Exploiting Model Transformation Examples for Easy Model Transformation Handling (Learning and Recovery)

Hajer Saada

Académie de Montpellier
Université Montpellier 2
Sciences et Techniques du Languedoc

Ph.D Thesis

présentée au Laboratoire d'Informatique de Robotique
et de Microélectronique de Montpellier pour
obtenir le diplôme de doctorat

par

Hajer Saada

Spécialité : **Informatique**
Formation Doctorale : **Informatique**
École Doctorale : **Information, Structures, Systèmes**

Exploiting Model Transformation Examples for Easy Model Transformation Handling (Learning and Recovery)

Soutenue le 04/12/2013, devant le jury composé de :

Reviewers
Gertrude Kappel, Professeur ..... University of Technology, Vienne, Autriche
Benoit Baudry, Chargé de Recherches HDR, .. INRIA Rennes Bretagne Atlantique, France

Examinator
Marie-Pierre Gervais, Professeur .... Université Paris Ouest Nanterre La Défense , France

Supervisor
Marianne Huchard, Professeur ..... Université Montpellier 2, France

Joint supervisor
Clémentine Nebut , Maitre de Conférences ..... Université Montpellier 2, France
Houari Sahraoui , Professeur ..... Université de Montréal, Canada

To my family

Acknowledgements

The research work presented in this thesis has been performed in the Laboratoire d'Informatique de Robotique et de Microélectronique de Montpellier (LIRMM), University Montpellier 2. This thesis has been done in collaboration between University Montpellier 2 and University of Montreal. My stay in France for pursuing the doctorate degree has been financially supported by the Government of Tunisia.

My first and sincere acknowledge goes to my supervisors Marianne Huchard, Clémentine Nebut and Houari Sahraoui for all I have learned from them and for their continuous help and support in all stages of this thesis. I would like to thank them for encouraging and helping me to shape my interest and ideas, and especially for letting me wide autonomy while providing appropriate advice. It was a real pleasure to work with them.

I would like to express my gratitude to the honorable reviewers, Dr. Gertrude Kappel and Dr. Benoit Baudry for accepting to review my thesis. I am thankful for their valuable comments and remarks. I am grateful to Dr. Marie-Pierre Gervais for her thorough examination of the thesis.

As a member of MaREL team, I got a lot of help and encouragement from all the members of the group. I forward my extreme appreciations to all MaREL members.

My greatest acknowledge goes to my closest friend Mohamed for the continuous help and advice, and for his effort in proof reading the thesis drafts and providing valuable feedback.

Thanks to my supportive friends, Najib, Kaouthar, Nadia, Xavier, Naoufal, Amine, Zak, Younes, Anas, Seza, Hamzeh, Raafat, Thibaut, Petr, Sarra, Azhar, Adel, Julien, Zakaria, and others who made the lab a friendly environment for working.

Thanks to my friends in the university of Tunisia, Jamel, Asma, Siwar, Nesma, Nesrine, Halima, Zeineb, Takwa..

Last but not the least, I would like to thank my mother and my father for always believing in me, for their continuous love and their supports in my decisions. I am also very grateful to my brothers, my sisters, my nieces and nephews.

Hajer Saada
December 2013

Abstract

Model Driven Engineering (MDE) considers models as first class artifacts. Each model conforms to another model, called its metamodel which defines its abstract syntax and its semantics. Various kinds of models are handled successively in an MDE development cycle. They are manipulated using, among others, programs called model transformations. A transformation takes as input a model in a source language and produces a model in a target language. The developers of a transformation must have a strong knowledge about the source and target metamodels which are involved and about the model transformation language. This makes the writing of the model transformation difficult.
In this thesis, we address the problem of assisting the writing of a model transformation and more generally of understanding how a transformation operates. We adhere to the Model Transformation By Example (MTBE) approach, which proposes to create a model transformation using examples of transformation. MTBE allows us to use the concrete syntaxes defined for the metamodels. Hence, the developers do not need in-depth knowledge about the metamodels. In this context, our thesis proposes two contributions. As a first contribution, we define a method to generate operational transformation rules from transformation examples. We extend a previous approach which uses Relational Concept Analysis as a learning technique for obtaining transformation patterns from 1-1 mapping between models. We develop a technique for extracting relevant transformation rules from these transformation patterns and we use the JESS language and engine to make the rules executable. We also study how we better learn transformation rules from examples, using transformation examples separately or by gathering all the examples. The second contribution consists in recovering transformation traces from transformation examples. This trace recovery is useful for several purposes as locating bugs during the execution of transformation programs, or checking the coverage of all input models by a transformation. In our context, we expect also that this trace will provide data for a future model transformation learning technique. We first address the trace recovery problem with examples coming from a transformation program. We propose an approach, based on a multi-objective meta-heuristic, to generate the *many-to-many* mapping between model constructs which correspond to a trace. The fitness functions rely on the lexical and structure similarity between the constructs. We also refine the approach to apply it to the more general problem of model matching.

Keywords: *MDE, model transformation, MTBE, operational rules, model transformation traces, model matching, FCA, JESS, meta-heuristic, genetic algorithm.*

Résumé

L'Ingénierie Dirigée par les Modèles (IDM) est un domaine de recherche en pleine émergence qui considère les modèles comme des éléments de base. Chaque modèle est conforme à un autre modèle, appelé son méta-modèle, qui définit sa syntaxe abstraite et ses concepts. Dans un processus IDM, différents types de modèles sont manipulés par des transformations de modèles. Une transformation génère un modèle dans un langage cible à partir d'un modèle dans un langage source. Pour concevoir une transformation, les développeurs doivent avoir une bonne connaissance des méta-modèles concernés ainsi que des langages de transformation, ce qui rend cette tâche difficile. Dans cette thèse, nous proposons d'assister l'écriture des transformations et plus généralement de comprendre comment une transformation opère. Nous adhérons à l'approche de transformation de modèles par l'exemple qui propose de créer une transformation de modèles à partir d'exemples de transformation. Cela permet d'utiliser la syntaxe concrète définie pour les méta-modèles, et cela évite donc de requérir que les développeurs aient une bonne maîtrise des méta-modèles utilisés. Dans ce contexte, nous proposons deux contributions. La première consiste à définir une méthode pour générer des règles de transformation opérationnelles à partir d'exemples. Nous nous basons sur une approche qui utilise l'Analyse Relationnelle de Concepts (ARC) comme technique d'apprentissage pour obtenir des patrons de transformation à partir d'un appariement de type 1-1 entre les modèles. Nous développons une technique pour extraire des règles de transformation opérationnelles à partir de ces patrons. Ensuite, nous utilisons le langage et le moteur de règles JESS pour exécuter ces règles. Nous étudions aussi comment mieux apprendre des règles de transformations à partir d'exemples, en utilisant séparément chaque exemple ou en réunissant tous les exemples. La deuxième contribution consiste à récupérer les traces de transformation à partir d'exemples de transformation. Ces traces peuvent être utilisées par exemple pour localiser des erreurs durant l'exécution des programmes de transformation ou vérifier la couverture de tous les modèles d'entrée par une transformation. Dans notre contexte, nous supposons que ces traces vont servir pour un futur apprentissage des règles de transformation. Nous traitons tout d'abord le problème de récupération des traces avec des exemples provenant d'un programme de transformation. Nous proposons une approche basée sur une méta-heuristique multi-objectifs pour générer des traces sous forme d'appariement de type n-m entre des éléments de modèles. La fonction objectif s'appuie sur une similarité lexicale et structurelle entre ces éléments. Une extension de cette méthode est proposée pour traiter le problème plus général de l'appariement entre modèles.

Mots clefs : *IDM, transformation de modèles, l'approche par exemple, règles opérationelles, traces de transformation, appariement des modèles, AFC, JESS, meta-heuristique, algorithmes génétiques*

Contents

Acronyms

MDE: Model Driven Engineering

UML: Unified Modeling Language

MDA: Model Driven Architecture

MT: Model Transformation

MTBE: Model Transformation By Example

OMG: Object Management Group

MOF: Meta-Object Facility

CIM: Computation Independent Model

PIM: Platform Independent Model

PSM: Platform Specific Model

MT: Model Transformation

RS: Relational Schema

CD: Class Diagram

API: Application Programming Interface

VB: Visual Basic

LHS: Left-Hand Side

RHS: Right-Hand Side

OCL: Object Constraint Language

MTBE: Model Transformation By Example

ILP: Inductive Logic Programming

RCA: Relational Concept Analysis

MTBD: Model Transformation By Demonstration

SBSE: Search Based Software Engineering

GA: Genetic Algorithms

SA: Simulated Annealing

PbE: Programming by Example

PSO: Particle Swarm Optimization

SA: Simulated Annealing

EA: Evolutionary Algorithms

Introduction

Research Context

Model Driven Engineering (MDE) [Schmidt, 2006] is a software development methodology which focuses on creating and exploiting models. MDE involves different principles including OMG's Model Driven Architecture (MDA) [Soley and the OMG Staff Strategy Group, 2000], that is based on the separation of business logic from platforms technology. One basic principle of MDE is "*everything is a model*". MDE provides supports for creating and editing models, transforming models to other models or programs, model-based testing, etc. The use of models helps to improve the productivity by maximizing compatibility between systems (via model reuse), the simplification of the process design and the communication between developers (via standardization) [Schmidt, 2006]. Each model conforms to another model, called its metamodel, which defines the structure, *i.e.*, concepts and their relationships that can be used to compose a model.

Besides models, model transformations represent another crucial element of MDE. They allow the definition of mapping between models. Transformations aim to automate the translation within and between different modeling languages, *e.g.*, the transformation of a design model to a program in C#. Transformations are defined at the metamodel level, and are applied at the model level. They transform a model in a source language to a model in a target language. For this end, models must be written in a modeling language (*e.g.*, the Unified Modeling Language - UML). A modeling language corresponds also to a metamodel (*e.g.*, UML metamodel) which defines the language concepts. Based on the modeling language, we can distinguish between two types of transformation: exogenous transformation and endogenous transformation [Mens and Gorp, 2006].

An exogenous transformation is a transformation between source and target models expressed using different languages, *e.g.*, the transformation of a UML class diagram to a Java code. Endogenous transformation is a transformation between source and target models expressed in the same language, *e.g.*, for maintenance activities (refactoring or optimization). It rewrites the input model to produce the output model for renaming,

adding or deleting some of its constructs. An endogenous transformation serves to change the structure of software without changing its behavior.

Thesis Problem

Several languages have been proposed to write model transformations. Although most of these languages are able to implement complex transformation problems, it may be difficult to use them for individuals who are not expert on specific transformation languages. In addition, the solution domain of a model transformation (abstract syntax of modeling languages) can be different from the problem domain (concrete syntax). Domain experts often give more easily transformation examples than complete and consistent transformation rules [Kessentini, 2010].

In this thesis, we stand for the idea that the high quality and the multiplicity of transformation examples may be exploited to assist the designers to write model transformation. Moreover, we believe that this assistance must be based on two main focuses:

1. An assistance for the definition of operational transformation rules which constitute the body of the transformation.
2. An assistance for the extraction of mapping links between source and target model of the transformation to infer a trace of the transformation.

Contributions

Our contribution deals with the assistance of model transformation by example design. More specifically, we provided solutions for two specific problems: 1) the definition of operational rules which constitute the transformation and 2) the extraction of links between the source and target models of the transformation.

Generation of operational transformation rules: We propose an approach to generate model transformation rules from transformation examples. Examples are given by experts. An example consists of a source model, a target model and the mapping between the two models. Our work is a continuation of the approach of *Dolques et al.* [Dolques *et al.*, 2009], that uses Relational Concept Analysis [Huchard *et al.*, 2007], as a learning technique, to derive transformation patterns organized in a lattice. Those patterns are abstract and they cannot be executed. Moreover, some of them are not relevant. Thus, we define in this first contribution an approach to analyze those patterns and select the most pertinent ones.

Furthermore, we propose a method to transform them into operational transformation rules written for the Java Expert System Shell (Jess) rule engine [Hill, 2003].
We also study how we better learn transformation rules using transformation examples. Thus, we perform experimentations to compare two learning strategies. While the first strategy uses examples separately to generate rules, the second one gathers all examples together and generates rules.

Recovering model transformation traces: As a second contribution of this thesis, we propose to generate fine-grained mappings from examples issued from a transformation. The transformation may be done or edited manually by experts. Thus recovering traces between models may be very essential in a development cycle to track their changes. For this end, we define an approach to derive *many-to-many* mapping between model elements. Our approach takes as input a source model in the form of a set of fragments (fragments are defined using the source metamodel cardinalities and Object Constraint Language (OCL) constraints), and a target model. It searches for each source a set of target elements by maximizing the lexical and structural similarities between them. Hence, it may lead to a huge number of possible combinations. Thus, NSGA-II, a metaheuristic method , is used to solve this problem.
The problem of transformation traces recovery may be similar to the model matching one. In the first context, source and target models are issued from a transformation program. Hence, it is easy to discover lexical similarities between the models constructs. However, if the transformation between source and target models is done manually, this may cause lexical and structural variations between the two models. Thus, the problem may be addressed in a model matching context.
We have consequently defined a variant of the model transformation recovery approach to deal with the model matching problem. The lexical similarity function is performed using a natural language process. Our approach produces matchings of type *many-to-many* between models.

Thesis Outline

This dissertation is organized as follows:

Chapter 1 introduces the techniques and tools used in this work. Definitions about Formal Concept Analysis and Relational Concept Analysis are given. Then, we present the

JESS rule engine. Finally, an overview on metaheuristic methods is presented and more details are given about NSGA-II, which is used in our second contribution.

Chapter 2 provides a literature review of the state of the art in model driven engineering, model transformation, model transformation traceability and search-based software engineering.

Chapter 3 reports our contribution for generating operational transformation rules from examples. An overview of the approach is given including the application of RCA to obtain the patterns, the selection of relevant patterns and the transformation of patterns into operational rules. Then, we explain how to write and execute model transformations with the Jess rule engine. Experimental evaluations are also presented to compare two strategies of model transformation learning. The first strategy consists in learning model transformations separately from different examples and the second one consists in gathering examples to learn transformation rules.

Chapter 4 presents our approach, which is based on metaheuristic methods, to address the transformation trace recovery problem as an optimization problem. After the problem definition, we describe how to adapt, a chosen meta-heuristic, NSGA-II to the recovery of traces from transformation examples. Then, we extend this approach to the model matching problem. Experimental evaluations are presented to validate our approaches.

Finally, we conclude this dissertation and we highlight some future directions of research.

1 Preliminaries

In this chapter, we introduce the techniques and tools used in this work. We introduce the definitions needed to understand Formal Concept Analysis and Relational Concept Analysis which will be used for transformation pattern learning. Then, we present the Java Expert System Shell rule engine which helps us making operational the transformation rules. Finally, a summary of the existing meta-heuristic methods is presented and more details are given about NSGA-II, the meta-heuristic we have chosen for addressing the problem of model transformation trace recovery.

1.1 Formal Concept Analysis and Relational Concept Analysis

Formal Concept Analysis (FCA) [Ganter and Wille, 1999] is a mathematical theory, based on lattice theory, which is used for machine learning, data mining, or knowledge structuring, as it groups entities described by characteristics into concepts, ordered in a lattice structure. Relational Concept Analysis (RCA) [Huchard *et al.*, 2007] is an extension of FCA to relational data. While FCA produces a single classification, using a formal context, RCA computes several connected classifications (lattices), using a relational context family.

Definition 1.1 (Formal context)**:** *A formal context is a triple* $K = (O, A, R)$ *where* O *and* A *are sets (objects and attributes, respectively) and* R *is a binary relation, i.e.,* $R \subseteq O \times A$.

Table 1.1 presents a formal context where several animals are described by their characteristics. A formal concept is a pair (E, I) composed of an object set $E \subseteq O$ and their shared attribute set $I \subseteq A$.
$E = \{o \in O | \forall a \in I, (o, a) \in R\}$ is the extent of the concept, while $I = \{a \in A | \forall o \in E, (o, a) \in R\}$ is the intent of the concept. For example, $(\{flamingo, chicken\}\{flying, feathered\})$ is a concept of our example.

	flying	nocturnal	feathered	migratory	with_crest	with_membrane
flying squirrel	×					×
bat	×	×				×
ostrich			×			
flamingo	×		×	×		
chicken	×		×		×	

Table 1.1 – A formal context for describing animals

Given a formal context $K = (O, A, R)$, and two formal concepts $C_1 = (E_1, I_1)$ and $C_2 = (E_2, I_2)$ of K, the concept specialization order $\leq_s$ is defined as follows: $C_1 \leq_s C_2$ if and only if $E_1 \subseteq E_2$ (and equivalently $I_2 \subseteq I_1$). C_1 is called a sub-concept of C_2. C_2 is called a super-concept of C_1. For example, $(\{flamingo, chicken\}\{flying, feathered\})$ is a sub-concept of $(\{flamingo, chicken, ostrich\}\{feathered\})$.

Let $\mathcal{C}_K$ be the set of all concepts of a formal context K. This set of concepts provided with the specialization order $(\mathcal{C}_K, \leq_s)$ has a lattice structure, and is called the concept lattice associated with K. Figure 1.1 shows the Hasse diagram of the concept lattice structuring our animals. In this diagram, extents and intents are presented in a simplified form: removing up-down inherited attributes and down-up inherited objects. It describe several concepts, *e.g.*, Concept_4, which groups the flying and feathered animals (flamingo, chicken) or Concept_5 which groups the feathered animals (flamingo, chicken and ostrich). The lattice highlights the structure of data: *e.g.* the group of flying animals is distinct of the group of feathered animals, implication rules can be extracted, like being with-crest implies being feathered (because Concept_7 is a subconcept of Concept_5).

While FCA builds upon a formal context, with a unique object set and a unique attribute set, RCA elaborates upon a relational context family, close to an entity-relationship model.

Definition 1.2 (Relational Context Family)**:** *A Relational Context Family $\mathcal{F}$ is a pair (K, R) with:*

- *K is a set of object-attribute contexts $K_i = (O_i, A_i, I_i)$, $i \in 1..n$ where O_i is a set of objects, A_i is a set of attributes and $I_i \subseteq O_i \times A_i$.*
- *R is a set of object-object contexts $R_j = (O_k, O_l, I_j)$, where (O_k, O_l) are the object sets of formal contexts $(K_k, K_l) \in K^2$, $I_j \subseteq O_k \times O_l$ and K_k is the* source/domain context*, K_l is the* target/range context*.*

One main principle of RCA consists in integrating the relations between objects as *relational attributes* that link objects of an object-attribute context to concepts formed on another object-attribute context. Given an object-object context $R_j = (O_k, O_l, I_j)$, there are different notable schemas between an object of domain O_k and concepts formed on O_l

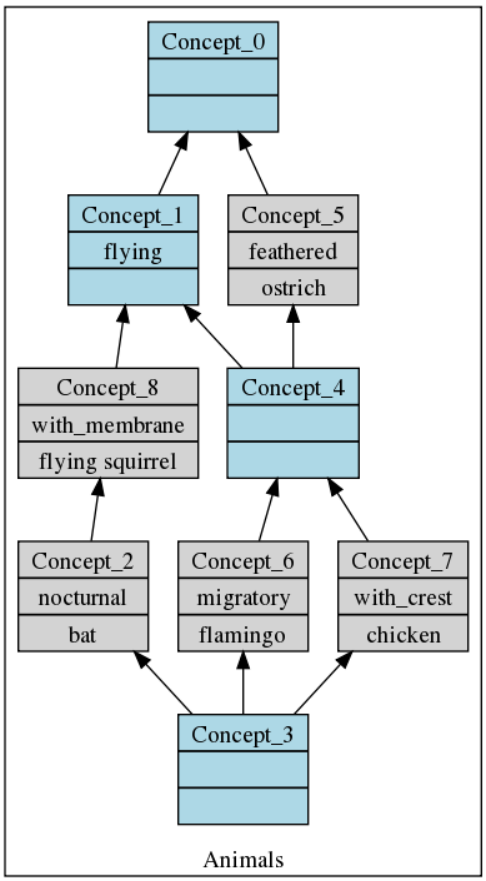

Figure 1.1 – The concept lattice for the formal context of Table 1.1

to obtain a relational attribute. Relational scaling is the process by which these links are established between objects and concepts.

There are several scaling operators, the most used are:

- **Existential (∃)**: an object is linked (by R_j) to at least one object of the extent of a concept: $\exists(R_j(o), Extent(C))$ is true iff $R_j(o) \cap Extent(C) \neq \emptyset$.
- **Universal strict (∀∃)**: an object is linked (by R_j) only to objects of the extent of a concept: $\forall\exists(R_j(o), Extent(C))$ is true iff $R_j(o) \subseteq Extent(C) \wedge \exists x \in R_j(o), x \in Extent(C)$

The RCA process is an iterative process which is sketched below.

RCA initialization step: Build, for i in $1..n$, $\mathbf{L}^0[i]$ the concept lattice of the context $\mathcal{K}_i$.

RCA Step p:

- apply the relational scaling to all object-object contexts using the lattices of step $p-1$ and the chosen scaling operators.

  For $R_j = (O_k, O_l, I_j)$, and the scaling operator S, this produces the scaled context $R_j^* = (O_k, A, I_j)$ where:

Pizza	thin	thick	calzone
okonomi			×
alberginia		×	
margherita	×		
languedoc	×		
four-cheeses	×		
three-cheeses	×		
frutti-di-mare	×		
quebec		×	
regina	×		
hawai		×	
lorraine	×		
kebab			×

Ingredient	fruit-vegetable	meat	fish	dairy	cereal-leguminous	veg-oil
tomato-sauce	×					
cream				×		
tomato	×					
basilic	×					
olive	×					
olive oil						×
soy	×					
mushroom	×					
eggplant	×					
onion	×					
pepper	×					
ananas	×					
mozza				×		
goat-cheese				×		
emmental				×		
fourme-ambert				×		
squid			×			
shrimp			×			
mussels			×			
ham		×				
bacon		×				
chicken		×				
maple-sirup	×					
corn					×	

Table 1.2 – Relational Context Family (RCF) / object-attributes contexts

has-topping	tomato-sauce	cream	tomato	basilic	olive	olive oil	soy	mushroom	eggplant	onion	pepper	ananas
okonomi	×					×	×	×				
alberginia	×					×	×		×	×		
margherita	×		×	×	×	×						
languedoc	×		×	×	×	×				×	×	
four-cheeses		×										
three-cheeses		×										
frutti-di-mare	×				×	×						
quebec	×											
regina	×							×				
hawai	×											×
lorraine		×								×		
kebab	×		×		×					×		

Table 1.3 – Relational Context Family (RCF) / object-object context / part 1

  - A is a set of relational attributes $S\ R_j.C$, where C is one concept of $\mathbf{L}^{p-1}[l]$, the lattice built on objects of O_l at step $p-1$
  - I_j contains $(o, S\ R_j.C)$ iff $S(R_j(o), Extent(C))$ is true.
- concatenate $\mathcal{K}_i$ with all the scaled relations R_j^* whose domain is O_i, this gives an extended context $\mathcal{K}_i^*$
- update lattices of step $p-1$ to build, for i in $1..n$, the lattice $\mathbf{L}^p[i]$ associated with the context $\mathcal{K}_i^*$

The process stops when an iteration does not add any new concept and we consider the last lattice family obtained as the output of the process.

Table 1.2, Table 1.3 and Table 1.4 present a relational context family. In Table 1.2, the left-

has-topping	mozza	goat-cheese	emmental	fourme-ambert	squid	shrimp	mussels	ham	bacon	chicken	maple-sirup	corn
okonomi												
alberginia												
margherita	×											
languedoc	×											
four-cheeses	×	×	×	×								
three-cheeses	×	×	×									
frutti-di-mare	×				×	×	×					
quebec	×							×			×	×
regina	×								×			
hawai	×							×				
lorraine			×						×			
kebab			×							×		

Table 1.4 – Relational Context Family (RCF) / object-object context / part 2

Pizza	thin	thick	calzone
okonomi			×
alberginia		×	
margherita	×		
languedoc	×		
four-cheeses	×		
three-cheeses	×		
frutti-di-mare	×		
quebec		×	
regina	×		
hawai		×	
lorraine	×		
kebab			×

has-topping	∃has-topping. Concept_7	∃has-topping. Concept_5	∃has-topping. Concept_6	∃has-topping. Concept_8	∃has-topping. Concept_9	∃has-topping. Concept_10	∃has-topping. Concept_11	∃has-topping. Concept_12
okonomi		x	x					x
alberginia		x	x					x
margherita		x	x			x		x
languedoc		x	x			x		x
four-cheeses		x				x		
three-cheeses		x				x		
frutti-di-mare		x	x		x	x		x
quebec		x	x	x		x	x	
regina		x	x	x		x		
hawai		x	x	x		x		
lorraine		x	x	x		x		
kebab		x	x	x		x		

Table 1.5 – Existential relational attributes

hand table presents an object-attribute context describing pizzas and the right-hand table presents an object-attribute context describing ingredients. Table 1.3 and Table 1.4 present an object-object context where the pizzas are described by their toppings (ingredients).

After the existential scaling, we obtain the table, in the right hand of Table 1.5, in which for example *okonomi* is associated to $\exists\ has - topping.Concept_6$. They are associated because in the relation *has-topping*, *okonomi* is associated with *mushroom*, which belongs to the extent of $Concept_6$ (Figure 1.2). Thus *okonomi* is associated with at least one element of the extent of $Concept_6$. This table is concatenated to the Pizza table, recalled in the left hand of Table 1.5, to obtain the new Pizza lattice of Figure 1.2. For example, *Concept*_21 groups pizzas with at least one dairy topping (dairy ingredients are grouped in *Concept*_10). This is represented by the relational attribute $\exists\ has - topping : Concept\_10$ which is owned by pizzas from the extent of *Concept*_21. *Concept*_18 contains pizzas with at least one meat topping (relational attribute $has - topping : Concept\_8$). Implication rules may also be

mined from this set of lattices. As *Concept*_18 is a sub-concept of *Concept*_21, we have $\exists$ *has* − *topping* : *Concept*_8 implies $\exists$ *has* − *topping* : *Concept*_10. This is interpreted by: having at least one meat topping implies having at least one dairy topping.

To conclude, while FCA is able to reveal structures in a single context, by building classifications and extracting implications rules from a single object set, RCA is able to build classifications and extracting implication rules from several object sets and links between these objects. Here we gave a simple example composed of only one relation, but RCA is able to deal with complex relational schemas, with several relations, and even cyclic relational schemas. The whole theoretical framework, with an analytical definition and a discussion about the convergence of the process are described in [Hacene *et al.*, 2013].

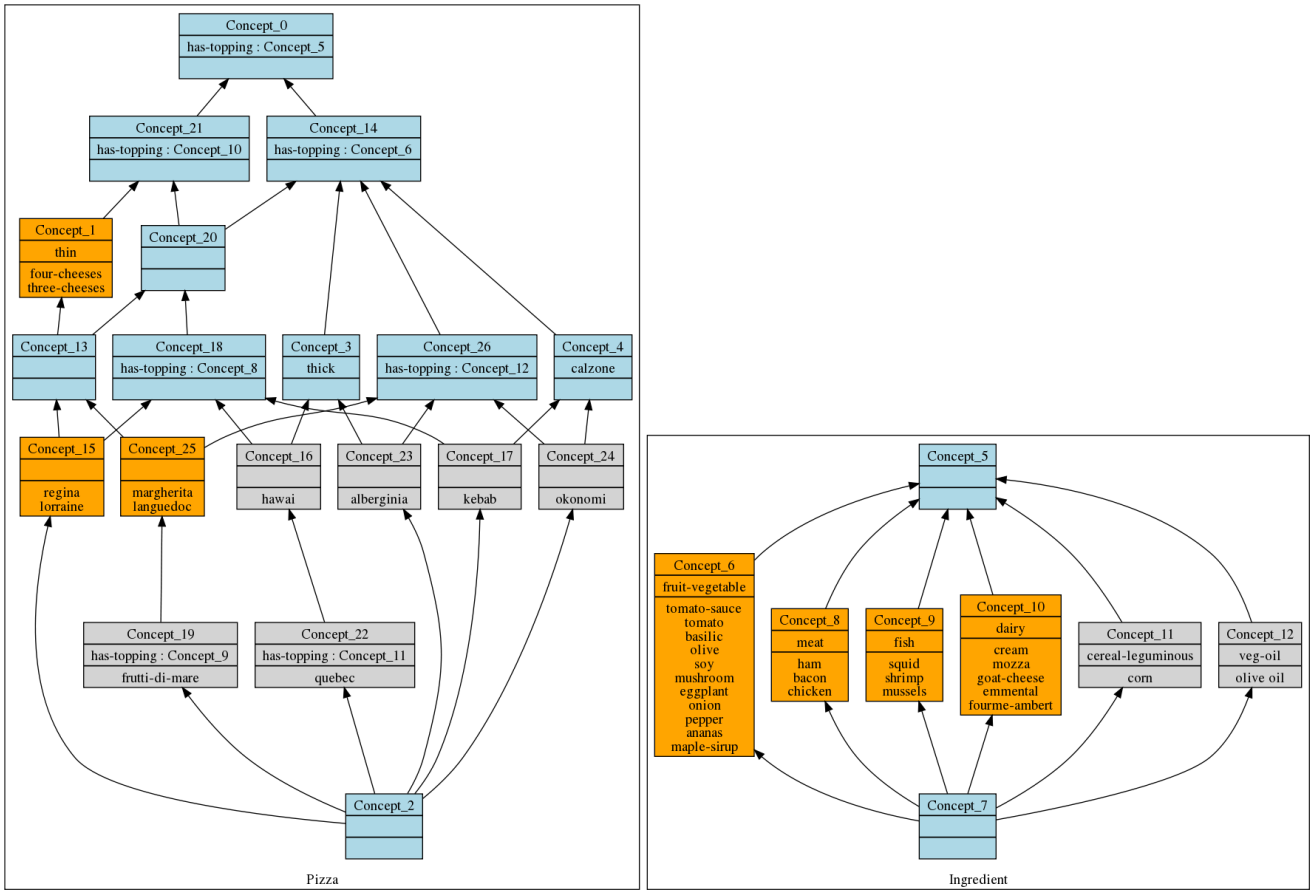

Figure 1.2 – The obtained lattices for the pizza example

1.2 Java Expert System Shell

Java Expert System Shell (Jess) [Hill, 2003] is a rule engine integrated in the Java platform. Java code can be referred by Jess code [Daniele, 2006]. With Jess, we can create Java

objects, implement Java interfaces, and call Java objects from its Java scripting environment. Despite this, Jess is mainly a declarative language.

A Jess program is usually composed of *facts* and *rules*. Facts encode data, while rules, activated by pattern matching, encode behavior [Hill, 2003]. A rule contains conditions, called left-hand-side (LHS), and actions, called right-hand-side (RHS). When the condition part is satisfied, the action part is executed. Conditions mainly test the presence of facts, whereas actions produce facts. Syntactically, a Jess rule is written as follows:

IF< (fact1)(fact2)...(factN) > THEN <(action1)(action2)...(actionM)>

The following example describes a very simple Jess rule which displays the name of each person who has a name.

```
1 (defrule welcome
2   (Person (firstname ?name))
3  =>
4   (printout t "Hello" ?name "!!!" crlf)
5 )
```

The conditions in LHS and facts conform to a *template*. A template in Jess is similar to a class in Java. It defines a fact type. A template has a name and a set of slots. A fact, *i.e.* a template instance, has specific values for these slots. The example below shows the declaration of *Person* template:

```
1 (deftemplate Person (slot firstname))
```

This example declares a template named *Person* with a property *firstname*. To instantiate a person fact, we use the command *assert*:

```
1 (assert (Person (firstname Peter)))
```

Figure 1.3 presents a simplified Jess metamodel. As mentioned above, a Jess model is composed of templates, facts and rules. A rule is composed of two expressions which present respectively the premise (LHS) and the conclusion (RHS). An expression contains a list of facts with conditions and tests defined on the facts themselves.

1.3 The NSGA-II Algorithm

During the past two decades, evolutionary algorithms (EAs) have gained popularity in dealing with software engineering tasks that could be modeled as optimization problems. These problems are generally too complex to be solved using deterministic methods.

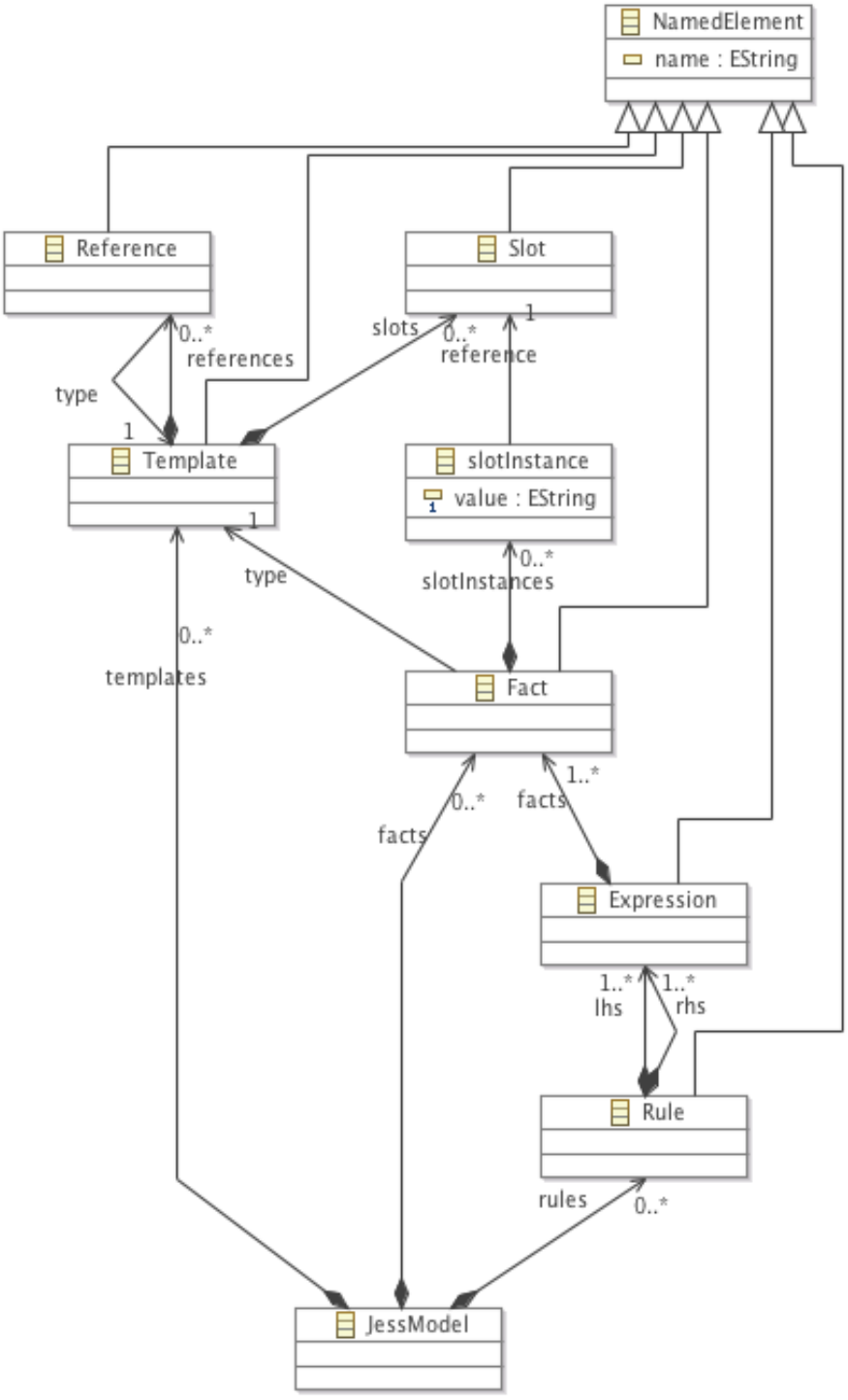

Figure 1.3 – Jess metamodel

EAs popularity could be explained by many reasons such as their simplicity, their applicability to a wide range of problems, as well as their ability to handle single and multiple objectives [Harman, 2011].

For problems with multiple (possibly conflicting) objectives, like the one studied in this

chapter, it is usually difficult to find a single optimal solution. Such kind of problems gives rise to a whole set of solutions, known as Pareto-optimal solutions [Deb *et al.*, 2002].

In this context, a number of multi-objective EAs have been proposed ([Horn *et al.*, 1994; Zitzler and Thiele, 1999; Knowles and Corne, 1999; Deb *et al.*, 2002]). Among those algorithms, the non-dominated sorting genetic algorithm (NSGA-II), proposed in [Deb *et al.*, 2002], is the one that is the most applied in the Search Based Software Engineering (SBSE) community [Harman *et al.*, 2012]. We decided to use it in this work as it allows to easily model the trace recovery as a multiobjective optimization problem.

NSGA-II procedure. The evolution of the population during an iteration of the NSGA-II procedure is presented in Figure 1.4 which is taken from the original paper. First, an initial population P_0 of N solutions is created. The individuals of P_0 are sorted based on the non-domination. Non-dominated individuals, corresponding to the best known solutions (with regard to at least one objective) at the current step, are grouped in the first non-dominated front (rank 1). Discarding the individuals of the first front, the current non-dominated individuals form the second non-dominated front (rank 2), and so on. Diversity is preserved thanks to a crowding distance which is calculated for each solution [Laumanns *et al.*, 2002]. Finally, a binary tournament selection operator, which is based on the crowding distance, selects the best solutions. At step t, an offspring population Q_t of size N is created using selection, crossover and mutation operators. Populations P_t and Q_t are combined to form the population R_t. From R_t, the best individuals in terms of non-dominance and diversity are kept to form P_{t+1}. Then those steps are repeated till some termination criteria are satisfied.

Fast non-dominated sorting principle. In order to identify solutions of the first non-dominated front in a population of size N, NSGA-II calculates first, for each solution p: 1) the domination count n_p, *i.e.* the number of solutions which dominate the solution p, and 2) S_p, a set of solutions that the solution p dominates. A solution s_1 dominates another solution s_2 if: (i) s_1 is no worse than s_2 in all objectives, and (ii) s_1 is strictly better than s_2 in at least one objective.

In the first non-dominated front, there are the solutions which have their domination count equal to zero. For each solution p with $n_p = 0$, each member $q \in S_p$ is visited, and its domination count is reduced by one. Then, if the domination count of a member q becomes zero, q is put in a separate list Q. These members belong to the second non-dominated front. Then, this procedure is continued with the members of Q to identify the third front.

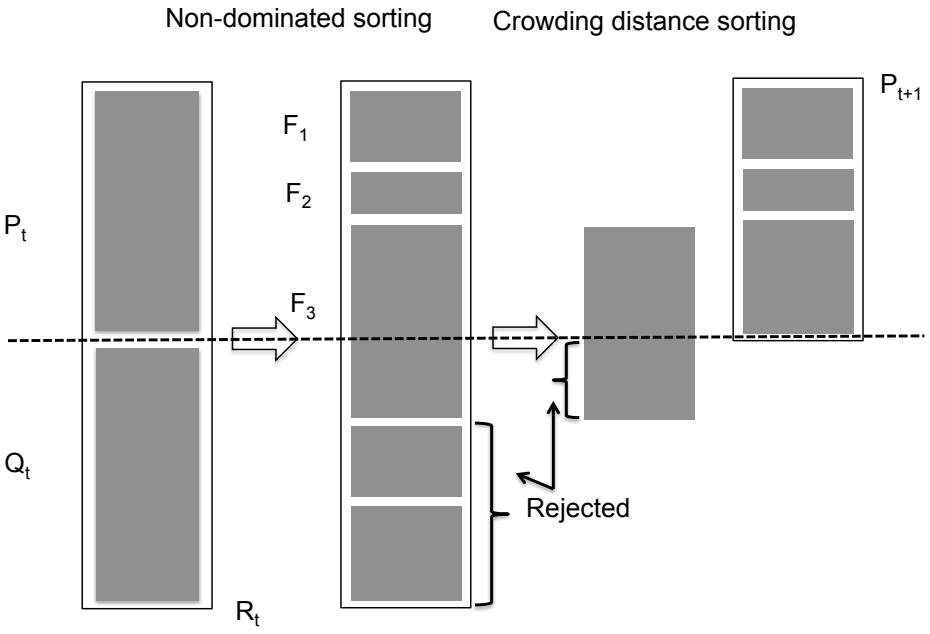

Figure 1.4 – NSGA-II main (Deb et Al, 2002).

If N_{obj} is the number of objectives and N is the size of the population, this algorithm has a low time complexity of $O(N_{obj}N^2)$ compared to the previous algorithms, which require $O(N_{obj}N^3)$.

Diversity preservation. The parent and offspring populations (each of size N) are combined and evaluated to choose the best solutions. Then, the N best solutions are selected to form the next populations. As mentioned earlier, the solutions of the first front are selected and if there is a room, those of the second front are included, and so on. When the number of solutions in the already selected fronts is less than N, and including the next front results in exceeding N, the solutions of this last front have to be ranked, and only part of them is selected to complete the count. This is done by selecting solutions that are maximally apart from their neighbors according to the crowding distance. This is measured as the distance of the biggest cuboid containing the two neighboring solutions of the same non-dominating front in the objective space. The goal of the crowding-distance-based ranking is to increase the diversity of solutions that are injected into the next population. In Chapter 4, we will give more details about implementation of this algorithm for our problem.

State Of The Art

2

Contents

This chapter provides an overview of research work related to this thesis. The approaches proposed in this thesis focus on six connected research areas: (1) model driven engineering, (2) model transformation, (3) generation of model transformation, (4) generation of model transformation by example, (5) model transformation traceability and (6) search-based software engineering. In this chapter, we give a survey on the existing work in these areas to present their principles and identify their limitations addressed by our work.

The remainder of this chapter is structured as follows: Section 2.1 presents the context of model driven engineering (MDE) with some definitions. Section 2.2 defines model transformation (MT) and Section 2.3 presents how to generate MT using the meta-model matching and the by-example approaches. Section 2.4 summarizes the existing work in the field of model transformation traceability and finally Section 2.5 positions our research work in search-based software engineering (SBSE).

2.1 Model Driven Engineering

MDE is a technique which aims to reduce the complexity of development and management of modern software applications through the exploitation of models. Despite it is a quite methodology, it gains more and more interest from the industry, which considers it

as a possible solution for the ever growing quality factors, performances, and maintainability. It allows models to be considered as data and then used as first class entities of development process.

According to *Rothenberg* [Rothenberg, 1989]: "Modeling in its broadest sense is the cost-effective use of something in place of something else for some cognitive purpose. It allows us to use something that is simpler, safer, or cheaper than reality instead of reality for some purpose. A model represents reality for the given purpose; the model is an abstraction of reality in the sense that it cannot represent all aspects of reality. This allows us to deal with the world in a simplified manner, avoiding the complexity, danger, and irreversibility of reality".

Although it was written several years before the creation of MDE, this definition perfectly describes the principals and the utility of modeling. A model is an abstraction. It is a simplification of a system that is sufficient to understand the modeled system. Models simplify the management of systems by presenting the requirements and the problems on different views. For instance, a class diagram facilitates the comprehension of an application independently from its platform.

Through this definition, we can have a general idea about the principals and the utility of models. In the following, we will focus on the meanings of models in the context of MDE.

Definition 2.1: *"A model is a description of (part of) a system written in a well-defined language" [Kleppe et al., 2003]. For example, a legend of a map provides a model for this map. The map can also be seen as a model of a region.*

According to the definition, the notion of model explicitly makes reference to the notion of well-defined language which defines the language concepts of a model: such a language is defined by a meta-model.

Definition 2.2: *"A meta-model is a model that defines the language for expressing a model" [Kleppe et al., 2003]. To handle a model, which is the goal of MDE, the language of the model must be defined. The models written with this language are said to conform to a meta-model. A meta-model is also considered as a model. It conforms to a meta-model: the meta-meta-model.*

Definition 2.3 (A meta-meta-model)**:** *A meta-meta-model is a model that defines the language for expressing the meta-modeling languages. A meta-meta-model may conform to itself. Thus, each modeling platform has a meta-meta-model, e.g, Ecore [Frank, 2004] is the meta-meta-model of Eclipse, or the meta-object family (MOF) [OMG, 2006] is the meta-model defined by the Object Management Group (OMG), etc.*

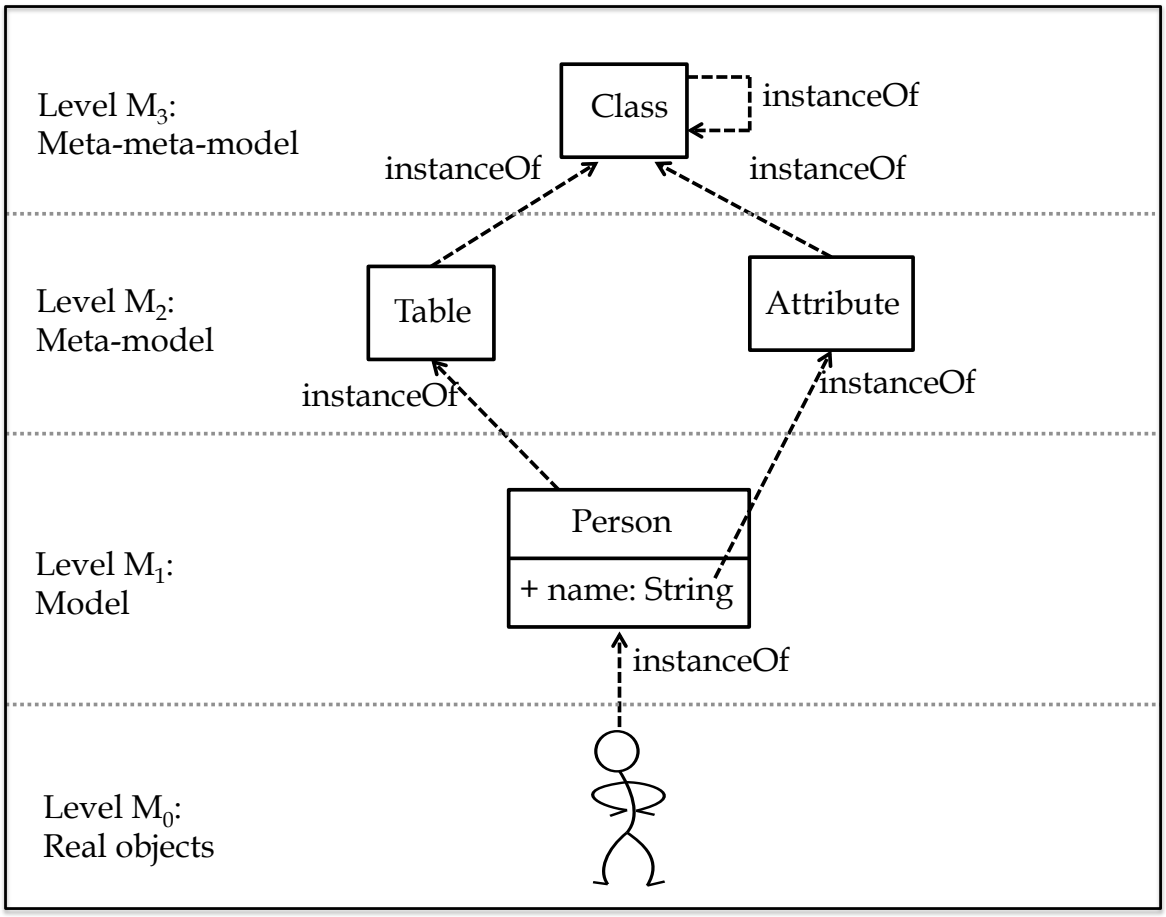

Figure 2.1 – Modeling in MDE

Figure 2.1 shows an example of modeling. Level M_0 consists on the real objects (a person). Level M_1 contains the representation which describes the concept of *Person* with its attribute *name*. The meta-model of this representation is shown at level M_2. It describes the concepts used in M_1 (Table, Attribute). These concepts are in turn defined at level M_3 which presents the meta-meta-model.

MDE gives models a predominating role in the software development process. Models are written in conformity with meta-models which capture the concepts of the modeling language. For example, the UML class diagrams define the concepts of class and attribute, the relational schema models define the concepts of table and column, etc.

Model Driven Architecture (MDA) [Soley and the OMG Staff Strategy Group, 2000] is the well known initiative of the OMG in this domain. It is sometimes viewed as a restriction of MDE to the languages introduced by the OMG. It also comes with a development methodology. Indeed, MDA advocates the construction of three models of a system:

1. Computation Independent Model (CIM): The goal of the analysis phase is to produce

the CIM. It presents what the system is expected to do. It hides all information related to the technology used for the system implementation.

2. Platform Independent Model (PIM): The goal of the design phase is to produce the PIM. It presents the operational view of the system independently from the platform. It defines a set of services to abstract all technical details. A PIM can be mapped to one or more platforms.

3. Platform Specific Model (PSM): The goal of the implementation phase is to produce the PSM. It combines the PIM with the specific details of the platform.

In an MDE process, models play an important role. To ensure the productivity of those models, model transformation (MT) is considered as a central concept. It provides mechanism for automating the manipulation of models.

2.2 Model Transformation

In [Kleppe *et al.*, 2003], the authors provide the following definitions of MT:

Definition 2.4: *"A transformation is the automatic generation of a target model from a source model according to a transformation definition".*

Definition 2.5: *"A transformation definition is a set of transformation rules that together describe how a model in a source language can be transformed into a model in a target language".*

Definition 2.6: *"A transformation rule is a description of how one or more constructs in a source language can be transformed into one or more constructs in a target language".*

The basic idea of MT is presented in Figure 2.2 where a model transformation program MT_p takes as input a source model M_s and produces a target model M_t. Being models, M_s and M_t conform to meta-models MM_s and MM_s. MT_p is composed of a set of rules which have a complete knowledge about MM_s and MM_s. It has a meta-meta-model MMT_p that defines the used transformation language.

2.2.1 Model Transformation Classification

Mens and *Gorp* [Mens and Gorp, 2006] extend the definition of MT by allowing several models as input and/or output: "Model transformation is the automatic generation of one or multiple target models from one or multiple source models according to a transforma-

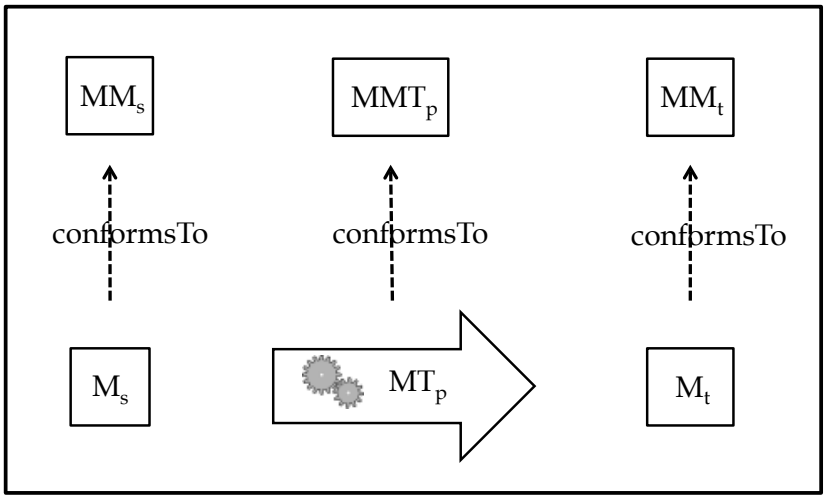

Figure 2.2 – Model Transformation process

tion definition". They give the example of a transformation that takes a PIM and transforms it into a number of PSMs. They classify MTs according to different criteria:

- ***Exogenous versus endogenous transformations***: We distinguish between two MT categories: (1) exogenous transformations in which the source and target models are expressed using different languages, and (2) endogenous transformations in which the source and target models are expressed in the same language. In exogenous transformations, the entire source model elements must be transformed to their equivalents in the target model, e.g., the transformation of a relational schema (RS) model into an UML class diagram (CD). However, there are just a few endogenous transformations, *e.g.*, for refactoring or improving performance of a code.
- ***Horizontal versus vertical transformations***: An horizontal transformation is a transformation where the source and target models are at the same abstraction level. For instance, refactoring (endogenous transformation) is considered as an horizontal transformation. A refinement of models (endogenous transformation) is considered as a vertical transformation because the source and target models reside at different abstraction levels.
- ***Syntactical versus semantical transformations***: A syntactical transformation transforms the syntax of the model, *e.g.*, the transformation of a concrete syntax into an abstract syntax. A more complex transformation, such as a refactoring, is considered as a semantical transformation.
- ***Technical space***: The technical space contains the concepts associated to a technology. It corresponds to the language used to present models. A technical space is deter-

mined in the meta-meta-model level. As an example, we note the MDA technical space defined by the OMG.

Several MT approaches have been proposed in the literature. *Czarnecki* and *Helsen* [Czarnecki and Helsen, 2006] classify these approaches as described in the following:

- ***Direct-Manipulation Approach***: It offers an internal model and users can manipulate the representation using any Application Programming Interface (API). It is usually manipulated as an object-oriented framework, which may also provide some minimal infrastructures. Users must implement transformation rules, tracing and scheduling in a programming language such as Java or Visual basic (VB). Examples of used tools in direct manipulation approaches are: Rational Rose, Rational XDE and Builder Object Network.
- ***Operational Approach***: This approach is similar to direct manipulation, but it offers more dedicated support for MT. A typical solution in this category is to extend the utilized meta-modeling formalism with facilities for expressing computations. An example would be to extend a query language such as OCL with imperative constructs. Kermeta [Muller *et al.*, 2005], QVT Operational mappings [OMG, 2002], MTL [Vojtisek and Jezequel, 2004] and XMF-mosaic [Clark *et al.*, 2004] belong to the operational category.
- ***Structural Approach***: In this category of approaches, a first phase consists in creating a hierarchal structure for the target model. Then, the second phase generates target elements for each source element and sets the attributes and the references in the target model. *OptimalJ* is an example that supports this approach.
- ***Template-Based Approach***: This approach uses template to generate code. A template consists of rules which are mapped on source model. Templates are expressed in the concrete syntax of the target model. They contain embedded metacode which have the form of annotations on model elements. In [Czarnecki, 2005], an example of template approach is given by *Czarnecki* and *Antkiewicz*. In this example, a template of an UML model is creating by using conditions and expressions to annotate model elements.
- ***Relational Approach***: It is a declarative approach. It focuses on mathematical relations and on source and target models relationships [Akehurst and Kent, 2002]. Relationships are specified using predicates and constraints. Operational approaches are bidirectional due to their mathematical foundation. They provide also backtracking. Most of them require a strict separation between source and target models.

Examples of relational approaches are ATL [Jouault *et al.*, 2008] and QVT Relations [OMG, 2005].

- ***Graph-Transformation-Based Approach***: This category is based on the theoretical work on graph transformations. The transformation rules consist of LHS (Left-Hand Side) and RHS (Right-Hand Side) graph patterns. The LHS pattern contains the pre-conditions of the rule and the RHS pattern represents the post-conditions of the rule. The LHS pattern is matched in the model being transformed and replaced by the RHS pattern in place. $LHS \bigcap RHS$ defines a graph part which must exist to apply the rule. Examples of graph-transformation-based approaches include VIATRA [Csertán *et al.*, 2002], AGG [Taentzer, 2000] and GReAT [Agrawal *et al.*, 2006] .

The difference between the two classifications is that *Czarnecki* and *Helsen* propose a hierarchical classification based on feature diagrams. In addition, they classify the specification of MTs. By contrast, the classification of *Mens* and *Gorp* is essentially multi-dimensional. They propose a taxonomy more targeted towards tools and techniques which support the activity of MT.

2.2.2 Model Transformation Languages

Since the introduction of MDE [Schmidt, 2006] and MDA [Soley and the OMG Staff Strategy Group, 2000], two kinds of languages are proposed to write MT:

- General purpose languages, in which we can add libraries or frameworks to manipulate models. For example EMF [Frank, 2004] and Java.
- Specific Language that are dedicated to model transformations, such as:

  ATL (Atlas Transformation Language) is a hybrid transformation language which supports declarative and imperative constructs. Thanks to its declarative style, ATL can simplify complex transformations algorithms. However, it is sometimes difficult to provide a complete declarative solution for complex transformation problem. In this case developers may resort to the imperative features of the language [Jouault *et al.*, 2008].

  Kermeta: is a meta-modeling, object-oriented and imperative programming language. It uses EMF tools to manipulate models. It offers constraints, checking and transformation. It does not support incremental model transformation.

  QVT (Query/View/Transformation) is the OMG standard language for specifying MT. It uses the Object Constraint Language (OCL). QVT defines three transformation

languages [Biehl, 2010] : (1) QVT Relational which is a high-level declarative transformation language which supports the specifications of bidirectional transformations, (2) QVT Core which is a low-level MT language which supports pattern matching and (3) QVT Operational which is an imperative MT language. The specified transformations are unidirectional.

GReAT (Graph Rewriting and Transformation) is a meta-model based MT language which uses meta-models to specify the abstract syntax of the source and the target models and the sequenced graph rewriting rules for specifying the transformation.

ETL (Epsilon Transformation Language) [Kolovos *et al.*, 2008] is a hybrid and rule-based transformation language. It can transform many source to many target models. Thanks to its rule inheritance, rules can be reused and extended.

2.3 Towards Model Transformation Generation

The evolution of model transformation languages consists in increasing the abstraction level of languages. Although most of these languages are able to implement complex model transformation problems, they may be difficult to use for users who are not experts on specific transformation languages. In addition, the solution domain of a model transformation (*i.e.*, the transformation language) can be largely different from the problem domain (*i.e.*, the source and the target model languages themselves) [Varró, 2006].
To address these challenges, several approaches are proposed to assist the specification and the design of MT.

2.3.1 Meta-model Matching for Model Transformation Generation

This approach is based on meta-models to generate MTs. Source and target models have to be similar, which is the case for example of model migration where the meta-models have two similar languages. This approach is inspired from the matching technique that is well known in the semantic web, schema, ontology and data warehouse domains [Rahm and Bernstein, 2001] [Shvaiko and Euzenat, 2005]. It takes as input a source schema and a target schema and generates a set of relations between them.

A first contribution proposed by [Lopes *et al.*, 2005] [Lopes *et al.*, 2006b] consists on defining an algorithm, called *SAMT4MDE*, that operates at the meta-model level to generate an alignment between source and target meta-models. It assumes that source and target models are similar in their structure and their terminology. String values of attributes are

used to find correspondences between source and target elements. Then they define a tool, called *MT4MDE*, that uses this alignment to generate MTs written in ATL.
In [Lopes *et al.*, 2009], *SAMT4MDE* is improved by using the structure similarity of the elements to find the correspondences between models.

In [Del Fabro and Valduriez, 2007], a similar approach is proposed to semi-automate the production of MTs. A first phase consists on discovering the relationships between the source and target models to create a weaving model. This latter is obtained by measuring the similarity between the attributes values (of type String) of the input models' elements. The *similarity flooding* algorithm [Melnik *et al.*, 2002] is also used to construct the adequate propagation models that capture the semantics of the relationships. The weaving model is then refined by an expert.

The contribution of *Falleri* [Falleri *et al.*, 2008] is inspired from the work of *Del Fabro* and *Valduriez*. They transform the source and target meta-models into directed labeled graphs. Those graphs are then exploited by the Similarity Flooding algorithm to compute the mapping between the meta-models elements and generate an Ecore alignment model from which MT is derived. They study several configurations for applying Similarity Flooding algorithm in the context of meta-model alignment with the aim of determining the best configurations.

In [Kappel *et al.*, 2006], the authors consider the mapping between two meta-models difficult because the meta-models represent an abstract syntax of the corresponding modeling language and a data structure for storing models. As a consequence, they do not make explicit certain language concepts. Thus, they propose to lift meta-models into ontologies to make the implicit concepts in the meta-model explicit in the ontology. Then, COMA++, an ontology alignment technique, is applied to compute the mapping between the source and target ontologies. Finally, alignments on ontologies are brought back to the meta-models.

Meta-model alignment is especially relevant when the source and target meta-models are semantically and structurally closed, *e.g.* when the transformation aims at migrating models from one meta-model version to another, but is inefficient on complex cases. When it can be applied, meta-model alignment reduces significantly the time of the development. Other approaches (MTBE for Model Transformation Based Example) take advantage of transformation examples to learn transformations in more complex cases. One of their strengths is that transformation examples, written in the concrete syntax, are easier to

manipulate than meta-models and their creation can be deferred to domain experts who do not need any programming skill.

2.3.2 Model Transformation By Example

Model Transformation by example (MTBE) [Kappel *et al.*, 2012] is a novel approach based on other by-example approaches like programming by example (pbE) [Lieberman, 1993], also known as programming by demonstration, which teaches a computer new behaviors by demonstrating actions or concrete examples.

Based on the by-example approaches, MTBE derives model transformation rules from a set of source and target models which describe the model transformation problem in a declarative way. The input models have to be established by the user. A matching between models is created to help the learning of rules. The advantage of this approach is that the concepts of the source and target models are used for the specification of the transformation.

2.3.2.1 MTBE approaches

The MTBE approach has been initiated by *Varró* [Varró, 2006]. An alignment between representative source and target example models is manually created. Transformation links are annotated by the transformation rule they illustrate (*e.g. ClassToEntity*). Transformation rules are derived from the transformation links and refined by the developer. Rules are validated on new source and target example models. If they are not satisfactory, the process iterates. The limitation of this approach is that it is not scalable for large industrial model transformation problems. In addition, it requires a manual intervention.

The proposal of [Varró, 2006] was extended in [Balogh and Varro, 2009], by using inductive logics programming (ILP [Muggleton and De Raedt, 1994]) to derive the transformation rules. ILP is a machine learning technique which derives a logic program from existing knowledge (source and target models), positive examples (pairs of model elements connected by transformation links) and negative examples (pairs of model elements that are not connected by transformation links). Considering only the immediate neighbors of each transformation-link end, the ILP engine infers an hypothesis for each transformation rule.

For the automation of this approach, a model transformation tool is implemented with an ILP engine. Small prototype mapping models are used to train the rules derivation.

Wimmer et al. [Wimmer *et al.*, 2007] propose a similar work which derive ATL transformation rules from examples. Both contributions use semantic correspondences between examples to derive rules. Examples are written in concrete syntax by taking advantage of the constraints explicitly applied by the transformation from the concrete syntax of a language to its abstract syntax. The main advantage of this solution is to be able to use the concrete syntax to define models and transformation links. However, model editors need to be written in a way that permits to extract constraints and to edit transformation links. This approach is applied on examples of business process models.

The work of *Garcia-Magarino et al.* [García-Magariño *et al.*, 2009] is also considered as a variant of MTBE approaches. In their approach, the authors generate many-to-many transformation rules from meta-models which satisfy some developer constraints for the simulation of input patterns of several elements.

Another MTBE approach [Dolques *et al.*, 2011; Dolques *et al.*, 2009] uses an extension of the anchorPrompt approach [Noy and Musen, 2001] to assist the transformation link discovery, and Relational Concept Analysis (RCA) [Huchard *et al.*, 2007] to derive commonalities between the source and target meta-models, models and transformation links. Compared to the ILP-based proposal, the RCA-based approach does not use annotations on transformation links and proposes a set of transformation patterns organized in a lattice.

Model Transformation By Demonstration (MTBD) [Langer *et al.*, 2010; Sun *et al.*, 2009; Brosch *et al.*, 2009], is a similar approach to MTBE. Through direct editing (*e.g.* add, delete, connect, update) of the source model, users are asked to demonstrate how the model transformation should be done. A recording engine was developed to capture user operations during a MT. The recorded fragments are then generalized to produce transformation patterns. However, this approach requires a high level of user intervention. The difference between the two cited works is that *Sun et al.* use the recorded fragments directly, however *Langer et al.* use differencing engine to generate ATL rules. In addition, the approach of *Sun* is applied to endogenous transformations while the approach of *Langer* is applied to both endogenous and exogenous ones.

Another track in MTBE consists in using the analogy to perform transformations using examples [Kessentini *et al.*, 2008; Kessentini *et al.*, 2009] [Kessentini *et al.*, 2010a]. The provided examples are manually decomposed into transformation blocks linking fragments of source models to fragments of target models. When a new source model has to be transformed, its constructs are compared to those in the example source fragments to select the similar ones. Blocks corresponding to the selected fragments, coming from differ-

Model transformation approaches	Model Transformation	
	By metamodels matching	By Example
[Lopes *et al.*, 2005]	×	
[Del Fabro and Valduriez, 2007]	×	
[Falleri *et al.*, 2008]	×	
[Wimmer *et al.*, 2007]		×
[Balogh and Varro, 2009]		×
[Kessentini *et al.*, 2008]		×
[Dolques *et al.*, 2010]		×
[Sun *et al.*, 2009]		×
[Langer *et al.*, 2010]		×
Our approach		×

Table 2.1 – Model Transformation approaches.

ent examples, are composed to propose a suitable transformation. Fragment selection and composition are performed through meta-heuristic algorithms. Thus, MT can be seen as an optimization problem where the transformation of a source model is obtained by finding, for each of its constructs, a similar transformation in the others examples. Particle Swarm Optimization (PSO) [Kennedy and Eberhart, 1995] and Simulated Annealing (SA) [Kirkpatrick *et al.*, 1983] heuristics are combined to automate MT . In [Kessentini *et al.*, 2010a], the approach is applied to Sequence Diagrams to Colored Petri Nets transformation.

Compared to the above-mentioned approaches, the analogy-based MTBE does not produce rules. This could be considered as a limitation if the goal is to infer reusable knowledge about transformations.

2.3.3 Synthesis

This section has introduced the existing work in the domain of generating model transformation. Table 2.1 summarizes the proposed approaches to generate model transformation. Diverse MTs have been identified and a number of techniques and tools have been developed to automate their generation and put them into practice. The context of our approach is model transformation by example (MTBE). The user has to create model transformation examples. An example consists of a source model and its corresponding model in the target language. Then several techniques can be used, such as relational concept analysis or inductive logic, to derive model transformation rules from the examples. These rules are abstract and not operational. They represent fragments of knowledge and must be arranged in a non-trivial way to perform the actual transformation. The approach of *Kessentini* consists in using search-based optimization techniques to directly generate the

By_Example Approaches	Exogenous transformation	Endogenous transformation	Matching	Rules generation	Rules execution	Techniques & Tools
[Wimmer *et al.*, 2007]	×		×	×		ad hoc method
[Kessentini *et al.*, 2008]	×		×			metaheuristic methods
[Sun *et al.*, 2009]		×	×			recording engine
[Balogh and Varro, 2009]	×		×	×		ILP
[Langer *et al.*, 2010; Brosch *et al.*, 2009]	×	×	×	×		differencing engine
[Dolques *et al.*, 2011]	×		×	×		FCA, RCA
Our approach	×		×	×	×	FCA, RCA, JESS

Table 2.2 – Model transformation by Examples approaches.

target model from the source model without the rules generation. This could be considered as a limitation if the goal is to infer reusable knowledge about transformations. In this context, the generation of operational rules from the existing examples can be preferable since it allows those rules to be executed on other source models to directly obtain the target models. Table 2.2 summarizes the proposed MTBE approaches. Most of them are specific to exogenous transformation and use matching techniques to derive transformation rules. The approach we propose is based on the work of *Dolques et al.* that uses RCA and FCA as learning techniques to derive transformation patterns. Those patterns are not operational. Thus, we propose to use the rule engine Jess to facilitate their manipulation and execution.

2.4 Model Transformation Traceability

In model-driven engineering, there is a concern on tracing model transformations during a software development. Some model transformation tools provide an integrated support for traceability such as QVT [OMG, 2005] and MOFScript [OMG, 2006]. With *ATL* [Bézivin *et al.*, 2003], developers can encode a trace as an output model. In [Jouault, 2005], *Jouault* proposes to attach traceability generation code to ATL program. *Grammel et Al.* [Grammel and Kastenholz, 2010] propose a generic framework for augmenting arbitrary model transformation approaches with a traceability mechanism. This framework is based on a domain-specific language for traceability. In [Kurtev *et al.*, 2007], the authors focus on generated trace relations as part of QVT transformations. In the same context, *Amar et al.* [Amar *et al.*, 2010] present an approach to automatically trace imperative model transformation in a Java/EMF environment. Finally, a recent work [van Amstel *et al.*, 2012] consists

in visualizing traceability in model transformations after adding a trace generator to the transformation engine of ATL. All those solutions generate trace links in parallel with the transformations. They depend on the existence of a transformation engine and could not be applied for trace recovery.

Another category consists in generating transformation trace independently from the transformations. This allows to handle cases where only source and target models are present without a knowledge on how the transformation was performed. In [Cysneiros *et al.*, 2003], the authors present an approach to support generation of bi-directional traceability relations between organizational requirements modeled in i*, and UML use cases and class diagrams. This approach is based on the use of rules, which express the different types of relations between model elements. It is applied to a specific type of transformation. The work in [Grammel *et al.*, 2012] consists in using graph-based model matching techniques to generate trace links. The mapping results are arranged into a cube to be analyzed and to extract trace links. This approach may have a higher complexity, especially when manipulating large-size models. In addition, it produces *one-to-one* matching links.

In a related field, refactoring could be seen as an endogenous transformation. Recovering refactorings from two versions of the same model is very similar to the traceability problem. In this context, an approach is proposed in [Xing and Stroulia, 2006] which is based on the design-level changes reported by the UMLDIFF algorithm [Xing and Stroulia, 2005] to detect and classify refactorings in evolving software models. We also mention the work of [Vermolen *et al.*, 2011], in which the authors provide an approach to detect complex evolution traces between two meta-model versions, using a matching result as input, to allow model migration. In [Kehrer *et al.*, 2011], *Kehrer et al.* address the problem of how to semantically lift low-level differences on models. They use a model transformer for finding instances of editing operations and annotating a low-level difference. In [ben Fadhel *et al.*, 2012], a very recent heuristic-based approach for detecting refactorings is proposed. It takes as input a list of possible refactorings, the initial model and the revised one, and generates as output a list of detected changes in terms of refactorings. Although the above-mentioned approaches produce very good results, they are specific to the particular case of refactoring and cannot be generalized easily to other transformation problems.

In the field of ontology engineering, [Hartung *et al.*, 2010] presents a rule-based approach to determine different evolution mappings between two versions of an ontology. The goal is to produce a minimal evolution mapping model using a rule-based system that finds the basic change operations.

Model matching technique can also produce correspondences between source and target models on the same abstraction level. It is related to the field of schema matching and ontology matching. The basic idea of the main approaches [Rahm and Bernstein, 2001; Choi *et al.*, 2006; Shvaiko and Euzenat, 2005], is to find semantic correspondences between elements of two schemas. They make the assumption that the relations between the two models being compared are identical. They compute a similarity between the elements using their names. They also compute a structural similarity between the elements. For this, they assume that there is the same kind of relations between the elements in the two compared models.

For model transformation, as mentioned in Section 2.3, *Fabro* and *Valduriez* [Fabro and Valduriez, 2009] create links between source and target metamodels by using the similarity flooding technique to construct propagation models which capture the semantics of the relationships between the two models. Then, links are designed by an expert and are used to produce transformation. *Dolques* [Dolques *et al.*, 2011] propose a semi-automatic matching approach for discovering links between source and target models. They assume that the target model results from a transformation from the source model. The approach uses and extends the Anchor-Prompt approach to discover the pairs of elements for which there is a strong assumption of matching. In [Lopes *et al.*, 2006a; Lopes *et al.*, 2009], the authors define an algorithm (SAMT4MDE) that assumes that source and target metamodels are similar in their structure. It finds correspondences between them using string values of attributes and structure similarity. The contribution of [Falleri *et al.*, 2008] consists in transforming the source and target metamodels to directed labeled graphs. Then they evaluate different parameterizations of the similarity flooding algorithm to compute the mapping between the two graphs.

2.4.1 Summary

This section has introduced the existing work in the domain of model transformation traceability. Table 2.3 summarizes the proposed approaches to recover model transformation. There exist two categories for model transformation traceability. The first one generates trace links in parallel with the transformation. It depends on the existence of transformation engine. The second one consists in generating transformation traces independently from the transformation. The approach proposed in [Cysneiros *et al.*, 2003] is not generalized and specific to two metamodels. The approach of *Grammel* [Grammel *et al.*, 2012] generates transformation links but it is not scalable for large models. Furthermore,

Traceability approaches	Transformation dependence	Automatisation	Many-to-many matching
[OMG, 2005]	×	×	
[OMG, 2006]	×	×	
[Jouault, 2005]	×	×	
[Kurtev *et al.*, 2007]	×	×	
[Grammel and Kastenholz, 2010]	×	×	
[Amar *et al.*, 2010]	×	×	
[van Amstel *et al.*, 2012]		×	
[Cysneiros *et al.*, 2003]		×	
[Grammel *et al.*, 2012]		×	
Our approach		×	×

Table 2.3 – Recovering model transformation traces approaches.

it produces one-to-one trace links. Thus, we propose to recover transformation links independently from the transformation. The goal is to find *m-to-n* matching links between arbitrary source and target models. Thus, we propose to use a meta heuristic method to associate for each m source elements their corresponding n target elements.

2.5 Search Based Software Engineering

Search based software engineering (SBSE) is the application of search based optimization in software engineering. It seeks to reformulate software engineering problems as search problems [Harman *et al.*, 2001]. It is inspired by the observation that many problems in software engineering can be formulated as optimization problems. Thus, several meta-heuristics algorithms such as genetic algorithms (GA) [Goldberg, 1989], simulated annealing (SA) [Laarhoven and Aarts, 1987] have been successfully applied to solve those problems, for example in cost estimation, testing and automated maintenance [Harman, 2007]. Model verification and module clustering have also been addressed using search-based techniques. In [Shousha *et al.*, 2008], *Shousa et al.* present an approach based on GA to detect deadlocks in UML models. In the context of MT, *Kessentini* [Kessentini, 2010] deals MT as a combinatorial optimization problem. He presents a search based approach that uses source and target models to learn MT. PSO and SA are applied to solve the problem. In this work, we model the software engineering problem of recovering model transformation traces as a search problem, thus the use of search techniques.

2.6 Conclusion

In this chapter, we survey the different approaches related to the MDE field and our observations which will be useful to introduce our contributions.
MT is one of the pillars of MDE [Guerra *et al.*, 2013]. Many approaches have been proposed to generate MT. MTBE is the closest work of our proposal. It consists on learning transformations rules from a set of examples. An example contains source and target models with matching links between the two models. Many learning methods are used to derive transformation rules, *i.e,* ad hoc methods, inductive logic programming, metaheuristic methods or relational concept analysis. We observe, from the study of the existing works on MTBE, that there does not exist operational transformation rules which can be apply to all kinds of source models to generate their corresponding models. All generated rules are abstract. In this context, we propose in this work to generate operational transformation rules from examples given by the users. The base of our work is the approach of *Dolques, et al.* which consists in using RCA to derive transformation patterns.

In MDE, there is also a concern on tracing model transformations. Model transformations traces can be obtained during a program transformation or independently from the transformation program. The first category depends on the existence of a transformation program while the second strategy is independent from any transformation program. This latter is specific to the cases where only source and target models are present without a knowledge about the transformation, *e.g.* when the transformation is done manually by an expert or when the program transformation is lost. Several approaches have been also proposed to recover model transformation traces independently from the transformation program. Some of them are related to specific metamodels and they are not generalized. The others have a higher complexity with large-size models. Furthermore, all proposed approaches produce *one-to-one* matching links between models. For this, we propose in this work an approach to recover model transformation traces from examples independently from their transformation program. The proposed approach is generic and does not depend on a specific metamodels. In addition, it produces *many-to-many* matching links between models. The scalability of this approach is ensured by the use of a multi-objective optimization method.

Model matching is a very important task, it is an essential part of different proposed approaches in the field of MDE (MTBE, recovering model transformation, etc) and other domains (data base, ontology, etc). For model transformation, different techniques have been used to create links between models, *i.e.* the similarity flooding, the anchor prompt,

etc. From the study of the proposed approaches, an observation consists on the generation of *one-to-one* matching links between models. This observation can be considered as a limitation of these approaches. This can be justified by the type of relationship between constructs in the models. For example, in a UML class diagram, we cannot separate an association or a generalization from their two classes. It may lose the semantic of the class diagram. Thus, we propose to produce matching links between n source elements and m target elements from source and target models. Due to the large number of possible combinations between source and target model elements, a metaheuristic method is used.

3 Generation of Operational Transformation Rules from Examples of Model Transformations

Contents

This chapter introduces our first contribution, which consists of the generation of operational transformation rules from examples of model transformations. We propose a two-step approach to generate the transformation rules. In a first step, transformation patterns are learned from the examples through a classification of the elements of the model

examples, and a classification of transformation links (expressing part of the transformation trace) using Formal Concept Analysis. In a second step, those transformation patterns are analyzed in order to select the more pertinent ones and to transform them into operational transformation rules written for the Jess rule engine. The generated rules are then executed on examples to evaluate their relevance through classical precision/recall measures. We also study how we better learn transformation rules from examples, using each example separately or by gathering all the examples.

This chapter is structured as follows. Section 3.1 recall the key concepts of Model Transformation (MT) and Model Transformation Based Examples (MTBE). Then, we introduce the problem and describe our two-step approach in Section 3.2. In Section 3.3, we briefly explain how RCA is used to extract information from examples and to generate transformation patterns. In this section, details are also given on how the obtained transformation patterns are filtered and refined. Section 3.3.3 describes the mapping of the transformation patterns into Jess rules. We present an evaluation of the approach and a discussion about the obtained results in Section 3.3.4. In Section 3.4, we analyze and compare two strategies for learning the transformation patterns. Section 3.5 concludes the chapter and describes future work.

3.1 Introduction

As we explained in the State-of-the-art section, Model Transformation is a key component of Model Driven Engineering (MDE). In model-driven development, the involved models are processed by programs as a matter of priority (rather than by hand). To ease the development of such programs handling models, several languages were introduced, *e.g.* graph transformation languages such as VIATRA [Csertán *et al.*, 2002], declarative or semi-declarative languages like ATL, or object-oriented and imperative languages such as Kermeta.

Implementing a model transformation requires two distinct skills: model-driven engineering skills (in particular, metamodeling and model-transformation environments), and domain-specific skills, *i.e.*, good knowledge about the specification of the transformation: the input domain, the output domain, and the transformation rules by themselves. While the first skills are possessed by model-driven engineering experts, the second ones are specific to domain experts. Experience shows that domain experts more easily give transformation examples than complete and consistent transformation rules [Kessentini, 2010].

In this context, MTBE [Varró, 2006] has emerged as a convenient way to let domain experts design transformations by giving an initial set of examples. An example consists of an input model, the corresponding transformed model, and fine-grained mappings between the constructs of both models. From those examples, an MTBE approach learns transformation rules. When those rules are operational, *i.e.*, they are written in a rule language disposing of a rule engine, they form the model transformation.

As part of our thesis, we present a Model Transformation By Example approach that goes from examples down to operational transformation rules. The learning mechanism used is based on RCA, a variant of Formal Concept Analysis [Ganter and Wille, 1999]. It results in a hierarchy of non-operational rules called transformation patterns. Such transformation patterns are analyzed and filtered to derive the more relevant ones. The selected transformation patterns are then transformed into concrete and operational transformation rules that can be processed by the Jess rule engine [Daniele, 2006]. The learning of the transformation patterns has been proposed in [Dolques *et al.*, 2009], in the current work we introduce the filtering of the obtained transformation patterns, and we explain how to obtain operational rules from the transformation patterns. Finally, since the obtained rules are operational, experiments have been carried out on a case study in order to measure the relevance of the generated rules.

3.2 Overview of the Rules Generation and Execution

Model-Transformation By Example (MTBE) consists in learning transformation programs/rules from examples. Usually, an example is composed of a source model, the corresponding transformed model, and transformation links between those two models. To illustrate MTBE, let us consider the well-known case of transforming UML class diagrams into relational schema, used, among others, in [Kessentini *et al.*, 2010b]. For this transformation, examples are given in the form of: an input UML model (such as the one given in Figure 3.1), the corresponding transformed relational model (such as the one given in Figure 3.2), and transformation links making explicit from which elements of the UML model, the elements of the relational model stem from. For instance, a transformation link is given to specify that class `Client` is mapped into table `Client`. A transformation link is equivalent to a link of an execution trace of the expected transformation, *i.e* two elements are related by a transformation link if the information contained in the first element is necessary to build the second one.

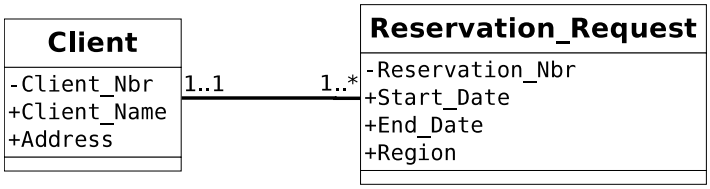

Figure 3.1 – Example for the UML2R transformation: input model

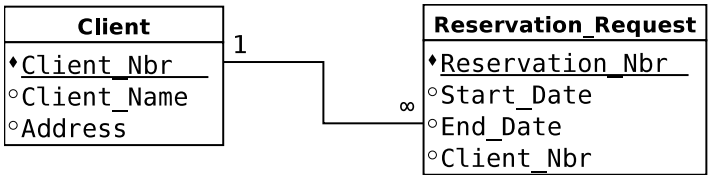

Figure 3.2 – Example for the UML2R transformation: transformed model

An MTBE process analyzes the examples and learns from them transformation rules such as *a class is transformed into a table*, or a UML property linked to a class and which is not a member of an association (*i.e.*, an attribute and not a role) is transformed into a column of a table. This process should produce operational rules, *i.e.*, rules that can be directly executed by a rule engine to transform any source model into a target model.

We propose to generate the operational rules in a two-step approach, as illustrated in

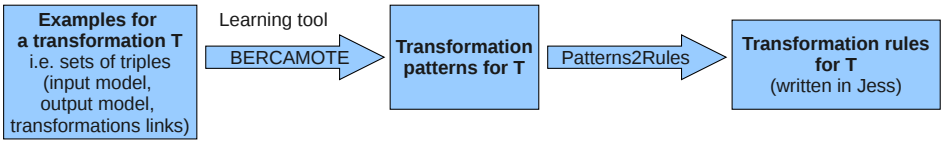

Figure 3.3 – A two-step approach for MTBE

Figure 3.3. The first step is the analysis of examples, that learns transformation patterns using Relational Concept Analysis. This step is supported by the Bercamote tool, that has been introduced in [Dolques *et al.*, 2009]. Each obtained transformation pattern describes a premise in the form of an input model pattern (based on the input metamodel), and a conclusion, in the form of the output model pattern (based on the output metamodel) that should be obtained after the execution of the transformation. The transformation patterns are ordered in a hierarchy. This hierarchy is analyzed to select the more relevant patterns, and sometimes to select in a transformation pattern the more pertinent fragment. We here target model-to-model transformations in which both models represent the same data but in different languages or using different structural constraints *e.g.* a transformation applying design patterns to enforce good structural modeling practices in a language. On

the contrary, our MTBE approach is not well-suited to learn transformations in which new values are computed, *e.g.*; we cannot learn a renaming policy that forces to use lowercase for attribute names. Widening the scope of the transformations that can be learned is possible but would impact on the complexity of the results and the efficiency of the approach.

The main contribution of this chapter deals with the second step, that makes the patterns operational. This is done by transforming them into rules that can be executed by a rule engine. To make the transformation patterns operational, we have transformed them into Jess rules and executed them using the Jess Rule engine. This step is detailed in Section 3.3.3.

3.3 A By-example Approach to Obtain Transformation Patterns

As stated in Section 3.2, a key step in our MTBE approach consists in generating transformation patterns. Such patterns describe how a source model element is transformed into a target model element, within a given source context and a given target context. This step has been presented in [Dolques *et al.*, 2009], and is summarized in the beginning of this section, whereas the end of this section is dedicated to the filtering of the obtained transformation patterns.

3.3.1 Obtaining the Transformation Patterns

To derive patterns from examples, a data analysis method is used, namely Formal Concept Analysis (FCA) and its extension to relational data, the Relational Concept Analysis (RCA) (introduced in Chapter 1).

We use RCA to classify: the source model elements, the target model elements and the transformation links. Which means that every one of them will be modeled as an Object-Attribute context in RCA. Those contexts will be linked by Object-Object contexts modeled after the following relations. Source and target model elements are classified using their metaclasses and relations. The transformation link classification relies on model element classifications and groups links that have similarities in their source and target ends: similar elements in similar contexts. From the transformation link classification, we derive a transformation pattern hierarchy, *i.e.*, a lattice of patterns, where patterns are organized by inclusion.

To illustrate the use of RCA to generate a transformation patterns hierarchy, we use a simple UML model (Figure 3.5) that is conform to the UML metamodel of Figure 3.4, its

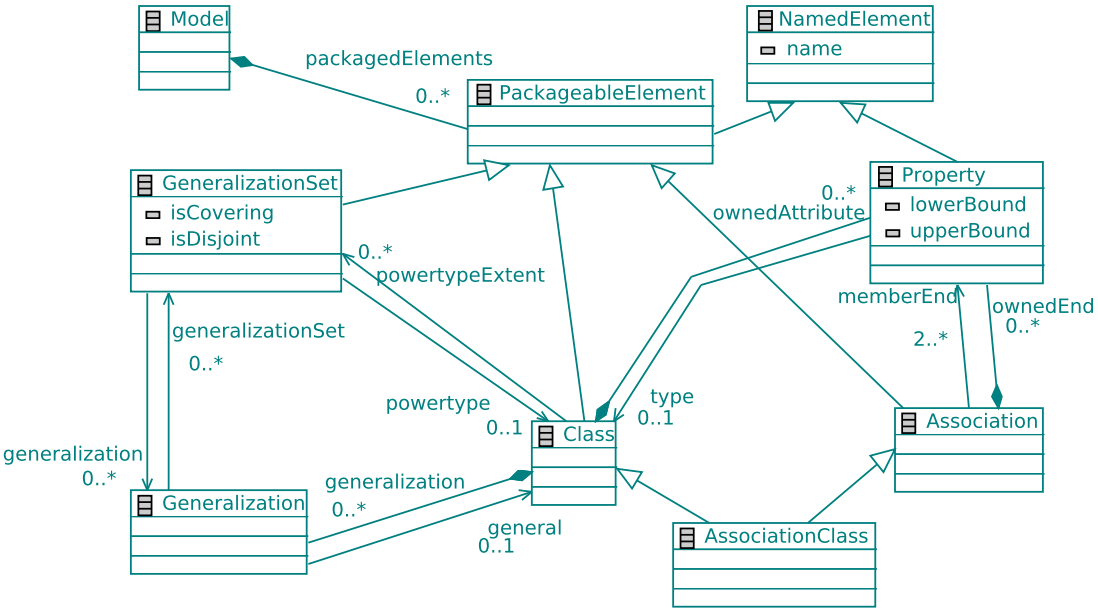

Figure 3.4 – A simplified UML metamodel

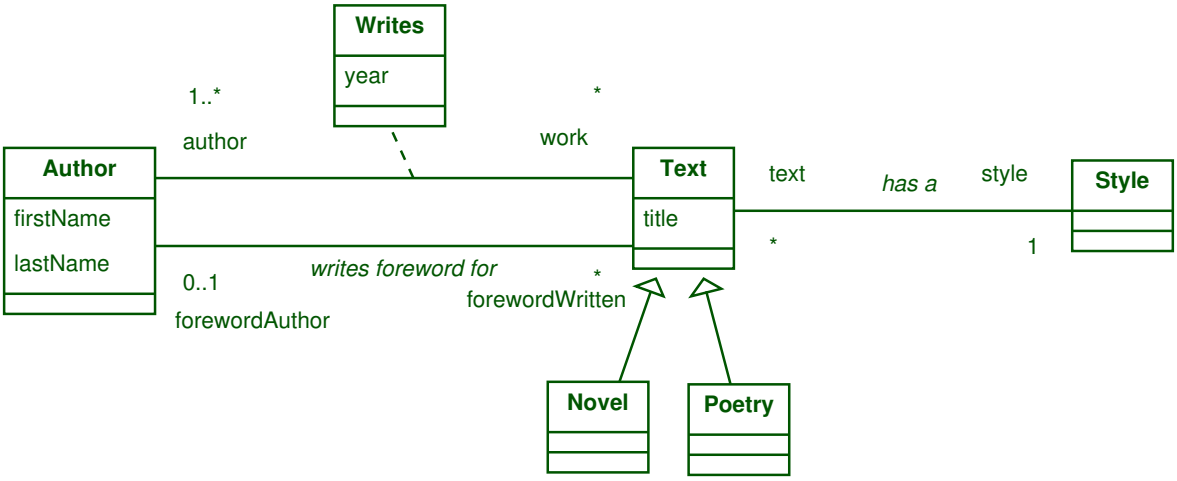

Figure 3.5 – A UML Model

corresponding entity relationship model (Figure 3.7) that conforms to the entity relationship metamodel of Figure 3.6 and the mapping links between the two models [1]. Figure 3.8 shows an excerpt of mappings between the two models. Data from this transformation example is encoded into five formal contexts (UML metamodel context, entity relationship metamodel context, UML model context, entity relationship model context and mapping link context). Then, RCA applied to these data leads to the corresponding five lattices. We present three of these lattices: the UML model lattice (Figure 3.9), the entity relationship model lattice (Figure 3.10) and the mapping link lattice (Figure 3.11). In this latter, a map-

1. Borrowed from Xavier Dolques

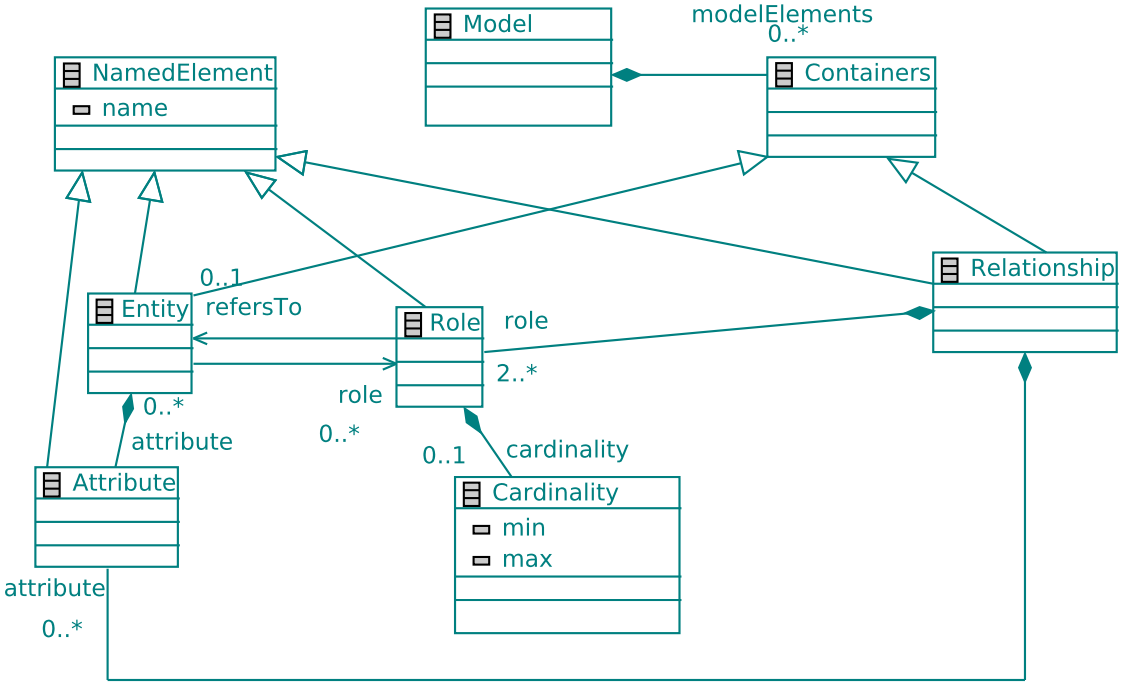

Figure 3.6 – An entity relationship metamodel

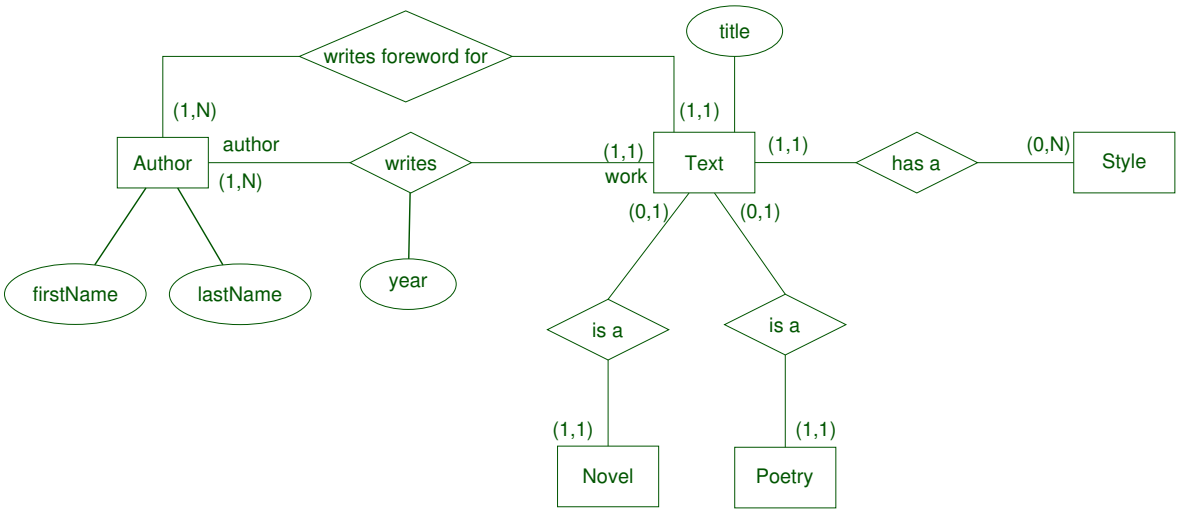

Figure 3.7 – An entity relationship model

ping link is described by two elements: 1) *traceA*, the source model element which is the source of the link and 2) *traceB*, the target model element which is the end of the link.

For example, `Concept_49` in lattice of Figure 3.9 groups the instances of the UML source metaclass Property which have a type which is a class (classes are grouped in `Concept_41`). `Concept_55` in Figure 3.10 groups the instances of ER metaclass Role which refer to an Entity and have a Cardinality. `Concept_82` in Figure 3.11 groups

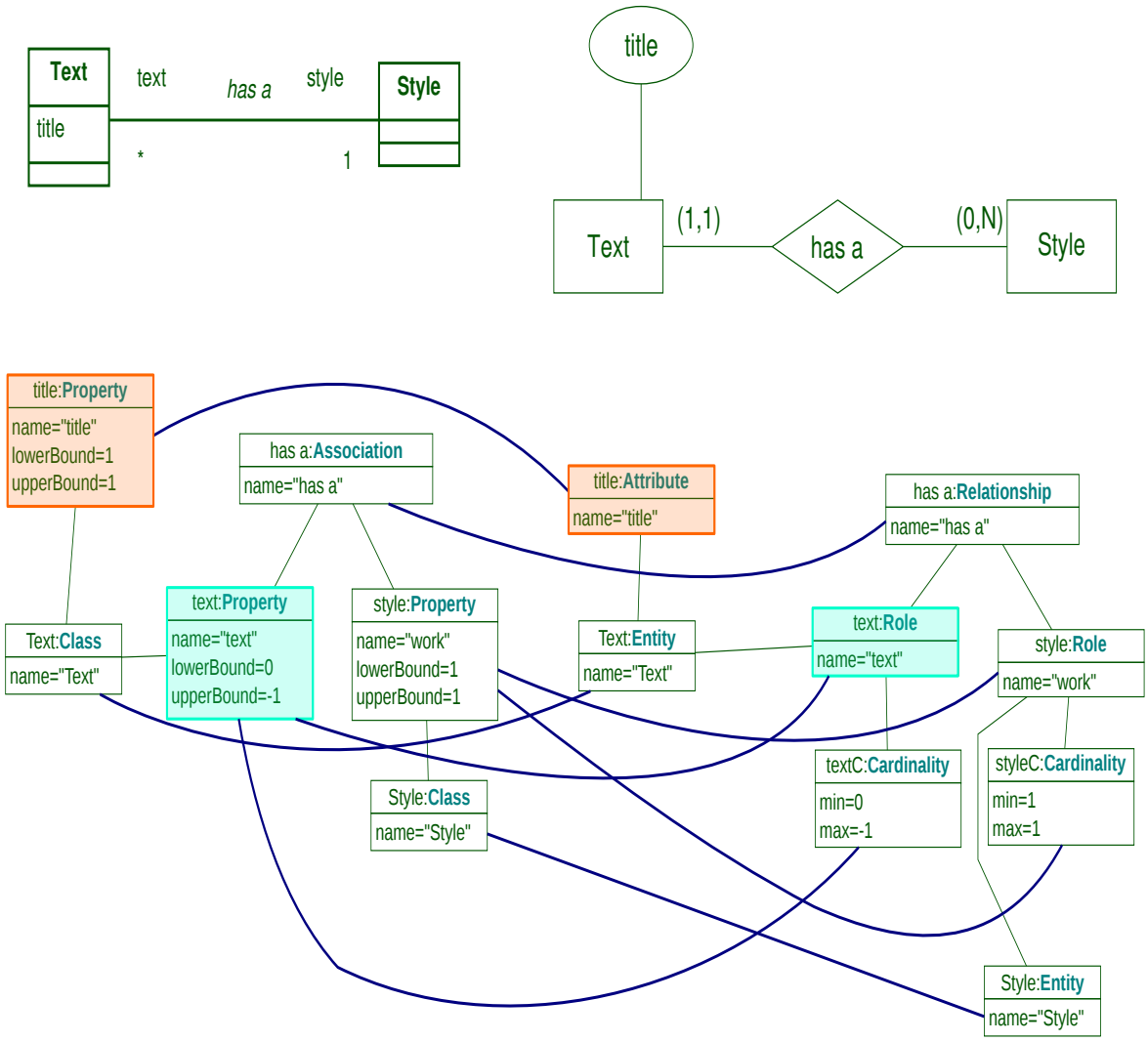

Figure 3.8 – An example of mapping between two excerpts of models of Figures 3.5 and 3.7

links going from `Concept_49` which groups UML properties connected to a class, to `Concept_55` which groups ER roles which refer to an Entity and have a Cardinality. The mapping link lattice is then automatically analyzed to generate a hierarchy of transformation patterns. An excerpt of this latter is presented in Figure 3.12. It contains six transformation patterns (in the six inner boxes). The transformation patterns in the bottom are always more specific than the ones in the top boxes. For example, the transformation pattern of Concept `TPatt_12-Concept_92` is more specific than the transformation pattern of Concept `TPatt_11-Concept_81`, which is indicated by the inclusion edge between the two boxes. The patterns are automatically named by the tool, they have a prefix beginning by *TPatt* for *transformation pattern*, then we find the number of the pattern, and finally the number of the concept representing the pattern, as generated by RCA/FCA algorithms.

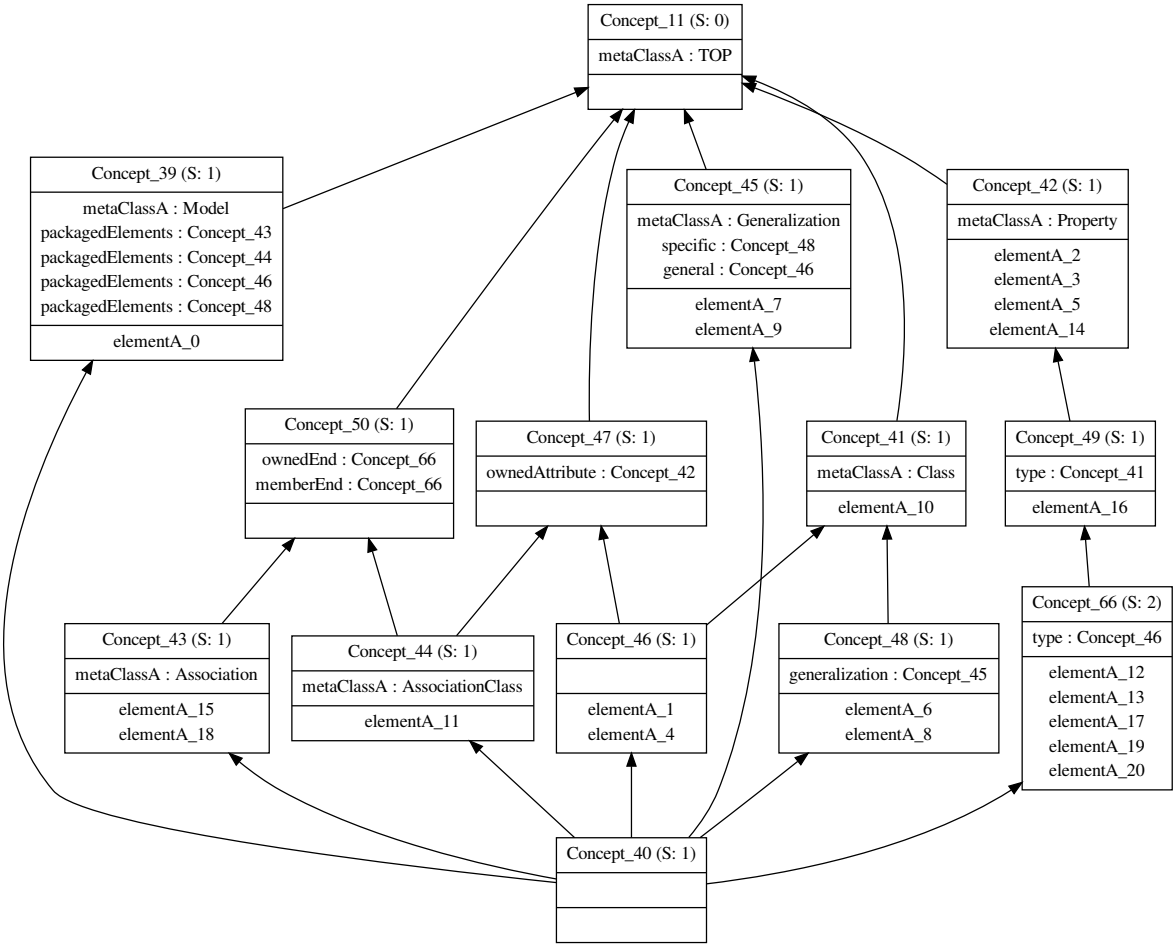

Figure 3.9 – The lattice obtained from the UML model of Figure 3.5

In each concept representing a transformation pattern, we have two types in two ellipses connected by a bold edge. The source ellipse of the bold edge represents the type T_s of the element to transform by the pattern. It can be seen as the main type of the premise. For instance, in Concept `TPatt_3-Concept_82`, we see that the pattern aims at transforming *properties*. This main type of the premise is linked, with non-bold edges, to the environment that an element of type T_s must have in order to be transformed by the pattern. Those edges are named according to the relation-role names between the type T_s and its environment in the metamodel. Those edges also have a cardinality defining the cardinality of the environment. Such an environment corresponds to the rest of the premise. For instance, in Concept `TPatt_3-Concept_82`, `Property` is linked to a `Class` with an edge named `type`. This means that the premise corresponds to a property, and that this property is linked to a class. The target ellipse of the bold edge represents the main type T_t of the conclusion of the pattern, *i.e.*, a T_s will be transformed into a T_t (with a specific environment). For example, in the transformation pattern `TPatt_3-Concept_82`, the conclusion

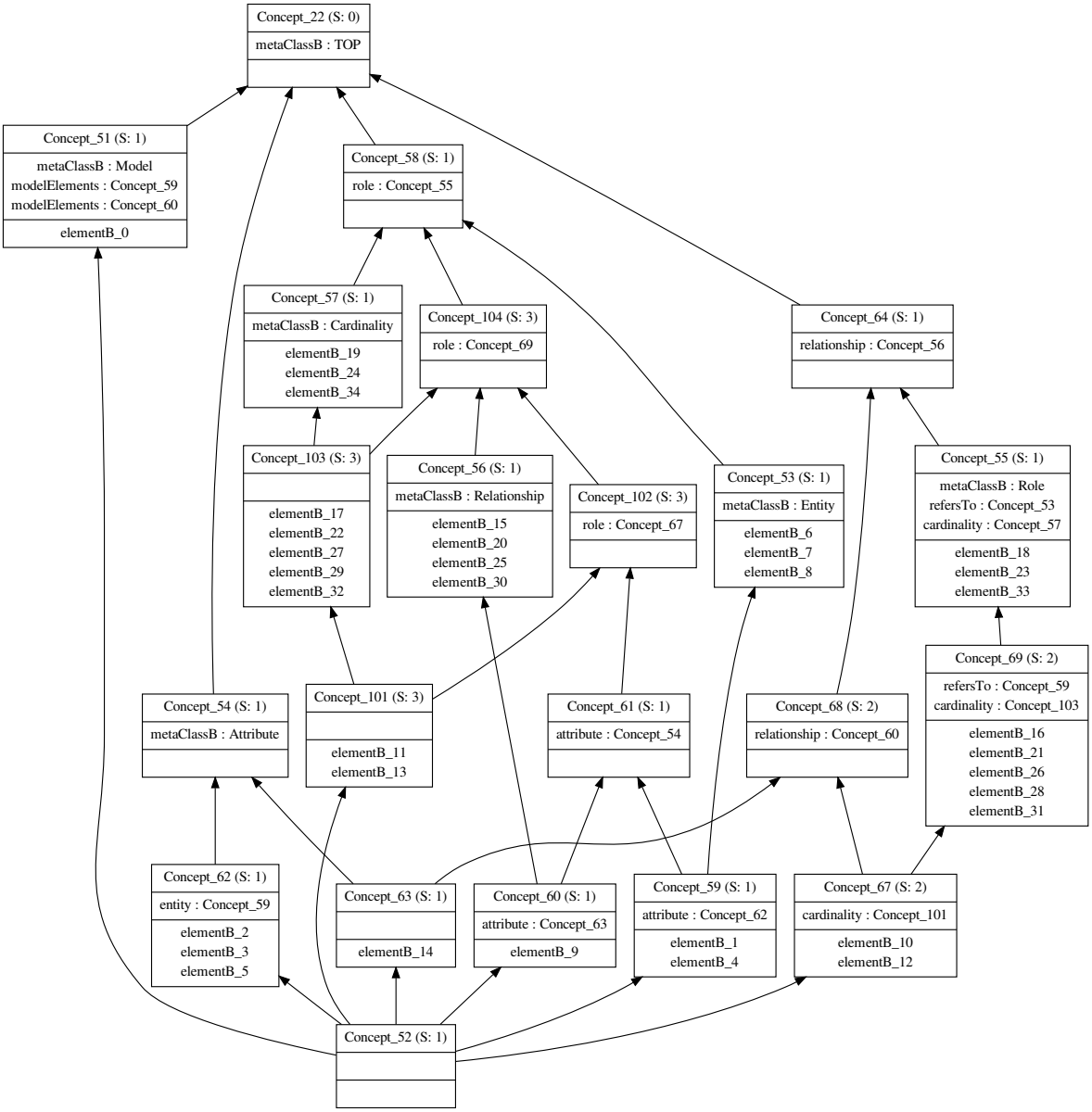

Figure 3.10 – The lattice obtained from the entity relationship model of Figure 3.7

corresponds to a role which refers to an entity, has a cardinality and has a relationship with another role. This role is in turn linked to other elements.

The transformation pattern `TPatt_3-Concept_82` has thus been deduced from a set of transformation links that were grouped together because they link a property (connected to a class) to a role (connected to a role, a cardinality and a relationship). This pattern is included in the pattern of sub-concept which is located below it (there is just an excerpt of this concept in the figure). This latter is more specialized because in addition to link the property to a class, it also links the class to a relationship. It also links the entity to other elements which are not presented in the figure.

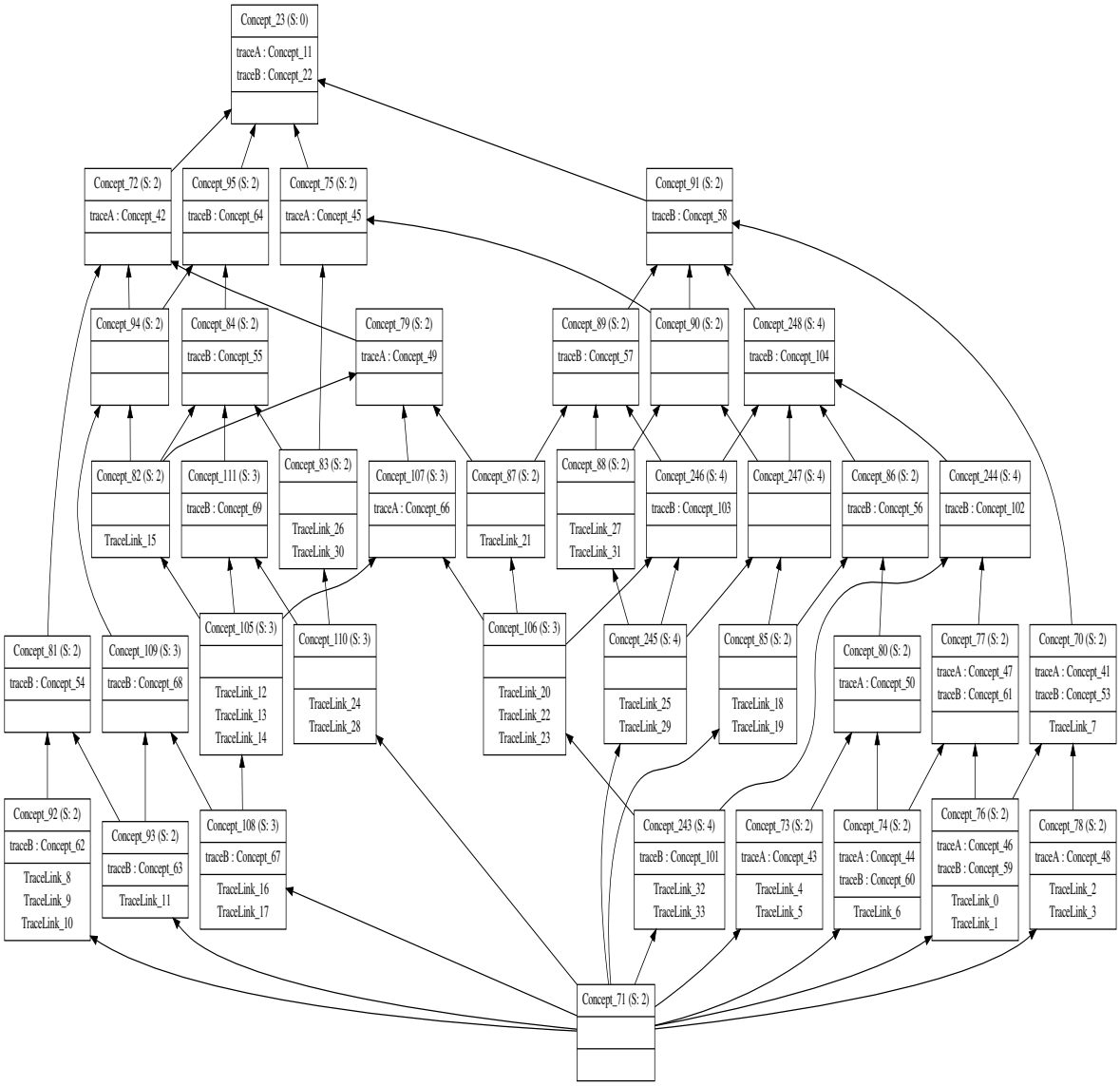

Figure 3.11 – The lattice obtained from mapping between the models of Figures 3.5 and 3.7

3.3.2 Patterns Lattice Simplification

After obtaining the lattice of transformation patterns, we select in this lattice the useful/relevant patterns or pattern fragments. In the lattice of Figure 3.12, for instance, concepts `TPatt_15-Concept_109` and `TPatt_14-Concept_94` are empty. They do not contain information about the transformation. They are present in the lattice to link other concepts (representing patterns) not shown in this excerpt. In the final transformation, those empty patterns are automatically removed from the lattice. When an empty concept is removed, we connect all its children with all its parents to keep the order structure of the lattice.

After the lattice pruning, the remaining patterns are analyzed for simplification pur-

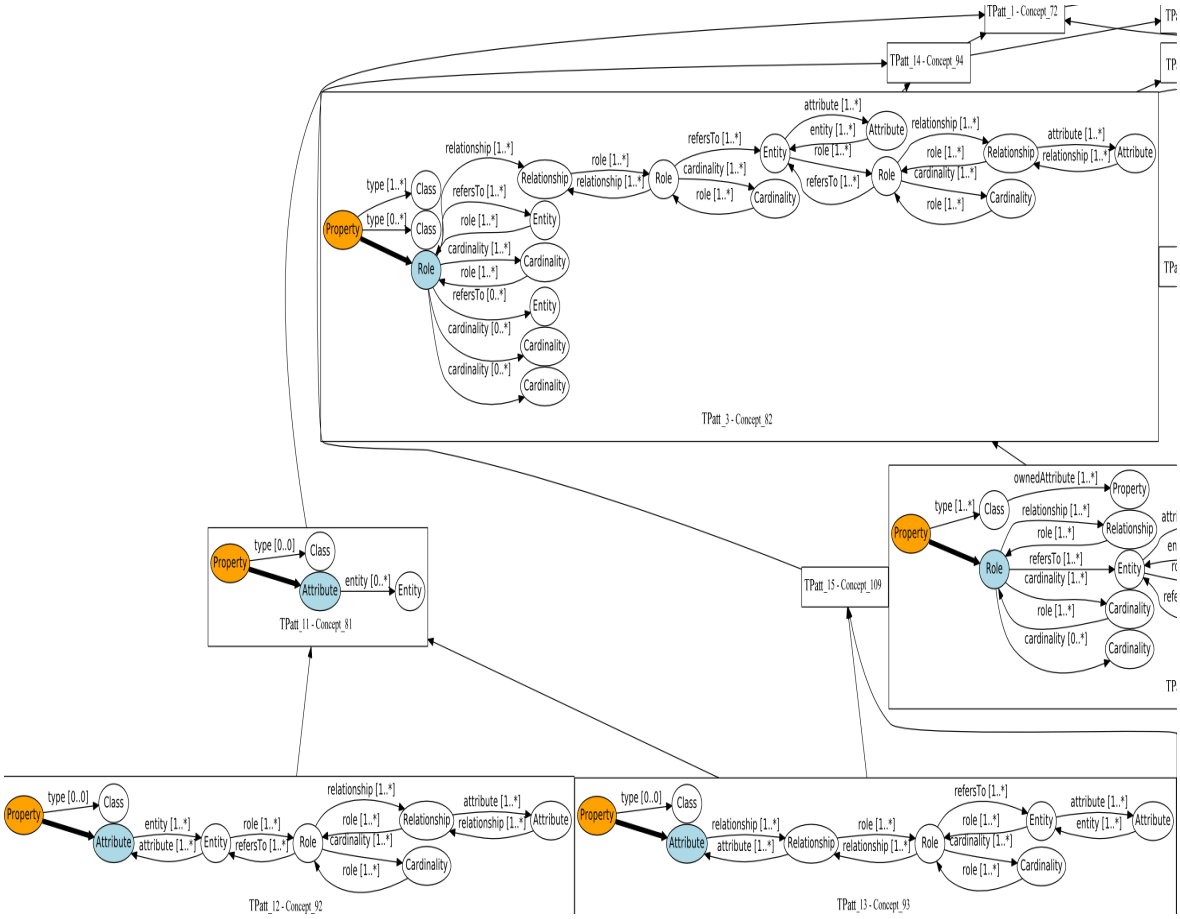

Figure 3.12 – An excerpt of the obtained hierarchy of transformation patterns

pose. We noticed that some patterns contain a deep premise or conclusion, *i.e.*, a long chain of linked objects. After observing many patterns of this type for many transformation problems, we found that after a certain depth, the linked elements are not useful. For instance, if we look again at the pattern `TPatt_3-Concept_82` in Figure 3.12, the important information is that a property linked to a class must be transformed into a role linked to an entity, a cardinality and a relationship. The other elements are redundant or details specific to some examples, that are not relevant to the transformation. Starting from this observation, we implemented a simplification heuristic that prunes the premises and conclusions after the first level (key element and its immediate neighbors).

After pruning the patterns according to the depth heuristic, some patterns could become identical. For redundant patterns, just the top ranked in the lattice is preserved, and all other are automatically removed. For removed concepts, their children are linked to their parents.

Figure Figure 3.13 shows an excerpt of the obtained hierarchy of the previous example (Sec-

tion 3.3.1) after simplification. For instance, the `TPatt_3-Concept_82` is transformed to the simplified `TPatt_1-Concept_66`.

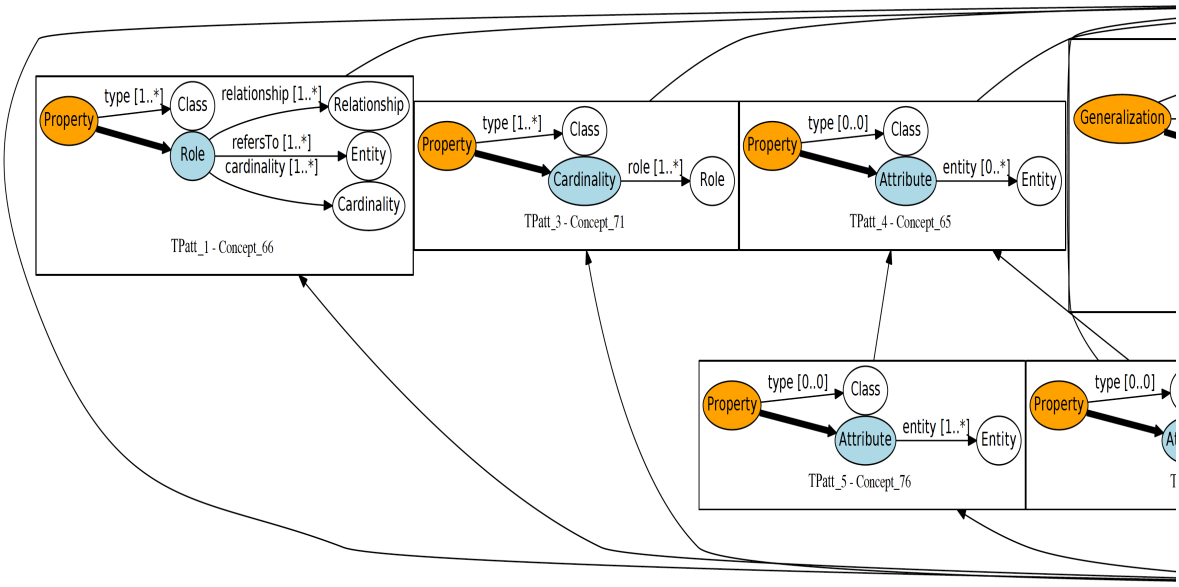

Figure 3.13 – An except of the obtained hierarchy of transformation patterns after simplification

3.3.3 Rules Generation

This section describes the mapping of transformation patterns into operational rules that can be executed using a rule engine. The rule engine used in our proposal is the Java Expert System Shell (*Jess*) introduced in Chapter 1.

In our context of model transformation, facts are model elements and templates are element types defined in the metamodel. A UML class diagram metamodel defines a set of templates such as `Class`, `Attribute`, and `Association`. A specific UML class diagram is described using facts that are instances of these templates such as, `Class Employee`, `Class Position`, and `Association has_position`. Fact `Class Employee` means that the model contains an element "Employee" which is an instance of the type "Class" in the metamodel.

Figure 3.14 illustrates the steps to follow in order to obtain operational rules from transformation patterns. The transformation process consists of three steps: Metamodel2Templates, Model2Fact, and TransformationPatterns2JessRules.

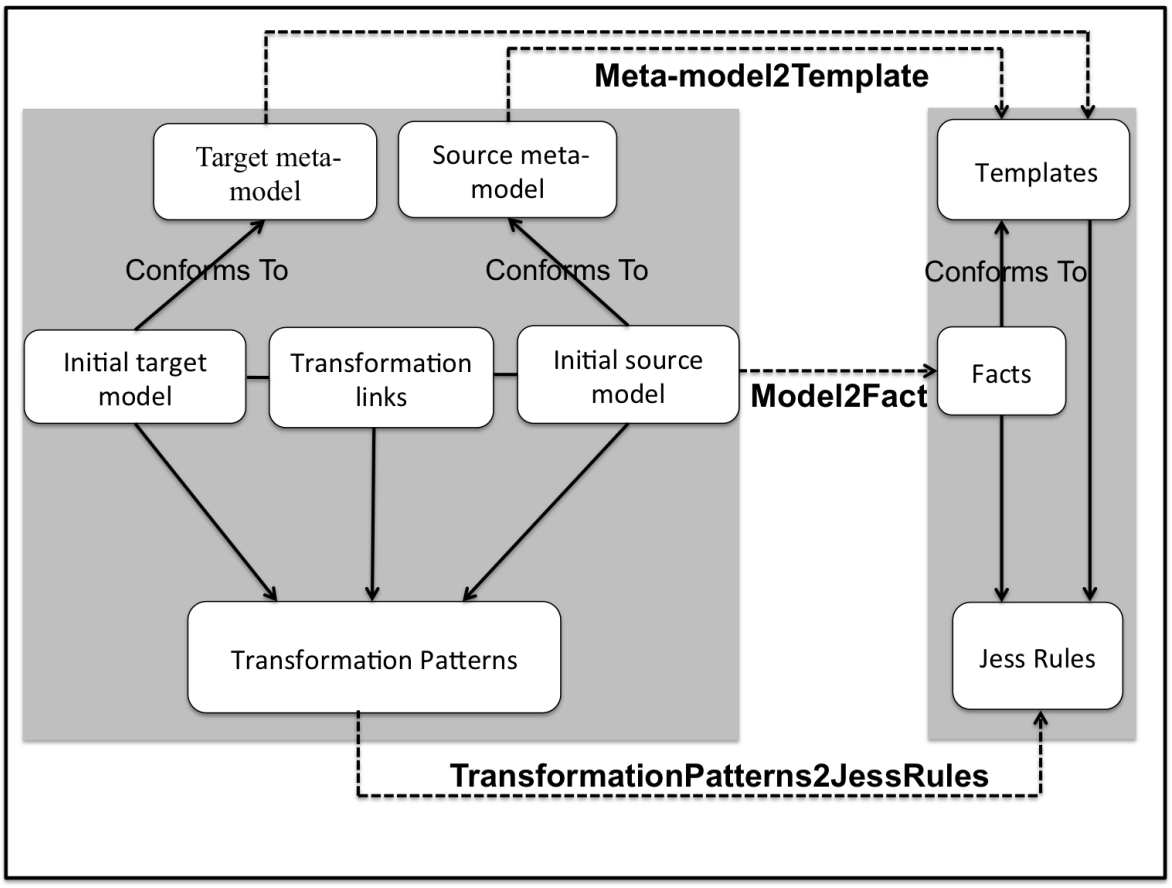

Figure 3.14 – Transformation Process

3.3.3.1 Meta-models2Templates

Step 1 consists in generating templates from the meta-models. Each metaclass of the metamodel is transformed into a template with the same name. Each meta-attribute is also transformed into a *slot* keeping the same name. The type of the slot is the type of the meta-attribute. To facilitate the description of relations between the metaclasses, each meta-reference is also transformed into a template. Such a template has two slots respectively containing the name of the source element and the target element of the meta-reference. We suppose that the name of each element is its identifier.

Concretely, since we work with the EMF framework, this step corresponds to the following transformations:

- each *EClass* is transformed into a template with the same name,
- each *EAttribute* is transformed into a *slot* with the same name and whose type is the *EDataType* of the *EAttribute*,
- each *EReference* is transformed into a template.

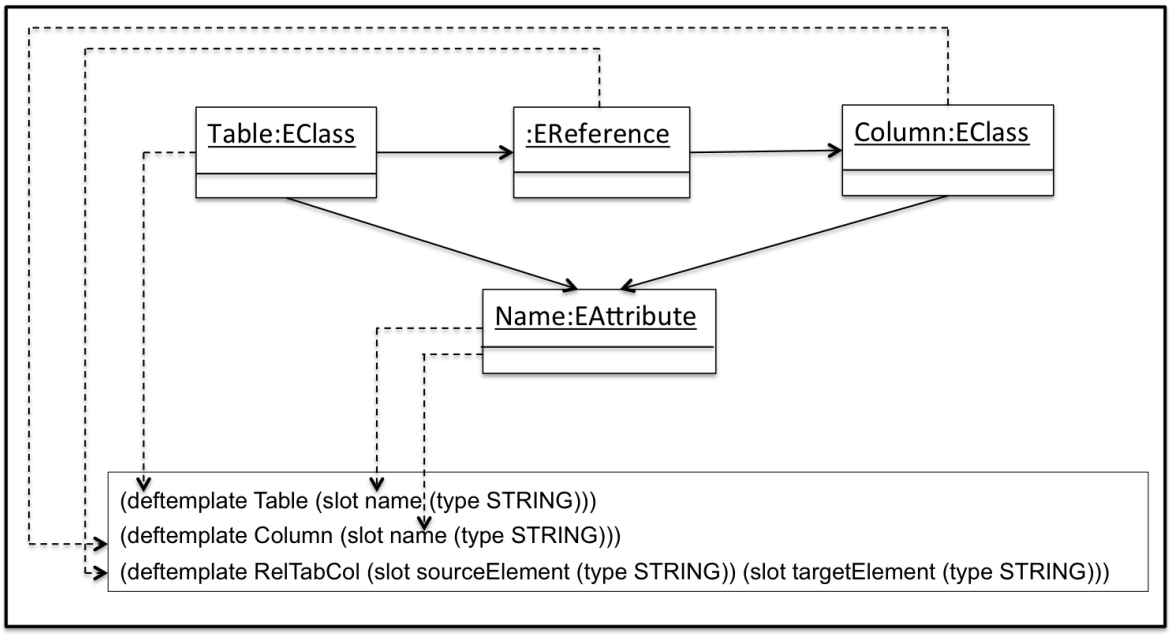

Figure 3.15 – Transformation of an extract of relational meta-model to Jess

In this section we work on a new model transformation which is the transformation of a UML model to relational schema model.

Figure 3.15 shows the transformation of a partial view of the relational schema meta-model. As indicated by the arrows, the *EClasses* table and column are transformed into templates. The *EAttribute* name is also converted to slot in each template. The *EReference* between table and column is transformed to a template which contains two slots containing the names of source and target elements of the *Ereference*.

3.3.3.2 Models2Facts

Step 2 aims at transformaing models into facts. A model is an instantiation of its meta-model. Accordingly, each instance of a meta-class present in the model is transformed into a fact with the same name. The instances of meta-attributes are transformed into slot values of the corresponding template. Each instance of meta-reference between two instances of meta-classes is also transformed into a fact which contains the names of relation elements.

A simple transformation example is presented in Figure 3.16. The three instances of metaclasses (the table and the two columns) are transformed into three facts. The two instances of meta-relations (from table to column) are transformed into the two facts instanciating the template *RelTabCol*.

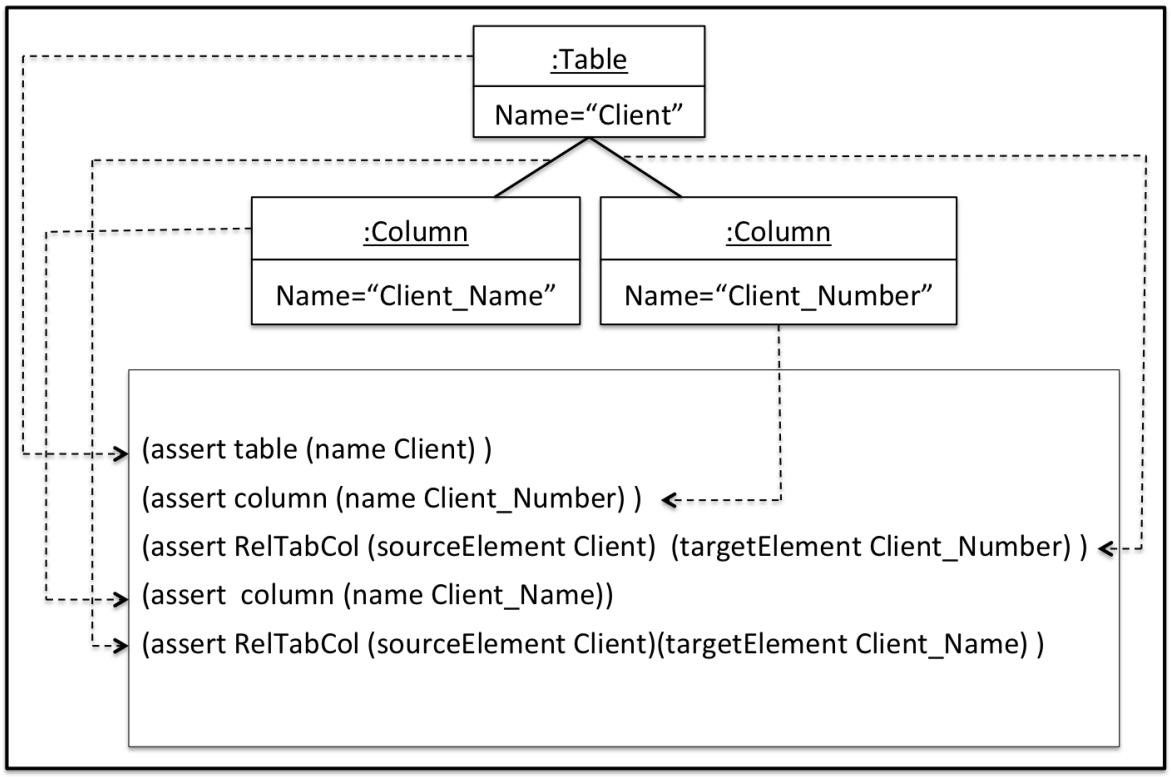

Figure 3.16 – Transformation of a partial view of relational schema model to Jess

3.3.3.3 TransformationPatterns2JessRules

Step 3 consists in the actual rule generation from transformation patterns. As it can be seen in Figure 3.17, there is a similarity between transformation-pattern structure and Jess-rule structure. Both of them are composed of two main parts. The premise of a pattern is equivalent to the LHS of a rule. Both describe the situation to find to fire the rule or to apply the transformation pattern. Similarly, the conclusion is equivalent to the RHS. Both are the action to perform or the conclusion to reach when the first part is satisfied.

The premise is a description of a set of source elements. These elements are linked together. Consequently, each element in the premise is transformed into a Jess condition corresponding to the test of the presence of a fact. As the premise elements are not named, we generate a slot name for each element. When more than one element is involved, conditions corresponding to relations are also generated. As relations do not have names, we named it by concatenating the three first letters of the relation elements names.

The conclusion of a transformation pattern is a description of a set of target elements together with their relations. It it similar to the premise. Consequently, each element in the conclusion is transformed into a Jess fact assertion. Names and relations between facts are also generated.

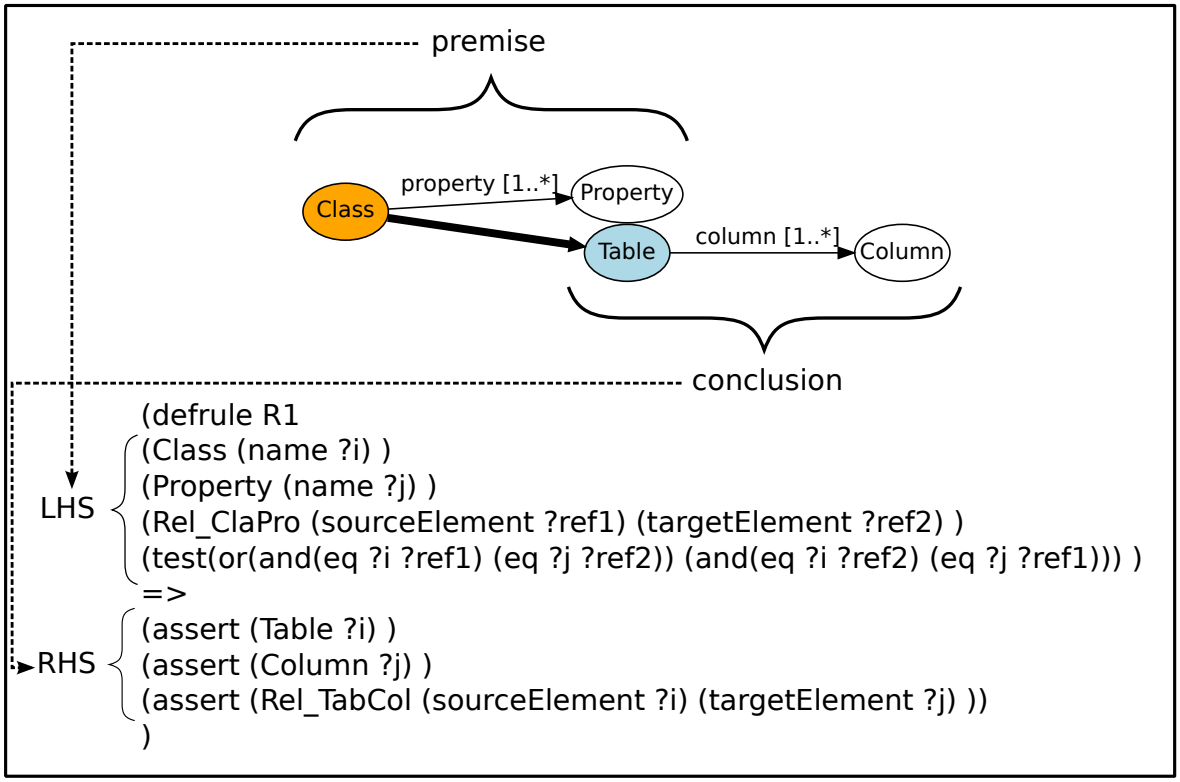

Figure 3.17 – Example of the transformation of a pattern into Jess

Figure 3.17 shows the transformation into a Jess rule of an example of transformation pattern (from UML class diagram to relational schema model). The premise of the transformation pattern is a class linked to a property. The corresponding Jess rule has for LHS four conditions, respectively checking: the existence of a class i, the existence of a property j, the existence of a relation from class to property, and that the existing relation links i to j. The conclusion of the transformation pattern is a table linked to a column. The corresponding RHS of the generated Jess rule contains three fact assertions, respectively stating: a table i, a column j, and a relation from i to j.

3.3.4 Tool Support and Case study

This section illustrates the rule generation process using a case study. It also reports on the efficiency of our approach through classical precision/recall measures. Like for testing, we compare the target models produced by our executable rules with the expected models. Precision and recall show to what extent the inferred rules perform the correct transformations.

Our case study concerns the transformation of class diagrams into relational schema. The rule generation is performed starting from a set of 30 examples of class diagrams and

their corresponding relational schema. Some of them were taken from [Kessentini, 2010], the others were collected from different sources on the Internet. We ensured by manual inspection that all the examples conform to valid transformations.

To take the best from the examples, a 3-fold cross validation was performed, *i.e.*, 30 examples divided into three groups of 10. For each fold, two groups (20 examples) were used for generating the rules, and the remaining third group was used for testing them. Each fold used a different group for testing. Testing consists in executing the generated rules on the source models of the testing examples and in comparing the obtained target models with those provided in the examples. This comparison allows calculating the precision (Equation 1) and the recall (Equation 2) measures.

We calculate precision and recall separately for each type T of fact (table, column, etc.).

$$P(T) = \frac{number\ of\ T\ with\ correct\ transformation}{total\ number\ of\ initial\ T} \tag{3.1}$$

$$R(T) = \frac{number\ of\ T\ with\ correct\ transformation}{total\ number\ of\ generated\ T} \tag{3.2}$$

Table 3.1, Table 3.2 and Table 3.3 show precision and recall averages (on all fact types) of the 10 generated transformations for the 3-folds. The precision and recall averages are higher than 0,70 in all cases. Some models were perfectly transformed (precision=1 and recall=1). For the others, the precision and recall could be better than the ones calculated automatically. This is due to the case of elements which have more than one transformation possibility. For example, if we have a generalization between two classes, we can transform it into a simple table which contains the attributes of general and specific classes. The second transformation method is to transform it into two tables. So, in the case of generalization, two rules are applied and this decreases the precision and the recall. The same problem exists for the aggregation which has also two transformation possibilities (1 or 2 tables).

Discussion

The study presented in this section is a first evaluation of our approach. This evaluation is a proof-of-concept to check if RCA-based derivation and pattern-to-rule mapping are effective. In this context, the obtained results are very satisfactory. They show that the

Examples	Fold1	
	Precision Average	Recall Average
1	1	1
2	0.77	0.75
3	0.70	0.75
4	0.94	0.75
5	1	1
6	1	0.77
7	0.88	0.77
8	1	0.77
9	0.90	0.77
10	0.90	0.85

Table 3.1 – Result of the first fold cross validation

Examples	Fold2	
	Precision Average	Recall Average
1	0.78	0.79
2	0.90	0.75
3	0.85	0.77
4	0.77	0.79
5	1	0.80
6	1	0.77
7	0.85	0.77
8	0.85	0.80
9	1	0.75
10	1	0.80

Table 3.2 – Result of the second fold cross validation

Examples	Fold3	
	Precision Average	Recall Average
1	0.80	0.75
2	1	1
3	1	0.85
4	1	0.80
5	0.77	0.75
6	1	0.77
7	1	1
8	1	0.80
9	0.85	0.77
10	0.88	0.80

Table 3.3 – Result of the third fold cross validation

proposed approach allows to find most of the expected transformation rules and that these rules are executable on actual models.

To help us improving the rule generation process, additional experiments have to be conducted, in particular to study the two following issues:

- First, we used a small number of examples, based on small meta-models. Larger meta-models and more numerous examples have to be considered in the future to draw a better portrait on the strengths and weaknesses of the approach.
- Second, we measured the correctness of the obtained model transformation by comparing elements of the produced and expected models without considering their relations. A better and comprehensive correctness measure should be defined in the future.

3.4 Strategies for learning Model Transformations from Examples

In this section, we analyze and compare two strategies for learning the transformation patterns with RCA. In the first one, each example is used alone to learn transformation patterns, and the transformation patterns obtained from all the examples are then gathered. In the second strategy, the examples are first gathered into a single large example, that is then used to learn the transformation patterns. The obtained transformation patterns are inspected and applied to test examples.

Our case study concerns the transformation of class diagrams into relational schema. The rule generation is performed starting from a set of 30 examples of class diagrams and their corresponding relational models. Some of them were taken from [Kessentini, 2010], the others were collected from different sources on the Internet. We ensured by manual inspection that all the examples conform to valid transformations. To take the best from the examples, a 3-fold cross validation was performed. We divide the j ($j \in 1..30$) examples into three groups of 10. For each fold i ($i \in 1..3$), we use two strategies to produce transformation rules:

- In the first one, we use the experimentation of section 3.3.4 which consists of using two groups (20 examples) separately for generating 20 pattern lattices (denoted l_{ij}). The l_{ij} lattices are analyzed and simplified, as explained in Section 3.3.2, to select automatically the relevant transformation patterns. Then, we transform them into operational rules written for Jess. The remaining third group is used for testing

them. Testing consists in executing the generated rules on the source models of the testing examples and in comparing the obtained target models with those provided in the examples.

- In the second one, we gather two groups (20 examples) for generating only one lattice of patterns (denoted L_i). L_i is analyzed and simplified to select automatically the relevant patterns. Those patterns are then transformed into operational rules. The remaining third group is used for testing them.

The goal is to compare in each fold i the results obtained from the two strategies. First, we compare the lattices generated from examples (lij), and the lattice generated from the union of those examples (Li). Then, we compare the results of executing the rules obtained from each strategy on the source models provided in the testing examples.

lij **vs** L_i

Compared to the first strategy, which produces small size lattices (from each lij we have about 9 patterns before simplification and 4 patterns after simplification), the second one produces large ones (from each L_i we have about 100 patterns before simplification and 50 patterns after simplification). Although the lattices L_i are larger and more difficult to analyze, they have more specific and complete transformation patterns compared to lij which are simple to analyze but contain transformation patterns that are proper to their examples. A single pattern of L_i can combine several patterns that exist in l_{ij}.

Figure 3.18 shows examples of different patterns obtained from l_{1j}. For instance, in the pattern of Figure 3.18(a), a transformation link is given to specify that a *class* linked to an *aggregation* is mapped into a *table* linked to *primary foreign key*. Pattern of Figure 3.18(b) shows that a class linked to a *property* is transformed into a table linked to a *column*. In the last pattern of Figure 3.18(c), the transformation specifies that a class linked to a property and a *generalization* is transformed into a table linked to a column and a *foreign key*.

If we compare these patterns with the pattern of lattice L_1 in Figure 3.19, we note that the information contained in the three patterns exists in the pattern of Figure 3.19. It is more complete. It combines all the information of transformation existing in Figure 3.18(a), Figure 3.18(b) and Figure 3.18(c).

So, if we combine various examples together, the generated lattice contains patterns which are more specific and combine different information. But, if we test each example separately, the obtained lattice contains less information. In addition, L_i contains all the patterns needed to transform a class diagram to a relational schema. The lattices l_{ij} contain

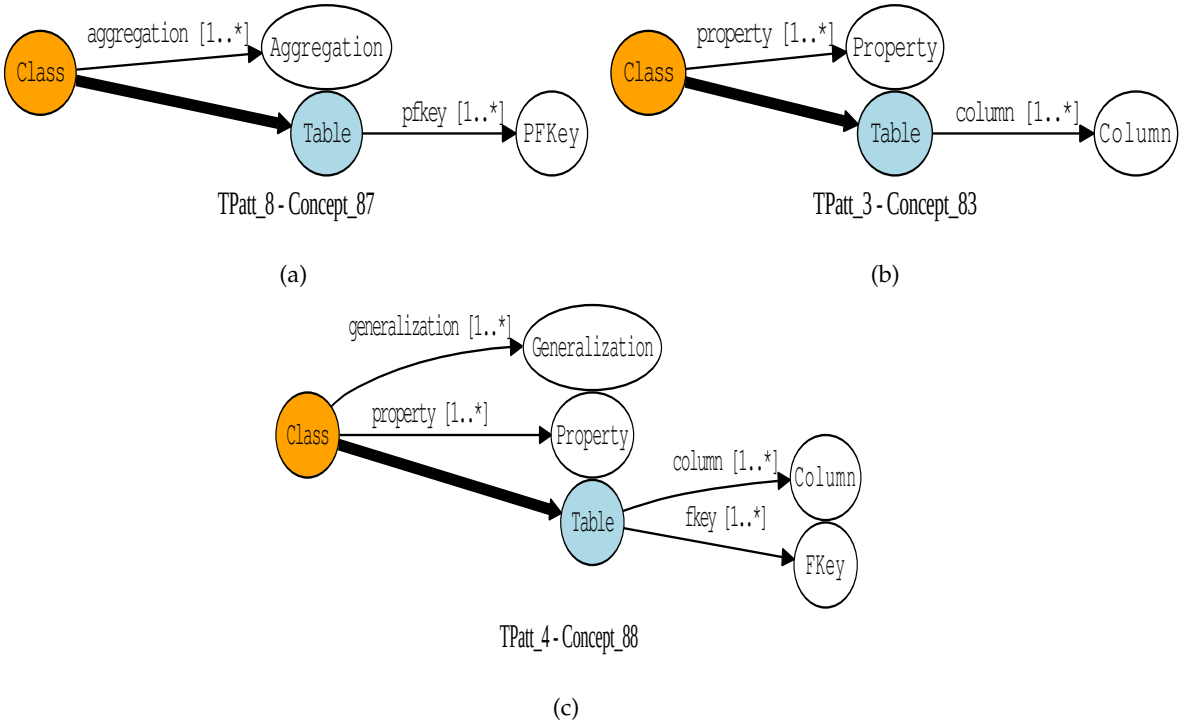

Figure 3.18 – Examples of transformation patterns extracted from lattices L_{l1}

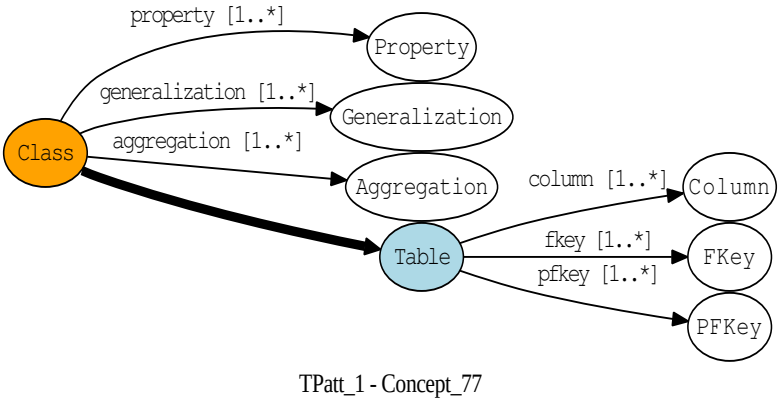

Figure 3.19 – Example of transformation pattern extracted from lattice L_1

just the transformation pattern proper to the transformation examples used. So, we need to merge several transformation examples to obtain all transformation rules of a class diagram into a relational schema.

Furthermore, in each fold, a L_i lattice contains about 50 transformation patterns and the union of l_{ij} produces about 40 ones (4 transformation patterns * 20 minus the redundant ones which exist). If we examine the patterns as an expert, we note that L_i contains about 12 relevant transformation patterns which are useful to the transformation. But they

Examples	Fold1			
	Recall Average		Precision Average	
	First Strategy	Second Strategy	First Strategy	Second Strategy
1	1	0.5	1	0.5
2	0.77	0.45	0.75	0.43
3	0.70	0.5	0.75	0.43
4	0.94	0.43	0.75	0.32
5	1	0.45	1	0.43
6	1	0.5	0.77	0.23
7	0.88	0.43	0.77	0.40
8	1	0.6	0.77	0.43
9	0.90	0.5	0.77	0.44
10	0.90	0.5	0.85	0.45

Table 3.4 – Result of the first fold cross validation

are less detailed and not applicable for all example types. On the other side, the union of l_{ij} contains about 10 relevant transformation patterns. Those patterns are easy to read and to apply because each one contains a piece of information of the transformation compared to L_i'rules which combine several pieces of information in the same pattern.

lij's rules execution vs L_i's rules execution

In this section, we compare the result of executing the rules obtained from the two strategies, which are transformed into Jess rules, on the source models provided in the testing examples. This comparison allows calculating the recall (Equation 1) and the precision (Equation 2) measures for each T. T represents the type of elements in the target meta-model (table, column, foreign key...)

$$R(T) = \frac{number\ of\ T\ with\ correct\ transformation}{total\ number\ of\ initial\ T} \tag{3.3}$$

$$P(T) = \frac{number\ of\ T\ with\ correct\ transformation}{total\ number\ of\ generated\ T} \tag{3.4}$$

Table 3.4, Table 3.5 and Table 3.6 show precision and recall averages on all element types of the 10 generated transformations for the 3-folds. As mentioned in Section 3.3.4, the precision and recall averages are higher than 0.7 in the first strategy. Some models were perfectly transformed (precision=1 and recall=1). Precision and recall decrease in the case of elements which have more than one transformation possibility. For example, if we have

Examples	Fold2			
	Recall Average		Precision Average	
	First Strategy	Second Strategy	First Strategy	Second Strategy
1	0.78	0.5	0.79	0.4
2	0.90	0.45	0.75	0.31
3	0.85	0.45	0.77	0.43
4	0.77	0.43	0.79	0.40
5	1	0.5	0.80	0.34
6	1	0.43	0.77	0.47
7	0.85	0.4	0.77	0.37
8	0.85	0.45	0.80	0.43
9	1	0.5	0.75	0.34
10	1	0.5	0.80	0.33

Table 3.5 – Result of the second fold cross validation

Examples	Fold3			
	Recall Average		Precision Average	
	First Strategy	Second Strategy	First Strategy	Second Strategy
1	0.80	0.49	0.75	0.33
2	1	0.44	1	0.34
3	1	0.5	0.85	0.44
4	1	0.45	0.80	0.44
5	0.77	0.40	0.75	0.35
6	1	0.5	0.77	0.40
7	1	0.4	1	0.33
8	1	0.33	0.80	0.23
9	0.85	0.35	0.77	0.3
10	0.88	0.4	0.80	0.39

Table 3.6 – Result of the third fold cross validation

a generalization between two classes, we can transform it into two tables or into a simple table which contains the attributes of general and specific classes. In this case, two rules are applied on the same example and this affects the performance results.

In the second strategy, precision and recall averages are low (less than 0.5) in the 3-folds. This is due to the fact that the generated rules are very large and contain different informations from different examples. Thus, the premises of the rules can not be matched for most of the examples because the examples are simple and do not contain all the transformation cases that have been learned. So, the Jess rule engine does not apply a part

of the rule premise when it is executed on an example, it searches for each example its corresponding rule and this decreases the precision and the recall.

3.4.1 Discussion

The study presented in this section is a comparison of two strategies for generating transformation rules using RCA. The first consists to generate from each example its rule lattice and the second consists to gather all the examples and generate only one rule lattice. Each one has its advantages and disadvantages:

- The first strategy produces simple and small transformation patterns which are easy to analyze and to manipulate, but they are proper to their examples. On the other side, the second one produces larger patterns but they are more specific and more complete. They combine different information about the transformation. An analysis on those patterns shows that the two strategies have the same number of relevant patterns (about 12 transformation patterns). The relevant ones of the first strategy are simple and applicable for each example. On the contrary, the relevant patterns of the second strategy are larger and mainly applicable for larger examples.
- The execution of the rules of the first strategy gives good results in our experiment. The rule engine searches and finds for each example the set of rules to apply. On the contrary the rules of the second strategy are more large and contain more information. Thus the rule engine does not find a rule to apply for the simple examples. We can work again on the obtained rules of L_i to execute them on all types of examples (for example by separating into smaller pieces), but as we found good results with the union of l_{ij}, it is not a promising track. We obtained a non-intuitive result: before the experimentation, we thought the best rules would be obtained with the second strategy.

Although the example used is a classical one, it is a good example of typical model transformations that we aim to learn. To confirm what is the best strategy to produce transformation rules, additional experiments have to be conducted with other model transformation kinds. Besides, the obtained result depends on the models on which we execute the rules. If we use larger models, the second strategy may have better results.

3.4.2 Summary

In this section, we studied an approach for inferring model transformations (composed of transformation rules) from transformation examples. We compare two strategies for ap-

plying this approach: inferring the rules from the example taken separately (then gathering the rules), or inferring the rules from the gathering of the examples. Although we thought the second strategy would produce better rules (more detailed), it appeared that the rules (less detailed) produced by the first approach execute better.

3.5 Conclusion

In this chapter, we presented an approach that aims at deriving model transformation rules from a set of model transformation examples.

A first step of the approach uses a data analysis method, RCA, to learn recurrent transformation patterns. In the second step, the transformation patterns are filtered, refined, and automatically transformed into Jess rules. Those rules constitute the expected transformation. Provided that meta-models and models are written as Jess facts (which is done by automatic transformation), the rules can be executed by the Jess engine to actually transform models. The approach is successfully evaluated on a case study used in previous research work.

An experimentation is also done to compare two strategies for generating transformation rules. The first strategy consists of generating from each example its rule lattice and then its transformation rules and the second strategy consists of gathering all the examples and generating only one lattice and their transformation rules. Results show that the first strategy produces better rules.

As future work, we plan to transform the obtained Jess facts (after rule application) to produce models conforming to the initial meta-models. Furthermore, future work includes learning rules whose premise and conclusion have several main elements and design heuristics to determine the best rule to apply when several rules are candidate for the same model element.

4 Model Transformation Traceability and Model Matching: metaheuristic approaches

Contents

We presented in Chapter 3 how to generate operational transformation rules from examples of model transformations. Our approach is successfully applied and evaluated on different examples.

In this chapter, we focus in recovering transformation traces from transformation examples. This trace recovery is useful for several purposes as locating bugs during the execution of transformation programs, or checking the coverage of all input models by a

transformation. As our goal in this thesis is to learn model transformations, we expect also that this trace will provide data for a future model transformation learning technique. We first address the trace recovery problem with examples coming from a transformation program. Our approach is based on multi-objective meta-heuristics. It takes as input source and target models. The recovered transformation traces take the form of *many-to-many* mappings between model elements from the source and the target models. Then this approach is refined to apply it to the more general problem of model matching.

This chapter is organized as follows. Section 4.1 introduces model transformations recovering traces. Section 4.2 is dedicated to the problem statement and the overview of our approach. While Section 4.3 describes our approach in detail, Section 4.4 explains the experimental evaluation of recovering traces approach on different examples and the obtained results. In Section 4.5, we present how to extend and apply our approach on the model matching problem and we conclude the chapter in Section 4.6.

4.1 Introduction

MDE involves the construction and manipulation of many models of different kinds in an engineering process [Paige *et al.*, 2011]. These models cover the whole software-development cycle, *e.g.*, requirement, design, implementation, maintenance, etc. They can be manipulated manually or automatically using model transformations. To ensure those transformation models' consistency and maintenance, recovering transformation trace can be essential in an MDE process.

Recovering transformation traces can be useful for different tasks [Grammel *et al.*, 2012]: (1) understanding the system complexity by the navigation on trace links on all the model transformation chains, (2) locating bugs during the execution of transformation programs, and (3) checking the coverage of all input models by a transformation. We can distinguish between two categories of strategies to generate transformation links: the first category [OMG, 2005], [OMG, 2006], [Bézivin *et al.*, 2003], [Jouault, 2005], [Grammel and Kastenholz, 2010], [Kurtev *et al.*, 2007], [Amar *et al.*, 2010], [van Amstel *et al.*, 2012] depends on the transformation program or engine. The corresponding approaches generate trace links through the execution of a model transformation. The second category consists in generating a transformation trace independently from a transformation program. Reference [Cysneiros *et al.*, 2003] proposes a mechanism to generate traces for a transformation from a requirement model conform to the meta-model i* towards the metamodel UML (use cases

plus class diagram), thus this approach is specific to two metamodels and cannot be generalized. [Grammel *et al.*, 2012] proposes an approach based on matching techniques to trace links between models. However this approach is not applied on large size models and only generates *one-to-one* matching links between models.

In this chapter, we are interested in the second category. We propose to recover a transformation trace independently from a transformation program. We consider that the transformation program is missing or the transformation was done manually. Our approach takes as input a source model and its corresponding target model. The aim is to find the *many-to-many* trace links between the two models, thus associating a group of m source elements to a group of n target elements. To this end, the source model is fragmented using the minimal cardinalities of its meta-model and the defined OCL constraints. Then, we search for each source fragment, the list of potential transformed fragments in the target model. A solution to our problem is a set of pairs of source and target fragments that maximize the lexical and structural similarities between them. A solution must also cover all the target model to ensure its transformation completeness. Due to the very large number of possible solutions, a multi-objective metaheuristic method (NSGA-II) is used to solve our problem.

4.2 Approach Overview

This section shows how recovering a transformation trace between a source model and a target model can be defined as an optimization problem. We also show the importance of heuristic search to explore the large space of possible transformation traces between the two models. Then, we present the main principles of our approach.

4.2.1 Problem Statement

Before describing the problem, let us start by defining key concepts involved in our work.

Definition 4.1: *In our context, a metamodel MM is an instance of the Ecore [Frank, 2004] meta-metamodel.*

Definition 4.2: *A model M is an instance of a metamodel MM.*

Our approach is based on a fragmentation of the source and target models.

Definition 4.3: *A fragment F is a set of connected constructs of a model M. A construct $e \in M$ is an instance of a meta-class $C \in MM$ ($\underline{e : C}$).*

We denote by $Frag(M)$ the set of all fragments that can be built from M.

We denote by $R\langle C_1 : \underline{R}, \overline{R} : C_2\rangle$, an e-reference of C_1, which has C_2 as a type, and such that $\underline{R}$ (resp. $\overline{R}$) is the minimal (resp. maximal) cardinality of R. For $R\langle C_1 : \underline{R}, \overline{R} : C_2\rangle$, eRe' means that we have $\underline{e : C_1}$, $\underline{e' : C_2}$ and e is connected to e' by R.

Definition 4.4: *A meaningful fragment MfF of a model M is a fragment that respects the minimum cardinalities of the references defined on the metamodel and the OCL constraints.*

Consequently, a fragment F of a model M which conforms to a metamodel MM (which is provided with a set of OCL constraints) is a MfF iff:

$$\forall \underline{e : C_1} \in F, (\exists C_2 | R\langle C_1 : \underline{R}, \overline{R} : C_2\rangle \in MM) \Rightarrow |\{\underline{e' : C_2} \in F | eRe'\}| \geq \underline{R} \text{ and the OCL constraints about minimum cardinalities are satisfied.}$$

We denote by $MeanFrag(M)$ the set of all meaningful fragments that can be built from M.

Let us consider the instance diagram of Figure 4.2 that conforms to the simplified UML class diagram metamodel $CDMM$ of Figure 4.1. Three fragments are circled, F_1, F_2, and F_3. F_1 (resp. F_2) is a meaningful fragment because it satisfies the minimal cardinalities defined on association (resp. generalization) meta-class in $CDMM$. An association must have two properties of type class. So, association *Reservation*, property *client* of type *Client* class and property *sRoom* of type *SimpleRoom* class form a meaningful fragment (F_1). A generalization consists of a relation between a general class and a specific class. Thus, the generalization between the Class *Room* and the class *SimpleRoom* constitutes a meaningful fragment (F_2). F_3 is composed of two connected constructs in the instance diagram (the association *Reservation* and its property named *client*). But in the metamodel $CDMM$, an association must have two properties, each one having a type class (according to the OCL constraint shown in $CDMM$). Thus, although F_3 conforms with the definition of fragment, it is not a meaningful fragment, because it violates the cardinality and OCL constraints in $CDMM$.

Definition 4.5: *A transformation trace between a source model M_s and a target model M_t is a set of pairs connecting a source meaningful fragment to a target fragment. A specific transformation trace of n pairs takes the following form $\{(MfF_{s_i}, F_{t_i}) \mid i \in \{1..n\}\} \subseteq MeanFrag(M_S) \times Frag(M_T)$.*

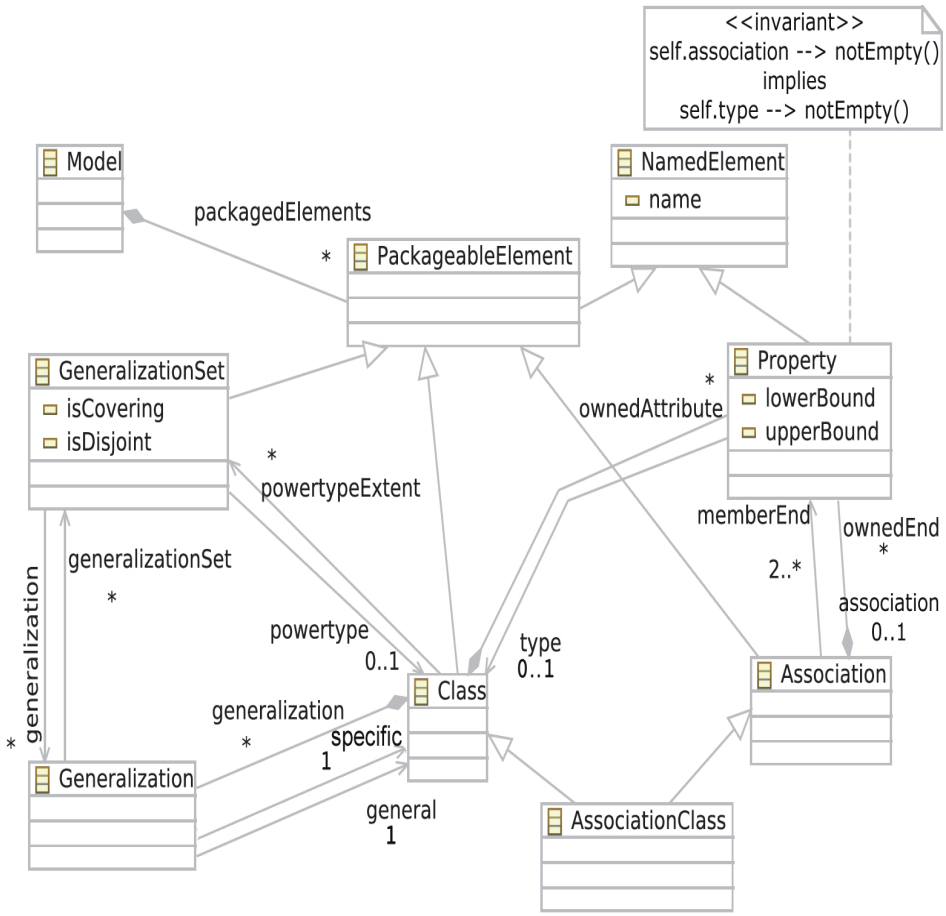

Figure 4.1 – A meta model for UML class diagrams

The size of the set of possible transformation traces is $2^{MeanFrag(M_s) \times Frag(M_t)}$. Searching this space, *i.e.*, finding the best match between each MfF in M_s with an F in M_t, is hard to perform with an exhaustive search method. This led us to use a metaheuristic search to solve the trace recovery problem.

4.2.2 Approach Overview

Our goal is to generate a transformation trace from two given source and target models. Figure 4.3 shows an overview of our approach. A first step consists in the decomposition of the source model into meaningful fragments according to the constraints of the metamodel. Then, a metaheuristic method is used to search for the best match between the identified source meaningful fragments and all the possible target fragments. To evaluate the quality of a match (candidate trace) two factors are considered:

- Lexical similarity between fragments in each pair, *i.e.*, both source and target fragment use similar identifiers.

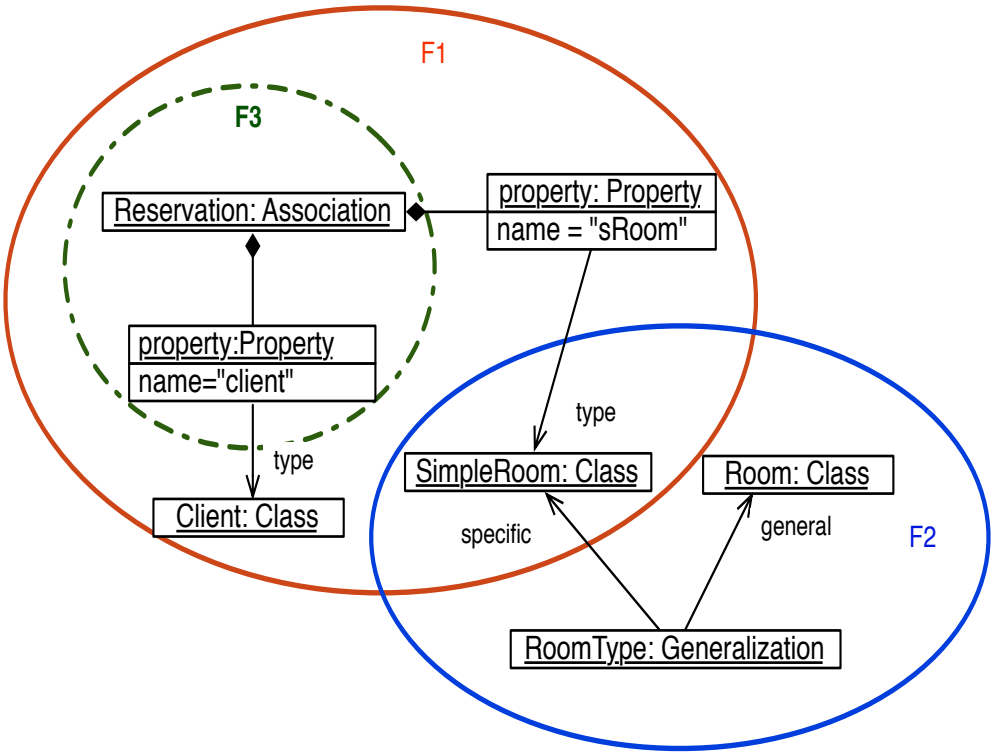

Figure 4.2 – An instance diagram of the class diagram metamodel of Figure 4.1

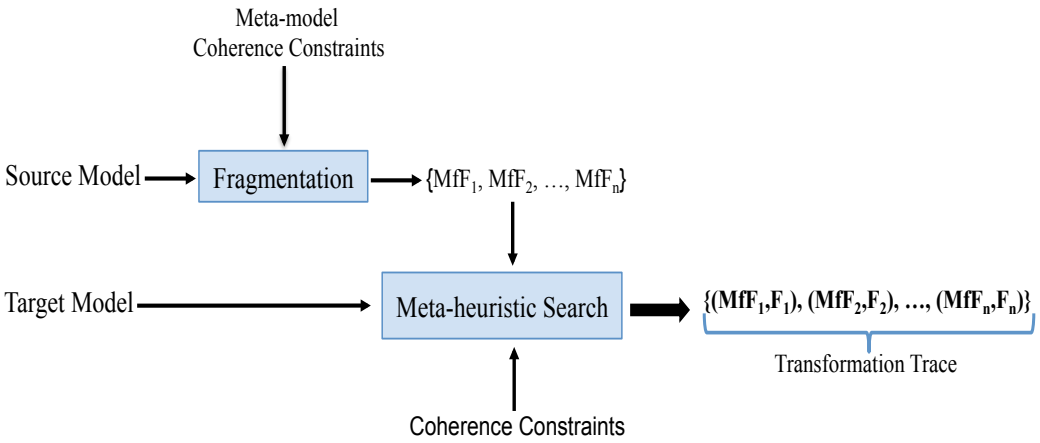

Figure 4.3 – Approach overview

- Structural consistency in the mapping of similar source fragments, *i.e.*, similar source fragments should be associated to similar target fragments.

In addition to the lexical and structural factors, to be acceptable, a candidate trace:

$$Trace = \{(MfF_{s_i}, F_{t_i}) \mid i \in \{1..n_{Trace}\}\} \subseteq MeanFrag(M_s) \times Frag(M_t)$$

should satisfy the completeness constraints that could be formalized as follows:

$$(\bigcup_{i=1}^{n_{Trace}} MfF_{s_i} = M_s) \wedge (\bigcup_{i=1}^{n_{Trace}} F_{t_i} = M_t)$$

Figure 4.4 illustrates an example of transformation trace between a simplified UML class diagram and its corresponding entity-relationship model. Note that the choice of this example is only motivated by clarity consideration, our approach does not depend from specific source and target metamodels. The shown trace can be described as follows:

- The class diagram is decomposed into three meaningful fragments according to the constraints of the metamodel of Figure 4.1.
- In terms of lexical similarity, MfF_1 matches well F_1 as they both contain the identifiers (*Text*, *title*, *Poem* and *Novel*). Comparable lexical similarities could be observed respectively between MfF_2 and F_2, and between MfF_3 and F_3.
- In terms of structural consistency, MfF_2 and MfF_3, which are fragments of the same type (a one-to-many association between two classes) are consistently matched to two fragments F_2 and F_3, which are also of the same type (a relation between two entities).

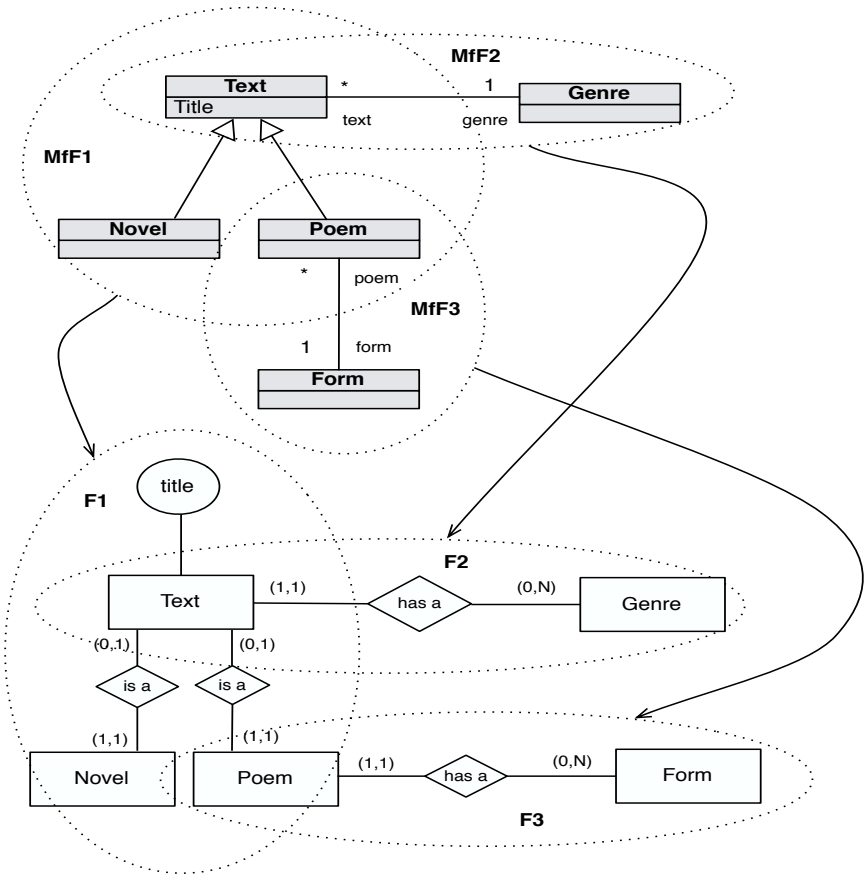

Figure 4.4 – An example of transformation links between a class diagram and an entity-relationship model

To find the good transformation trace between a pair of source and target models, we perform a heuristic search guided by the lexical and structural factors as well as by the completeness constraints. Thus, the trace recovery can be seen as a multi-objective optimization problem. To implement our approach, we select NSGA-II algorithm (introduced in Chapter 1) and adapt it to our specific problem (see details in Section 4.3).

4.3 Adapting NSGA-II to the Transformation Trace Recovery Problem

In this section, we describe the adaptation of NSGA-II to recover a transformation trace from a pair of source and target models. Applying this type of algorithm to a specific problem requires to specify the encoding of a solution, the fitness functions (one per objective), to guide the search process, and the operators to derive new solutions from existing ones. These elements are respectively detailed in subsections 4.3.1, 4.3.2 and 4.3.3.

4.3.1 Solutions Representation

A crucial element in our approach is the encoding of a transformation trace between the source and target models for each candidate solution. The solution encoding impacts both the fitness evaluation and the derivation operators. In our case, a solution s, is a set of fragment pairs, $s = \{fp_i, i \in \{1, 2, ...n_s\}\}$. Each fragment pair fp_i is, in turn, encoded as a pair $fp_i = (MfF_i, F_i)$ where MfF_i is a source meaningful fragment in the source model and F_i is its corresponding fragment in the target model.

As stated in Section 4.2, the source model is divided into fragments according to three criteria: 1) compliance with the minimum cardinalities for the references defined in the source meta-model, 2) compliance with the OCL constraints specified in the source meta-model and 3) source model coverage, *i.e.*, each construct in the source model must belong to at least one fragment. The target model is randomly divided to associate a fragment with each source meaningful fragment. We conjecture that:

- Dependent constructs (with cardinality constraints) in the source model have to be transformed together and the corresponding constructs in the target model do not necessarily need to form a meaningful fragment.
- When transforming a source model, some constructs may not have corresponding constructs in the target model. Furthermore, some new constructs in the target model may be created independently from the source ones.

Thus, after obtaining the n_s source meaningful fragments, an integer x is generated randomly such that $-y < x < y$. y is a parameter of our algorithm which is the maximum variation of the number of target fragments with respect to the source ones. Then, the target model is divided into $n_s + x$ fragments.

More concretely, for each target-model fragment f, we start by randomly choose an integer $1 \leqslant t \leqslant 4$ such as t corresponds to the size of f (thus we have a maximum of 4 constructs in a fragment). *i.e.*, if $t = 3$, we select randomly a construct, call it c, from the target model. Then, if c is connected to other constructs, we extend the fragment by randomly selecting two of them. If c is connected to just one construct c_1, we can extend the fragment by one of the constructs connected to c_1. Then, c is removed from the set of potential starting constructs for the next fragments. Nevertheless, c can still be included in other fragments thanks to its connections with other constructs.

When both source and target fragment sets are created, each source MfF is randomly associated with a target model fragment F. A solution is then a vector whose dimensions are the MfFs and values are the Fs. Figure 4.5 shows another transformation trace candidate s randomly generated by our algorithm in addition to the one proposed in Figure 4.4. $s = \{(MfF_1, F_3), (MfF_2, F_2), (MfF_3, F_1)\}$.

To create the initial population of N solutions, our algorithm randomly generates a set of solutions s_i, $i \in \{1, 2, ..N\}$. Each s_i represents a possibility of target-model fragmentation and association of the resulting fragments to the meaningful ones of the source model.

4.3.2 Solutions Evaluation

The fitness functions evaluate a candidate trace solution. We defined three fitness functions corresponding to three objectives:

1. An MfF in a source model corresponds to a F in a target model when MfF and F use similar vocabulary, *i.e.*, are similar in terms of properties values of type string.
2. In a solution s, a set of MfF which have the same type, *i.e.*, same construct types with the same connections, must be matched to a set of F in the target model which have the same type.
3. In a solution s, the obtained fragments must cover the target model.

The two first objectives approximate the semantic equivalence between model fragments belonging to two different metamodels. The third objective ensures that a solution

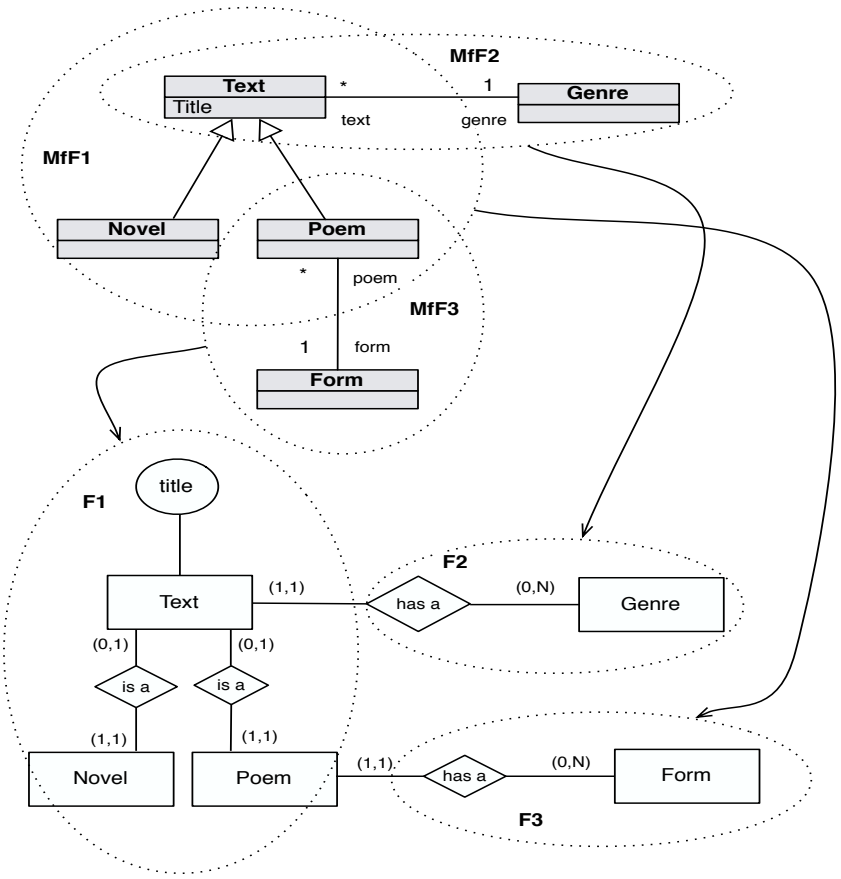

Figure 4.5 – A trace randomly generated for the models of Figure 4.4

is complete as it recovers all the transformation trace. The three objectives should be maximized.

Lexical similarity. For this fitness function, we take our inspiration from information retrieval methods, which sort documents according to queries by extracting information about the terms' occurrences within documents. The extracted information is used to find the similarity between queries and documents. In our case, the similarity is used to compare the property values of MfF_i and F_i in each fp_i in a solution s. All the terms (distinct property values) in s are extracted in a list l. l defines the dimensions of vectors associated to each source or target fragment in s. For each fragment and each term, the corresponding dimension is set to 1 if the term exists in the fragment or to 0 otherwise. Then, the similarity is calculated between each pair MfF_i and F_i using the cosine similarity between the two concerned vectors. The resulting similarity ranges from -1, meaning that MfF_i and F_i do not share any term, to 1, meaning that MfF_i and F_i use exactly the same terms. The lexical similarity $LexSim(s)$ of a solution s is equal to the average of the contained pairs' lexical similarities.

Structure similarity. In order to measure the structure similarity in a solution s, we start by classifying the set of its fragment pairs per type of their respective meaningful fragments MfF_i.

After classifying the solutions per type of their MfF, we measure for each two pairs of fragments, which have the same type of MfF, the structural similarity of the matched target models. To this end, we use also the cosine similarity, but between vectors whose dimensions are the construct types in the metamodel. Indeed, for each construct type instantiated in the target model, a term is created. Then for each target model fragment, the dimension is set to 1 if it contains a construct of the corresponding type, and to 0 otherwise. The structural similarity $StrSim(s)$ of a solution s is the average of the target-fragment similarities of the pairs having the same MfF type.

Target model coverage The coverage of the target model is the most important objective because it ensures that the fragments obtained in the solution cover all the target model. The coverage $Cov(s)$ of a solution s is measured by the number of distinct constructs in the matched target fragments divided by the number of constructs in the target model.

4.3.3 Operators Definition

In NSGA-II, in each iteration, the N solutions selected from the previous generation are used to create N new solutions to complete the population using genetic operators. This is done to improve the existing solutions by mixing their genetic material (crossover) and/or by creating new material (mutation). Before applying the operators, the solutions are selected according to their fitness values.

Binary Tournament selection. In our work, binary tournament selection is used. It consists in randomly choosing some solutions in the population, and selecting the fittest two for reproduction. The selection criteria are the rank of the containing front and the crowding distance for solutions within the same front. Several tournaments are ran to produce the N needed solutions.

Crossover operator. The crossover consists in producing new solutions from the existing ones. When two solutions are selected using the binary tournament method, two offspring solutions are created, with a given crossover probability, by exchanging parts of the parent solutions. This consists in randomly deciding for a cut point in the solution vector, and all the target fragments beyond that point in either parent is swapped between the two parents. For instance, Figure 4.6 shows an example, where two offspring solutions

(s_k and s_l) are obtained after exchanging the target fragments between the traces s_i and s_j beyond the point cut x=1.

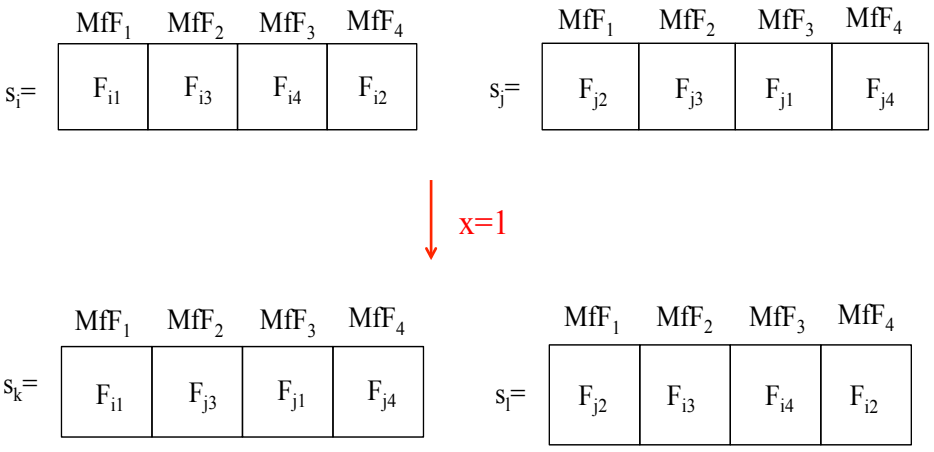

Figure 4.6 – An example of crossover

Mutation operator. After performing the crossover, the obtained solutions could be mutated with a given mutation probability. Mutation allows the introduction of new genetic material while the population evolves. For our problem, we define two mutation strategies: extending a target fragment with a new construct or deleting a construct from a target fragment. We recall that a transformation-trace solution is a set of fragment pairs; each one contains a source and a target fragment (MfF and F). Like a MfF, a target fragment contains constructs connected with references. For the first mutation strategy, a pair $fp_i = (MfF_i, F_i)$ is randomly chosen. Two kinds of construct can be added to fp_i: a randomly chosen construct, not already included in F_i, or a construct which has a reference to another one in F_i. For instance, in the second case, a construct c_1 is selected randomly in F_i. If c_1 has references to several constructs, we randomly choose a connected construct c_2 in the target model M_t which is not already included in F_i, and we add c_2 to F_i.

The second mutation strategy consists in deleting a construct from a target fragment F_i in a pair $fp_i = (MfF_i, F_i)$ also randomly chosen from a solution. We randomly select a construct c from F_i. Then, we check if c is connected at least to one construct in F_i. If we find that c is not linked to any construct in F_i, it will be deleted.

The two mutation strategies can be randomly combined to produce more variations. For each offspring solution, one of the following strategies can be applied: (i) adding a construct, (ii) deleting a construct, (iii) adding a construct and deleting a construct and (iv) adding two constructs and deleting one construct. For instance, let us come back to the obtained solution s of Figure 4.5. $s = \{(MfF_1, F_3), (MfF_2, F_2), (MfF_3, F_1)\}$. Suppose that

the pair $(Mf F_2, F_2)$ is selected for mutation. Then, suppose that the relationship *has a* is picked. If the first mutation strategy is applied, the relationship *has a* has a reference to the class *Text* which is not included in F_2. Then, the mutation adds this class to the fragment F_2.

4.4 Evaluation

To evaluate the feasibility of our approach for recovering transformation traces, we conducted an experiment on six model transformations. The obtained transformation links are compared to ideal transformation links.

4.4.1 Experimental Setting

4.4.1.1 Experimental data

Our case study is composed of existing six model transformations collected from the literature or written by the authors in previous projects. These six model transformations are described below.

- UML class-diagram to relational schema (Cl2Rs). We use for this well-known model transformation the specifications given in [Kessentini, 2010].
- Ecore meta-model to Jess [Hill, 2003] meta-model (Ec2Je). This transformation aims at encoding an ecore meta-model into Jess templates. This transformation was written by the authors and published in [Saada *et al.*, 2012a].
- Relational schema to Jess model (Rs2Je). This transformation takes as input a relational schema and generates Jess facts that encode it. Note that this transformation was developed in the same context as the previous one, *i.e.*, the generated Jess facts are instances of the Jess templates generated from a metamodel.
- UML state machine to labeled transition System (St2Lt) [Luong *et al.*, 2008]. This transformation maps a UML state machine to a labeled transition system, masking internal behavior of the state machine. This transformation was developed in order to verify properties on the composition of components described by state machines.
- Abstract syntax examples to graphical syntax examples (As2Gs) [Pfister *et al.*, 2012]. This transformation takes as input a model (which is an annotated meta-model of a domain, where annotations indicate how the elements will appear in visual notation) and produces a model that conforms to a metamodel of visual notation.
- Application of the design pattern State to a UML model owning at least one class

Examples	Nbr_{trace}	Source Fragments		Target Fragments	
		(min_F)	(max_F)	(min_F)	(max_F)
Cl2Rs	19	2	4	1	4
Ec2Je	22	1	3	1	2
Rs2Je	21	2	4	1	3
St2Lt	22	3	4	2	7
As2Gs	20	2	4	1	4
St2St	25	2	3	3	5

Table 4.1 – The actual traces of our examples

described by a StateMachine (St2St). This transformation was developed by the authors for educational purpose and serves as practical example in a master model-engineering class.

For each of the above-mentioned model transformations, we wrote a source model. To have realistic models, we decided to set their size to at least 50 constructs. As the selected transformations are implemented, we applied them to the source models to generate their counterpart target models. Afterwards, we used each pair of source/target models as a case to test the trace recovery approach.

4.4.1.2 Experimental protocol

In order to evaluate the relevance of the transformation trace generated by our approach, we also defined, for each pair of models, the actual transformation trace. For the two model transformations Cl2Rs and Rs2Je, we used the transformation rules generated in our previous work [Saada *et al.*, 2012a] which allow to obtain for each meaningful fragment its corresponding fragment using the rule engine Jess. For the rest of model transformations used in this evaluation, a domain expert manually built the actual trace.

Descriptive statistics about the actual (expected) traces are given in Table 4.1. The number of fragment mappings in each trace is given in column Nbr_{trace}. This ranges from 19 for the Cl2Rs transformation to 25 for the St2St one. The size of the source fragments in the actual traces varies in general from 2 (min_F) to 4 (max_F). The variation in size for target fragments is larger with fragments containing up to 7 constructs (for St2Lt), which makes the mapping recovery more difficult.

For each source and its corresponding target model, we use our algorithm to generate a transformation trace. As we are dealing with multiobjective optimization, usually, many

solutions are present in the Pareto front. Usually, a user could look at the proposed solutions and select one. In the case of problems for which the user does not have enough knowledge to choose a solution, a ranking could be necessary to make a recommendation. For the purpose of this evaluation, as our three objectives are normalized in the interval $[0,1]$, it is possible to calculate a distance to the optimal solution s_o, *i.e.*, the one having $LexSim(s_o) = 1$, $StrSim(s_o) = 1$, and $Cov(s_o) = 1$. For a candidate solution s, such a distance $d(s)$ could be calculated as follows:

$$\sqrt{(1 - LexSim(s))^2 + (1 - StrSim(s))^2 + (1 - Cov(s))^2} \tag{4.1}$$

Distance d gives equal importance to all the objectives. However, we believe that consistency and completeness are very important when selecting the final solution. Indeed, having incomplete solutions or similar fragments that are transformed differently could invalidate a solution. Therefore, we propose a variation d_2 of the distance d where only the consistency and completeness objectives are considered. $d_2(s)$ of a candidate solution s is defined as follows:

$$\sqrt{(1 - StrSim(s))^2 + (1 - Cov(s))^2} \tag{4.2}$$

The solution having the minimal d_2 distance is then evaluated by computing its precision and recall. The precision of a solution s is defined as the average precision of its fragment-pairs. For a pair $fp_i = (MfF_i, F_i) \in s$ and the expected mapping (MfF_i, EF_i) of MfF_i, the mapping precision of fp_i is defined as the number of correctly assigned constructs in F_i among the total number of constructs in F_i. Formally:

$$pr(fp_i) = \frac{F_i \bigcap EF_i}{F_i} \tag{4.3}$$

Like for the precision, we evaluate also the pairs' average recall of a solution. For a pair $fp_i = (MfF_i, F_i) \in s$ and the expected mapping (MfF_i, EF_i) of MfF_i, the mapping recall of fp_i is defined as the number of correctly assigned constructs in F_i among the total number of constructs in EF_i. Formally:

$$re(fp_i) = \frac{F_i \bigcap EF_i}{EF_i} \tag{4.4}$$

As mentioned in Section 4.3, the trace recovery algorithm uses a set of parameters. For the case study, these are set as follows:

- Crossover probability is usually high. It is set to 0.8.
- Mutation probability is set to 0.4. As we are dealing with relatively large models, the number of possible fragments in the target model could be very large and so is the number of combinations between the source and target fragments. Usually, the mutation probability is low in evolutionary algorithms, but we increase it in our case to compensate for the random generation of the initial population. Indeed, we have no guarantee that the initially generated target fragments include or are close to the expected ones.
- In each transformation example, a population of 500 solutions was randomly generated. Then we kept only those having a score above a threshold for the three objectives. This was done to start with a decent genetic material.
- We ran the algorithm with a number of iterations equal to twice the size of the population.
- The maximal variation y of the number of target fragments with respect to the source ones is set to 1. This means that in a solution, we could have a MfF without assigned F (part of the source model that does not need to be transformed like in the endogenous transformations) or a F without any MfF (part of the target model that is generated from the other parts of the target model and not from the source one).
- With meta-heuristic search, we can obtain, for the same source and target models with the same parameters, different results on different executions. Therefore, we took the best result from three executions.

4.4.2 Results and Discussion

We first present a summary of the results for the six studied transformations and then, give the details about one case, namely Cl2Rs.

4.4.2.1 Results for the six examples

Table 4.2 shows the distances best solutions obtained for the six transformation examples. The distances varies from 0.09 for Cl2Rs to 0.35 for St2Lt. The distance results are confirmed by the precision and recall scores reported in Figure 4.8.

Except for the case St2Lt, the precision scores are all at least equal to 90% and the recall scores are greater or equal to 84%. This is very encouraging since the examples involve different types of transformations. The scores of St2Lt are intriguing although both precision (75%) and recall (70%) are interesting. In general, many events are generated

	Distance d_2 from our best solution to the optimal one
S_{Cl2Rs}	0.09
S_{Ec2Je}	0.13
S_{Rs2Je}	0.19
S_{St2Lt}	0.35
S_{As2Gs}	0.21
S_{St2St}	0.23

Table 4.2 – Distance from our best solutions to the optimal ones

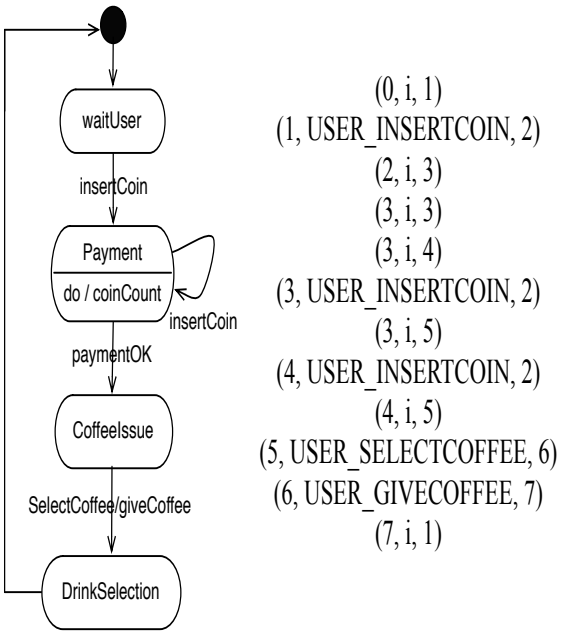

Figure 4.7 – UML state machine to LTS transformation example

and synthetically labeled "i". Moreover, the different states are indicated by numbers. In this context, our lexical and structural approximations are limited to capture the semantic equivalence between some types of fragments. To illustrate this limitation, let us consider the small example of Figure 4.7. This example shows the transformation of a coffee state machine to an LTS system. The operation *coinCount* in the state *Payment* is repeated as long as the amount introduced by the user is not sufficient. A set of events named "i" are generated that correspond to non observable events, for example to represent the internal

action *coinCount*. This can hardly be captured by our objectives, because we do not have neither lexical, nor structural clues to indicate the link between "i" and the internal event.

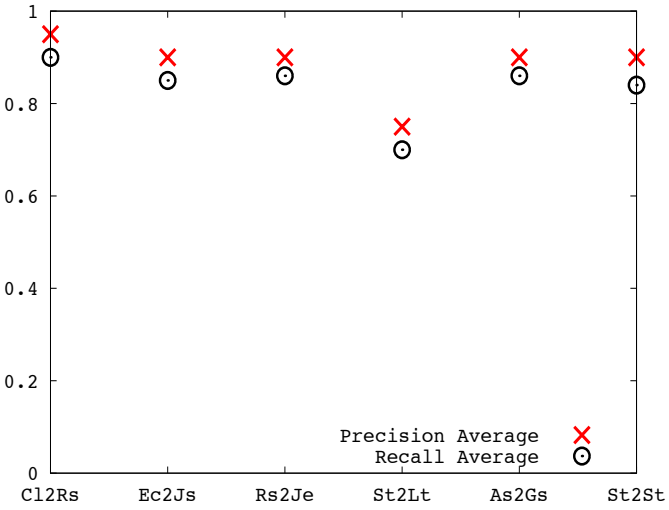

Figure 4.8 – Evaluation results

4.4.2.2 Detailed results for the Cl2Rs example

For the Cl2Rs example, the generated Pareto front contains 15 solutions. They are almost similar and they are all close to the optimal solution in terms of distance. Since the source and target models are pretty large, we show, in Figure 4.9, only an excerpt of the best obtained solution. The partial models in the figure as well as the corresponding full models are well covered by the solution. The association class *Register* with its related classes *Student* and *Module* are mapped to the three tables *Register*, *Student* and *Module*. The association class *Intervene* with its related classes *Teacher* and *Module* are also mapped to the three tables *Intervene*, *Teacher* and *Module*. Moreover, the general class *Person* and its specific classes are well mapped to the three tables *Person*, *Student* and *Teacher*.

Figure 4.10 shows snapshots of the best-solution quality during the successive iterations of our algorithm for the Cl2Rs example. One can notice that precision and recall scores increase (quasi linearly) during the first 50 iterations. For illustration purpose, to calculate the intermediate precision and recall for the different generations during the execution of our algorithm, we selected for each iteration the best solution in the Pareto front using the distance d_2. Note that in a normal execution, the selection of the best solution using d_2 is done only at the last iteration. The best solution in this example was found on the 50^{th} iteration with a precision average of 0.9 and a recall average of 0.85. After that, all the obtained solutions converged towards this solution. We also examined how the

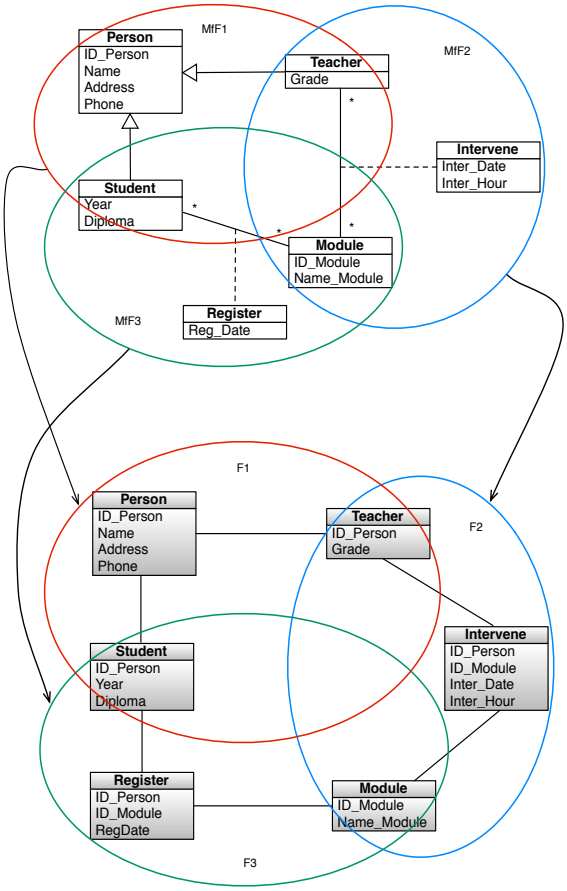

Figure 4.9 – Excerpt of the solution obtained for the Cl2Rs example

mutation strategies help introducing new material to converge towards a good solution. We noticed, for example, that at the 35^{th} iteration, MfF2 (Figure 4.9) is mapped to a fragment that contains just the class *Intervene* and the class *Teacher*. However, as the class *Intervene* contains the primary/foreign key *ID _Module* which has a reference to the table *Module* in the target model, this fragment was extended later with the table *Module* by the first mutation strategy.

4.4.2.3 Performance

The execution time is very important since we use a metaheuristic search to explore a large space. In our experiments, we used a simple MacBook (2.4GHz CPU and 2G of RAM). The execution time for recovering transformation trace on our examples (about 50 constructs), with a number of iterations up to 1000, is less than 120 seconds. This is very acceptable since the recovery process is not intended to be executed on a short-period basis. We tried with models having different sizes and with the same parameter values

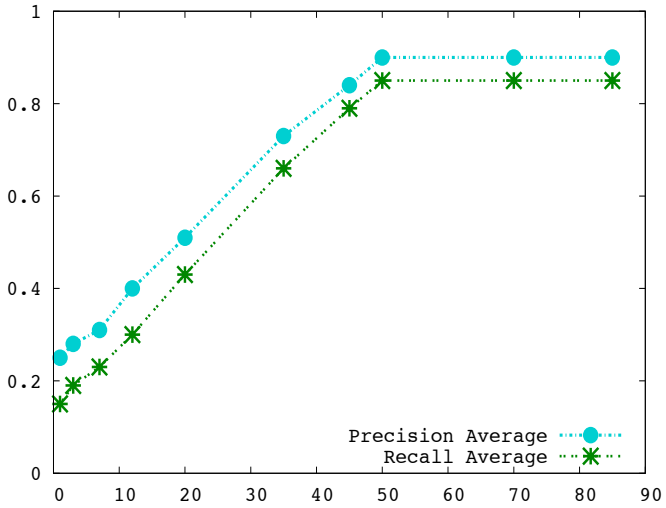

Figure 4.10 – Improvement of the solution quality over the iterations for the Cl2Rs example

(population size and number of iterations), the execution time increases quasi linearly with the models' size.

4.4.3 Threats to Validity

The experiment is here conducted on six examples of model transformations. The choice of the transformation examples could be a threat to the validity of our evaluation. To circumvent this threat, we carefully chose examples from different transformations (structural/behavioral, exogenous/endogenous, different fragment sizes, etc.). The only possible limitation to the generalizability is the relative fixed size of the examples (about 50 constructs). We plan to try our approach with models having a larger variation in size in the future.

Another threat concerns the fact that our experimental setting is semi-real as all the target models are generated using the known transformation mechanisms, and none was derived manually by an expert. A possible issue is that transformation engines tend to use the vocabulary of the source model whereas a human expert could use derived vocabulary (synonyms, abbreviations, etc.). These situations could be handled by a more subtle way to measure the lexical similarity.

4.5 The Model Matching Problem

Applying MDE leads to the creation and the manipulation of a large number of models. Detect mapping between those models is a very complex task. It needs to be manually checked especially when models are edited manually. This alignment is then used to generate model transformation when the goal is to learn model transformation from examples.

In the previous sections, we proposed an approach to automatically generate model transformation traces from source and target models. We used NSGA-II, a metaheuristic method, to solve the problem of the huge number of possible combinations between models elements. Recovering model transformation traces may be similar to the problem of model matching. Both of them aim to derive mappings between models. The difference exists in the nature of examples. While traces are derived from examples issued from a program transformation, model mappings are extracted from examples given without any knowledge of their transformation.

Question 1 *Can our approach for recovering model transformation traces be applied to the model matching problem?*

We propose in this section to apply our approach to the model matching problem. As the examples are not issued from a program transformation, transformed elements may be different from the ones of the source model or may use different naming conventions. Thus, we propose to use natural language processing techniques, which identify the original forms of the words, to improve the lexical similarity. We extract the property value lemmas of MfF_i and F_i in each fragment pair fp_i in a solution s using *TreeTagger* [Schmid, 1994; Schmid, 1995], a tool for annotating text with part-of-speech and lemma information. It is used to tag various languages including English, French German, etc. Then, all the distinct lemmas in s are extracted in a list li. li represents the dimensions of vectors associated to each source or target fragment in s. For each fragment and each term, the corresponding dimension is set to 1 if the term exists in the fragment or to 0 otherwise. Then, the similarity is calculated between each pair MfF_i and F_i using the cosine similarity between the two concerned vectors. The resulting lexical similarity ranges from -1, meaning that MfF_i and F_i do not share any term, to 1, meaning that MfF_i and F_i use exactly the same terms. The lexical similarity $LexSim(s)$ of a solution s is equal to the average of the contained pairs' lexical similarities.

4.5.1 Evaluation

To illustrate the ability of our approach to derive mappings from source and target models, we conducted an experiment on six source and target models coming from several sources on the Internet. The size of models varies between 20 and 40 constructs.

- UML class diagram to Relational Schema model (cl2rs).
- EMF metamodel to Kermeta metamodel (em2ker).
- Kermeta metamodel to EMF metamodel (ker2em).
- UML class diagram to Java code model (cl2jc).
- Ecore metamodel to Jess metamodel (ec2je).
- Book model to publication model (bo2pu).

Source and target models are not obtained by a transformation program and they are not written by the same person. Thus they may have different vocabularies.

As mentioned before, our algorithm uses a set of parameters. For these examples, there are set as follows:

- Crossover probability is set to 0.8.
- Mutation probability is set to 0.35
- The initial population is set to 400 solutions for each example.
- We ran the algorithm with a number of iterations equal to twice the size of the population.
- The maximum variation y of the number of target fragments with respect to the source ones is set to 1. This means that in a solution, we can have a MfF without a corresponding fragment, or an F without an assigned MfF.
- With a metaheuristic method, we can obtain, for the same example, with the same parameters, different results on different executions. Thus, we took the best result from four executions.

Testing the examples consists in generating the mapping from each source and target models and comparing the obtained mapping with those provided by an expert. This comparison allows calculating the precision and the recall for each pair $fp_i = (MfF_i, F_i)$ in the obtained solutions.

The precision of a pair is defined as the number of correctly assigned constructs ($C_{correct}$) among the total number of constructs ($C_{totalNbr}$) (Equation 4.5).

The recall of a pair is defined as the number of correctly assigned constructs among the number of expected constructs (C_{expert}) (Equation 4.6).

$$Precision(fp_i) = \frac{C_{correct}}{C_{totalNbr}} \tag{4.5}$$

$$Recall(fp_i) = \frac{C_{correct}}{C_{expert}} \tag{4.6}$$

The precision (resp. the recall) of a solution is defined as the average precision (resp. recall) of its fragment pairs.

Results and Discussion

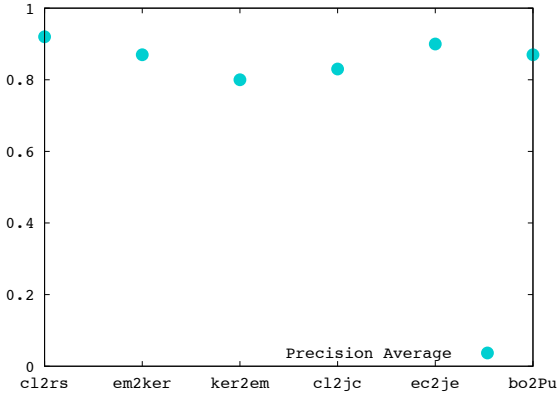

Figure 4.11 – Precision average measured on the six examples

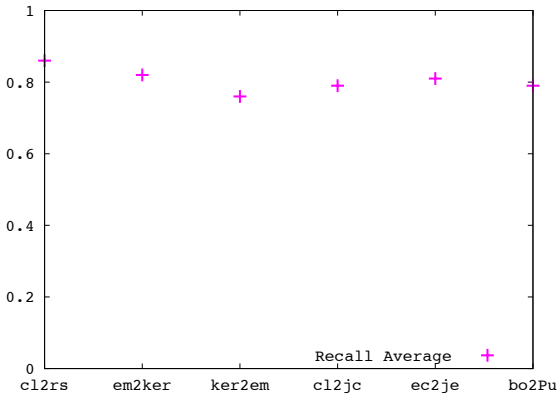

Figure 4.12 – Recall average measured on the six examples

During our experiments, we obtained good results confirmed by the precision and recall averages shown in Figure 4.11 and Figure 4.12. The precision scores are all between 0.87 and 0.92 and the recall scores are higher than 0.76 in all cases. The scores of UML class diagram to relational schema model are interesting (0.92 precision and 0.86 recall). This is very encouraging since we used different types of examples.

Results and Discussion

The execution time is very important since we use a metaheuristic method. In our experiments, we used a simple macBook (2.4 GHz CPU and 2G of RAM). The execution time for generating a mapping between source and target models with a number of iterations up to 800, is less than 90 seconds. We note also that the execution time increases quasi linearly with the models' size.

Threats to validity

The experiment is here conducted on six source and target models. Those models have different size, vocabulary and structure. To help us improving the model matching algorithm, additional experiments have to be conducted, especially to study the two following issues:

- The relatively fixed size of the used examples. Larger models and more examples have to be considered in the future.
- The correctness of the obtained mapping is manually measured by an expert and this may be a hard task especially when using larger models. An automatic measure may be defined in the future.

4.6 Conclusion

In this chapter, we proposed a novel approach for the recovery of transformation trace for two arbitrary source and target models. Our approach does not require any knowledge of the transformation. To implement this approach, we adapted the NSGA-II algorithm to explore the space of mapping possibilities between the two models. We evaluated our approach on six different cases corresponding to six distinct transformation problems. The obtained results indicate that recovered traces are very similar to the expected ones. For five of them, the precision is higher than 90% and the recall higher than 84%.

We also tested this approach on the model matching problem. Furthermore, we improved the lexical similarity function by using a natural language processing technique to annotate text. In order to validate this proposal, we performed experiments on six source and targets models and compared, using retrieval information metrics, the obtained matchings to the expected ones. The results are promising. For all the examples, precision average is higher than 0.8 and recall average is higher than 0.76.

Despite these encouraging results, there is still a room for improvement. First, we plan to conduct more experiments to test our approach on other transformation types and compare our results with the ones of the other approaches. From the algorithmic perspec-

tive, we will explore other functions to approximate the semantic equivalence between the source and target model. This can be done, for example, by more sophisticated lexical similarity techniques such as word dictionary. Another direction worth to explore is the use of examples of past traces to recover a trace for a newly seen pair of models. In addition, the constraint of meaningfulness of the source fragments could be relaxed to handle a wider spectrum of transformation problems.

As the principal goal of this thesis is to learn model transformation from examples, we plan to use the generated *many-to-many* mapping links as an input to our first contribution (Chapter 3) to derive transformation rules.

Conclusions and perspectives

In this Chapter, we summarize the results and conclusions of the dissertation. We also discuss opportunities for extending our work.

Main Contributions

The main objective of this thesis was to assist the writing of a model transformation and understand how a transformation operates. This problem is crucial and one of the most studied issues in the field of Model Driven Engineering. Our main contributions are as follows.

Generation of operational transformation rules from examples. We proposed in Chapter 3 an approach to generate operational transformation rules from transformation examples. We extended a previous work which uses the Relational Concept Analysis as a learning technique to generate transformation patterns. These patterns are not all relevant. Thus, we developed a technique to extract the pertinent ones. Then, we used the language and the rule engine JESS to transform them into executable rules. Through an experimentation on different model transformations, we showed the effectiveness of our approach.
This work was published in [Saada *et al.*, 2012a].

Comparison of strategies to better learn model transformations. To learn transformation rules from examples, it is important to know how we better use examples to obtain pertinent rules. Thus, we performed an experimentation to compare two strategies for learning model transformations from examples. The first one consists in using each example separately to derive transformation rules and then gathering the rules, or gathering all the examples and derive transformation rules. The execution of the obtained rules of the first strategy gave good results in our experiment. The rule engine searches and finds for each example the set of rules to apply. On the contrary, the rules of the second strategy are larger and contain more information. Thus the rule engine does not find a rule to apply for the simple examples.
This work was published in [Saada *et al.*, 2012b].

Recovering transformation traces from transformation examples. In Chapter 4, we focused on recovering transformation traces from transformation examples. These traces between models are essential in an MDE process where the models are handled successively using model transformations. They can be useful for different problems, *e.g.*, locating bugs during the execution of a transformation. Our underlying purpose is to use in future work those traces to learn model transformations. We proposed an approach based on a multi-objective meta-heuristic to derive a *many-to-many* matching between models coming from a transformation program. Thus, we searched to associate a group of m source elements to a group of n target elements according to the lexical and structural similarities between them. Our approach is evaluated through different model transformations examples.
This work was published in [Saada *et al.*, 2013b].

An approach to the model matching problem. The problem of transformation traces recovery is similar to the model matching problem. In the first problem, examples are issued from a transformation program. Thus, it is easy to find lexical and structural similarity between models. However, in the second context, the transformation between source and target models may be manually done , or the models may not come from a transformation at all and this may cause more lexical and structural variations between models. Hence, we defined a variant of our previous approach to deal with the model matching problem. We used a natural language processing technique to improve the lexical similarity. Our approach produces matchings of type *many-to-many* between models.
This work was published in [Saada *et al.*, 2013a].

Ongoing work and open issues

This section covers the issues not handled by our proposed approaches and discusses some possible extensions of our current work.

First, the performance of our contributions depends on the availability of good quality examples, which could be difficult to collect. Second, since we used an optimization technique, the process may spend more time with large size models. Thus, we plan to conduct more experiments to better test our approaches.

For the generation of transformation rules, we plan to transform the obtained JESS facts (after the rule application) to produce models conforming to the initial meta-models. Furthermore, we plan to learn rules whose premise and conclusion have several main elements

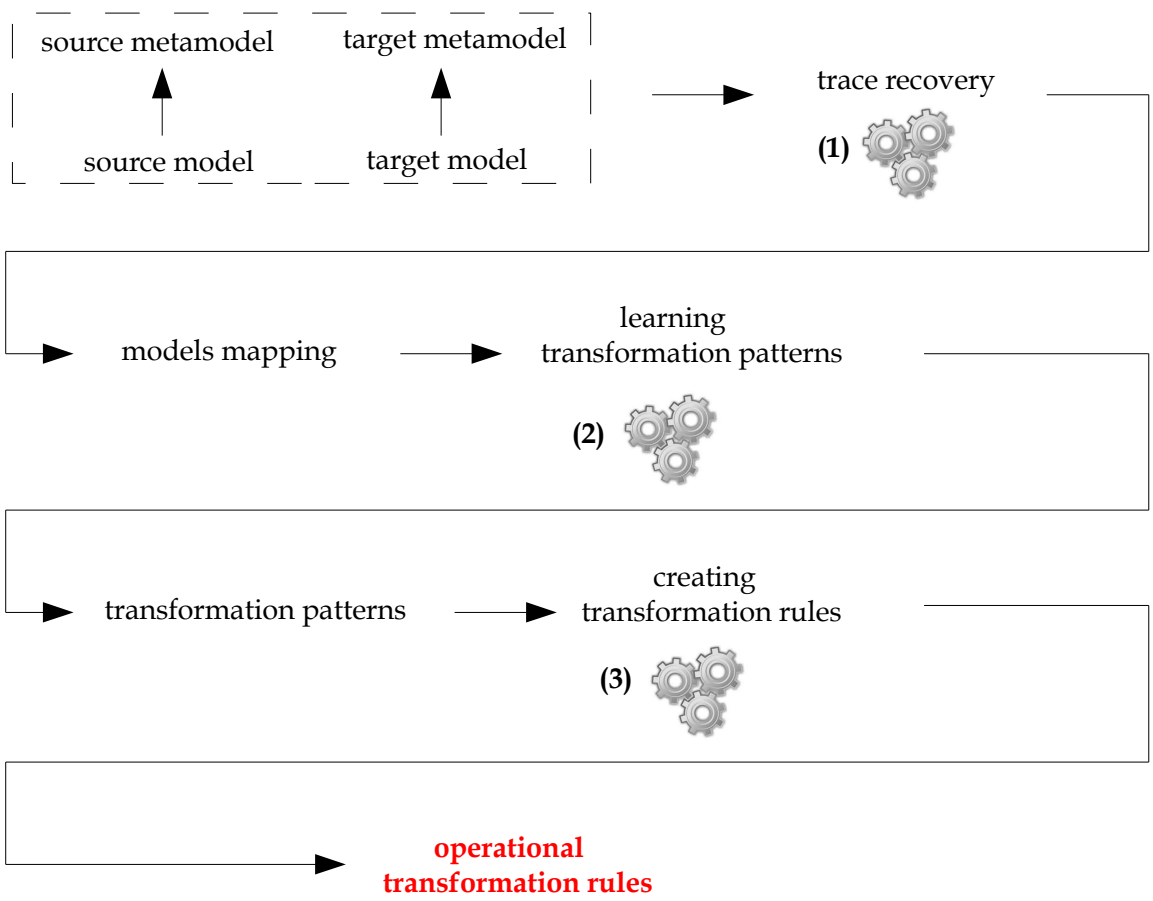

Figure 4.13 – Learning transformation rules process

and design heuristics to select the best rule to apply when several rules are candidate for the same model element.

Recovering traces from model transformations needs other functions to approximate the semantic equivalence between the source and target models. Thus, we plan to enhance our lexical similarity techniques using for example a word dictionary. In addition, the constraint of meaningfulness of the source fragments used could be relaxed to handle a wider spectrum of transformation problems. We are looking also to use another metaheuristic to compare it with NSGA-II.

In this thesis, we based our proposed work on a previous work which learns transformation patterns from mappings of type *1-1* between the model constructs. As future work, we plan to use our contributions and restart the process to generate operational rules as mentioned in Figure 4.13:

1. Extract mappings from examples is very important in a learning process. Thus, we plan to use our recovering traces approach to generate *many-to-many* mappings between models constructs.

2. As a learning technique, we plan to test RCA with the obtained *many-to-many* mappings to obtain transformation patterns. However, the current learning process with RCA deals with *1-1* mapping. A first idea to deal with *n-m* matchings is to propose a new encoding, such that a matching has not one source and one target elements but several ones. The current encoding does not allow us to express that a group of source elements is transformed into a group of target elements. Thus, the notion of group has to be reified in the encoding of mappings.

   Our studied learning strategies will help us also to better use transformation examples (separately or gathered).

3. The operationalization of transformation patterns is very important. It helps to test the rules and execute them on different examples. Thus, we plan to use our approach to generate operational rules from the new obtained patterns.

Finally, we can extend our work by applying our approaches on the metamodel level. In this thesis, we focused our work on the model level. We plan to extract mapping between metamodels and then generate transformation rules. In this level, a considerable lexical and structural variation may exist between two given metamodels that are written in an independent way. Concerning the lexical variation, the lexical similarity may be modified so as to use dictionaries that may contain information or synonyms for example. However metamodels deal with very specific domains, and thus technical vocabulary, such that the used vocabulary may not be present in several purpose dictionaries, e.g., WordNet [Miller, 1980]. It will be better to use specific and technical dictionaries, but such dictionaries are not likely to exist for all the metamodels. An alternative way is to build such a dictionary, for example using the dedicated game *JeuxDeMots* [Lafoucarde, 2011]. However, this requires the interaction of experts for each involved metamodel to build the dictionary.

Related Publications

I have started my PhD in January 2011. In the following, we give our publications during this thesis.

- International Conferences and workshops
  - **Hajer Saada**, Marianne Huchard, Clémentine Nebut, Houari Sahraoui. «Model Matching for Model Transformation: A meta-heuristic Approach», 2^{nd} International Conference on Model Driven Engineering and Software Development (MODELSWARD), 2013.
  - **Hajer Saada**, Marianne Huchard, Clémentine Nebut, Houari Sahraoui. «Recovering Model Transformation Traces using Multi-Objective Optimization», 28th IEEE/ACM International Conference on Automated Software Engineering (new ideas paper) (ASE), 2013.
  - **Hajer Saada**, Xavier Dolques, Marianne Huchard, Clémentine Nebut, Houari Sahraoui. «Generation of operational transformation rules from examples of model transformations», 15th IEEE/ACM International Conference on Model Driven Engineering Languages and Systems (MODELS), 2012.
  - **Hajer Saada**, Xavier Dolques, Marianne Huchard, Clémentine Nebut, Houari Sahraoui. «Learning Model Transformations from Examples using FCA: One for All or All for One ?», 9th International Conference on Concept Lattices and Their Applications (CLA), 2012.
  - Xavier Dolques, Marianne Huchard, Clémentine Nebut, **Hajer Saada**. «Formal and Relational Concept Analysis approaches in Software Engineering: an overview and an application to learn model transformation patterns in examples». ICESE'11: First ICESE Virtual Workshop, Search-based Model-Driven Engineering, Qatar (2011).
- National Conferences
  - **Hajer Saada**, Xavier Dolques, Marianne Huchard, Clémentine Nebut, Houari Sahraoui. «Generation of operational transformation rules from examples», actes des 4èmes journées nationales du GDR GPL, 2012.

Bibliography

[Agrawal *et al.*, 2006] Aditya Agrawal, Gabor Karsai, Sandeep Neema, Feng Shi, and Attila Vizhanyo. The design of a language for model transformations. *Software and System Modeling*, 5(3):261–288, 2006. (Cited on page 23.)

[Akehurst and Kent, 2002] David Akehurst and Stuart Kent. A relational approach to defining transformations in a metamodel. pages 243–258. Springer, 2002. (Cited on page 22.)

[Amar *et al.*, 2010] Bastien Amar, Herve Leblanc, Bernard Coulette, and Clementine Nebut. Using aspect-oriented programming to trace imperative transformations. In *Proceedings of the 2010 14th IEEE International Enterprise Distributed Object Computing Conference*, pages 143–152, 2010. (Cited on pages 29, 32, and 62.)

[Balogh and Varro, 2009] Zoltan Balogh and Daniel Varro. Model transformation by example using inductive logic programming. *Software and Systems Modeling*, 8(3):347–364, 2009. (Cited on pages 26, 28, and 29.)

[ben Fadhel *et al.*, 2012] Ameni ben Fadhel, Marouane Kessentini, Philip Langer, and Manuel Wimmer. Search-based detection of high-level model changes. In *ICSM*, pages 212–221, 2012. (Cited on page 30.)

[Biehl, 2010] Matthias Biehl. Literature study on model transformations. Technical Report ISRN/KTH/MMK/R-10/07-SE, Royal Institute of Technology, 2010. (Cited on page 24.)

[Brosch *et al.*, 2009] Petra Brosch, Philip Langer, Martina Seidl, Konrad Wieland, Manuel Wimmer, Gerti Kappel, Werner Retschitzegger, and Wieland Schwinger. An example is worth a thousand words: Composite operation modeling by-example. In *Proceedings of the 12th International Conference on Model Driven Engineering Languages and Systems*, MODELS '09, pages 271–285, Berlin, Heidelberg, 2009. Springer-Verlag. (Cited on pages 27 and 29.)

[Bézivin *et al.*, 2003] Jean Bézivin, Grégoire Dupé, Frédéric Jouault, Gilles Pitette, and Jamal Eddine Rougui. First experiments with the atl model transformation language: Transforming xslt into xquery. In *OOPSLA 2003 Workshop*, 2003. (Cited on pages 29 and 62.)

[Choi *et al.*, 2006] Namyoun Choi, Il-Yeol Song, and Hyoil Han. A survey on ontology mapping. *SIGMOD Rec.*, 35(3):34–41, 2006. (Cited on page 31.)

[Clark *et al.*, 2004] Tony Clark, Andy Evans, Paul Sammut, and James Willans. Applied metamodelling, a foundation for language driven development. Technical report, 2004. (Cited on page 22.)

[Csertán *et al.*, 2002] György Csertán, Gábor Huszerl, István Majzik, Zsigmond Pap, András Pataricza, and Dániel Varró. Viatra: Visual automated transformations for formal verification and validation of uml models. In *Proceedings of the 17th IEEE international conference on Automated software engineering*. IEEE Computer Society, 2002. (Cited on pages 23 and 36.)

[Cysneiros *et al.*, 2003] Gilberto A. A. Cysneiros, Filho Andrea, and Zisman George Spanoudakis. Traceability approach for i* and uml models. In *in Proceedings of 2nd International Workshop on Software Engineering for Large-Scale Multi-Agent Systems (SELMAS'03*, 2003. (Cited on pages 30, 31, 32, and 62.)

[Czarnecki and Helsen, 2006] Krzysztof Czarnecki and Simon Helsen. Feature-based survey of model transformation approaches. *IBM SYSTEMS JOURNAL*, 45(3):621–645, 2006. (Cited on page 22.)

[Czarnecki, 2005] Krzysztof Czarnecki. Mapping features to models: A template approach based on superimposed variants. In *GPCE 2005 - Generative Programming and Component Enginering. 4th International Conference*, pages 422–437. Springer, 2005. (Cited on page 22.)

[Daniele, 2006] Laura Maria Daniele. *Towards a Rule-based Approach for Context-Aware Applications*. PhD thesis, University of Twente The Netherlands, May 2006. (Cited on pages 12 and 37.)

[Deb *et al.*, 2002] Kalyanmoy Deb, Samir Agrawal, Amrit Pratap, and T. Meyarivan. A fast and elist multiobjective genetic algorithm: Nsga-II. *IEEE Trans, Evolutionary Computation*, 6(2):182–197, 2002. (Cited on page 15.)

[Del Fabro and Valduriez, 2007] Marcos Didonet Del Fabro and Patrick Valduriez. Semi-automatic model integration using matching transformation and weaving models. In *International Conference SAC'07*, pages 963–970. ACM, 2007. (Cited on pages 25 and 28.)

[Dolques *et al.*, 2009] Xavier Dolques, Marianne Huchard, and Clémentine Nebut. From transformation traces to transformation rules: Assisting model driven engineering approach with formal concept analysis. In *Supplementary Proceedings of ICCS'09*, pages 15–29, 2009. (Cited on pages 4, 27, 37, 38, and 39.)

[Dolques *et al.*, 2010] Xavier Dolques, Marianne Huchard, Clémentine Nebut, and Philippe Reitz. Learning transformation rules from transformation examples: An approach based on relational concept analysis. In *Companion proceedings of EDOC'10*. IEEE Computer Society Press, 2010. à paraître. (Cited on page 28.)

[Dolques *et al.*, 2011] Xavier Dolques, Aymen Dogui, Jean-Rémy Falleri, Marianne Huchard, Clémentine Nebut, and François Pfister. Easing model transformation learning with automatically aligned examples. In *7th European Conference, ECMFA 2011*, pages 189–204, 2011. (Cited on pages 27, 29, and 31.)

[Fabro and Valduriez, 2009] Marcos Didonet Del Fabro and Patrick Valduriez. Towards the efficient development of model transformations using model weaving and matching transformations. *Software and System Modeling*, 8(3):305–324, 2009. (Cited on page 31.)

[Falleri *et al.*, 2008] Jean-Rémy Falleri, Marianne Huchard, Mathieu Lafourcade, and Clémentine Nebut. Meta-model matching for automatic model transformation generation. In *MODELS'08, LNCS 5301*, pages 326–340. Springer, 2008. (Cited on pages 25, 28, and 31.)

[Frank, 2004] Budinsky Frank. *Eclipse Modeling Framework (Eclipse Series)*. Addison-Wesley Professional, 2004. (Cited on pages 18, 23, and 63.)

[Ganter and Wille, 1999] Bernhard Ganter and Rudolf Wille. *Formal Concept Analysis, Mathematical Foundations*. Springer, 1999. (Cited on pages 7 and 37.)

[García-Magariño *et al.*, 2009] Iván García-Magariño, Jorge J. Gómez-Sanz, and Rubén Fuentes-Fernández. Model transformation by-example: An algorithm for generating many-to-many transformation rules in several model transformation languages. In *ICMT*, pages 52–66, 2009. (Cited on page 27.)

[Goldberg, 1989] David E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edition, 1989. (Cited on page 32.)

[Grammel and Kastenholz, 2010] Birgit Grammel and Stefan Kastenholz. A generic traceability framework for facet-based traceability data extraction in model-driven software development. In *Proceedings of the 6th ECMFA Traceability Workshop*, ECMFA-TW '10, pages 7–14. ACM, 2010. (Cited on pages 29, 32, and 62.)

[Grammel *et al.*, 2012] Birgit Grammel, Stefan Kastenholz, and Konrad Voigt. Model matching for trace link generation in model-driven software development. In *Proceed-*

ings of the 15th international conference on Model Driven Engineering Languages and Systems, MODELS'12, pages 609–625, 2012. (Cited on pages 30, 31, 32, 62, and 63.)

[Guerra *et al.*, 2013] Esther Guerra, Juan Lara, Dimitrios S. Kolovos, Richard F. Paige, and Osmar Marchi Santos. Engineering model transformations with transML. *Softw. Syst. Model.*, 12(3):555–577, July 2013. (Cited on page 33.)

[Hacene *et al.*, 2013] Mohamed Rouane Hacene, Marianne Huchard, Amedeo Napoli, and Petko Valtchev. Relational concept analysis: mining concept lattices from multi-relational data. *Ann. Math. Artif. Intell.*, 67(1):81–108, 2013. (Cited on page 12.)

[Harman *et al.*, 2001] Mark Harman, Ub Ph, and Bryan F. Jones. Search-based software engineering. *Information and Software Technology*, 43:833–839, 2001. (Cited on page 32.)

[Harman *et al.*, 2012] Mark Harman, S. Afshin Mansouri, and Yuanyuan Zhang. Search-based software engineering: Trends, techniques and applications. *ACM Comput. Surv.*, 45(1):11:1–11:61, 2012. (Cited on page 15.)

[Harman, 2007] Mark Harman. The current state and future of search based software engineering. In *2007 Future of Software Engineering*, FOSE '07, pages 342–357, Washington, DC, USA, 2007. IEEE Computer Society. (Cited on page 32.)

[Harman, 2011] M. Harman. Software engineering meets evolutionary computation. *IEEE Computer*, 44(10):31–39, 2011. (Cited on page 14.)

[Hartung *et al.*, 2010] Michael Hartung, Anika Gross, and Erhard Rahm. Rule-based generation of diff evolution mappings between ontology versions. *CoRR*, 2010. (Cited on page 30.)

[Hill, 2003] Ernest Friedman Hill. *Jess in Action: Java Rule-Based Systems*. Manning Publications Co., Greenwich, CT, USA, 2003. (Cited on pages 5, 12, 13, and 73.)

[Horn *et al.*, 1994] J. Horn, N. Nafpliotis, and D.E. Goldberg. A niched pareto genetic algorithm for multiobjective optimization. In *Proceedings of the First IEEE Conference on Evolutionary Computation*, pages 82–87. IEEE, 1994. (Cited on page 15.)

[Huchard *et al.*, 2007] Marianne Huchard, Mohamed Rouane Hacène, Cyril Roume, and Petko Valtchev. Relational concept discovery in structured datasets. *Ann. Math. Artif. Intell.*, 49(1-4):39–76, 2007. (Cited on pages 4, 7, and 27.)

[Jouault *et al.*, 2008] Frederic Jouault, Freddy Allilaire, Jean Bezivin, and Ivan Kurtev. Atl: A model transformation tool. *Sci. Comput. Program.*, 72(1-2):31–39, June 2008. (Cited on page 23.)

[Jouault, 2005] Frédéric Jouault. Loosely coupled traceability for atl. In *In Proceedings of the European Conference on Model Driven Architecture (ECMDA) workshop on traceability*, pages 29–37, 2005. (Cited on pages 29, 32, and 62.)

[Kappel *et al.*, 2006] Gerti Kappel, Elisabeth Kapsammer, Horst Kargl, Gerhard Kramler, Thomas Reiter, Werner Retschitzegger, Wieland Schwinger, and Manuel Wimmer. Lifting metamodels to ontologies : A step to the semantic integration of modeling languages. In *Proceedings of MoDELS 2006*, pages 528–542, 2006. (Cited on page 25.)

[Kappel *et al.*, 2012] Gerti Kappel, Philip Langer, Werner Retschitzegger, Wieland Schwinger, and Manuel Wimmer. Conceptual modelling and its theoretical foundations. chapter Model transformation by-example: a survey of the first wave, pages 197–215. Springer-Verlag, Berlin, Heidelberg, 2012. (Cited on page 26.)

[Kehrer *et al.*, 2011] Timo Kehrer, Udo Kelter, and Gabriele Taentzer. A rule-based approach to the semantic lifting of model differences in the context of model versioning. In *Proceedings of the 2011 26th IEEE/ACM International Conference on Automated Software Engineering*, ASE '11, pages 163–172, 2011. (Cited on page 30.)

[Kennedy and Eberhart, 1995] James Kennedy and Russel C. Eberhart. Particle swarm optimization. In *Proceedings IEEE International Conference on Neural Networks*, pages 1942–1948, 1995. (Cited on page 28.)

[Kessentini *et al.*, 2008] Marouane Kessentini, Houari Sahraoui, and Mounir Boukadoum. Model transformation as an optimization problem. In *MODELS'08, LNCS 5301*, pages 159–173. Springer, 2008. (Cited on pages 27, 28, and 29.)

[Kessentini *et al.*, 2009] Marouane Kessentini, Houari Sahraoui, and Mounir Boukadoum. Méta-modélisation de la transformation de modèles par l'exemple : approche méta-heuristiques. In Bernard Carré and Olivier Zendra, editors, *LMO'09: Langages et Modèles à Objets*, pages 75–90, Nancy, mars 2009. Cepaduès. (Cited on page 27.)

[Kessentini *et al.*, 2010a] Marouane Kessentini, Arbi Bouchoucha, Houari Sahraoui, and Mounir Boukadoum. Example-based sequence diagrams to colored petri nets transformation using heuristic search. In *Modelling Foundations and Applications*, pages 156–172. Springer, 2010. (Cited on pages 27 and 28.)

[Kessentini *et al.*, 2010b] Marouane Kessentini, Houari Sahraoui, Mounir Boukadoum, and Omar Ben Omar. Model transformation by example : a search-based approach. *Software and Systems Modeling Journal*, 2010. (Cited on page 37.)

[Kessentini, 2010] Marouane Kessentini. *Transformation by Example*. PhD thesis, University of Montreal, 2010. (Cited on pages 4, 32, 36, 52, 54, and 73.)

[Kirkpatrick *et al.*, 1983] Scott Kirkpatrick, Charles Daniel Gelatt, and Mario P. Vecchi. Optimization by simulated annealing. *Science*, 220:671–680, 1983. (Cited on page 28.)

[Kleppe *et al.*, 2003] Anneke Kleppe, Jos Warmer, and Wim Bast. *MDA Explained. The Model Driven Architecture: Practice and Promise*. Addison-Wesley, 2003. (Cited on pages 18 and 20.)

[Knowles and Corne, 1999] J. Knowles and D. Corne. The pareto archived evolution strategy: A new baseline algorithm for pareto multiobjective optimisation. In *Proceedings of the Congress on Evolutionary Computation*, volume 1, pages 98–105. IEEE, 1999. (Cited on page 15.)

[Kolovos *et al.*, 2008] Dimitrios S. Kolovos, Richard F. Paige, and Fiona A. Polack. The epsilon transformation language. In *Proceedings of the 1st international conference on Theory and Practice of Model Transformations*, ICMT '08, pages 46–60, 2008. (Cited on page 24.)

[Kurtev *et al.*, 2007] I. Kurtev, M. Dee, A. Göknil, and K.G. Berg van den. Traceability-based change management in operational mappings. In *ECMDA Traceability Workshop 2007*, pages 57–67, 2007. (Cited on pages 29, 32, and 62.)

[Laarhoven and Aarts, 1987] P. J. M. Laarhoven and E. H. L. Aarts, editors. *Simulated annealing: theory and applications*. Kluwer Academic Publishers, Norwell, MA, USA, 1987. (Cited on page 32.)

[Lafoucarde, 2011] Mathieu Lafoucarde. Jeuxdemots, 2011. http://www.jeuxdemots.org/jdm-accueil.php. (Cited on page 90.)

[Langer *et al.*, 2010] Philip Langer, Manuel Wimmer, and Gerti Kappel. Model-to-model transformations by demonstration. In *ICMT*, pages 153–167, 2010. (Cited on pages 27, 28, and 29.)

[Laumanns *et al.*, 2002] M. Laumanns, L. Thiele, K. Deb, and E. Zitzler. Combining convergence and diversity in evolutionary multiobjective optimization. *Evolutionary computation*, 10(3):263–282, 2002. (Cited on page 15.)

[Lieberman, 1993] Henry Lieberman. Programming by example, 1993. http://web.media.mit.edu/ lieber/PBE/index.html. (Cited on page 26.)

[Lopes *et al.*, 2005] Denivaldo Lopes, Slimane Hammoudi, Jean Bezivin, and Frederic Jouault. Generating transformation definition from mapping specification: Applica-

tion to web service platform. In *CAiSE'05, LNCS 3520*, pages 309–325, 2005. (Cited on pages 24 and 28.)

[Lopes *et al.*, 2006a] D. Lopes, S. Hammoudi, and Z. Abdelouahab. Schema matching in the context of model driven engineering: From theory to practice. In *Advances in Systems, Computing Sciences and Software Engineering*, pages 219–227. Springer, 2006. (Cited on page 31.)

[Lopes *et al.*, 2006b] Denivaldo Lopes, Slimane Hammoudi, and Zair Abdelouahab. Schema matching in the context of model driven engineering: From theory to practice. In Tarek Sobh and Khaled Elleithy, editors, *Advances in Systems, Computing Sciences and Software Engineering*, pages 219–227. Springer, 2006. (Cited on page 24.)

[Lopes *et al.*, 2009] Denivaldo Lopes, Slimane Hammoudi, and Zair Abdelouahab. A step forward in semi-automatic metamodel matching: Algorithms and tool. In Joaquim Filipe and José Cordeiro, editors, *Proceeding of ICEIS 2009*, pages 137–148. Springer, 2009. (Cited on pages 25 and 31.)

[Luong *et al.*, 2008] Hong-Viet Luong, Thomas Lambolais, and Anne-Lise Courbis. Implementation of the conformance relation for incremental development of behavioural models. In *MoDELS*, pages 356–370, 2008. (Cited on page 73.)

[Melnik *et al.*, 2002] Sergey Melnik, Hector Garcia-Molina, and Erhard Rahm. Similarity flooding : A versatile graph matching algorithm and its application to schema matching. In *ICDE*, pages 117–128. IEEE Computer Society, 2002. (Cited on page 25.)

[Mens and Gorp, 2006] Tom Mens and Pieter Van Gorp. A taxonomy of model transformation. *Electr. Notes Theor. Comput. Sci.*, 152:125–142, 2006. (Cited on pages 3 and 20.)

[Miller, 1980] George A. Miller. Wordnet, a lexical database for english, 1980. http://wordnet.princeton.edu/. (Cited on page 90.)

[Muggleton and De Raedt, 1994] Stephen Muggleton and Luc De Raedt. Inductive logic programming: Theory and methods. *Journal of Logic Programming*, 19/20:629–679, 1994. (Cited on page 26.)

[Muller *et al.*, 2005] Pierre-Alain Muller, Franck Fleurey, and Jean-Marc Jézéquel. Weaving executability into object-oriented meta-languages. In L. Briand and S. Kent, editors, *Proceedings of MODELS/UML'2005*, 2005. (Cited on page 22.)

[Noy and Musen, 2001] Natalya F. Noy and Mark A. Musen. Anchor-prompt: Using non-local context for semantic matching. In *Proc. of the Workshop on Ontologies and Information Sharing at IJCAI-2001*, pages 63–70, Seattle (USA), 2001. (Cited on page 27.)

[OMG, 2002] OMG. Object management group: Mof 2.0 query view transformation rfp, 2002. (Cited on page 22.)

[OMG, 2005] OMG. Object management group: Mof 2.0 query view transformation, 2005. (Cited on pages 23, 29, 32, and 62.)

[OMG, 2006] OMG. Meta object facility core specification. Technical report, Object Management Group, 2006. (Cited on pages 18, 29, 32, and 62.)

[Paige *et al.*, 2011] Richard F. Paige, Nikolaos Drivalos, Dimitrios S. Kolovos, Kiran J. Fernandes, Christopher Power, Goran K. Olsen, and Steffen Zschaler. Rigorous identification and encoding of trace-links in model-driven engineering. *Softw. Syst. Model.*, 10(4):469–487, October 2011. (Cited on page 62.)

[Pfister *et al.*, 2012] Francois Pfister, Vincent Chapurlat, Marianne Huchard Huchard, and Clementine Nebut. A proposed tool and process to design domain specific modeling languages. Technical report, LGI2P, Ecole Des Mines, 2012. (Cited on page 73.)

[Rahm and Bernstein, 2001] Erhard Rahm and Philip A. Bernstein. A survey of approaches to automatic schema matching. *The VLDB Journal*, 10(4):334–350, 2001. (Cited on pages 24 and 31.)

[Rothenberg, 1989] Jeff Rothenberg. *AI, Simulation & Modeling*, chapter The Nature of Modeling, pages 75–92. John Wiley & Sons, Inc., 1989. (Cited on page 18.)

[Saada *et al.*, 2012a] Hajer Saada, Xavier Dolques, Marianne Huchard, Clémentine Nebut, and Houari Sahraoui. Generation of operational transformation rules from examples of model transformations. In *MoDELS*, pages 546–561, 2012. (Cited on pages 73, 74, and 87.)

[Saada *et al.*, 2012b] Hajer Saada, Xavier Dolques, Marianne Huchard, Clementine Nebut, and Houari A. Sahraoui. Learning model transformations from examples using fca: One for all or all for one? In *CLA*, pages 45–56, 2012. (Cited on page 87.)

[Saada *et al.*, 2013a] Hajer Saada, Marianne Huchard, Clementine Nebut, and Houari A. Sahraoui. Model matching for model transformation: A meta-heuristic approach. will be published in the Model-Driven Engineering and Software Development proceeding, 2013. (Cited on page 88.)

[Saada *et al.*, 2013b] Hajer Saada, Marianne Huchard, Clementine Nebut, and Houari A. Sahraoui. Recovering model transformation traces using multi-objective optimization. will be published in the Automated Software Engineering proceeding, 2013. (Cited on page 88.)

[Schmid, 1994] Helmut Schmid. Probabilistic part-of-speech tagging using decision trees, 1994. (Cited on page 81.)

[Schmid, 1995] Helmut Schmid. Improvements in part-of-speech tagging with an application to german. In *In Proceedings of the ACL SIGDAT-Workshop*, pages 47–50, 1995. (Cited on page 81.)

[Schmidt, 2006] Douglas C. Schmidt. Model-driven engineering. *IEEE Computer*, 39(2), February 2006. (Cited on pages 3 and 23.)

[Shousha *et al.*, 2008] Marwa Shousha, Lionel Briand, and Yvan Labiche. A uml/spt model analysis methodology for concurrent systems based on genetic algorithms. In *Proceedings of the 11th international conference on Model Driven Engineering Languages and Systems*, MoDELS '08, pages 475–489, Berlin, Heidelberg, 2008. Springer-Verlag. (Cited on page 32.)

[Shvaiko and Euzenat, 2005] Pavel Shvaiko and Jérôme Euzenat. A survey of schema-based matching approaches. pages 146–171, 2005. (Cited on pages 24 and 31.)

[Soley and the OMG Staff Strategy Group, 2000] Richard Soley and the OMG Staff Strategy Group. Model driven architecture. Technical report, Object Management Group, 2000. (Cited on pages 3, 19, and 23.)

[Sun *et al.*, 2009] Yu Sun, Jules White, and Jeff Gray. Model transformation by demonstration. In *MoDELS*, pages 712–726, 2009. (Cited on pages 27, 28, and 29.)

[Taentzer, 2000] Gabriele Taentzer. Agg: A tool environment for algebraic graph transformation. In *in AGTIVE, ser. Lecture Notes in Computer Science*, pages 481–488. Springer, 2000. (Cited on page 23.)

[van Amstel *et al.*, 2012] Marcel van Amstel, Mark G. J. van den Brand, and Alexander Serebrenik. Traceability visualization in model transformations with trace vis. In *ICMT*, pages 152–159, 2012. (Cited on pages 29, 32, and 62.)

[Varró, 2006] Dániel Varró. Model transformation by example. In *Proc. MODELS 2006, LNCS 4199*, pages 410–424. Springer, 2006. (Cited on pages 24, 26, and 37.)

[Vermolen *et al.*, 2011] Sander Vermolen, Guido Wachsmuth, and Eelco Visser. Reconstructing complex metamodel evolution. In *SLE*, pages 201–221, 2011. (Cited on page 30.)

[Vojtisek and Jezequel, 2004] Didier Vojtisek and Jean-Marc Jezequel. Mtl and umlaut ng - engine and framework for model transformation. Technical report, IRISA, INRIA France, 2004. (Cited on page 22.)

[Wimmer *et al.*, 2007] Manuel Wimmer, Michael Strommer, Horst Kargl, and Gerhard Kramler. Towards model transformation generation by-example. In *HICSS '07: Proc. of the 40th Annual Hawaii International Conf. on System Sciences*, page 285b. IEEE Computer Society, 2007. (Cited on pages 27, 28, and 29.)

[Xing and Stroulia, 2005] Zhenchang Xing and Eleni Stroulia. Umldiff: an algorithm for object-oriented design differencing. In *ASE*, pages 54–65, 2005. (Cited on page 30.)

[Xing and Stroulia, 2006] Zhenchang Xing and Eleni Stroulia. Refactoring detection based on umldiff change-facts queries. In *WCRE*, pages 263–274, 2006. (Cited on page 30.)

[Zitzler and Thiele, 1999] Eckart Zitzler and Lothar Thiele. Multiobjective evolutionary algorithms: a comparative case study and the strength pareto approach. *IEEE Trans. Evolutionary Computation*, 3(4):257–271, 1999. (Cited on page 15.)

List of Figures

List of Tables

Vers une assistance à la manipulation de transformations de modèles par l'exploitation d'exemples de transformation

L'Ingénierie Dirigée par les Modèles (IDM) est un domaine de recherche en pleine émergence qui considère les modèles comme des éléments de base. Chaque modèle est conforme à un autre modèle, appelé son méta-modèle, qui définit sa syntaxe abstraite et ses concepts. Dans un processus IDM, différents types de modèles sont manipulés par des transformations de modèles. Une transformation génère un modèle dans un langage cible à partir d'un modèle dans un langage source. Pour concevoir une transformation, les développeurs doivent avoir une bonne connaissance des méta-modèles concernés ainsi que des langages de transformation, ce qui rend cette tâche difficile.
Dans cette thèse, nous proposons d'assister l'écriture des transformations et plus généralement de comprendre comment une transformation opère. Nous adhérons à l'approche de transformation de modèles par l'exemple qui propose de créer une transformation de modèles à partir d'exemples de transformation. Cela permet d'utiliser la syntaxe concrète définie pour les méta-modèles, et cela évite donc de requérir que les développeurs aient une bonne maîtrise des méta-modèles utilisés. Dans ce contexte, nous proposons deux contributions. La première consiste à définir une méthode pour générer des règles de transformation opérationnelles à partir d'exemples. Nous nous basons sur une approche qui utilise l'Analyse Relationnelle de Concepts (ARC) comme technique d'apprentissage pour obtenir des patrons de transformation à partir d'un appariement de type 1-1 entre les modèles. Nous développons une technique pour extraire des règles de transformation opérationnelles à partir de ces patrons. Ensuite, nous utilisons le langage et le moteur de règles JESS pour exécuter ces règles. Nous étudions aussi comment mieux apprendre des règles de transformations à partir d'exemples, en utilisant séparément chaque exemple ou en réunissant tous les exemples. La deuxième contribution consiste à récupérer les traces de transformation à partir d'exemples de transformation. Ces traces peuvent être utilisées par exemple pour localiser des erreurs durant l'exécution des programmes de transformation ou vérifier la couverture de tous les modèles d'entrée par une transformation. Dans notre contexte, nous supposons que ces traces vont servir pour un futur apprentissage des règles de transformation. Nous traitons tout d'abord le problème de récupération des traces avec des exemples provenant d'un programme de transformation. Nous proposons une approche basée sur une méta-heuristique multi-objectifs pour générer des traces sous forme d'appariement de type n-m entre des éléments de modèles. La fonction objectif s'appuie sur une similarité lexicale et structurelle entre ces éléments. Une extension de cette méthode est proposée pour traiter le problème plus général de l'appariement entre modèles.

Mots-clés *IDM, transformation de modèles, l'approche par exemple, règles opérationelles, traces de transformation, appariement des modèles, AFC, JESS, meta-heuristique, algorithmes génétiques*

Exploiting Model Transformation Examples for Easy Model Transformation Handling (Learning and Recovery)

Model Driven Engineering (MDE) considers models as first class artifacts. Each model conforms to another model, called its metamodel which defines its abstract syntax and its semantics. Various kinds of models are handled successively in an MDE development cycle. They are manipulated using, among others, programs called model transformations. A transformation takes as input a model in a source language and produces a model in a target language. The developers of a transformation must have a strong knowledge about the source and target metamodels which are involved and about the model transformation language. This makes the writing of the model transformation difficult.
In this thesis, we address the problem of assisting the writing of a model transformation and more generally of understanding how a transformation operates. We adhere to the Model Transformation By Example (MTBE) approach, which proposes to create a model transformation using examples of transformation. MTBE allows us to use the concrete syntaxes defined for the metamodels. Hence, the developers do not need in-depth knowledge about the metamodels. In this context, our thesis proposes two contributions. As a first contribution, we define a method to generate operational transformation rules from transformation examples. We extend a previous approach which uses Relational Concept Analysis as a learning technique for obtaining transformation patterns from 1-1 mapping between models. We develop a technique for extracting relevant transformation rules from these transformation patterns and we use the JESS language and engine to make the rules executable. We also study how we better learn transformation rules from examples, using transformation examples separately or by gathering all the examples. The second contribution consists in recovering transformation traces from transformation examples. This trace recovery is useful for several purposes as locating bugs during the execution of transformation programs, or checking the coverage of all input models by a transformation. In our context, we expect also that this trace will provide data for a future model transformation learning technique. We first address the trace recovery problem with examples coming from a transformation program. We propose an approach, based on a multi-objective meta-heuristic, to generate the *many-to-many* mapping between model constructs which correspond to a trace. The fitness functions rely on the lexical and structure similarity between the constructs. We also refine the approach to apply it to the more general problem of model matching.

Keywords *MDE, model transformation, MTBE, operational rules, model transformation traces, model matching, FCA, JESS, meta-heuristic, genetic algorithm.*

LIRMM — 161, rue Ada — 34095 Montpellier cedex 5 — France