne (au niveau du bloc) et la cohérence externe (entre les blocs).

Par conséquent, nous définissons la fonction objectif comme la somme des qualités individuelles de transformation des n éléments du modèle à transformer (équation1).

$$f = \sum_{j=1}^{n} a_j \times (ic_j + ec_j) \qquad (1)$$

Le paramètre a_j représente l'adéquation de la transformation proposée pour le j^{éme} élément. En effet, a_j est égal à 1 si le bloc associé au j^{éme} élément contient un élément similaire, sinon a_j est égal à 0. Deux éléments sont similaires dans le cas où ils ont le même type de prédicat. Par exemple : *Association* et *Classe* représentent deux types de prédicats différents.

Les deux paramètres ic_j et ec_j correspondent, respectivement, à la cohérence interne et externe :

$$ic_j = \frac{\text{nombre de paramètres équivalents avec ceux du j}^{ième}\ concept}{\text{nombre des paramétres du j}^{ième}\ \text{concept}} \qquad (2)$$

$$ec_j = \frac{\text{nombre de concepts équivalents reliés au j}^{ième}\ \text{concept}}{\text{nombre des concepts reliés au j}^{ième}\ \text{concept}} \qquad (3)$$

Dans ce qui suit, nous illustrerons, via l'exemple des fig.6 et fig.7, le calcul de la fonction objectif. L'association *payable* (6^{éme} dimension) est définie par le prédicat suivant :

Association (1,n,1,1, Payable, Commande, Facture).

Le prédicat *Association* contient 7 paramètres. Les quatre premiers paramètres indiquent les multiplicités (1 .. n et 1 .. 1). Le cinquième paramètre est le nom de l'association, et les deux derniers paramètres correspondent aux noms des classes associées. Par exemple, considérons qu'une solution *s1* attribue le bloc B42 à cette association. Le bloc B42 est défini comme suit :

Begin B42
Classe(Client) : Table(Client).
Attribut(NClient, Client, unique) : Colonne(NClient, Client, pk).
…
Classe(Réservation) : Table(Réservation).
Attribut(NReservation, Réservation, unique) : Colonne(NReservation, Réservation, pk).
…

Association (1,n,0,n, Effectuer, Client, Réservation) : Colonne(N_Client, Réservation, fk).
End B42

Dans ce cas, a_6 est égal à 1, car le bloc B42 contient le prédicat *Association* entre les classes *Client* et *Réservation*. Comme mentionné précédemment le prédicat association contient 7 paramètres. Par rapport à l'association *Payable* (élément à transformer), l'association *Effectuer* contient cinq paramètres équivalents (en caractère gras) sur les 7 paramètres :

*Association (**1**,**n**,1,1, **Payable**, **Commande**, **Facture**).*
*Association (**1**,**n**,0,n, **Effectuer**, **Client**, **Réservation**).*

En effet, lors d'une comparaison entre deux prédicats, on suppose que les noms (classes, prédicats) sont équivalents. Par conséquent, $ic_6 = 5 / 7 = 0.71$. D'autre part, conformément au bloc B42, pour assurer la cohérence externe, les deux classes liées *Facture* et *Commande* doivent être transformées en tables. Par contre, *s1* affecte, respectivement, les blocs B28 et B3 aux éléments Facture (dimension2) et Client (Dimension4). Nous supposons que dans ces deux blocs, les classes sont transformées en deux tables. Cette transformation est compatible avec le bloc B42 donc C6 = 2 / 2 = 1.

La fonction objectif évalue, indirectement, la complétude de la transformation. En effet, une solution qui ne transforme pas un sous-ensemble de constructions sera pénalisée (a_j égal à 0). Pour permettre une meilleure lecture des résultats, la fonction objectif est normalisée dans le domaine [0,1] comme suit :

$$f_{nor} = \frac{f}{2 * n} \quad (4)$$

5 Validation

Pour évaluer notre approche, nous avons effectué une validation sur des projets industriels. Nous présentons, dans la première sous-section, le contexte de nos expérimentations. Ensuite, nous analyserons les résultats obtenus. Enfin, nous évaluerons l'impact de la taille de la base d'exemples sur la qualité de la transformation des modèles.

5.1 Contexte des expérimentations

Nous avons implémenté l'algorithme de SA en Java. La base d'exemples utilisée contient 12 exemples fournis par un partenaire industriel. Ces modèles ont servi dans le développement logiciel de la compagnie. Chaque exemple représente une transformation d'un diagramme de classes UML vers son équivalent en schéma relationnel. Les diagrammes utilisés dans nos expérimentations sont de grandes tailles, allant de 28 à 92 éléments (table 2). En effet, dans le monde industriel les modèles utilisés sont de cette taille. Le nombre de blocs dans la base d'exemples est de 257 blocs. Pour l'évaluation, nous avons utilisé une validation croisée. En effet, chacun des 12 diagrammes de classes UML est transformé à l'aide des 11 autres exemples de transformation. Ensuite, les modèles cibles obtenus (en SR) sont comparés avec ceux fournis par notre partenaire industriel.

Exemples	1	2	3	4	5	6	7	8	9	10	11	12
Nombre des éléments	72	83	49	53	38	47	78	34	92	28	59	63

TAB. 2 – *Taille des exemples*.

Pour évaluer la qualité des modèles cibles générés, nous définissons la précision qui est le pourcentage des éléments qui ont été bien transformés.

$$P = \frac{\text{nombre de concepts bien transformés}}{\text{nombre total des concepts}} \quad (3)$$

Dans notre cas, l'algorithme du SA a été appliqué avec les paramètres suivants :

- La température initiale est choisie au hasard dans l'intervalle [0, 1]
- Le coefficient de refroidissement α est 0.98
- La condition d'arrêt (seuil de température) est de 0.1
- L'intervalle d'itérations pour chaque valeur de la température est de 10000 (SA est lent pour converger)
- PSO est utilisé pour générer, rapidement, une solution initiale. Par la suite, le nombre des particules a été limité à 10 et le nombre d'itérations à 20.

Avec ces paramètres, la transformation de diagrammes de classes de grandes tailles ne prend que quelques secondes.

5.2 Résultats

La table 3 illustre les précisions des 12 modèles transformés via la méta-heuristique SA. La précision moyenne est de 74% et la plupart des modèles ont été, également, transformés avec une précision acceptable (plus de 70%) surtout en absence de règles. En plus, la validation manuelle (qualitative) a donné des précisions de l'ordre de 90%. Le modèle 4 a la plus faible précision (67%).

En comparaison avec les résultats de PSO présentés dans (kessentini et al., 2008), la précision est meilleure pour le cas des grands modèles. Les résultats mentionnés dans le tableau 3 confirment que notre choix de la fonction objectif est un bon estimateur pour la précision de la transformation. Par contre, un modèle cible (solution) peut avoir une meilleure valeur de la fonction objectif qu'un autre modèle, mais une précision plus faible (cas des modèles 4 et 9 dans la table 3). Cette contradiction peut être expliquée par le fait que la qualité de la transformation dépend du nombre d'éléments du modèle. En effet, contrairement à la fonction objectif, la précision est liée au nombre total des éléments dans le modèle source à transformer.

Dans le cas où le nombre d'exemples disponibles est faible, la figure 8 montre que notre approche propose, également, de bonnes solutions de transformation dans cette situation. Avec 11 exemples seulement, nous avons obtenu une bonne qualité de transformation de modèle. En plus, avec 8 et 9 exemples dans la base, la précision dépasse le 70%. Pour les performances de calcul, toutes les expérimentations ont été menées en quelques minutes sur un ordinateur portable avec une configuration de base.

Les résultats précédents ont été obtenus en comparant les schémas relationnels (modèles cibles) générés automatiquement par notre approche avec ceux fournis par notre partenaire industriel. En conséquence, une bonne alternative de transformation d'un modèle source qui est différente du modèle de référence (celui de notre partenaire industriel) n'est pas prise en compte dans le calcul de la précision. En effet, après avoir étudié les modèles cibles générés, nous avons constaté qu'au moins 25% des transformations rejetées sont valables.

Pour certains modèles, les transformations proposées sont partielles puisqu'elles contiennent des éléments qui n'existent pas dans la base d'exemples. Par contre, certains éléments ont été transformés bien qu'il n'existe pas leurs similaires dans la base d'exemples. Par

exemple, la transformation d'une association avec la multiplicité (1 .. N, 1 .. N) est approximée par celle d'une association (0 .. N, 0 .. N) mais avec une pénalité dans la fonction objectif. L'un des avantages de notre approche est cette capacité d'adapter une transformation similaire à un nouvel élément.

Modèle source	Nombre des éléments	Fonction objectif	Précision
1	72	0.735	0.696
2	83	0.784	0.723
3	49	0.632	0.690
4	53	0.619	0.672
5	38	0.742	0.733
6	47	0.737	0.704
7	78	0.743	0.790
8	34	0.845	0.813
9	92	0.648	0.667
10	28	0.846	0.873
11	59	0.796	0.730
12	63	0.772	0.720
Précision moyenne :			0.740

TAB. 3 – *Taille des exemples*.

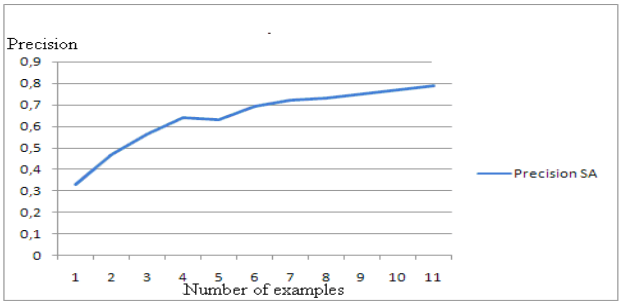


FIG. 8 – *Impact de la variation de la taille de la base d'exemples*.

Conclusion

Dans ce papier, nous avons proposé une nouvelle approche MOTOE basée sur les méta-heuristiques afin d'automatiser la transformation. MOTOE se base sur l'historique des projets de transformations déjà réalisés (exemples) pour générer automatiquement un modèle cible. Dans le cadre de notre approche, la transformation de modèle est considérée comme un problème d'optimisation où différentes possibilités de transformation sont proposées à partir de la base d'exemples et évaluées selon leurs similarités. L'espace de recherche est exploré par la méta-heuristique SA. EOP est exécutée pour obtenir une première solution afin d'initialiser une première solution. Cette solution est ensuite affinée en utilisant l'algorithme SA.

Notre approche est différente des approches existantes de transformation basées sur les règles puisque nous n'utilisons pas des règles pour faire la transformation. Par la suite, notre approche permet de remédier aux problèmes de définition des règles tels que l'exhaustivité, la cohérence et la non-redondance. Ce que nous avons proposé est différent de ce qui existe en transformation de modèle par l'exemple (Varro, 2006; Wimmer, et al., 2007). En effet, nous n'utilisons pas les exemples pour générer des règles et un nombre réduit d'exemples suffit pour générer un modèle cible de bonne qualité.

Nous avons présenté une méta-modélisation de notre approche indépendante des formalismes à transformer. Ensuite, nous avons illustré MOTOE par la transformation des diagrammes de classes UML vers des schémas relationnels. Dans ce cadre, nous avons effectué une validation à grande échelle sur des projets industriels. Les résultats de cette validation montrent que les modèles générés sont comparables à ceux proposés par les experts. Nous avons constaté que certains éléments ont été bien transformés bien que leurs équivalents n'existent pas dans la base d'exemples. En effet, MOTOE permet d'adapter la transformation d'un élément similaire à un élément donné du modèle. Nous avons, également, démontré que la combinaison EOP-SA donne de bons résultats de transformation pour les modèles de grande taille. D'autre part, un nombre réduit d'exemples permet d'avoir une transformation avec une précision acceptable. En effet, la validation montre que 9 exemples sont suffisants.

Nous allons démontrer prochainement que notre approche peut être appliquée pour d'autres formalismes (par exemple, diagrammes de séquence vers réseau de Pétri par exemple). Nous travaillerons, également, sur l'adaptation de notre approche pour d'autres problèmes de transformation tels que la génération de code (modèle vers code) et la refactorisation (code vers code). En ce qui concerne l'évaluation de la qualité des transformations, la fonction objectif pourrait être améliorée. En effet, dans ce travail, nous avons donné la même importance à la transformation de tous les éléments du modèle. Dans le monde réel, certains types d'éléments peuvent être plus importants que d'autres.

Références

Aamodt, A., et E.Plaza (1994). Case-Based Reasoning: Foundational Issues, Methodological Variations, and System Approaches, *Artificial Intelligence Communications* 7, pp. 39-52.

Behrens, U., M. Flasinski, L.Hagge, J. Jurek, et K.Ohrenberg (1996). Recent developments of the ZEUS expert system ZEX, IEEE Trans. Nucl. Sci., NS-43, pp. 65-68.

Cypher, A. (1993). Watch What I Do: Programming by Demonstration. The MIT Press.

Czarnecki, K., et S. Helsen (2003). Classification of model transformation approaches.OOSPLA'03 Workshop on Generative Techniques in the Context of Model-Driven Architecture, Anaheim, USA, pp. 65-78.

Egyed, A. (2002). Automated abstraction of class diagrams. ACM Trans. Softw. Eng.Methodol. 11(4): 449-491.

Interactive Objects Software GmbH, Project Technology, Inc. MOF 2.0 Query/Views/Transformations RFP. OMG Document ad/2003-08-11.

Jouault, F., et I. Kurter (2005). Transforming models with ATL. In: Proc. Of the Model Transformations in Practice Workshop at MoDELS 2005, Jamaica.

Kennedy, J., et R.C Eberhart (1995). Particle swarm optimization. In: Proc. IEEE Intl.Conf. on Neural Networks, pp. 1942–1948.

Kessentini, K., H.Sahraoui et M.Boukadoum (2008). Model Transformation as an Optimization Problem. In Proc.MODELS 2008, Vol. 5301 of LNCS. Springer, pp. 159-173.

Kirkpatrick, D.S., Jr. Gelatt, et M. P. Vecchi (1983). Optimization by simulated annealing. *Science*, 220(4598), pp. 671-680.

Kleppe, A., J.Warmer J and W. Bast (2003). MDA Explained. The model driven architecture: practice and promise. Addison-Wesley, pp. 53-69.

Krishnamurthy, R., S.P. Morgan et M.M. Zloof (1983). Query-By-Example: Operations on Piecewise Continuous Data. Proc. 9th International Conference on Very Large Data Bases, October 31 - November 2, 1983, Florence, Italy, pp. 305-308

Muller, P., F.Fleurey, et J. M. Jézéquel (2005). Weaving Executability into Object-Oriented Meta-languages. Proc. of MoDELS'05, Montego Bay, Jamaica, pp 264-278.

Taenzer, G. (2003). AGG: a graph transformation environment for system modeling and validation. In: Proc. Tool Exihibition at Formal Methods 2003, Pisa, Italy, pp. 446-453.

Varro, D. (2006). Model transformation by example. In Proc.MODELS 2006, Vol. 4199 of LNCS. Springer, pp. 410–424.

Varro, D., et A. Pataricza (2004). Generic and meta-transformations for model transformation engineering. UML 2004. LNCS, vol. 3273. Springer, Heidelberg, pp.290-304.

Varro, D., et Z.Balogh (2007). Automating Model Transformation by Example Using Inductive Logic Programming, ACM Symposium on Applied Computing | Model Transformation Track, pp. 978 - 984.

Wimmer, M., M. Strommer, H. Kargl, et G. Kramler (2007).Towards model transformation generation by-example. In Proc. of HICSS-40 Hawaii International Conference on System Sciences. Hawaii, USA, pp. 285-300.

Summary

Most of the contributions in MT so far have been concerned with defining languages to express transformation rules. However, the task of defining, expressing, and maintaining these rules has been at best difficult, and it is often unrealizable, especially for proprietary and non-widely used formalisms. At the same time, in many situations, experts show transformation examples more easily than they can express complete and consistent transformation rules. In other situations, companies have accumulated examples from past experiences. Our work starts from these observations to view the transformation problem as one to solve with insufficient knowledge, i.e. with only examples of source-to-target model transformations. In this context, we use a search-based approach, based on simulated annealing (SA), to automate the transformation process. It consists of finding in the examples combinations of transformation fragments that best cover the source model. The results of validating our approach, with industrial projects, show that the obtained models are comparable to those proposed by experts in industrial organization.