From transformation traces to transformation rules: Assisting model driven engineering approach with formal concept analysis

Xavier Dolques, Marianne Huchard, Clémentine Nebut

From transformation traces to transformation rules: Assisting Model Driven Engineering approach with Formal Concept Analysis

Xavier Dolques[1], M. Huchard[1], and C. Nebut[1]

LIRMM, Université de Montpellier 2 et CNRS
161, rue Ada, 34392 Montpellier cedex 5, France
{dolques, huchard, nebut}@lirmm.fr

Abstract. In this paper we are interested in semi-automatically generating labelled graph (model) transformations conform to a particular syntax (meta-model). Those transformations are basic operations in model driven engineering. They are usually developed by specialised programmers and for every change the source code must be updated. Our proposition is about generating transformation rules between two particular syntaxes using transformation examples (transformation traces) as input data. Examples are easier to write than a transformation program and often are already available. We are proposing a method based on FCA using relational descriptions of objects to find transformation rules. This method has been implemented and tested on transformations such as LaTeX to HTML.

1 Introduction

Model Driven Engineering (MDE) is a recent paradigm that gives models a predominating role in the software development process. A well known initiative of the Object Management Group (OMG) in this domain is Model Driven Architecture (MDA) [1]. Rather than developing programs in specific implementation languages, engineers are encouraged to produce and maintain high-level models describing the domain and the specific problem they deal with. Models are written in conformity with metamodels that capture concepts of the modeling language, *e.g.* classes and attributes for UML class diagrams, or entities and relationships for Entity-Relationship (ER) diagrams. Tools that assist modeling or that semi-automatically transform models in several directions [2], *e.g.* going from abstract models to code models or translating models from a modeling language to another, accompany the approach. One consequence of this generalized usage of models inside a wide range of tools and contexts is the huge amount of metamodels. Some of them are regrouped in *zoos*, like the Atlantic Zoo [1]. Successive versions of modeling and meta-modeling languages are another source of diversity. Then, the success of MDE approach strongly depends on the

[1] http://www.emn.fr/x-info/atlanmod/index.php/Atlantic

easiness to develop metamodel-to-metamodel transformations, either dedicated to a development step, or to guarantee tool interoperability.

The transformations are developed with general-purpose languages like Java, or with specialized languages, as ATL [3] or QVT [4]. Most of them are rather simple, because they mainly associate a pattern in the target model to a pattern in the source model. One case of such transformation changes the metamodel which is used to express concepts of a domain: going from UML to ER, or going from UML to Java are usual transformation tasks for software developers.

Programmers of model transformations must have serious skills in the chosen transformation language and in the involved metamodels, however in practice few programmers have this kind of skills. For example, many programmers cleverly handle UML models, but do not know the underlying metamodel.

In this paper, we propose a method inspired by "Programming By-example" approaches [5] to alleviate the writing of transformations. Engineers only need to handle models in their usual (concrete) syntax and to describe main cases of a transformation, namely *examples*. Such a transformation example includes the source model, the target model and trace links that make explicit how elements from the source model are transformed into elements of the target model. The transformation rules are generated from the transformation traces, using formal concept analysis extended by relations.

We introduce our problem into detail in Section 2, through a running example. Then, in Section 3, part of this example is used to present the extension of Formal Concept Analysis we will use to take into account relations inside models and traces. Section 4 describes the method we use to generate transformation rules from the lattices. Section 5 reports a case study. Related works are presented in Section 6 and we conclude in Section 7.

2 Problem overview using an example

A classical transformation used during the workshop MTIP'2005 [6] is used to illustrate our proposal. UML models (class diagrams) are converted into Entity-Relationship models. Simplified metamodels (also called abstract syntax) of UML and ER formalisms are presented in Figure 1.

The simplified UML metamodel describes the main concepts, also called metaclasses, that are used in UML structural models (class diagrams): classes which are a subset of types, properties owned by classes, associations connecting classes through properties. Attributes `upper` and `lower` indicate how many values a property can have (or how many objects can be connected when property is used in the context of an association). A simple UML model, given in usual concrete syntax in the lhs of Fig. 2, gathers two classes `Account` and `Client`, respectively owning properties `number` and `name`, and an association `owns` associating `Account` and `Client` through properties `owner` and `ownedAccount`. An account is linked to exactly one client while a client can have an unlimited number of accounts.

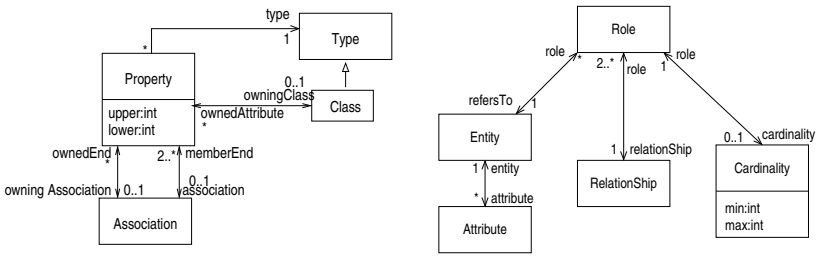

Fig. 1. Simplified metamodels for UML (lhs) and Entity-Relationship (rhs) used in [7].

The metaclasses of the simplified ER metamodel are entities which have attributes, relationships which connect entities through roles and cardinalities that restrict the connection number. The rhs of Fig. 2 presents an ER model composed of two entities `Account` and `Client`, with attributes `number` and `name`, and connected via the relation `possess` and roles `owner` and `ownedAccount`.

Fig. 2 shows a transformation trace given by a designer who indicated in dashed lines the correspondences between the source and target model.

Another view of the transformation trace is shown in Figure 3, where models are written using abstract syntax. Each element identifier (*e.g.* `Account`) is followed by the name of the metaclass (*e.g.* `Class`) it belongs to. Going from concrete syntax to abstract syntax is discussed in [7].

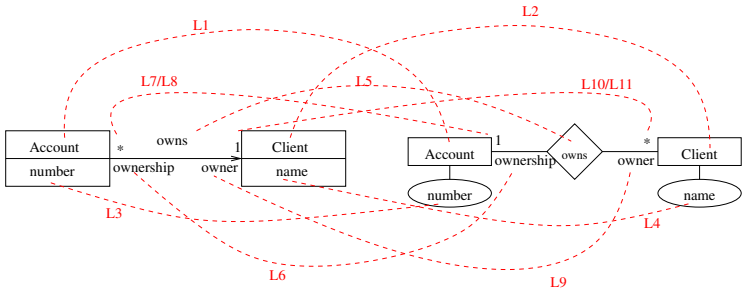

Fig. 2. UML (lhs) and ER (rhs) models in concrete syntax, and transformation links.

This example illustrates a common task in MDE: conversion between similar metamodels. A UML *Property* can be converted into an *Attribute* or a *Role* depending on the context thus even in simple cases, it is not possible to create a transformation rule using only the metaclass of the source elements: Analysing the neighborhood of a *property* converted into an *attribute* (for example *property* number), we find that the property is only connected to a *Class* through *owningClass*. In the case of a *property* converted into a *role* (for example *property* ownership) connections are with an *association* with a *Class* through *type*.

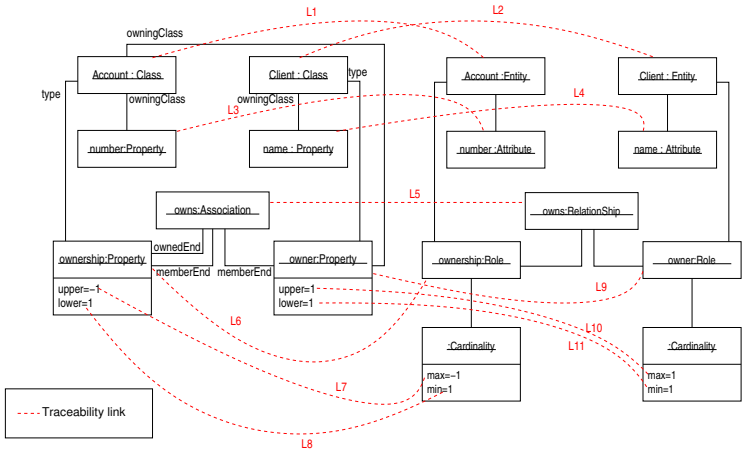

Fig. 3. UML model, ER model, in abstract syntax, and transformation links. Inspired by [7]. Some role names are omitted for the sake of clarity.

The found transformation rules can be expressed as follows, in a syntax close to declarative transformation languages.

```
Rule c10
For all Property p
where p is connected to a Class through owningClass
   and p is not connected to an Association through association
create Attribute a
```

As we have seen through this very restricted example, inferring the transformation rules mainly consists in finding common features of source elements and target elements connected by the transformation. Among these common features, the neighborhood of the elements has to be considered. For all these reasons, Formal Concept Analysis can be a relevant approach, if we consider an extension able to include and exploit links in the description of elements.

3 Relational Concept Analysis

In this section, we briefly recall Formal Concept Analysis [8] and Relational Concept Analysis, the extension we will use in our approach.

Formal Concept Analysis A **formal context** is denoted by $K = (O, A, I)$, O is an object set, A is an attribute set and $I \subseteq O \times A$. $(o, a) \in I$ when a is an attribute of o.

The UML example of Figure 3 is encoded in a formal context where O are the model elements and A is divided in two subsets: A_1 encodes the types and A_2 encodes the types of the neighbors (Tab.1). $(o, a) \in (O, A_1)$ is in I if a is the metaclass of o; $(o, a) \in (O, A_2)$ if a is the metaclass of a neighbor of o.

Table 1. Formal context encoding the UML model of Fig. 3.

	type			neighbor type		
	Class	Property	Association	Class	Property	Association
Account	X				X	
Client	X				X	
owner		X		X		X
ownership		X		X		X
owns			X		X	
number		X		X		
name		X		X		

A **concept** is a pair (X, Y) with $X \subseteq O$, $Y \subseteq A$ and $X = \{o \in O | \forall y \in Y, (o, y) \in I\}$ is the extent (covered objects), $Y = \{a \in A | \forall x \in X, (x, a) \in I\}$ is the intent (shared attributes).

The concept lattice associated to Table 1 is given in Figure 4 with a simplified labelling. Each box represents a concept: the name in the upper part, the simplified intent in the middle part and the simplified extent in the lower part.

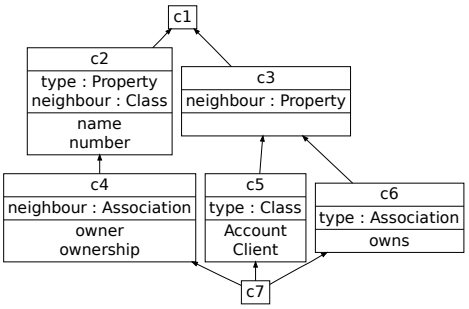

Fig. 4. Concept lattice associated with Tab.1

The example of Fig. 4 shows a classification of the model elements depending of their type and their neighbor type. The elements with type *Property* (concept $c2$) have in their neighborhood at least an element of type *Class*. But some of these elements, regrouped in the subconcept $c4$, also contain in their neighborhood at least an element of type *Association*.

As shown in this example, it is possible with only one incidence relation I to represent several kinds of characteristics of an object set. To have an accurate description, we can use as formal attribute the type of the neighbors of an element, and we could use also the link name which connects the element to its neighbor. Nevertheless, in a single step, we cannot take into account the created concepts. One solution is then to create a context which associates initial objects and new created concepts through a special incidence relation. Applying such an approach characterizes an element by its neighbors, the neighbors of its neighbors, etc. to have a definition of the concept as accurate as possible.

Relational Concept Anaysis Relational Concept Analysis [9] is one of the extensions of Formal Concept Analysis that considers links between objects in the concept construction. Considering these links leads to take into account concepts created in one step of the process to enhance the object description and create new abstractions at next steps. Connections can be made with other FCA-based proposals to deal with relational descriptions or complex structures including [10,11,12,13] to mention just a few.

Relational Concept Analysis (RCA) computes concepts based on a relational context family composed of one or two formal contexts, as well as *relational* contexts. A relational context describes a relation between objects of two formal contexts (not necessarily different).

A **relational context family** $\mathcal{R}$ is a kind of multicontext [14] presented as a pair (K, R). K is a set of formal contexts $K_i = (O_i, A_i, I_i)$, R is a set of relational contexts $R_j = (O_k, O_l, I_j)$ (O_k et O_l are the object sets of K_k et K_l de K). O_k is called the source of R_j.

With RCA, data are divided into several object sets. In our example, we consider the elements of the models (context $K_1 = (O_1, A_1, I_1)$) and the metaclasses of the metamodels (context $K_2 = (O_2, A_2, I_2)$), these two contexts are shown in Figure 5. K_1 objects are described by two kinds of characteristics: their metaclass and their neighbors. The relation which connects an element to its metaclass (resp. neighbor) is represented by a relational context included in $O_1 \times O_2$ (resp. $O_1 \times O_1$).

Each association end of the metamodel (e.g. `owningClass`) is encoded into a separated relational context (Fig. 7). The attribute set of K_1 is empty. K_2 elements are described by a unique identifier in order to generate a first lattice where each object belongs to a concept different from the others (Fig. 6). The metaclasses are not used as attributes in K_1 to obtain a clearer modeling and an easy evolution. Relations describing metaclasses can be easily added, for example inheritance. For the sake of clarity, inheritance is not encoded and metaclass *Type* does not appear in K_2.

New abstractions emerge iterating two steps. The first step is classical concept lattice construction. In the second step, formal contexts are added to relational contexts enhanced by concepts created in previous lattice construction, then lattices are built.

Initialisation step. Lattices are built at this step using FCA. For each formal context K_i, a lattice $\mathcal{L}_i^0$ is created (in our example, it is shown in Fig. 6).

Step n+1. For each relational context $R_j = (O_k, O_l, I_j)$, an enhanced relational context $R_j^s = (O_k, A, I)$ is created. A is the concept set of the lattice $\mathcal{L}_l^n$ (created at step n). Incidence relation I contains the set of pairs (o, a) s.t. $S(R(o), Extent(a))$ is *true*, where S is a *scaling* operator. The scaling operator we use in the rest of the paper is $S_\exists(R(o), Extension(a))$, which is *true* iff $\exists x \in R(o), x \in Extent(a)$. Other operators could be used, and especially $S_{\forall\exists}(R(o), Extent(a))$, which is *true* iff $\forall x \in R(o), x \in Extent(a) \wedge \exists x \in R(o), x \in Extent(a)$. For each formal context K_i extended with enhanced relations with source O_i, the lattice $\mathcal{L}_i^{n+1}$ is created.

K_1	
Account	
Client	
owner	
ownership	
owns	
number	
name	

K_2	idClass	idProperty	idAssociation
Class	X		
Property		X	
Association			X

Fig. 5. Initial formal contexts K_1 (up) et K_2 (down) for the UML model.

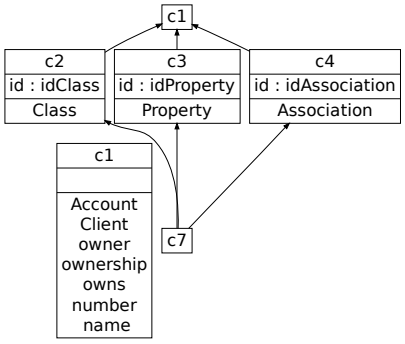

Fig. 6. Lattice at initialization step for contexts K_1 (lhs) and K_2 (rhs)

owningClass	Account	Client
number	x	
name		x
owner	x	

ownedAttribute	number	name	owner
Account	x		x
Client		x	

type	Account	Client
ownership	x	
owner		x

owningAssociation	owns
ownership	x

memberEnd	ownership	owner
owns	x	x

association	owns
ownership	x
owner	x

ownedEnd	ownership
owns	x

meta-class	Class	Property	Association
Account	X		
Client	X		
owner		X	
ownership		X	
owns			X
number		X	
name		X	

Fig. 7. Relational contexts. Contexts in the upper part correspond to relations in $K_1 \times K_1$. Context in the lower part is a relation in $K_1 \times K_2$.

The computed lattice stemming from the context K_1 extended by relational contexts is shown in Figure 8. The final lattice stemming from the context K_2 is still the lattice of Fig. 6 because K_2 is never the source of a relational context.

4 Generating transformation rules

Our approach includes three steps: first, classification of the elements of source and target models of the transformation; second, classification of the links that show how elements are connected by the transformation; third, transformation of the resulting concepts into transformation rules.

4.1 Classification of model elements

Elements of a model (necessarily conform to metamodel) yet belong to an explicit classification driven by their metaclass. But the elements that instantiate a same

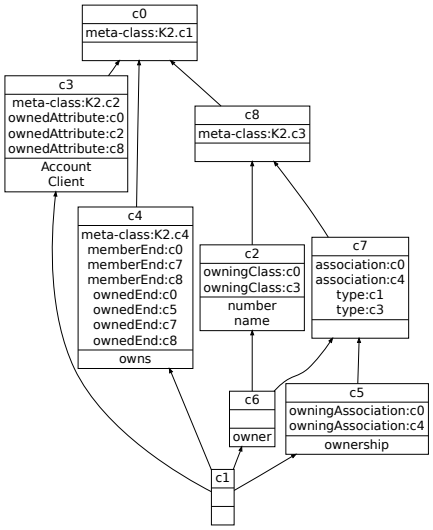

Fig. 8. Final lattice obtained by RCA for K_1.

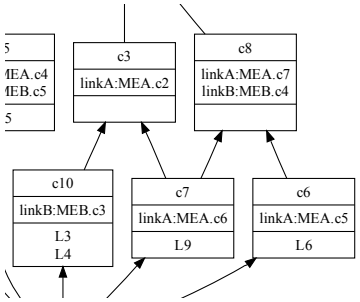

Fig. 9. Part of final MapLink lattice obtained by RCA. MEA (resp. MEB) refers to ModelElementA (resp. ModelElementB).

metaclass may have different meanings and be transformed in different ways: *e.g.* UML properties can be transformed into ER attribute or roles. To capture these different meanings, we study the neighborhood of elements: an UML *Property* used as a role is linked to an *association*, while an UML *property* used as an attribute has a link *owningClass* with a class and no link with an association.

The use of RCA allows us to obtain a classification of the elements of a model taking into account this neighborhood. We have seen in the previous section parts of the used modeling. For a given model, a first formal context *ModelElement* is created which contains all the model elements (it is equivalent to the context K_1 of Section 3). Objects of this context are the model elements and the set of attributes is an empty set. A second formal context called *MetaModelElement*, contains the metamodel elements (it is equivalent to the context K_2 of section 3). A relational context connects *ModelElement* and *MetaModelElement*: the incidence relation is here the link between an element and its metaclass (its type), this is equivalent to the context *metaclass* in Figure 7). This leads to classify elements of *ModelElement* according to their metaclasses. Each relation end R of the metamodel is encoded into a relational context in $ModelElement \times ModelElement$ (Tables of Figure 7). Using those relations refines the classification of *ModelElement*.

The result of this encoding is a concept lattice that describes the elements available in a model. Concept intents will be used to generate part of the transformation rules. Figure 8 presents the concept lattice describing the elements of the UML model. For example, concept c_2 describes *Properties* which are connected

through *owningClass* links to *Classes*. More precisely, elements of the simplified extent of c_2 (namely *number* and *name*) are not connected to anything else.

4.2 Classification of transformation links

Transformation links are given in a transformation example to describe the correspondences between several elements of two models. We consider here 1-1 links (one source element transformed into one target element), or 1-n links (one source element transformed into n target elements). Links are encoded into a formal context *MapLinks*: objets are the links and there are no attributes. From the *MapLinks* context, we build the lattice which is used to generate the transformation rules.

For two models A and B involved in the transformation, we create two relational contexts $LinkA \subseteq MapLinks \times ModelElementA$ and $LinkB \subseteq MapLinks \times ModelElementB$. An element of *MapLinks* is connected to a concept of the *ModelElement* lattice if one element of the concept extent is end of the link. The lattice[2] represents a classification of links based on their ends.

4.3 Concept interpretation and rule generation

A transformation link is characterized by the concept containing its end in model A and the concept containing its end in model B. The concepts of the lattice built on top of *MapLinks* regroup links which have common characteristics in their two ends. For example, concept c_{10} (lattice of Figure 9) regroups links that connect properties linked to classes and not to associations (concept c_2, lattice Fig. 8 classifying UML elements), and attributes (concept c_3 of the lattice classifying ER elements). We propose to extract the transformation rules using these characteristics. The description of source elements ($linkA$ values) can be seen as premise of the rule, while the description of target elements ($linkB$ values) can be interpreted as the conclusion. Concept c_{10} leads to the Rule 10 of Section 2: the premise is derived from the UML concept c_2 and expresses that the involved source elements are precisely *the properties p where p is connected to a Class through owningClass and p is not connected to an Association through association*. The conclusion of the rule is derived from the ER concept c_3, expressing that the target element is an ER attribute.

In other words, we consider a rule as a mapping taking as parameter a source element conform to metamodel A. If this element satisfies given required characteristics, it is transformed into target elements conform to metamodel B and satisfying other required characteristics. All required characteristics are indicated inside the concepts.

For a rule stemming from a concept c of the lattice associated with *MapLink*, required properties are obtained analysing the concept *MEA.c'* (for *ModelElementA.c'*), which is the most specialized concept of *ModelElementA* in c intent. There are three categories of characteristics:

[2] All the lattices are at this url: http://www.lirmm.fr/~dolques/publications/data/iccs09

Mandatory characteristics, described in *MEA.c'* intent. They are common to all c links.

Authorized characteristics, described in the intents of concepts specializing *MEA.c'*, under the condition that the concept extent includes the end of a link of c. For example, in the case of the concept c_8 of *MapLink* regrouping links L_6 and L_9 (Figure 3), *MEA.c'* corresponds to $MEA.c_7$; *ownership* (end of L_6) which belongs to $MEA.c_5$ and *owner* (end of L_9) which belongs to $MEA.c_6$ are then authorized.

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 *MEA.c'*. In our example, if we consider the concept of *MapLink* regrouping links L_3 et L_4, *MEA.c'* corresponds to $MEA.c_2$. The rule must not include description coming from $MEA.c_6$ since no element of its extent is end of L_3 or L_4.

5 Case Study

This section provides a proof-of-concept of our technique in the form of a case study. We are interested here in the quality and usability of the obtained rules.

By *quality*, we mean the rules adequacy with what a developer would have done by hand: ideally, the results obtained with our approach should give similar or identical rules. The usability evaluates if the number of obtained rules does not explode, as could unhappily be expected with lattices.

5.1 Preparation

In order to gather experimental data, we organized a session with graduate students around model transformation, during their *Model Driven Engineering* class. The models and metamodels were written using the Eclipse Modeling Framework (EMF) and the Sample Reflective Ecore Model Editor, the students were familiar with them. We created small models conform to metamodels that students were used to handle. These models were given to the students as source examples and they were asked to create by-hand a transformed model conform to another given metamodel. They were also asked to write the trace links between the two models and the transformation using an imperative language.

These models were given to the students as source examples, as well as the specification of a model transformation. The transformation was specified with written natural language, and also orally explained by the teacher. The target metamodels of the transformations were also familiar to the students. The students were asked to create by-hand the result model. The students were told that their work will be used for a research experiment but they did not know the exact purpose of the experiment. They had all the time they wanted to produce their data, but they worked alone and two nearby students in the classroom were given different models. We then gathered all the data and applied our RCA-technique on them. We have then studied the lattices to extract the rules.

The metamodels used for this case study are simplified versions of LaTeX and HTML, the subjects were asked to make transformations from one to another in both directions.

Table 2. Data obtained from the case study.

	d15	d16	d43	d44	d25	d26	d29
Transformation	latex to html				html to latex		
Source MetaModel size	3	3	3	3	7	7	7
Target MetaModel size	7	7	7	7	3	3	3
Source Model size	9	12	9	12	13	13	13
Target Model size	11	14	14	22	12	11	12
Trace size	9	12	13	21	10	11	10
Source MetaModel coverage	3	3	3	3	3	4	3
Rules Space size	5	5	5	4	4	4	3
Detected problems	2	2	4	3	1	1	0
Real problems	2	2	2	1	1	1	0
Number of links	9	12	13	14	9	11	10
Number of correct links	8	11	11	14	8	11	10
Number of links from rules without problems	4	10	4	14	8	11	10
Number of correct links from rules without problems	4	10	4	14	8	11	10

From the obtained data, we gathered the results presented in Table 2. At first, we measured the size of all the handled models. The metamodel size is the number of its *EClass*. The model size corresponds to the number of its metamodel *EClass* instances. A trace size is given by its number of links.

We also measured the source metamodel coverage, using the number of *EClass* for which instances were traced. The final lattice used to generate the rules contains concepts that cannot be used as they do not contain enough information to make a rule. The Rules Space defines the set of concepts that are usable as rules.

During the experiments, a lot of factors could cause error, and we needed to know whether the technique was sensible to errors. We have determined that concepts with the same source but different targets could cause problems, that is what is measured for the detected problems. We then checked if these problems were real and if all problems were detected. By problems, we mean trace link errors or an incoherent target model.

For the last metrics, we applied the rules we obtained on the source example, and we measured the difference between the result obtained and the target model example. As in some cases the rules do not cover all the elements in the source model, we have based our measurement on trace links. As some problems have

appeared in the data used to generate the rules, we also made measurement excluding links from the rules that cause problems due to trace errors.

5.2 Evaluation

The results on the different models seem satisfying as every rule applied, if it has been generated from sane data, behaves in the expected way. We also see that errors can be treated in most cases: they were all detected on our data and good rules can be obtained from the remaining sane model elements. But we can also see some cases of false positives, that could be more frequent with bigger models.

A problem that could arise when working with RCA is the size of the lattice, and especially the number of possible rules. But, this does not appear to be a problem with the transformation we have here.

5.3 Threats to validity

- *Conclusion validity:* Our results are valid as a proof-of-concept: we conducted our experiments on a little number of models. The subjects knowledge was good enough.
- *Internal validity:* The high number of problems comes from the used tools (Eclipse Model Editor) and the edition of XMI files, this is highly error-prone.
- *Construct validity:* The correctness of the transformation is measured by the transformation of an element in another element with the good type, but we do not take into account the relations between the transformed elements.
- *External validity:* We are using here little metamodels and examples, as it is needed to make a proof-of-concept, but it makes results hardly generalizable on bigger metamodels. However, metamodels are reasonably different so that the transformations to generate are not obvious.

6 Related Work

In Model-Driven Engineering domain, the automatic generation of model transformation is a recent and active research topic. Roots and inspiration can be found in the domains of ontology and schema matching [15,16] and programming by-example or by-demonstration [17,5].

Metamodel alignement-based approaches search for mappings that provide transformation rules. In [18], metamodels are mapped to a pivot ontology, then an ontology-based reasoning is used to generate a relational-QVT transformation. In [19], refactoring is applied to metamodels in order to make explicit hidden concepts of metamodels and obtain an ontology where all concepts are reified before mapping. *Similarity Flooding* [20] propagates similarity values in a labeled graph whose vertices are potential mappings. It is used in several proposals for metamodel alignment such as in [21] and [22].

Another track of research aims at inferring transformation rules from transformation examples (traces). Particle swarm optimization is used in [23] to generate a consistent transformation of a model. The transformation of a model source element is encoded as a particle which has to be placed in the space of possible transformations. Graph transformation rules are semi-automatically derived from mappings given by a user between two models in [24]; Analysis is then based on model element neighboring, inductive logics and interaction with an expert. ATL rules are derived from transformation examples written in concrete syntax in [7].

Meta-model alignment approaches are especially suitable to give mappings in the case of rather simple transformations in the context of tool interoperability or version changes. Approaches based on rule inference have a larger application spectrum including complex transformations or transformations where metamodels are quite different. RCA allows us

to map connected elements (patterns) rather than isolated elements. Lattices classify results and help navigation among generated rules to choose the relevant ones. Compared to the opaqueness of the optimization approach which just provides a transformation result, inferred rules provide a transformation procedure, and are a clear and easy-to-handle artefact.

7 Conclusion and future work

In this paper, we propose to generate model transformation rules using examples of transformed models and transformation links between source and target elements. This allows engineers involved in Model Engineering tasks to rapidly 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. On simple models, we show that the results are satisfying as the method produces correct rules most of the time and is usable on erroneous data. From these encouraging results we plan to test our approach on bigger and more complex transformations. But to achieve this goal, we need to complete our tools for generating and manipulating rules. By using other scaling operators we expect to enhance the produced rules and detect new patterns for rule premise and conclusion.

References

1. Soley, R., the OMG Staff: Model driven architecture. Technical report, Object Management Group (2000)
2. Mens, T., Gorp, P.V.: A taxonomy of model transformation. Electr. Notes Theor. Comput. Sci. **152** (2006) 125–142
3. Jouault, F., Kurtev, I.: Transforming models with atl. In Bruel, J.M., ed.: MoDELS Satellite Events, Springer (2005) 128–138
4. OMG: MOF^{TM} Query / Views / Transformations. Technical report, OMG (2008)
5. Lieberman, H.: Your Wish is My Command: Giving Users the Power to Instruct their Software. Morgan Kaufmann (2000)

6. MTIP: Model transformations in practice workshop. http://sosym.dcs.kcl.ac.uk/events/mtip05/ (2005)
7. Wimmer, M., Strommer, M., Kargl, H., Kramler, G.: Towards model transformation generation by-example. In: HICSS, IEEE Computer Society (2007) 285
8. Ganter, B., Wille, R.: Formal Concept Analysis, Mathematical Foundations. Springer (1999)
9. Huchard, M., Hacene, M.R., Roume, C., Valtchev, P.: Relational concept discovery in structured datasets. Ann. Math. Artif. Intell. **49**(1-4) (2007) 39–76
10. Priss, U.: Classification of meronymy by methods of relational concept analysis. In: Online Proceedings of the 1996 Midwest Artificial Intelligence Conf., Bloomington, Indiana. (1996)
11. Prediger, S., Wille, R.: The lattice of concept graphs of a relationally scaled context. In: Proc. of the 7th Intl. Conf. on Conceptual Structures (ICCS'99), Springer (1999) 401–414
12. Ganter, B., Kuznetsov, S.: Pattern structures and their projections. In Delugach, H., Stumme, G., eds.: Conceptual Structures: Broadening the Base, Proc. of the 9th Intl. Conf. on Conceptual Structures (ICCS'01), Stanford, CA. Volume 2120 of LNCS., Springer (2001) 129–142
13. Ferré, S., Ridoux, O., Sigonneau, B.: Arbitrary relations in formal concept analysis and logical information systems. In: ICCS 2005. Volume 3596 of LNCS., Springer (2005) 166–180
14. Wille, R.: Conceptual structures of multicontexts. In: Conceptual Structures: Knowledge Representation as Interlingua, Springer (1996) 23–39
15. Rahm, E., Bernstein, P.A.: A survey of approaches to automatic schema matching. VLDB J. **10**(4) (2001) 334–350
16. Shvaiko, P., Euzenat, J.: A survey of schema-based matching approaches. In: J. Data Semantics IV, Volume 3730 of LNCS. (2005) 146–171
17. Cypher, A., Halbert, D.C., Kurlander, D., Lieberman, H., Maulsby, D., Myers, B.A., Turransky, A.: Watch What I Do: Programming by Demonstration. The MIT Press (1993)
18. Roser, S., Bauer, B.: An approach to automatically generated model transformations using ontology engineering space. In: Proceedings of Workshop on Semantic Web Enabled Software Engineering (SWESE). (2006)
19. Kappel, G., Kapsammer, E., Kargl, H., Kramler, G., Reiter, T., Retschitzegger, W., Schwinger, W., Wimmer, M.: Lifting metamodels to ontologies: A step to the semantic integration of modeling languages. In: MoDELS. Volume 4199 of Lecture Notes in Computer Science., Springer (2006) 528–542
20. Melnik, S., Garcia-Molina, H., Rahm, E.: Similarity flooding: A versatile graph matching algorithm and its application to schema matching. In: ICDE, LNCS 2593. (2002) 117–128
21. Lopes, D., Hammoudi, S., Abdelouahab, Z.: Schema matching in the context of model driven engineering: From theory to practice. In: Advances in Systems, Computing Sciences and Software Eng., Springer (2006) 219–227
22. Falleri, J.R., Huchard, M., Lafourcade, M., Nebut, C.: Meta-model Matching for Automatic Model Transformation Generation. In: MODELS'08, LNCS 5301, Springer (2008) 326–340
23. Kessentini, M., Sahraoui, H., Boukadoum, M.: Model Transformation as an Optimization Problem. In: MODELS'08, LNCS 5301, Springer (2008) 159–173
24. Balogh, Z., Varró, D.: Model transformation by example using inductive logic programming. Software and Systems Modeling (2008) Appeared online.