

# Learning transformation rules from transformation examples: An approach based on Relational Concept Analysis

Xavier Dolques, Marianne Huchard, Clémentine Nebut and Philippe Reitz

LIRMM

Université Montpellier 2, CNRS

Montpellier, France

Email: {lastname}@lirmm.fr

**Abstract**—In Model Driven Engineering (MDE), model transformations are basic and primordial entities, thus easing their design and implementation is an important issue. A quite recently proposed way to create model transformations consists in deducing a transformation from examples of transformed models. Examples are easier to write than a transformation program and are often already available. We propose in this paper a method based on a machine learning method of the lattice domain, the Relational Concept Analysis, and an implementation of this method.

a running example. Then, in Section III, we describe how to obtain the information from our examples to generate the rules using an extension of Formal Concept Analysis taking into account relations inside models and mapping links. Section IV describes the method we use to generate transformation rules from the lattices, and the corresponding tool. Section V presents related work and we conclude in Section VI.

## I. INTRODUCTION

Model transformation is a crucial topic in model driven development (MDD) [1], [2]. The many model transformation languages and tools can only be used by transformation experts, with a strong knowledge concerning the transformation language by itself, the metamodels of the involved source and target models, and the underlying meta-model. Domain experts generally do not have sufficient skills in model-driven technologies. Model Transformation By Example (MTBE) [3] is a proposal to let them design model transformations, based on their knowledge of the domains, masking the technological complexity of MDD. The domain expert is asked to give examples: a set of representative source models are created, as well as the corresponding target models, and links explaining in which target element(s) one or several source model elements are transformed. From those examples, transformation rules are automatically deduced, using a learning approach.

In this paper, we present a Model Transformation By Example approach, from examples to transformation rules. The proposed underlying learning approach is based on Relational Concept Analysis (RCA). RCA is an extension of the Formal Concept Analysis theory, that classifies a set of objects based on their properties. Applying RCA on the examples results in a classification of source and target elements of the transformation. Then, the classification of the mappings between source and target results in a set of rules partially ordered in a lattice. We propose an analysis of these rules to filter the relevant rules.

The following of this paper is structured as follows. We introduce our problem into details in Section II, through

## II. PROPOSAL OVERVIEW USING AN EXAMPLE

In this section, we provide the reader with the flavor of our approach using the example of a simple model-to-model transformation that aims at transforming association members described by their role inside the association (normal member, treasurer, chairman) into persons described by their responsibility level. For example, *the chairman Joe in the association GreenFingers* is transformed into *a responsible person* while *the normal member Arthur in the association GreenFingers* is transformed into *a person without responsibility*.

Figure 1 shows the two metamodels involved in the transformation. The source metamodel represents associations. An association is composed of members that have a name. Each member of the association plays a role (*chairman*, *treasurer*, or *normalMember*). The target metamodel describes persons and responsibilities. A person has a name and is a *Responsible Person* or a *Person Without Responsibility*.

Two models conform to those metamodels are given in Figure 2 in the form of instance diagrams of the metamodels. On the left, the association *GreenFingers* is composed of Joe, the chairman, Jenny, the treasurer, and Arthur and Cindy two normal members. The association members are transformed into persons, respectively *Responsible Persons* or *Persons without Responsibility*, according to their role inside the association: chairmans and treasurers are transformed into responsible persons while normal members are transformed into persons without responsibility. Let us note that the transformation of members is thus based on their neighborhood,

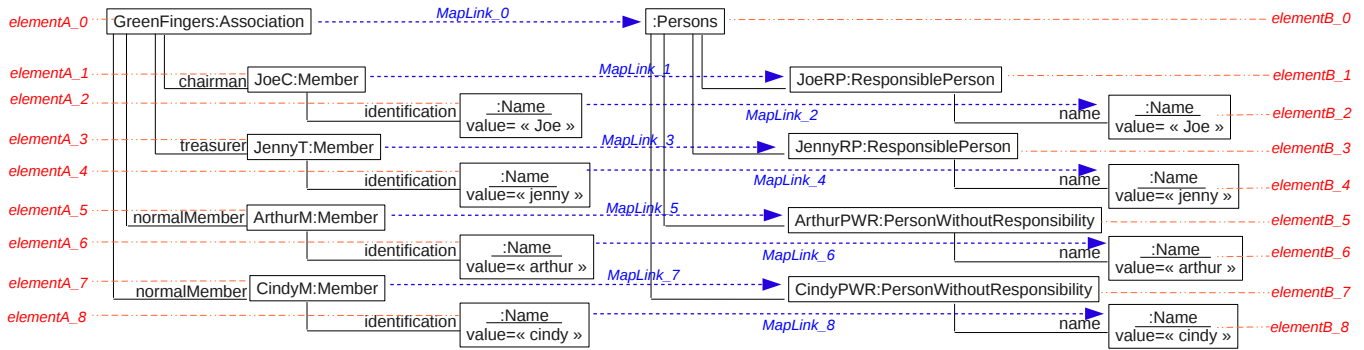


Figure 2. Models (in the form of instance diagrams) for associations (lhs) and responsibilities (rhs) and the transformation example (*MapLinks*).

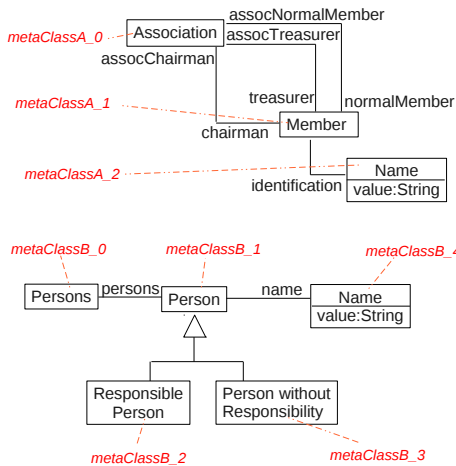


Figure 1. Metamodels for associations (up) and responsibilities (down).

here the roles `assocChairman`, `assocTreasurer`, or `assocNormalMember`.

Figure 2 also shows a transformation example from the association towards persons and responsibilities. The nine mapping links of the transformation have been given by a designer who indicated with directed dashed lines the correspondences between the source and the target models.

Inferring the transformation rules mainly consists in finding common features of source (resp. target) elements connected by the transformation. Among these common features, the neighborhood of the elements has to be considered (e.g. member are transformed differently according to the role they have w.r.t. the association). Formal Concept Analysis [4], which is a clustering method able to find commonalities in the description of objects, appears to be a relevant approach for such a type of problem. To be more precise in learning the transformation rules, we will use Relational Concept Analysis [5], an extension of FCA. RCA takes into account objects described by their relations with other objects.

Our approach takes as input a transformation example and includes three steps: first, classification of the elements

of source and target models of the transformation example (here two instance diagrams); second, classification of the mapping links that show how elements are connected by the transformation example; third, interpretation of the resulting concepts into transformation rules. Next section describes the first two steps.

### III. CLASSIFICATION OF MODEL ELEMENTS AND LINKS

In this section we show how to characterize patterns involved in the transformation example through model elements and link classification.

From now on, we use the suffixes A and B to differentiate source models from target models.

#### A. Classification of source model elements

In our example, the source model elements are the boxes from the left of Figure 2. Each of them is an instance of a metaclass from the source metamodel of Figure 1. We aim at grouping the source model elements by common characteristics in order to identify later which characteristics induce which transformation rule. This will include classification of the metaclasses which are part of the description of the elements themselves. But we want to capture the fact that the elements that instantiate a same metaclass may have different meanings and may be transformed in different ways: for example a `Member` is transformed into a `Responsible Person` or into a `Person Without Responsibility` depending whether it is linked through a `assocNormalMember`.

We thus take into account the following characteristics to classify the model elements:

- the metaclass of each model element, a metaclass being characterized by its name. For example, we take into account that `elementA_1` (JoeC) has for metaclass `metaClassA_1` (named `Member`), and that `elementA_2` (the name Joe) has for metaclass `metaClassA_2` (named `Name`), etc.
- the way model elements are linked with roles. For example, we take into account that `elementA_1`

(JoeC) is linked to elementA\_2 (the name Joe) by the role identification.

Technically, those characteristics are presented in the form of binary tables that are required for the RCA process.

FCA and RCA are clustering techniques which extract concepts, namely groups of entities sharing characteristics from binary tables describing these entities. In RCA, the description takes into account relations between the entities. This approach gives sound mathematical foundations to our classification construction. Here the entities are our metaclasses and model elements, which will be clustered. For metaclasses, we limited the description to their name for the sake of simplicity, but it could include relations inside the metamodel. For model elements, we use their metaclasses and the associations in the model.

The application of RCA results in lattices, that organize concepts (entity groups) in a partial order structure. The lattices built using the RCA tool eRCA<sup>1</sup> for our example are presented in Figure 3.

Each concept is a box with 3 parts: the top part is the name, the middle part is the intent of the concept (shared characteristics) and the bottom part is the extent (covered entities). Inheritance applies in the lattice: if an attribute (resp. object) is represented in a concept intent (resp. extent), it will be owned by all the concepts below (resp. above). For our example, we obtain two lattices, one for the metaclasses, and one for the model elements. To illustrate the lattices, we here detail several concepts. In the *metamodelA* lattice, there are three non trivial concepts, each one representing one of the metaclasses. The only given description being the name, each concept groups only one meta-class. For example Concept<sub>3</sub> represents Member meta-class. In the *modelA* lattice, groups are more informative. Concept<sub>26</sub> represents elementA<sub>0</sub> (GreenFingers). Concept<sub>32</sub> groups elementA<sub>5</sub> (Arthur) and elementA<sub>7</sub> (Cindy) which share: characteristic assocNormalMember; Concept<sub>26</sub> (being associated with GreenFingers through the role normal member); inherited characteristic metaClassA: Concept<sub>3</sub> (being an instance of Member); inherited identification: Concept<sub>29</sub> (having a name).

### B. Classification of target model elements

The target metamodel and model elements are similarly classified into two lattices (presented in Figure 4), exploiting the same kind of description for the model elements: their meta-class (the meta-class being described by its name), and the way the model elements are linked by roles.

We present into more details several concepts of the obtained lattices. In the *metamodelB* lattice, there are three non trivial concepts, each one representing

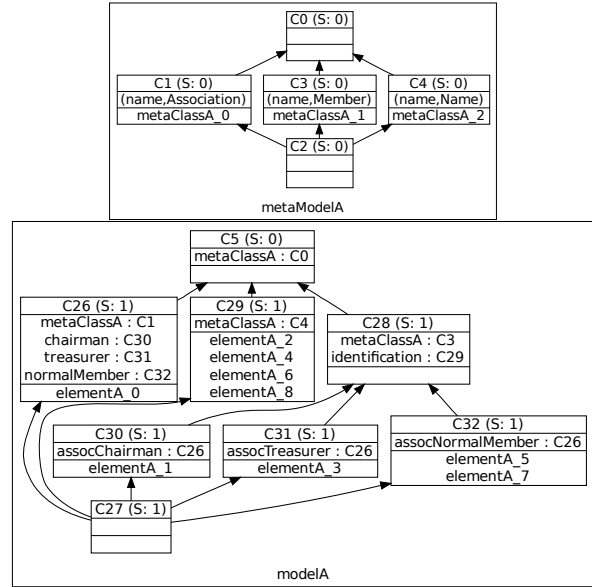


Figure 3. Lattices obtained by RCA for the RCF describing source metamodel and model

one of the metaclasses. The only given description being the name, each concept groups only one meta-class. For example Concept<sub>11</sub> represents Person Without Responsibility meta-class. In the *modelB* lattice, groups are more informative. Concept<sub>36</sub> groups elementB<sub>5</sub> (Arthur) and elementB<sub>7</sub> (Cindy) which share: characteristic metaClassB: Concept<sub>11</sub> (being an instance of person without responsibility); inherited characteristic name: Concept<sub>37</sub> (having a name).

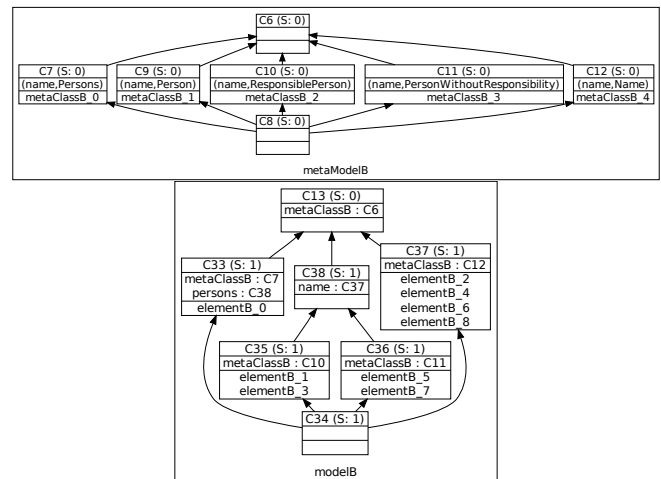


Figure 4. Lattice obtained by RCA for the RCF describing target metamodel and model

### C. Classification of transformation links

Transformation links are given in a transformation example to describe the correspondences between several

<sup>1</sup><http://code.google.com/p/erca/>

elements of two models. We consider here 1-1 links (one source element transformed into one target element). n-m links are encoded by several 1-1 links.

We explain here how we classify those links, using RCA. A mapping link is characterized by its source element denoted *mappingA* and its target element denoted *mappingB*. For example, mapping link *MapLink\_1* has for *mappingA* (source) *elementA\_1* (the member JoeC) and for *mappingB* (target) *elementB\_1* (the responsible person JoeRP)

Figure 5 shows the classification obtained for the mapping links. Detailing the concept *Concept\_46*, we understand that the links *MapLink\_5* and *MapLink\_7* are grouped because they share: a source described by *Concept\_32* (members connected to an association through the normal member role) and a target described by *Concept\_36* (persons without responsibility). Such a concept can also be understood as a rule: *transform members connected through normal member role to an association into persons without responsibility*. This mapping link lattice is thus a good structure to identify the transformation rules, provided that interpretation mechanisms are defined. In the next section, we go deeper in the rule generation mechanisms and choice by an expert in a representation of the mapping link lattice with good visual properties.

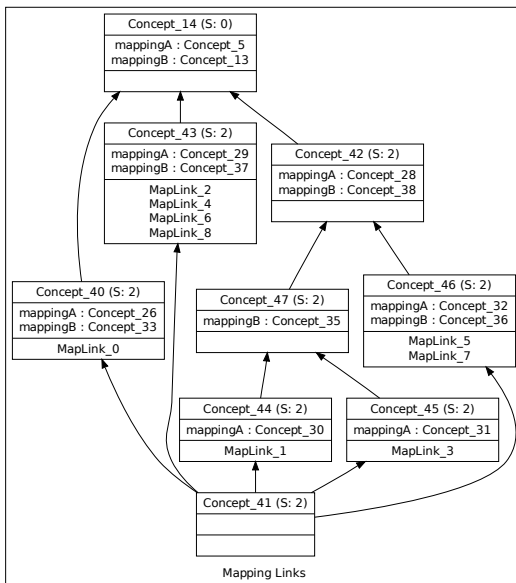


Figure 5. Lattice of mapping links

#### IV. CONCEPT INTERPRETATION AND RULE GENERATION

The result from the RCA process is a set of five concept lattices where a concept intent often has references to other concepts (not necessarily in the same lattice). These lattices are therefore quite difficult to read that is why the transformation rules are extracted from the *MapLink*

lattice by transforming it into a rule lattice presented in Figure 6. The concept characterizing the source elements (*mappingA* values in Figure 5) is used to get the premise of the rule, while the concept characterizing the target elements (*mappingB* values in Figure 5) is used to build the conclusion.

Figure 6 is a visualization of the rule lattice where boxes are the rules. The structure of the lattice is the same as the structure of the *MapLink* lattice (Figure 5). For example, Rule 4 is deduced from *Concept\_46* of the *MapLink* lattice. Some of the concepts from this lattice cannot be translated as rules. Some of the upper concepts describe too general mappings: e.g. the *Concept\_42* is referring to *Concept\_38* from *modelB* and this concept groups all the elements which have a *name* reference. We consider that having a type for the premise and conclusion element is a mandatory condition to create a rule. The bottom concept has usually an empty extent so it should not either be considered.

The deduction is represented by a larger arrow, its origin represents the premise and its target represents the conclusion. A premise consists in an element constrained by its neighborhood. In the figure the neighbors are linked to an element by thin arrows, the label of the arrow represents the name of the role of the neighbor and its cardinality.

For example, *Member* in Rule 4 comes from the *Concept\_32* which describes members. *Member* is connected to an identification and also through normal member role to an association (neighbourhood at distance 1), this association being connected to members via the roles chairman, treasurer (neighbourhood at distance 2), etc. The conclusion of the rule is an element (at the end of the large arrow) and its neighbourhood. For Rule 4, the conclusion is *Concept\_36*, that is *Person Without Responsibility* with a name (neighbourhood at distance 1). This neighbourhood of the source or target element can be exploited going more or less deeply in this neighbourhood, to form the rule.

The lattice offers a structure to navigate among the rules thus choosing the relevant rules is facilitated. The upper rules are more general, like the rule 5 which describes that a *Member* who is not a normal member can be transformed into a *Responsible Person*. This rule is specialized by rules 6 and 7 which describe respectively that a chairman is transformed into a *Responsible Person* and that a treasurer is transformed into a *Responsible Person*.

To facilitate the reading of the next part, we will use the following convention: *Lattice.c* refers to the concept *c* owned by the lattice *Lattice*.

For a rule stemming from a concept *MapLink.c*, required properties for the premise (resp. conclusion) are obtained by analysing the concept *ModelA.c'* (resp. *ModelB.c'*), which is the most specialized concept of *ModelA* (resp. *ModelB*) in *c* intent. There are three categories of characteristics:

**Mandatory characteristics**, described in *ModelA.c'* (resp. *ModelB.c'*) intent. They are the characteristics com-

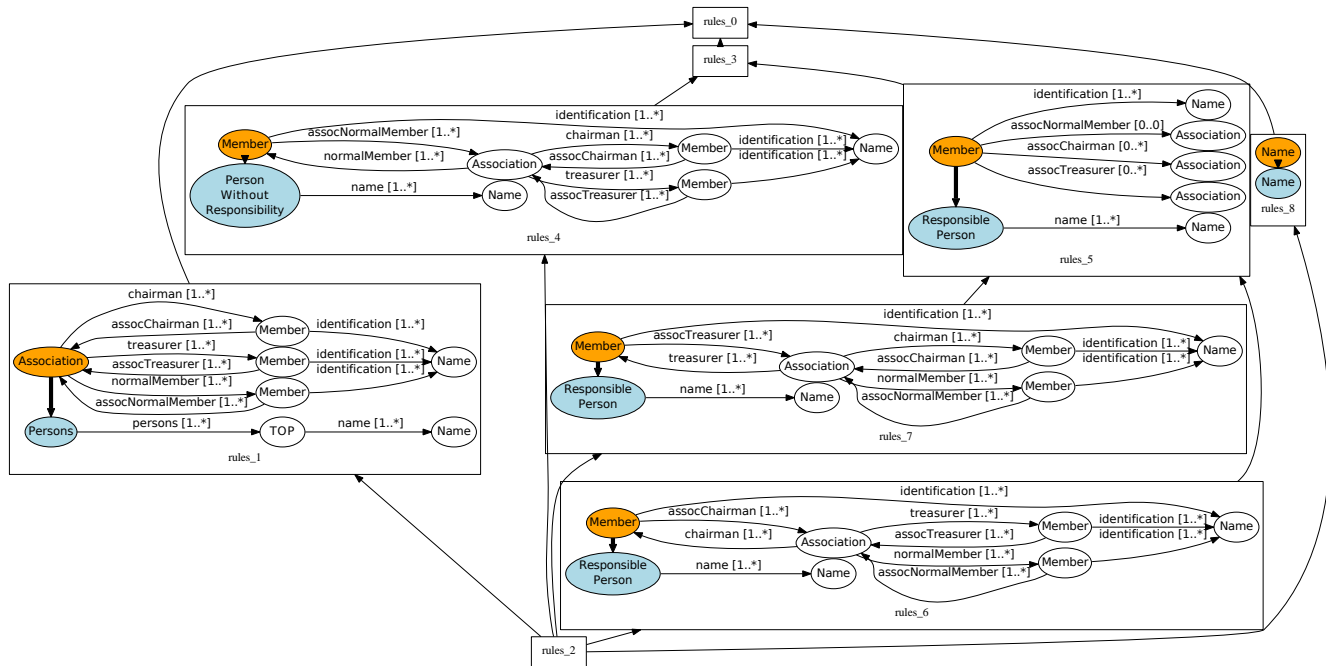


Figure 6. Rule lattice

mon to all the mapping links contained in *Maplink.c* extent. They are represented in the Rule Lattice by the associations with (1,\*) cardinality

**Authorized characteristics**, described in the intents of concepts specializing *ModelA.c'* (resp. *ModelB.c'*), under the condition that the concept extent includes the end of a link of *c*. They are the characteristics that may appear but have no incidence in the way an element is to be transformed. They are represented in the Rule Lattice by the associations with (0,\*) cardinality. We see two authorized characteristics in rule 5: if a *Member* is transformed in a *Responsible Person* it can be a *chairman* or a *treasurer* in the *association*.

**Forbidden characteristics**, described by the intents of the concepts which do not include in their extent the end of a link of *c*. These characteristics are especially important if they belong to concepts specializing *ModelA.c'* (resp. *ModelB.c'*). They are the characteristics that never appear in the source or target of the rule transformation. They are represented in the Rule Lattice by the associations with (0,0) cardinality. We see an authorized characteristic in rule 5: if a *Member* is transformed in a *Responsible Person* it cannot be a *Normal Member*.

We developed a tool supporting our approach. The tool, mainly written in Java/EMF, supports as an Eclipse plug-in the full process from the mapping to the transformation rules. The mappings are taken as input of the process in the form of models conform to the mapping metamodel we defined. From the mapping and the corresponding models, contexts are generated for the models and the mappings, according to the description given in section III. The RCA

process by itself is applied, using the MDE tool described in [6]. The main work achieved by our tool is then to build the rules.

## V. RELATED WORK

The automatic generation of model transformation is a recent research topic. Roots and inspiration can be found in the domains of ontology and schema matching [7], [8] and programming by-example or by-demonstration [9], [10]. Here we only describe the by-example approaches. In [11], ATL rules are derived from transformation examples and mappings written in concrete syntax. Links are made from the concrete syntax to the abstract syntax, in particular with OCL constraints. The algorithm to build a metamodel mapping uses these constraints. That requires the links between the abstract and concrete syntax to be well defined and implemented in an editor, making it hardly generalizable for any metamodel without a prior development cost. The mapping generation is not described into details, so that it is difficult to precisely compare the two approaches. However, the main contribution of [11] is to provide an approach directly dealing with concrete syntax, whereas we are based on abstract syntax. Our approach would clearly benefit from a bridge from mappings in concrete syntax to mappings in abstract syntax.

Kessentini et al. [12], [13] discuss the transformation model as an optimization problem. The particle swarm optimization in [12] and the simulated annealing in [13] are used to generate a consistent model transformation. The transformation of the source model is encoded as solutions

to potential elementary transformation solutions, which are placed in the search space of possible transformations. A possibility of transformation is associated to each element of the model to transform. The quality of this transformation is evaluated and refined at each iteration. The optimization problem is to find the best possible combination of transformations that maximizes the overall quality of processing. This optimization approach based on meta-heuristics is fundamentally different from ours: we produce rules whereas they produce a black-box way of generating the output models.

Varró [3] lays the foundations for a generation process of transformations by examples, based on inductive logic programming algorithms [14]. This process is presented as iterative and interactive, since the transformation designer must interfere to provide examples illustrating critical cases of the transformation to be inferred, but also to select or generalize the transformation rules obtained at the end of an iteration. Varro's approach uses pre-defined mappings between the metamodel elements, defining to which type of target model element a type of source model element is transformed. If we consider the example introduced in our paper, that means that we have to specify as input that elements conform to Association are transformed into elements conform to Persons. Those initial informations are refined using the examples, to obtain what is called contextual conditions. In our approach, we just use the examples to learn the rules to obtain at the same time the mappings between the meta-model elements and the contextual conditions. Furthermore, we classify the rules in the rule lattice in order to help the programmer to choose the right ones.

References [15], [16] describe the early specification of the approach detailed in this paper. The novelty in this paper consists in the definition of a rule lattice that gathers the informations from all the lattices generated and presents the rules in a practical structure, and a tool implementing the approach.

## VI. CONCLUSION AND FUTURE WORK

In this paper, we have presented an approach and a tool to generate model transformation rules using examples of transformed models and mapping links between source and target elements. This allows engineers involved in Model Engineering tasks to quickly have a transformation program even if they are not familiar with transformation languages and metamodels. Using Formal Concept Analysis, rules are classified through a lattice which helps navigation and choice. Improving the tool and the underlying process using in particular other RCA scaling operators, we expect to enhance the produced rules and detect new patterns for rule premise and conclusion.

## REFERENCES

- [1] T. Stahl, M. Völter, and K. Czarnecki, *Model-driven software development: technology, engineering, management*. John Wiley & Sons, 2006.
- [2] S. Sendall and W. Kozaczynski, "Model transformation: The heart and soul of model-driven software development," *IEEE Software*, vol. 20, no. 5, pp. 42–45, 2003.
- [3] D. Varró, "Model transformation by example," in *Proc. MODELS 2006, LNCS 4199*. Springer, 2006, pp. 410–424.
- [4] B. Ganter and R. Wille, *Formal Concept Analysis, Mathematical Foundations*. Springer, 1999.
- [5] M. Huchard, M. R. Hacene, C. Roume, and P. Valtchev, "Relational concept discovery in structured datasets," *Ann. Math. Artif. Intell.*, vol. 49, no. 1-4, pp. 39–76, 2007.
- [6] G. Arévalo, J.-R. Falleri, M. Huchard, and C. Nebut, "Building abstractions in class models: Formal concept analysis in a model-driven approach," in *MoDELS*. Springer, 2006, pp. 513–527.
- [7] E. Rahm and P. A. Bernstein, "A survey of approaches to automatic schema matching," *VLDB J.*, vol. 10, no. 4, pp. 334–350, 2001.
- [8] P. Shvaiko and J. Euzenat, "A survey of schema-based matching approaches," in *J. Data Semantics IV, Volume 3730 of LNCS*, 2005, pp. 146–171.
- [9] A. Cypher, D. C. Halbert, D. Kurlander, H. Lieberman, D. Maulsby, B. A. Myers, and A. Turransky, *Watch What I Do: Programming by Demonstration*. The MIT Press, 1993.
- [10] H. Lieberman, *Your Wish is My Command: Giving Users the Power to Instruct their Software*. Morgan Kaufmann, 2000.
- [11] M. Wimmer, M. Strommer, H. Kargl, and G. Kramler, "Towards model transformation generation by-example," in *HICSS*. IEEE Computer Society, 2007, p. 285.
- [12] M. Kessentini, H. Sahraoui, and M. Boukadoum, "Model Transformation as an Optimization Problem," in *MODELS'08, LNCS 5301*. Springer, 2008, pp. 159–173.
- [13] —, "Méta-modélisation de la transformation de modèles par l'exemple : approche par méta-heuristiques," in *LMO'09: Langages et Modèles à Objets*. Cépaduès, 2009, pp. 75–90.
- [14] Z. Balogh and D. Varró, "Model transformation by example using inductive logic programming," *Software and Systems Modeling*, 2008, appeared online.
- [15] X. Dolques, M. Huchard, and C. Nebut, "Génération de transformation de modèles par application de l'ARC sur des exemples," in *LMO'09: Langages et Modèles à Objets*, 2009, pp. 61–75.
- [16] —, "From transformation traces to transformation rules: Assisting model driven engineering approach with formal concept analysis," in *Supplementary Proceedings of ICCS'09*, 2009, pp. 15–29.