CHAPTER FOUR

Model Transformation Using Multiobjective Optimization

Mohamed Wiem Mkaouer, Marouane Kessentini
SBSE Laboratory, CIS Department, University of Michigan, Michigan, USA

Contents

Abstract

The evolution of languages and software architectures provides a strong motivation to migrate/transform existing software systems. Thus, more attention is paid to the transformation aspects in model-driven engineering (MDE) along with the growing importance of modeling in software development. However, a major concern in MDE is how to ensure the quality of the model transformation mechanisms. Most of existing work in model transformation has relied on defining languages to express transformation rules. The main goal of existing transformation approaches is to provide rules generating

Advances in Computers, Volume 92
ISSN 0065-2458
http://dx.doi.org/10.1016/B978-0-12-420232-0.00004-0

target models, from source models, without errors. However, other important objective is how to minimize the complexity of transformation rules (e.g., the number of rules and number of matching in the same rule) while maximizing the quality of target models. In fact, reducing rule complexity and improving target model quality are important to (1) make rules and target models easy to understand and evolve, (2) find transformation errors easily, and (3) generate optimal target models. In this chapter, we consider the transformation mechanism as a multiobjective problem where the goal is to find the best rules maximizing target model quality and minimizing rule complexity. Our approach starts by randomly generating a set of rules, executing them to generate some target models. Of course, only solutions ensuring full correctness are considered during the optimization process. Then, the quality of the proposed solution (rules) is evaluated by (1) calculating the number of rules and matching metamodels in each rule and (2) assessing the quality of generated target models using a set of quality metrics. To this end, we use the nondominated sorting genetic algorithm (NSGA-II) to automatically generate the best transformation rules satisfying the two conflicting criteria. We report the results of our validation using three different transformation mechanisms. The best solutions provided well-designed target models with a minimal set of rules.

1. INTRODUCTION

Model transformation plays an important role in model-driven engineering (MDE) [1]. The research efforts by the MDE community have produced various languages and tools [2–5] for automating transformations between different formalisms using mapping rules. These transformation rules can be implemented using general programming languages such as Java or C#, graph transformation languages like AGG [2] and the VIsual Automated model TRAnsformations (VIATRA) [3], or specific languages such as ATLAS Transformation Language (ATL) [4, 5] and the Query/View/Transformation (QVT) [6]. Sometimes, transformations are based on invariants (preconditions and postconditions specified in languages such as the Object Constraint Language (OCL) [7]).

One major challenge is to automate transformations while preserving the quality of the produced models [1]. Thus, the main goal is to reduce the number of possible errors when defining transformation rules. These transformation errors have different causes such as transformation logic (rules) or source/target metamodels. Existing approaches and techniques have been successfully applied to transformation problems with a minimum number of errors. Especially at the model level, correctness is the gold standard characteristic of models: it is essential that the user understands exactly how the target model deviates from fidelity to the source model in order to be able to rely on any results. However, other important objectives are how to

minimize the complexity of transformation rules (e.g., the number of rules and number of matching in the same rule) while maximizing the quality of target models to obtain well-designed ones. In fact, reducing rule complexity and improving target model quality are important to (1) make rules and target models easy to understand and evolve, (2) find transformation errors easily, and (3) generate optimal target models.

The majority of existing approaches [1, 2, 5] formulate the transformation problem as a single-objective problem that maximizes rule correctness. In this case, the proposed transformation rules produce target models without errors. However, these rules are sometimes complex (e.g., size) and applying them may generate very large target model, for example, complex transformation rules in mapping from dynamic Unified Modeling Language (UML) models to colored Petri nets (CPN); their systematic application will generally results in large PNs [8]. This could compromise the subsequent analysis tasks, which are generally limited by the number of the PN states. Obtaining large PNs is not usually related to the size of the source models but to the rule complexity [9]. In addition, it is important to take into consideration the quality of produced target models (e.g., maximizing good design practices by reducing the number of bad smells [10] in a generated class diagram (CLD) from a relational schema (RS)). Another category of approaches [11, 12] proposes an additional step to minimize complexity, using refactoring operations [13, 14], after generating transformation rules. However, it is a difficult and fastidious task to modify, evolve, and improve quality of already generated complex rules.

In this chapter, to overcome some of the previously mentioned limitations, we propose to alternatively view transformation rule generation as a multiobjective problem. We generate solutions matching the source metamodel elements to their equivalent target ones, taking into consideration two objectives: (1) minimizing rule complexity and (2) maximizing target model quality. We start by randomly generating a set of rules, executing them on different source models to generate some target models, and then evaluate the quality of the proposed solution (rules). Of course, during the optimization process, we select only solutions ensuring full correctness (generating correct target models/rules). Correctness is the gold standard characteristic of models: it is essential that the user understand exactly how the target model deviates from fidelity to the source model in order to be able to rely on any results. To ensure the transformation correctness, we used a list of constraints to satisfy when generating target models. For the first objective, it calculates the number of rules and number of matching metamodels in each rule (one-to-one, many-to-one, etc.). For the second objective, we

use a set of software quality metrics [15] to evaluate the quality of generated target models. To search for solutions, we selected and adapted, from the existing multiobjective evolutionary algorithms [16], the nondominated sorting genetic algorithm (NSGA-II) [17]. NSGA-II aims to find a set of representative Pareto-optimal solutions in a single run. In our case, the evaluation of these solutions is based on the two mentioned conflicting criteria.

The primary contributions of the chapter can be summarized as follows:

1. We introduce a new approach for model transformation using multiobjective techniques. Our proposal does not require to define rules manually, but only to input a set of source models and equivalent target models (without traceability links); it generates well-designed target models/rules without the need to refactor them; it takes into consideration the complexity of the generated rules; and it can be applied to any source or target metamodels (independent from source and target languages). However, different limitations are discussed in Section 6.3.
2. We report the results of an evaluation of our approach; we used three different transformation mechanisms to evaluate our proposal: CLDs to RS and vice versa and sequence diagrams (SDs) to CPN. The generated rules for both mechanisms achieved high-quality scores with a minimal set of rules.

The rest of this chapter is organized as follows. Section 2 is dedicated to the related work, while Section 3 describes the problem statement. The overview of our multiobjective proposal is described in Section 4. Section 5 explains the experimental method, the results of which are discussed in Section 6. The chapter concludes with Section 7.

2. STATE OF THE ART

2.1. Model Transformation Languages

Kleppe *et al.* [18] provide the following definition of model transformation, as illustrated in Fig. 4.1: a transformation is the automatic generation of a target model from a source model, according to a transformation definition. A transformation definition is a set of transformation rules that describe together how a model in the source language can be transformed into a model in the target language. A transformation rule is a description of how one or more constructs in the source language can be transformed into one or more constructs in the target language.

In the following, a classification of endogenous transformation (model-to-model) approaches is briefly reported. Then, some of the available

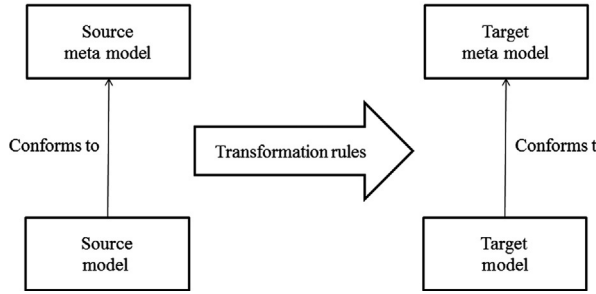

Figure 4.1 Model transformation process.

endogenous transformation languages are separately described. The classification is mainly based upon [18, 19]. Several endogenous transformation approaches have been proposed in the literature. In the following, classifications of model-to-model endogenous transformation approaches discussed by Czarnecki and Helsen [18, 19] are described.

2.1.1 Direct Manipulation Approach

It offers an internal model representation and some Application Programming interfaces (API) to manipulate it. It is usually implemented as an object-oriented framework, which may also provide some minimal infrastructure. Users have to implement transformation rules, scheduling, tracing, and other facilities in a programming language.

An example of used tools in direct manipulation approaches is Builder Object Network (BON), a framework that is relatively easy to use and is still powerful enough for most applications. BON provides a network of C++ objects. It provides navigation and update capabilities for models using C++ for direct manipulation.

2.1.2 Operational Approach

It is similar to direct manipulation but offers more dedicated support for model transformation. A typical solution in this category is to extend the utilized metamodeling formalism with facilities for expressing computations. An example would be to extend a query language such as OCL with imperative constructs. Examples of systems in this category are Embedded Constraint Language [20], QVT Operational Mappings [21], XMF [22], MTL [23], and Kermeta [24].

2.1.3 Relational Approach

It groups declarative approaches in which the main concept is mathematical relations. In general, relational approaches [25] can be seen as a form of constraint solving. The basic idea is to specify the relations among source and target element types using constraints that, in general, are nonexecutable. However, the declarative constraints can be given executable semantics, such as in logic programming where predicates can describe the relations. All of the relational approaches are side effect-free and, in contrast to the imperative direct manipulation approaches, create target elements implicitly. Relational approaches can naturally support multidirectional rules. They sometimes also provide backtracking. Most relational approaches require strict separation between source and target models, that is, they do not allow in-place update. Examples of relational approaches are QVT Relations and ATL [26]. Moreover, in Ref. [3], the application of logic programming has been explored for the purpose.

2.1.4 Graph Transformation-Based Approach

They exploit theoretical work on graph transformations and require that the source and target models be given as graphs [27]. Performing model transformation by graph transformation means to take the abstract syntax graph of a model and to transform it according to certain transformation rules. The result is the syntax graph of the target model. More precisely, graph transformation rules have an LHS and an RHS graph pattern. The LHS pattern is matched in the model being transformed and replaced by the RHS pattern in place. In particular, LHS represents the preconditions of the given rule, while RHS describes the postconditions. $LHS \cap RHS$ defines a part that has to exist to apply the rule, but that is not changed. $LHS - LHS \cap RHS$ defines the part that shall be deleted, and $RHS - LHS \cap RHS$ defines the part to be created. The LHS often contains conditions in addition to the LHS pattern, for example, negative conditions. Some additional logic is needed to compute target attribute values such as element names. Graph Rewriting and Transformation Language (GReAT) [28] and AToM3 [29] are systems directly implementing the theoretical approach to attributed graphs and transformations on such graphs. They have built-in fixed-point scheduling with nondeterministic rule selection and concurrent application to all matching locations.

Mens and Van Gorp [30] provide a taxonomy of model transformations. One of the main differences with the previous taxonomy is that Czarnecki and Helsen propose a hierarchical classification based on feature diagrams, while

the Mens *et al.* taxonomy is essentially multidimensional. Another important difference is that Czarnecki *et al.* classify the specification of model transformations, whereas Mens *et al.* taxonomy is more targeted toward tools, techniques, and formalisms supporting the activity of model transformation.

For these different categories, many languages and tools have been proposed to specify and execute exogenous transformation programs. In 2002, the Object Management Group (OMG) issued the Query/View/Transformation request for proposal [21] to define a standard transformation language. Even though a final specification was adopted at the end of 2008, the area of model transformation continues to be a subject of intense research. Over the last years, in parallel to the OMG effort, several model transformation approaches have been proposed from both academia and industry. They can be distinguished by the used paradigms, constructs, modeling approaches, tool support, and suitability for given problems. We briefly describe next some well-known languages and tools.

ATL [5] is a hybrid model transformation language that contains a mixture of declarative and imperative constructs. The former allows dealing with simple model transformations, while the imperative part helps in coping with transformations of higher complexity. ATL transformations are unidirectional, operating on read-only source models and producing write-only target models. During the execution of a transformation, source models may be navigated through, but changes are not allowed. Transformation definitions in ATL form modules. A module contains a mandatory header section, import section, and a number of helpers and transformation rules. There is an associated ATL Development Toolkit available as open source from the GMT Eclipse Modeling Project [31]. A large library of transformations is available at Ref. [4].

GReAT [32] is a metamodel-based graph transformation language that supports the high-level specification of complex model transformation programs. In this language, one describes the transformations as sequenced graph rewriting rules that operate on the input models and construct an output model. The rules specify complex rewriting operations in the form of a matching pattern and a subgraph to be created as the result of the application of a rule. The rules (1) always operate in a context that is a specific subgraph of the input and (2) are explicitly sequenced for efficient execution. The rules are specified visually using a graphical model builder tool called the Generic Modeling Environment (GME) [33].

AGG is a development environment for attributed graph transformation systems that support an algebraic approach to graph transformation. It aims at

specifying and rapid prototyping applications with complex, graph-structured data. AGG supports typed graph transformations including type inheritance and multiplicities. It may be used (implicitly in "code") as a general-purpose graph transformation engine in high-level Java applications employing graph transformation methods.

The source, target, and common metamodels are represented by type graphs. Graphs may additionally be attributed using Java code. Model transformations are specified by graph rewriting rules that are applied non-deterministically until none of them can be applied anymore. If an explicit application order is required, rules can be grouped in ordered layers. AGG features rules with negative application conditions to specify patterns that prevent rule executions. Finally, AGG offers validation support that is consistency checking of graphs and graph transformation systems according to graph constraints, critical pair analysis to find conflicts between rules (that could lead to a nondeterministic result), and checking of termination criteria for graph transformation systems. An available tool support provides graphical editors for graphs and rules and an integrated textual editor for Java expressions. Moreover, visual interpretation and validation are supported.

VIATRA2 [3] is an eclipse-based general-purpose model transformation engineering framework intended to support the entire life cycle for the specification, design, execution, validation, and maintenance of transformations within and between various modeling languages and domains. Its rule specification language is a unidirectional transformation language based mainly on graph transformation techniques. More precisely, the basic concept in defining model transformations within VIATRA2 is the (graph) pattern. A pattern is a collection of model elements arranged into a certain structure fulfilling additional constraints (as defined by attribute conditions or other patterns). Patterns can be matched on certain model instances, and upon successful pattern matching, elementary model manipulation is specified by graph transformation rules. There is no predefined order of execution of the transformation rules. Graph transformation rules are assembled into complex model transformations by abstract state machine rules, which provide a set of commonly used imperative control structures with precise semantics.

VIATRA2 is a hybrid language since the transformation rule language is declarative, but the rules cannot be executed without an execution strategy that should be specified in an imperative manner. Important specification features of VIATRA2 include recursive (graph) patterns, negative patterns

with arbitrary depth of negation, and generic and meta-transformations (type parameters and rules manipulating other rules) for providing reuse of transformations.

A conclusion to be drawn from studying the existing endogenous transformation approaches, tools, and techniques is that they are often based on empirically obtained rules [34]. In fact, the traditional and common approach toward implementing model transformations is to specify the transformation rules and automate the transformation process by using an executable model transformation language. Although most of these languages are already powerful enough to implement large-scale and complex model transformation tasks, they may present challenges to users, particularly to those who are unfamiliar with a specific transformation language. Firstly, even though declarative expressions are supported in most model transformation languages, they may not be at the proper level of abstraction for an end user and may result in a steep learning curve and high training cost. Moreover, the transformation rules are usually defined at the metamodel level that requires a clear and deep understanding about the abstract syntax and semantic interrelationships between the source and target models. In some cases, domain concepts may be hidden in the metamodel and difficult to unveil (e.g., some concepts are hidden in attributes or association ends, rather than being represented as first-class entities). These implicit concepts make writing transformation rules challenging. Thus, the difficulty of specifying transformation rules at the metamodel level and the associated learning curve may prevent some domain experts from building model transformations for which they have extensive domain experience.

To address these challenges inherited from using model transformation languages, an innovative approach called model transformation by example (MTBE) is proposed that will be described in the Section 2.2.

2.2. Model Transformation by Example

Examples play a key role in the human learning process. There are numerous theories on learning styles in which examples are used. For a description of today's popular learning style theories, see Refs. [35, 36].

Our work is based on using past transformation examples. Various "by-example" approaches have been proposed in the software engineering literature [37].

What does by example really mean? What do all by-example approaches have in common? The main idea, as the name already suggests, is to give the software examples of how things are done or what the user expects and let it do the work automatically. In fact, this idea is closely related to fields such as machine learning or speech recognition. Common to all by-example approaches is the strong emphasis on user-friendliness and a "short" learning curve. According to Baudry *et al.* [38], the by-example paradigm dates back to 1970—see "Learning Structure Descriptions from Examples" in Ref. [39].

Programming by example [36] is the best-known by-example approach. It is a technique for teaching the computer a new behavior by demonstrating actions on concrete examples. The system records user actions and generalizes a program that can be used for new examples. The generalization process is mainly based on user responses to queries about user intentions. Another well-known approach is query by example (QBE) [40]. It is a query language for relational databases that are constructed from filled sample tables with examples: rows and constraints. QBE is especially suited for queries that are not too complex and can be expressed in terms of a few tables. In web engineering, Lechner and Schrefl [41] present the language TBE (XML transformers by example) that allows defining transformers for WebML schemes by example, that is, stating what is desired instead of specifying the operations to get it. Advanced XSLT tools are also capable of generating XSLT scripts using examples from schema levels (like MapForce from Altova) or document (instance)-level mappings (such as the pioneering XSLerator from IBM Alphaworks or the more recent Stylis Studio).

The problems addressed by the previously mentioned approaches are different from ours in both the nature and the objectives.

The commonalities of the by-example approaches for transformation can be summarized as follows: All approaches define an example as a triple consisting of an input model and its equivalent output model and trace between the input and output model elements. These examples have to be established by the user, preferably in a concrete syntax. Then, generalization techniques such as hard-coded reasoning rules, inductive logic, or relational concept analysis are used to derive model transformation rules from the examples, in a deterministic way that is applicable for all possible input models that have a high similarity with the predefined examples.

Varro and Balogh [42, 43] propose a semiautomated process for MTBE using inductive logic programming. The principle of their approach is to derive transformation rules semiautomatically from an initial prototypical

set of interrelated source and target models. Another similar work is that of Wimmer *et al.* [44] that derives ATL transformation rules from examples of business process models. Both contributions use semantic correspondences between models to derive rules. Their differences include the fact that [44] presents an object-based approach that finally derives ATL rules for model transformation, while [45] derives graph transformation rules. Another difference is that they, respectively, use an abstract versus a concrete syntax: Varro uses IPL when Wimmer relies on an *ad hoc* technique. Both models are heavily dependent on the source and target formalisms. Another similar approach is that of Dolques *et al.* [46] that aims to alleviate the writing of transformations and where engineers only need to handle models in their usual (concrete) syntax and to describe the main cases of a transformation, namely, the examples. 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, and they are classified through a lattice that helps navigation and choice. This approach requires the examples to cover all the transformation possibilities and it is only applicable for one-to-one transformations.

Recently, a similar approach to MTBE, called model transformation by demonstration (MTBD), was proposed [47]. Instead of the MTBE idea of inferring the rules from a prototypical set of mappings, users are asked to demonstrate how the MT should be done, through direct editing (e.g., add, delete, connect, and update) of the source model, so as to simulate the transformation process. A recording and inference engine was developed, as part of a prototype called MT-Scribe, to capture user operations and infer a user's intention during an MT task. A transformation pattern is then generated from the inference, specifying the preconditions of the transformation and the sequence of operations needed to realize the transformation. This pattern can be reused by automatically matching the preconditions in a new model instance and replaying the necessary operations to simulate the MT process. However, this approach needs a large number of simulated patterns to be efficient, and it requires a high level of user intervention. In fact, the user must choose the suitable transformation pattern. Finally, the authors do not show how MTBD can be useful to transform an entire source model and only provide examples of transforming model fragments. On the other hand, the MTBD approach, in contradiction with other by-example approaches, is applied to endogenous transformations.

Another very similar by demonstration approach was proposed by Langer *et al.* [48]. The difference from Sun *et al.*'s work, which uses the recorded fragments directly, is that Langer *et al.* use them to generate ATL rules. Another difference is that the Langer approach is related to exogenous transformation.

Brosch *et al.* [49] introduced a tool for defining composite operations, such as refactorings, for software models in a user-friendly way. This by-example approach prevents modelers from acquiring deep knowledge about the metamodel and dedicated model transformation languages. However, this tool is only able to apply refactoring operations and does not detect automatically refactoring operations.

The commonalities of the by-example approaches for the exogenous transformation can be summarized as follows: All approaches define an example as a triple consisting of an input model and its equivalent output model and trace between the input and output model elements. The examples have to be established by the user, preferably in a concrete syntax. Then, generalization techniques such as hard-coded reasoning rules, inductive logic, or relational concept analysis are used to derive model transformation rules from the examples, in a deterministic way that is applicable to all possible input models that have a high similarity with the predefined examples.

None of the mentioned approaches claims that the generation of the model transformation rules is correct or complete. In particular, all approaches explicitly state that some complex parts of the transformation involving complex queries, attribute calculations such as aggregation of values, nondeterministic transformations, and counting of elements have to be developed by the user, by changing the generated model transformations. Furthermore, the approaches recommend developing the model transformations using an iterative methodology. This means that, after generating the transformations from initial examples, these examples can be adjusted or the transformation rules should be changed if the user is not satisfied with the outcome. However, in most cases, deciding that the examples or the transformation rules need changing is not an obvious process to the user.

2.3. Traceability-Based Model Transformation

Some other metamodel matching works can also be considered as variants of by-example approaches. Garcia-Magarino *et al.* [11] propose an approach to generate transformation rules between two metamodels that satisfy some

manually introduced constraints by the developer. In Ref. [50], the authors propose to automatically capture some transformation patterns in order to generate matching rules at the metamodel level. This approach is similar to MTBD, but it is used at the metamodel level.

Most current transformation languages [11, 51, 52] build an internal traceability model that can be interrogated at execution time, for example, to check if a target element was already created for a given source element. This approach is specific to each transformation language and sometimes to the individual transformation specification. The language determines the traceability metamodel, and the transformation specification determines the label of the traces (in case of QVT/relational, the traceability metamodel is deduced from the transformation specification). The approach taken only provides an access to the traces produced within the scope of the current transformation. Marvie describes a transformation composition framework [53] that allows manual creation of linkings (traces). These linkings can then be accessed by subsequent transformation, although this is limited to searching specific traces by name, introducing tight coupling between subtransformations.

In order to transform models, they need to be expressed in some modeling language (e.g., UML for design models and programming languages for source code models). The syntax and semantics of the modeling language itself are expressed by a metamodel (e.g., the UML metamodel). Based on the language in which the source and target models of a transformation are expressed, a distinction can be made between endogenous and exogenous transformations.

2.4. Search-Based Software Engineering

Our approach is largely inspired by contributions in search-based software engineering (SBSE). SBSE is defined as the application of search-based approaches to solve optimization problems in software engineering [54]. Once a software engineering task is framed as a search problem, there are numerous approaches that can be applied to cope with that problem, from local searches such as exhaustive search and hill climbing to metaheuristic searches such as genetic algorithms (GAs) and ant colony optimization [55].

Many contributions have been proposed for various problems, mainly in cost estimation, testing, and maintenance [40, 56]. Module clustering, for example, has been addressed using exhaustive search [55], GAs [56], and simulated annealing (SA) [57]. In those studies that compared search

techniques, hill climbing was perhaps surprisingly found to produce better results than metaheuristic GA searches. Model verification has also been addressed using search-based techniques. Shousha *et al.* [58] propose an approach to detect deadlocks in UML models, but the generation of a new quality predictive model, starting from a set of existing ones by using SA, is probably the problem that is the most similar to MT by examples. In that work, the model is also decomposed into fine-grained pieces of expertise that can be combined and adapted to generate a better prediction model. To the best of our knowledge, inspired among others by the road map paper of Harman [56], the idea of treating model transformation as a combinatorial optimization problem to be solved by a search-based approach was not studied before our proposal.

2.5. Summary

This section has introduced the existing work in different domains related to our work. The closest work to our proposal is MTBE. Once the examples have been established, generalization techniques, such as hard-coded reasoning rules, inductive logic [43], or relational concept analysis or pattern, are used to derive model transformation rules from the examples, in a deterministic way that is applicable for all possible input models that have a high similarity with the predefined examples.

Table 4.1 summarizes an existing transformation by-example approaches according to given criteria. The majority of these approaches are specific to exogenous transformation and based on the use of traceability.

One conclusion to be drawn from studying the existing by-example approaches is that they use semiautomated rule generation, with the

Table 4.1 By-Example Approaches

By-Example Approaches	Exogenous Transformation	Endogenous Transformation	Traceability	Rule Generation
[43]	X		X	X
[44]	X		X	X
[47]		X	X	
[46]	X		X	X
[48]	X		X	X
[49]		X	X	

generated rules further refined by the user. In practice, this may be a lengthy process and require a large number of transformation examples to assure the quality of the inferred rules. In this context, the use of search-based optimization techniques can be a more preferable transformation approach since it directly generates the target model from the existing examples, without using the rule step. This also leads to a higher degree of automation than existing by-example approaches.

As shown in Section 2.4, like many other domains of software engineering, MDE is concerned with finding exact solutions to these problems or those that fall within a specified acceptance margin. Search-based optimization techniques are well suited for the purpose. For example, when testing model transformations, the use of deterministic techniques can be unfeasible due to the number of possibilities to explore for test case generation, in order to cover all source metamodel elements. However, the complex nature of MDE problems sometimes requires the definition of complex fitness functions [59]. Furthermore, the definition is specific to the problem to solve and necessitate expertise in both search-based and MDE fields. It is thus desirable to define a generic fitness function, evaluating a quality of a solution that can be applied to various MDE problems with low adaptation effort and expertise.

To tackle these challenges, our contribution combines search-based and by-example techniques. The difference with case-based reasoning approaches is that many subcases can be combined to derive a solution, not just the most adequate case. In addition, if a large number of combinations have to be investigated, the use of search-based techniques becomes beneficial in terms of search speed to find the best combination.

3. MOTIVATIONS AND PROBLEM STATEMENT

In this section, we emphasize the motivations of our work and the specific problems that are addressed by our multiobjective approach.

3.1. Defining Transformation Rules

Although there is a consensus about the necessity of defining transformation rules, our experience with industrial partners showed that there are many open issues that need to be addressed when defining a transformation mechanism. Sometimes, the transformation may not be obvious to realize, due to different reasons [9]. The process of defining rules manually for model transformation is complex, time-consuming, and error-prone. Thus, we need to

define an automated solution to generate rules automatically instead of the manual process. One solution is to propose a semiautomated approach for a rule generation in order to help the designer. In the majority of existing approaches, the rules are generated from traceability links interrelating different source and target model examples. However, defining traces is a fastidious task because they are manually defined. Generating transformation rules can be difficult since the source and target languages may have elements with different semantics; therefore, one-to-one mappings are not often sufficient to express the semantic equivalence between metamodel elements. Indeed, in addition, to ensure structural (static) coherence, the transformation should guarantee a behavioral coherence in terms of time constraints and weak sequencing. In addition, various rule combination possibilities may be used to transform between the same source and target languages: how to choose between different possible rule combinations having the same correctness.

3.2. Reducing Transformation Complexity

In general, the majority of existing transformation approaches generates transformation rules without taking into consideration complexity (but only correctness). In such situations, applying these rules could generate large target models, it is difficult to test complex rules and detect/correct transformation errors, and it is a fastidious task to evolve complex rules (modifying the transformation mechanism) when the source or target metamodels are modified. Some transformation approaches [12, 60, 61] propose to refactor the rules after defining them. However, it is difficult to manipulate and modify complex rules. For this reason, it is better to optimize the complexity when generating the rules.

3.3. Improving Transformation Quality

The majority of model maintenance work [10, 12, 62, 63] is concerned with the detection and correction of bad design fragments, called design defects or bad smells, after the generation of target models [8]. Design defects refer to design situations that adversely affect the development of models [2]. In Ref. [62], Beck defines 22 sets of symptoms of common defects. For UML CLDs, these include large classes, feature envy, long parameter lists, and lazy classes. In most of the existing model transformation work, the main goal is to generate correct target models. The quality of target models is not considered when generating transformation rules. However, it is important

to ensure that generated transformation rules provide well-designed target models with a minimum number of bad smells. Otherwise, each target model should be revised to improve its design quality, which can be a fastidious task. In fact, detecting and fixing design defects is, to some extent, a difficult, time-consuming, and manual process [8].

4. APPROACH OVERVIEW

This section shows how the previously mentioned issues can be addressed using our proposal. This section starts by presenting an illustration of an example of the transformation mechanism. Then, we provide an overview of the approach and we discuss the computational complexity of our problem.

A model transformation mechanism takes as input a model to transform, the source model, and produces as output another model, the target model. The source and target models must conform to specific metamodels and, usually, relatively complex transformation rules are defined to ensure this.

We can illustrate this definition of the model transformation mechanism with the case of CLD-to-RS transformation. Our choice of CLD-to-RS transformation is motivated by the fact that it is well known and reasonably complex; this allows us to focus on describing the technical aspects of our approach. In Section 6, we show that our approach can also be applied to more complex transformations such as SDs to CPNs [64].

Figure 4.2A shows a simplified metamodel of the UML CLD, containing concepts like class, attribute, and relationship between classes. Figure 4.2B shows a partial view of the RS metamodel, composed of table, column, attribute, etc. The transformation mechanism, based on rules, will then specify how the persistent classes, their attributes, and their associations should be transformed into tables, columns, and keys.

Figure 4.3 shows the example of a source model, as CLD containing four classes and two association links, and its related target model.

The associated target model is expressed as an RS. Four classes are mapped to tables (Client, Order, Order details, and Product). The two association links become foreign keys. Finally, attributes in subclasses are mapped into columns of the derived table from the parent class. The CLD-to-RS transformation is used to illustrate our approach described in the rest of this chapter.

The general structure of our approach is introduced in Fig. 4.4. The following two sections give more details about our proposals.

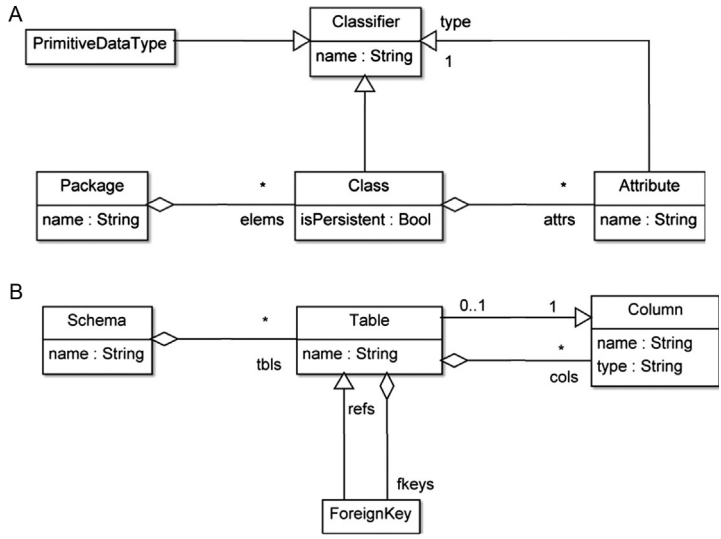

Figure 4.2 Class diagram and relational schema metamodels. (A) Class diagram metamodel. (B) Relational schema metamodel.

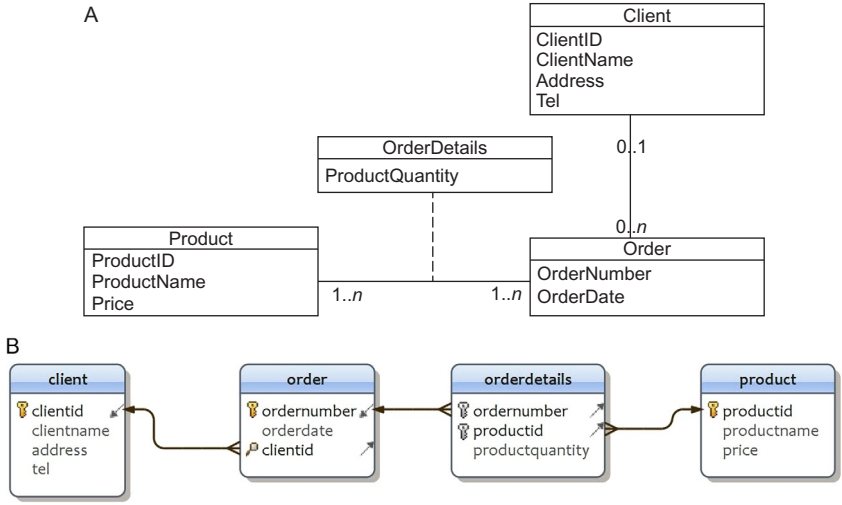

Figure 4.3 Models of class diagram with equivalent relational schema diagram. (A) Class diagram example. (B) Relational schema diagram example.

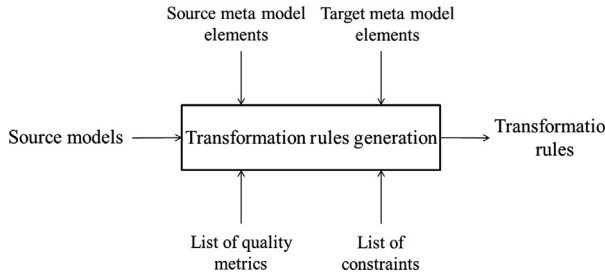

Figure 4.4 Overview of the approach: General architecture.

As described in Fig. 4.4, the number of source models and the expected target ones is used to generate the transformation rules. In fact, our approach takes as inputs a set of source models with their equivalent target models, a list of quality metrics, and another list of constraints (to ensure transformation correctness) and takes as controlling parameters a list of source and target metamodel elements. Our approach generates a set of rules as output.

The rule generation process combines source and target metamodel elements within rule expressions. Some logical expressions (union OR and intersection AND) can be used to combine between metamodel elements. Consequently, a solution to the transformation problem is a set of rules that transforms well the source models to target models within the satisfaction of the list of all transformation constraints. For example, the following rule states that a class is transformed to a table with the same name having a primary key:

```
R1: IF Class(A) THEN Table(A) AND Column(idA, A, pk).
```

In this example of a rule, a class, a table, and a primary key column correspond to some extracted elements from the source and target metamodels. The first part of the rule contains only elements from the source metamodel. Consequently, the second part of the rule contains only elements from the target metamodel.

To ensure the transformation correctness when generating transformation rules, the idea is that the transformation of source models into target models is coupled with a contract consisting of pre- and postconditions. Hence, the transformation is tested with a range of source models that satisfy the preconditions to ensure that it always yields target models that satisfy the postconditions. If the transformation produces an output model that violates a postcondition, then the contract is not satisfied and the transformation needs to be corrected. The contract is defined at the metamodel level and

conditions are generally expressed in OCL. We used these constraints as input in our approach.

After ensuring the transformation correctness, our multiobjective optimization process uses two criteria to evaluate the generated solutions. The first criterion consists of minimizing the rule complexity by reducing the number of rules and the number of matching metamodels in each rule. The second criterion consists of maximizing the quality of generated target models, based on different quality metrics. Quality metrics provide useful information that helps in assessing the level of conformance of a software system to a desired quality such as *evolvability* and *reusability*. For instance, Ehrig *et al.* and Marinescu [12, 63] propose different metrics to evaluate the quality of RSs such as depth of relational tree of a table T that is defined as the longest referential path between tables, from the table T to any other table in the schema database; referential degree of a table T (RD(T)) consisting of the number of foreign keys in the table T; percentage of complex columns metric of a table T; and size of a schema (SS) defined as the sum of the tables size (TS) in the schema.

We selected also a set of quality metrics that can be applied on CLDs as target models. These metrics include number of associations (Naccoc), the total number of associations; number of aggregations (Nagg), the total number of aggregation relationships; number of dependencies (Ndep), the total number of dependency relationships; number of generalizations (Ngen), the total number of generalization relationships (each parent–child pair in a generalization relationship); number of aggregations hierarchies, the total number of aggregation hierarchies; number of generalization hierarchies, the total number of generalization hierarchies; and maximum DIT, the maximum of the DIT (depth of inheritance tree) values for each class in a CLD. The DIT value for a class within a generalization hierarchy is the longest path from the class to the root of the hierarchy; number of attributes (NA), the total number of attributes; number of methods (LOCMETHOD), the total number of methods; and number of private attributes (NPRIVFIELD), number of private attributes in a specific class.

During the multiobjective optimization process, our approach combines randomly source and target metamodel elements within logical expressions (union OR and intersection AND) to create rules. In this case, the number n of possible combinations is very large. The rule generation process consists of finding the best combination between m source metamodel elements and k target metamodel elements. In addition, a huge number of possibilities to execute the transformation rules exist (rule execution sequence). In this

context, the number NR of possible combinations that have to be explored is given by $NR = ((n+k)!)^m$.

This value quickly becomes huge. Consequently, the rule generation process is a combinatorial optimization problem. Since any solution must satisfy two criteria (complexity and quality), we propose to consider the search as a multiobjective optimization problem instead of a single-objective one. To this end, we propose an adaptation of the NSGA-II proposed in Ref. [17]. This algorithm and its adaptation are described in Section 5.

5. MULTIOBJECTIVE MODEL TRANSFORMATION

In this section, we describe the NSGA-II that is used to generate model transformation rules. After ensuring transformation correctness, this algorithm takes into consideration two objectives: (1) minimizing rule complexity (number of rules and number of matching metamodels in each rule) and (2) maximizing target model quality using quality metrics.

5.1. NSGA-II Overview

The NSGA-II is a powerful search method. It is stimulated by natural selection that is inspired from the theory of Darwin. Hence, the basic idea is to make a population of candidate solutions evolving toward the best solution in order to solve a multiobjective optimization problem. NSGA-II was designed to be applied to an exhaustive list of candidate solutions, which creates a large search space.

The main idea of the Pareto NSGA-II is to calculate the Pareto front that corresponds to a set of optimal solutions, so-called nondominated solutions, or also Pareto set. A nondominated solution is the one that provides a suitable compromise between all objectives without degrading any of them. Indeed, the concept of Pareto dominance consists of comparing each solution x with every other solution in the population until it is dominated by one of them. If any solution does not dominate it, the solution x will be considered nondominated and will be selected by the NSGA-II to be one of the set of Pareto front. If we consider a set of objectives f_i, $i \in 1, \ldots, n$, to maximize, a solution x dominates x' *if* $\forall i, f_i(x') \leq f_i(x)$ and $\exists j \mid f_j(x') < f_j(x)$.

The first step in NSGA-II is to create randomly the initial population P_0 of individuals encoded using a specific representation. Then, a child population Q_0 is generated from the population of parents P_0 using genetic operators such as crossover and mutation. Both populations are merged and a subset of individuals is selected basely on the dominance principle to create

the next generation. This process will be repeated until it reaches the last iteration according to stop criteria.

To be applied, NSGA-II needs to specify some elements that have to be considered in its implementation: (1) the representation of individuals used to create a population, (2) a fitness function according to each objective to evaluate the candidate solutions, and (3) the crossover and mutation operators that have to be designed according to the individual's representation. In addition, a method to select the best individuals has to be implemented to create the next generation of individuals. The result of NSGA-II is the best individuals (with highest fitness scores), produced along all generations. In the following sections, we show how we adapted all of these concepts to guide our search-based transformation approach.

5.2. NSGA-II Adaptation

We adapted NSGA-II to the problem of generating transformation rules, taking into consideration both complexity and model quality dimensions. We consider each one of these criteria as a separate objective for NSGA-II. The algorithm's pseudocode is given in Fig. 4.5.

As Fig. 4.5 has shown, the algorithm takes as input a set of source and target metamodel elements and a set of source models and its equivalent target ones. Lines 1–5 construct an initial based population on a specific representation, using the list of metamodel elements, given at the inputs. Thus, the initial population stands for a set of possible transformation rule solutions that represents a set of source and target metamodel elements, selected and combined randomly. Lines 6–30 encode the main NSGA-II loop whose goal is to make a population of candidate solutions that evolve toward the best rule combination, that is, the one that minimizes as much as possible the number of rules and matching metamodels in the same rule and maximizes the target model quality by improving quality metric values. During each iteration t, a child population Q_t is generated from a parent generation P_t (line 7) using genetic operators. Then, Q_t and P_t are assembled in order to create a global population R_t (line 8). After that, each solution S_i in the population R_t is evaluated using the two fitness functions, complexity and quality (lines 11–18):

- Complexity function (line 13) calculates the number of rules and matching metamodels in each rule.
- Quality function (line 14) represents the quality score of based target models on a combination of quality metrics.

```
Input : Source metamodel elements SMM
Input : Target metamodel elements TMM
Input : source models SM
Input: Correctness constraints CC
Input: Quality metrics QM
Output : Near-optimal transformation rules
1: initialize_population(P, Max_population)
2: P0:= set_of(S)
3: S:= set_of(Rules:SMM:TMM)
4: SM:= Source_Models
5: iteration:=0
6: repeat
7:      Qt:= Gen_Operators(Pt)
8:      Rt:=Pt U Qt
9:      for all Si∈Rt do
10:        TM:= execute_rules(Rules, SM);
11:         Correctness(Si) := calculate_constraints_coverage(SM, TM, CC);
12:         if (Correctness(Si) ==1 ) then
13:            Complexity(Si) := calculate_complexity(Rules);
14:            QualityModels(Si) := calculate_qualityModels(TM, QM);
15:        else
16:            Complexity(Si) == 0;
17:            QualityModels(Si) == 0;
18:        End if
19:     end for
20:     F:=fast-non-dominated-sorting(Rt)
21:     Pt+1 :=Ø
22:     while |Pt+1|<Max_size
23:         Fi := crowding_distance_assignment(Fi)
24:         Pt+1 := Pt+1+Fi
25:     end while
26:     Pt+1 :=Pt+1[0:Max_size]
27:     iteration:= iteration +1;
28: until (iteration ==max_iterations)
29: best_solutions = Pareto_front(Rt)
30: return best_solutions
```

Figure 4.5 High-level pseudocode for NSGA-II adaptation to our problem.

These two functions take the value 0 if the transformation correctness is not ensured. The correctness function (line 11) represents the percentage of source/target metamodel constraints that are satisfied by the proposed solution S_i. We consider during the optimization process only solutions that satisfy all correctness constraints.

Once quality and complexity are calculated, solutions are sorted in order to return a list of nondominated fronts F (line 20). When the whole current population is sorted, the next population P_{t+1} will be created using solutions that are selected from sorted fronts F (lines 21–26). When two solutions are in the same front, that is, same dominance, they are sorted by the crowding distance, a measure of density in the neighborhood of a solution. The algorithm terminates (line 28) when it achieves the termination criterion (maximum iteration number). The algorithm returns the best solutions that are extracted from the first front of the last iteration (line 29).

We give more details in the following subsections about the representation of solutions, genetic operators, and fitness functions.

5.2.1 Solution Representation

An individual is a set of declarative IF–THEN rules. To ease the manipulation of the source and target metamodels and their transformation, the metamodels are described using a set of predicates that corresponds to the included element. For example, Fig. 4.6 shows the rule interpretation of an individual containing two rules. So, the mapping between predicates *Class* (*A*) and *Table* (*A*) indicates that the class *A* is transformed to a table with the same name.

Similarly, the mapping between *Association*(*1*,*n*,*1*,*n*,*N*,*A*, *B*) and *Table*(*N*) *AND Column*(*idA*, *N*,*pfk*) *AND Column*(*idB*, *N*,*pfk*) indicates that the association link *N* is transformed to a table with the same name containing two primary foreign keys *pfk*, *idA* and *idB* that are primary keys, respectively, in tables *A* and *B*.

Consequently, a transformation rule has the following structure:

`IF "Combination of source metamodel elements" THEN "Combination of target metamodel elements"`

Rule 1 : Class(A) THEN Table(A)
Rule 2 : Association(1,n,1,n,N,A, B). THEN Table(N) AND Column(idA, N,pfk) AND Column(idB, N,pfk).

Figure 4.6 Rule interpretation of an individual.

As shown in Fig. 4.6, the IF clause contains a combination of source metamodel elements. These elements are combined using logic operators (AND and OR). Consequently, THEN clauses highlight the equivalent target metamodel elements. Some other additional rules determine the sequence of applying transformation rules.

One of the most suitable computer representations of rules is based on the use of trees. In our case, the rule interpretation of an individual will be handled by a tree representation, which is composed of two types of nodes: terminals and functions. The terminals (leaf nodes of a tree) correspond to source or target metamodel elements. The functions that can be used between these elements correspond to logical operators, which are union (OR) and intersection (AND).

Consequently, the rule interpretation of the individual of Fig. 4.6 has the following tree representation of Fig. 4.7. The sequence of applying the rules is determined randomly.

5.2.2 Generation of an Initial Population

To generate an initial population, we start by defining the maximum tree length including the number of nodes and levels. Because the individuals

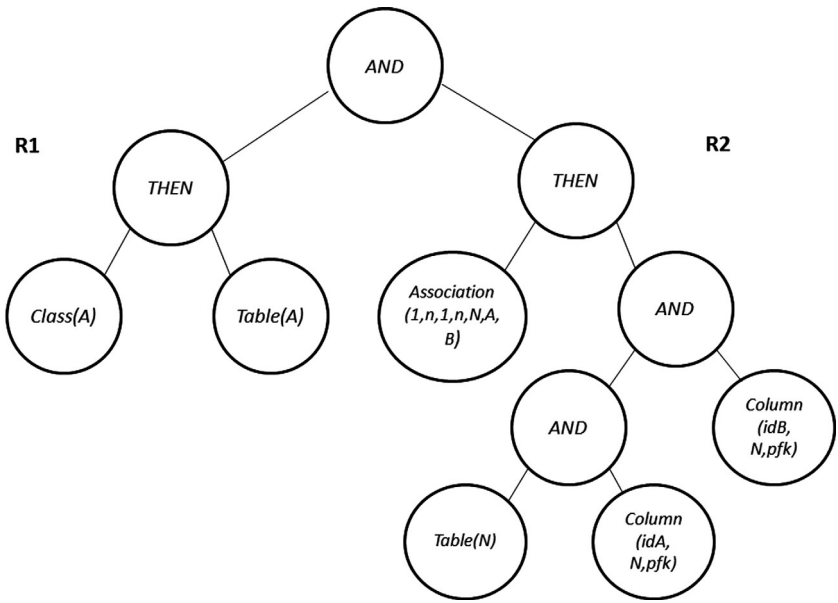

Figure 4.7 Solution representation.

will evolve with different tree lengths (structures), we randomly assign for each one:

- One source or target metamodel element to each terminal node
- A logic operator (AND or OR) to each function node

5.2.3 Selection and Genetic Operations

5.2.3.1 Selection

There are many selection strategies where fittest individuals are allocated more copies in the next generations than the other ones. Thus, to guide the selection process, NSGA-II uses a comparison operator, based on a calculation of the crowding distance, to select potential individuals to construct a new population P_{t+1}. Furthermore, for our initial prototype, we used stochastic universal sampling (SUS) to derive a child population Q_t from a parent population P_t, in which each individual's probability of selection is directly proportional to its relative overall fitness value (average score of the two fitness values) in the population. We use SUS to select elements from P_t that represents the best elements to be reproduced in the child population Q_t using genetic operators such as mutation and crossover.

5.2.3.2 Crossover

Two parent individuals are selected, and a subtree is picked on each one. Then, the crossover operator swaps the nodes and their relative subtrees from one parent to the other. Each child thus combines information from both parents.

Figure 4.8 shows an example of the crossover process. In fact, the rule R1 and a rule R2 are combined to generate two new rules. The right subtree of R1 is swapped with the left subtree of R2.

As result, after applying the cross operator, the new rule R1 will be:

```
Rule 1: Class(A) THEN Table(A).
Rule 2: Association (1,n,1,n,N,A, B). THEN Table(N) AND Column(idA, N,pfk) AND Column(idB, N,pfk).
```

5.2.3.3 Mutation

The mutation operator can be applied to either function or terminal nodes. This operator can modify one or many nodes. Given a selected individual, the mutation operator first randomly selects a node in the tree representation of the individual. Then, if the selected node is a terminal (source or target metamodel element), it is replaced by another terminal (another metamodel element).

If the selected node is a function (e.g., AND operator), it is replaced by a new function (i.e., AND becomes OR). If a tree mutation is to be carried out, the node and its subtrees are replaced by a new randomly generated subtree.

To illustrate the mutation process, consider again the example that corresponds to a candidate rule. Figure 4.9 illustrates the effect of a mutation to

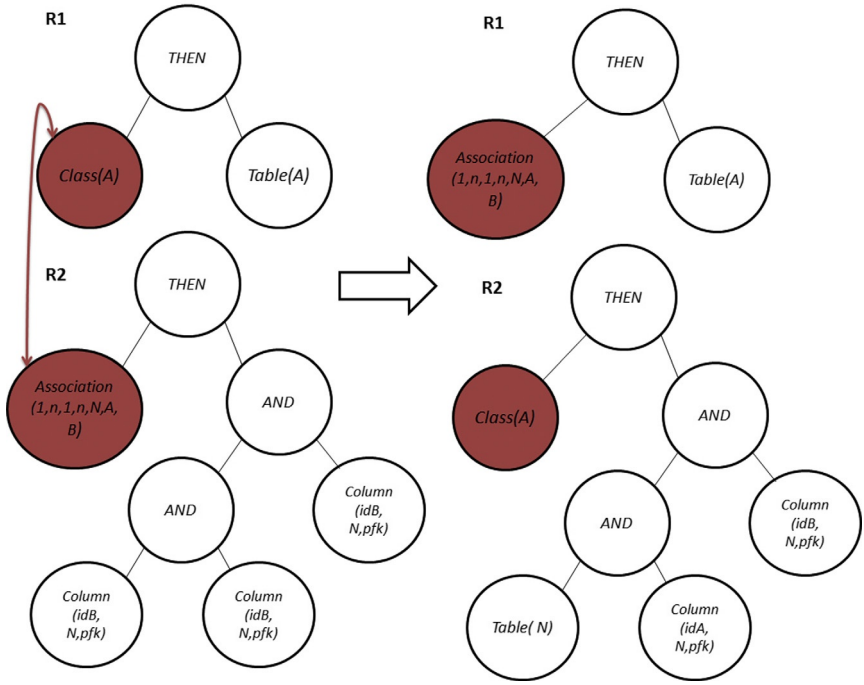

Figure 4.8 Crossover operator.

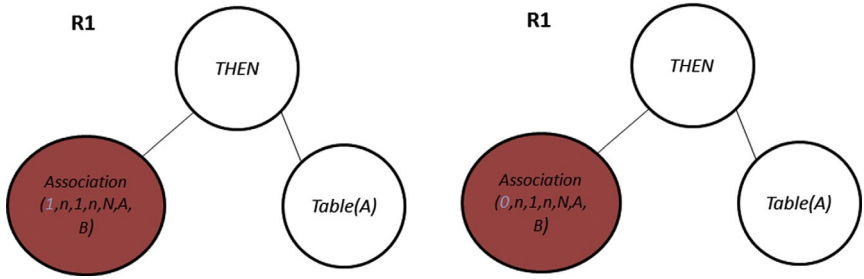

Figure 4.9 Mutation operator.

modify the metamodel element association link in the rule R1. Thus, after applying the mutation operator, the new rule R1 will be:

```
Rule 1: Association (0,n,0,n,N,A, B). THEN Table(A).
```

When the crossover and mutation operators are executed, many pre- and postconditions should be satisfied to ensure that the rule modifications are valid. We specified these conditions for each metamodel element.

5.2.4 Multicriteria Evaluation

In the majority of existing work, the fitness function evaluates a generated solution by verifying its ability to ensure transformation correctness. In our case, in addition to ensuring transformation correctness, we define two new fitness functions in our NSGA-II adaptation: (1) rule complexity and (2) target model quality.

To ensure transformation correctness, different constraints are defined manually including two parts: pre- and postconditions. The preconditions constrain the set of valid models and the postconditions declare a set of properties that can be expected on the output model. For example, a table should contain at least one primary key or a foreign key should be a primary key in another table. As described in Fig. 4.4, the transformation correctness constraints are verified before evaluating the rule complexity and model quality. If the proposed solution generates correct transformation rules, then complexity and quality criterion can be evaluated. Thus, the correctness C parameter takes 1 if all constraints are satisfied otherwise 0:

$$C = \begin{cases} 1, & \text{if all transformation correctness constraints are satisfied} \\ 0, & \text{otherwise} \end{cases}$$

5.2.4.1 Complexity Criterion

In our approach, we define the complexity function, to minimize, as the sum of number of generated rules and number of metamodel elements in each rule:

$$f_1 = c^*(n+m) \tag{4.1}$$

where n is the number of rules to define and m is the number of metamodel elements in the same rule. Of course, the complexity function takes 0 if the transformation correctness is not ensured ($c=0$).

5.2.4.2 Quality Criterion

The quality criterion is evaluated using the fitness function given in Eq. (4.2). The quality value increases when the metric values (m_i) are in the range of well-designed model thresholds ($m_{i,\text{best\_minOrmax}}$). This function, to minimize, returns a real value that represents the difference between good metric values (expected) and those extracted from the generated target models. The choice of good metric thresholds is based on our previous works in model quality improvements:

$$f_2 = \sum_{i=0}^{\text{nbMetrics}} \text{Min}\left(\left|m_{i,\min} - m_i\right|, \left|m_{i,\max} - m_i\right|\right) \tag{4.2}$$

In this case, the quality of generated target models is maximized when f_2 is minimized. To illustrate the fitness function, we consider that a solution generated contains these four rules:

```
R1: IF Class(A) THEN Table(A) AND Column(idA,A,pk).
R2: IF Attribute(a,A) THEN Column(a,A,_).
R3: IF Association(0,1,0,n,N,A,B) THEN Column(idA,B,fk).
R4: IF Association(1,n,1,n,N,A, B) THEN Table(N) AND Column(idA,idB,
N,pfk).
```

To evaluate this solution, let us consider the CLD source model of Fig. 4.3A. After executing this set of four rules, we obtain the RS target model of Fig. 4.3B. We consider, for example, that the correctness is ensured based on two constraints: (C1) each table should contain, at least, one primary key; and (C2) a foreign key in a table *A* should be a primary key in another table *B*. To evaluate the design quality of target models, we use two quality metrics:

- RD(T) consists of the number of foreign keys in the schema: $m_{1,\text{best}} = (\min = 1;\ \max = 3)$.
- SS defined as the sum of the TS in the schema: $m_{2,\text{best}} = (\min = 3;\ \max = 5)$.

In such scenario, the parameters of the complexity fitness function take the following values: $c = 1$ since both correctness constraints are satisfied by the target model; $n = 4$, which corresponds to the number of rules; and $m = 3 + 2 + 2 + 3 = 10$ (*number of matching metamodels*). Thus, the complexity score of the generated solution is

$$f_1 = 1^*(4 + 10) = 14$$

Regarding the quality dimension, based on Fig. 4.3B, RD and SS take, respectively, the values 3 (0 + 1 + 2 + 0) and 4. Thus, the quality fitness function is defined as follows:

$$f_2 = \text{Min}(|1-3|,|3-3|) + \text{Min}(|3-4|,|5-4|) = 0 + 1 = 1$$

6. VALIDATION

To evaluate the feasibility of our approach, we conducted an experiment with three transformation mechanisms. We start by presenting our research questions. Then, we describe and discuss the obtained results.

6.1. Research Questions

Our study addresses two research questions, which are defined here. We also explain how our experiments are designed to address them. The goal of the study is to evaluate the efficiency of our approach for generating correct transformation rules while minimizing rule complexity and maximizing the quality of generated target models. The three research questions are then:

- RQ1: To what extent can the proposed approach minimize rule complexity?
- RQ2: To what extent can the proposed approach maximize the quality of generated target models?
- RQ3: To what extent can the proposed multiobjective approach perform compared to mono-objective search algorithms?

To answer RQ1, we compared the complexity of the generated rules with expected ones that are defined manually: number of rules and number of elements in each rule.

To answer RQ2, the transformation result is checked for quality using two methods: (1) we calculate the dissimilarity between reference metric threshold and those related to generated target models and (2) we evaluate the variation in terms of size between generated target models using NSGA-II and those provided manually by experts.

To answer RQ3, we implemented a mono-objective GA where the goal is to generate a minimal set of correct transformation rules (one objective is used, which is the complexity). Then, we compared the results to those generated by our NSGA-II approach; the comparison is based on complexity and quality criteria.

6.2. Settings

To evaluate the feasibility of our approach, we conducted an experiment on generating rules for CLD to RS and vice versa (RS to CLD) and SD to CPN. We used 12 large class diagrams with their equivalent RSs. The examples were provided by an industrial partner. The size of the CLDs varied from 28 to 92 model elements, with an average of 58. In addition, we collected the transformations of 10 SDs to SDs from the Internet and textbooks. We ensured by manual inspection that all the transformations are valid. The size of the SDs varied from 16 to 57 constructs, with an average of 36. The 10 SDs contained many complex fragments: loop, alt, opt, par, region, neg, and ref.

As described in Section 2, we selected a set of 12 quality metrics for CLD, 9 for RS, and 2 for CPN (number of places and transitions). Based on our previous work, we define the threshold range for each of those metrics. As described previously, we implemented a set of constraints to ensure the correctness of generated target models during the optimization process.

6.3. Results and Discussions

In this section, we present the answer to each research question in turn, indicating how the results answer each. Figure 4.10 shows the rule complexity and target model quality for all the three transformation mechanisms, based on the two fitness function values. These two fitness functions, to minimize, correspond to (1) complexity, the number of rules and matching metamodels in each rule and (2) dissimilarity, the difference between the solution-calculated metric values and the reference metric values; so, decreasing the dissimilarity will increase the solution's quality. For all the transformation mechanisms, different solutions generate well-designed target models with a minimal set of rules.

As shown in Fig. 4.10, NSGA-II converges to Pareto-optimal solutions that are considered as good compromises between quality and complexity. In this figure, each point is a solution with the complexity score represented in the x-axis and the dissimilarity score (deviation from reference metric threshold) in the y-axis. The best solutions exist in the middle representing the Pareto front that minimizes dissimilarity with reference metric threshold and the rule complexity. The user can choose a solution from this front depending on his preferences in terms of compromise. However, at least for our validation, we need to have only one best solution that will be suggested by our approach. To this end and in order to fully automate

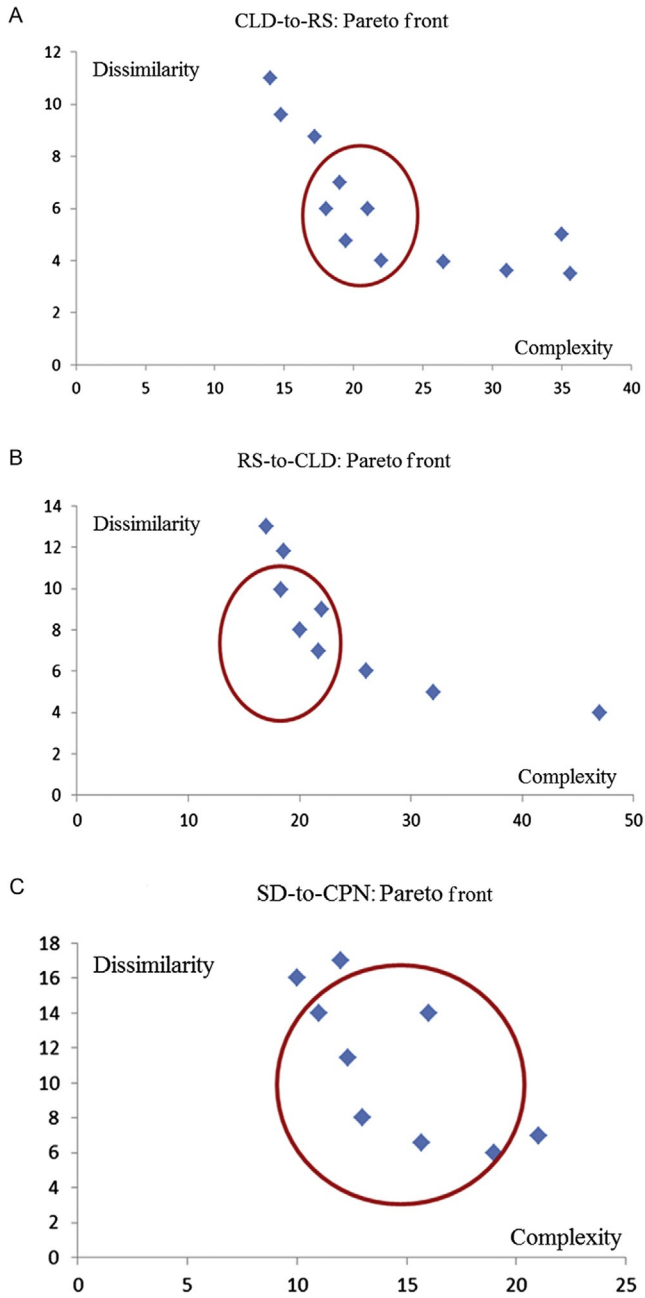

Figure 4.10 Pareto front optimal solutions. (A) CLD-to-RS transformation results. (B) RS-to-CLD transformation results. (C) SD-to-CPN transformation results.

our approach, we propose to extract and suggest only one best solution from the returned set of solutions. Equation (4.3) is used to choose the solution that corresponds of the best compromise between quality and complexity. Hence, we select the nearest solution to the ideal one in terms of Euclidian distance:

$$\text{bestSol} = \min_{i=0}^{n} \left(\sqrt{(\text{Dissimilarity } [i])^2 + (\text{Complexity } [i])^2} \right) \tag{4.3}$$

where n is the number of solutions in the Pareto front returned by NSGA-II.

Since the two objectives of quality and complexity are conflicting/contradicting, the results of Fig. 4.10 confirm that a solution that scores better in complexity is better than any other solution that is of lower quality.

As described in Figs. 4.10 and 4.11, the majority of proposed transformation rules generate good quality of target models with minimal complexity compared to those provided manually by experts or a mono-objective GA. For all the three transformation mechanisms, the dissimilarity of generated target models using NSGA-II is lower than those generated by the manual and GA methods, which means that NSGA-II quality is much better than the other transformation mechanisms. In fact, when experts write rules manually, they did not take into consideration, in general, the quality of produced models but only the correctness. Since the mono-objective algorithm has considered only correctness when generating transformation rules, then, it is evident that NSGA-II performs better in terms of target model quality. The generated rules using NSGA-II are less complex than those generated by an expert for all the three transformation mechanisms. In fact, experts ensure that the rules are correct as a main goal. However, GA provides less complex rules for CLD to RS and RS to CLD than our NSGA-II algorithm. This can be explained by the reason that these two transformation mechanisms are not complex. However, with more complex transformation mechanisms, such as SD to CPN, it is difficult to obtain a minimal set of rules without specifying complexity as a separate objective in addition to correctness. In addition, based on NSGA-II algorithm, we can sacrifice a small complexity decrease to improve the quality of generated target models.

Figure 4.11 shows that, in general, we generate, approximately, the same number of rules for all transformation mechanisms. The number of generated rules is comparable to those provided by our expert in terms of number of matching metamodels. The different generated rules are verified manually and we did not find any errors.

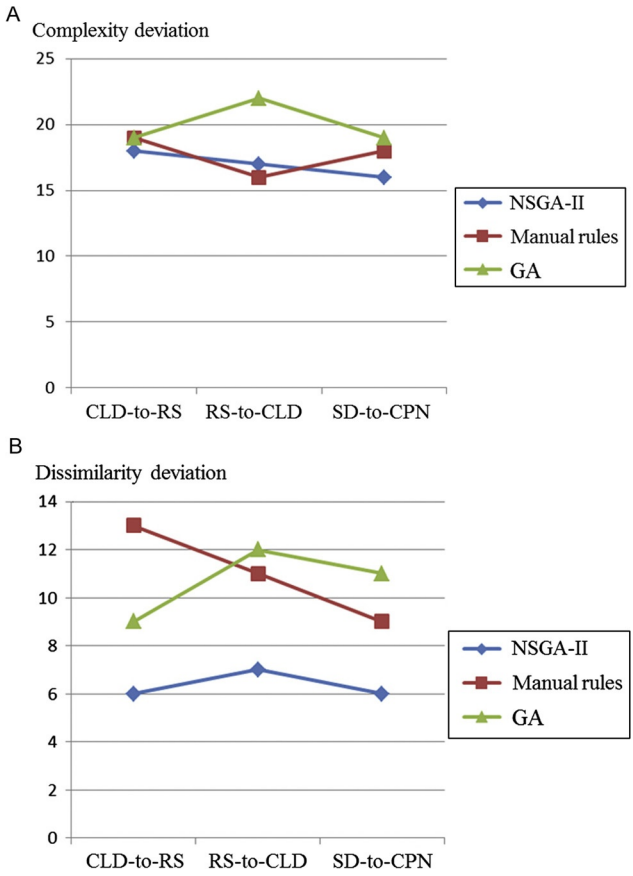

Figure 4.11 Comparison between NSGA-II, manually defined rules and GA.

As described in Figs. 4.10 and 4.11, the average of quality deviation, from reference metric values, for all transformed source models is low. This is confirming the good quality of generated target models. After a manual investigation of the results, we found that most of quality deviation is due to the bad quality of source models to transform. In conclusion, our approach produces good refactoring suggestions, both from the point of views of complexity and target model quality. The generated rules might vary depending on search space exploration, since solutions are randomly generated, though guided by a metaheuristic. To ensure that our results are relatively stable, we compared the results of multiple executions for NSGA-II as shown in Fig. 4.12; we, consequently, believe that our technique is stable, since the quality and complexity scores are approximately the same for different executions (each fold).

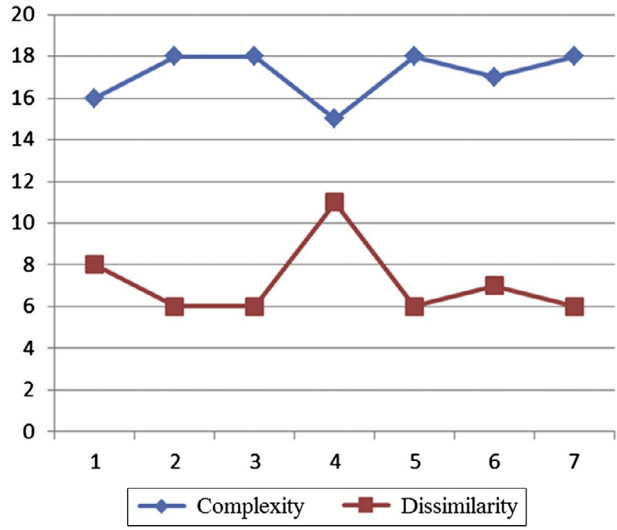

Figure 4.12 An example of seven executions on CLD-to-RS (best solutions).

Since we viewed the maintainability defects correction problem as a combinatorial problem addressed with heuristic search, it is important to contrast the results with the execution time. We executed our algorithm on a standard desktop computer (i7 CPU running at 4 GHz with 4 GB of RAM). The execution time for finding the optimal rules with a number of iterations (stopping criteria) fixed to 1000 was <1 h. This indicates that our approach is reasonably scalable from the performance standpoint. However, the execution time depends on the source and target metamodels.

As described in Table 4.2, we used the CPN-SD transformation mechanism to compare between the quality of the generated CPNs using mono-objective GA (minimizing only rule complexity) and multiobjective approach.

When developing our approach, we conjectured that the multiobjective approach produces CPNs less complex/better quality (e.g., in size) than the one obtained by a mono-objective approach. Table 4.2 compares the obtained CPN sizes by using both approaches for the 10 source models to transform.

The size of a CPN is defined by the number of elements. In all cases, a reduction in size occurs when using our multiobjective approach, with an average reduction of 13% in comparison with mono-objective. The obtained results confirm our assumption that systematic application of rules using a mono-objective approach results in larger CPNs.

Table 4.2 Complexity Comparison

CPN Size (Mono-objective)	CPN Size (Multiobjective)	Variation (%)
13	11	15
22	19	14
24	24	0
31	26	17
36	33	9
39	29	25
44	37	16
52	43	18
54	46	15
Average variation		13

7. CONCLUSION

In this chapter, we introduced a new multiobjective approach for generating model transformation rules. Our algorithm starts by randomly generating a set of rules, executing them to generate some target models, and then evaluates the complexity by reducing the number of generated rules and the quality of generated target models, based on some quality metric thresholds. Our approach differs from rule-based transformation approaches as it does not require writing rules. To our best knowledge, our proposal represents the first work that uses multiobjective techniques to automated model transformations. It also differs from existing by-example approaches by the fact that no traceability links are needed in the examples.

We have evaluated our approach on three transformation mechanisms. The experimental results indicate that the quality of derived target models is comparable and sometimes better than those defined by experts in the base of examples in terms of correctness with a minimal set of rules.

Finally, we discussed some limitations and open research directions that were related to our proposal. First, all our performance contribution depends on the availability of examples, which could be difficult to collect. However, as we have shown in the experiments, only few examples are needed to obtain good results. Second, due to the nature of our solution,

that is, an optimization technique, the process could be time-consuming for large models. Furthermore, as we use heuristic algorithms, different executions for the same input could lead to different outputs. This can be a disadvantage for some MDE applications, for example, when a model transformation is required to be a deterministic process and the generated target model is unique. Nevertheless, having different and equivalent output models is close to what happens in the real world where different experts may propose different target models.

Different future work directions can be explored. The application of new search-based techniques like artificial immune system to model evolution or model refactoring is challenging. We are working on an extension of our first contribution about exogenous transformation by example. The idea is to generate transformation rules from examples using heuristic search. Our approach starts by randomly generating a set of rules, executing them to generate some target models. Then, it evaluates the quality of the proposed solution (rules) by comparing the generated target models to the expected ones in the base of examples. In this case, the search space is large and a heuristic search is needed.

We are actually working to extend our proposal to other problems. A new technique for predicting "buggy" changes, when modifying an existing version of a model, can be proposed. The idea is to classify the changes as clean or not. The change classification determines whether a new model change is more similar to prior "buggy" or clean changes in the base of examples. In this manner, a change classification can predict the existence of "bugs" in model changes.

Furthermore, we are working on a transformation composition using examples. We propose a solution based on a music-inspired approach. We draw an analogy between the transformation composition process and finding the best harmony when composing music. Say, for example, that we have a transformation mechanism M1 that transforms formalism T1 into T2, but the metamodel of T2 evolved into T3, after deleting or adding elements. We want to generate new transformation rules that transform T1 into T3. The idea is to compose two transformation mechanisms T1 to T2 and T2 to T3. To this end, we propose to view the transformation rule generation as an optimization problem where rules are automatically derived from available examples. Each example corresponds to a source model and its corresponding target model, without transformation traces from T1 to T3. Our approach starts by composing a set of rules (T1 to T2 and T2 to T3), executing them to generate some target models, and then evaluating the quality of the proposed solution (rules) by comparing the generated target models and the expected ones in the base of examples.

REFERENCES

[1] R. France, B. Rumpe, Model-driven development of complex software: a research roadmap, in: L. Briand, A. Wolf (Eds.), International Conference on Software Engineering: Future of Software Engineering, IEEE Computer Society Press, Los Alamitos, 2007.

[2] G. Taentzer, AGG: a graph transformation environment for modeling and validation of software, applications of graph transformations with industrial relevance, Springer, Berlin, Heidelberg, vol. 3062, 2004, pp. 446–453.

[3] D. Varro, A. Pataricza, Generic and meta-transformations for model transformation engineering, The Unified Modeling Language. Modelling Languages and Applications, in: T. Baar, A. Strohmeier, A. Moreira, S.J. Mellor (Eds.), vol. 3273, 2004, pp. 290–304.

[4] ATLAS Group, The ATLAS Transformation Language. http://www.eclipse.org/gmt, 2000.

[5] F. Jouault, I. Kurter, Transforming models with ATL, in: Proceedings of the International Conference on Satellite Events at the MoDELS (MoDELS), Jean-Michel Bruel (Ed.), Springer-Verlag, Berlin, Heidelberg, 2005, pp. 128–138.

[6] Compuware, SUN. MOF 2.0 Query/Views/Transformations RFP, Revised Submission. OMG Document ad/2003-08-07. http://www.omg.org/cgi-bin/doc?ad/2003-08-07, 2003.

[7] T. Clark, J. Warmer, Object modeling with the OCL, in: The Rationale Behind the Object Constraint Language, vol. 2263, Springer, London, 2002.

[8] O. Ribeiro, J. Fernandes, Some rules to transform sequence diagrams into coloured Petri nets, in: K. Jensen (Ed.), Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN Tools, Aarhus, 2006, pp. 237–256.

[9] A. Ouardani, P. Esteban, M. Paludetto, J. Pascal, A meta-modeling approach for sequence diagrams to Petri nets transformation, in: The European Simulation and Modeling Conference, 2006, pp. 345–349.

[10] A. Ouni, M. Kessentini, H.A. Sahraoui, M. Boukadoum, Maintainability defects detection and correction: a multi-objective approach, Journal of Automated Software Engineering (JASE), Springer, US, vol. 20, 2013, pp. 47–79.

[11] I. Garcia-Magarino, J.J. Gomez-Sanz, R.F. Ferandez, Model transformation by-example: an algorithm for generating many-to-many transformation rules in several model transformation languages, in: International Conference on Theory and Practice of Model Transformations, Berlin, 2009, pp. 52–66.

[12] H. Ehrig, K. Ehrig, C. Ermel, Refactoring of model transformations, in: Graph Transformation and Visual Modeling Techniques, 2009.

[13] K. Dhambri, H.A. Sahraoui, P. Poulin, Visual detection of design anomalies, in: IEEE European Conference on Software Maintenance and Reengineering, 2008, pp. 279–283.

[14] S. Forrest, A.S. Perelson, L. Allen, R. Cherukuri, Self-nonself discrimination in a computer, in: IEEE Symposium on Security and Privacy, Washington, 1994, pp. 202–212.

[15] M. Harman, J.A. Clark, Metrics are fitness functions too, in: IEEE Metrics, 2004, pp. 58–69.

[16] E. Zitzler, L. Thiele, Multiobjective optimization using evolutionary algorithms: a comparative case study, IEEE Transactions on Evolutionary Computation, vol. 3(4), 1999, pp. 257–271

[17] K. Deb, A. Pratap, S. Agarwal, T. Meyarivan, A fast and elitist multiobjective genetic algorithm: NSGA-II, IEEE Trans. Evol. Comput. 6 (2002) 182–197.

[18] G. Kleppe, J. Warmer, W. Bast, MDA Explained: The Model Driven Architecture: Practice and Promise, Addison-Wesley, Boston, 2003.

[19] K. Czarnecki, S. Helsen, Classification of model transformation approaches, in: OOSPLA Workshop on Generative Techniques in the Context of Model-Driven Architecture, Anaheim, 2003.

[20] J. Gray, Aspect-Oriented Domain-Specific Modeling: A Generative Approach Using a Meta-weaver Framework, (Ph.D. thesis), Vanderbilt University, 2002.

[21] OMG, MOF 2.0 Query/Views/Transformation RFP, 2002.
[22] Xactium. Xmf-mosaic. http://xactium.com.
[23] D. Vojtisek, J. Jézéquel, MTL and Umlaut NG: engine and framework for model transformation, in: INRIA Technical Report, 2004.
[24] J. Falleri, M. Huchard, C. Nebut, Towards a traceability framework for model transformations in Kermeta, in: The European Conference on MDA Traceability Workshop, Bilbao, 2006.
[25] D.H. Akehurst, S. Kent, A relational approach to defining transformations in a metamodel, in: J.M. Jézéquel, H. Hussmann, S. Cook (Eds.), The Unified Modeling Language 5th International Conference, Dresden, vol. 2460, 2002, pp. 243–258.
[26] F. Jouault, F. Allilaire, J. Bézivin, I. Kurtev, ATL: a model transformation tool, Science of Computer Programming, vol. 72, 2008, pp. 31–39.
[27] M. Andries, G. Engels, A. Habel, B. Hoffmann, H.J. Kreowski, S. Kuske, D. Kuske, D. Plump, A. Schürr, G. Taentzer, Graph transformation for specification and programming, in: Technical Report 7/96, Universität Bremen, 1996.
[28] G. Karsai, Lessons learned from building a graph transformation system, in: Graph Transformations and Model-Driven Engineering, 2010, pp. 202–223.
[29] J. De Lara, H. Vangheluwe, AToM3: a tool for multi-formalism and meta-modelling, Fundamental Approaches to Software Engineering, Springer, Berlin, Heidelberg, vol. 2306, 2002, pp. 174–188.
[30] T. Mens, P. Van Gorp, A taxonomy of model transformation, Electronic Notes in Theoretical Computer Science, vol. 152, 2006, pp. 125–142.
[31] Eclipse. Generative Modeling Technologies (GMT) Project, 2006.
[32] A. Agrawal, G. Karsai, S. Neema, F. Shi, A. Vizhanyo, The design of a language for model transformations, Software & Systems Modeling, Springer-Verlag, vol. 5, 2006, pp. 261–288.
[33] Á. Lédeczi, A. Bakay, M. Maroti, P. Völgyesi, G. Nordstrom, J. Sprinkle, G. Karsai, Composing domain-specific design environments, IEEE Computer Society Press, Los Alamitos, vol. 34, 2001, pp. 44–51.
[34] A.F. Egyed, Heterogeneous View Integration and Its Automation, (Ph.D. Dissertation) University of Southern California, Los Angeles, 2000.
[35] P. Baker, M. Harman, K. Steinhofel, A. Skaliotis, Search based approaches to component selection and prioritization for the next release problem, in: IEEE International Conference on Software Maintenance, Washington, 2006, pp. 176–185.
[36] A. Repenning, C. Perrone, Programming by example: programming by analogous examples, Commun. ACM 43 (2000) 90–97.
[37] M. Kessentini, H.A. Sahraoui, M. Boukadoum, Example-based model-transformation testing, Automat. Softw. Eng. 2 (2011) 199–224.
[38] B. Baudry, F. Fleurey, J. Jézéquel, Y. Le Traon, Automatic test cases optimization using a bacteriological adaptation model: application to NET components, in: IEEE International Conference on Automated Software Engineering, Washington, 2006, pp. 253–256.
[39] O. Cinnéide, P. Nixon, Automated software evolution towards design patterns, in: International Workshop on Principles of Software Evolution, New York, 2001, pp. 162–165.
[40] R. Krishnamurthy, S.P. Morgan, M. Zloof, Query-by-example: operations on piecewise continuous data (extended abstract), in: Proceedings of the 9th International Conference on Very Large Data Bases (VLDB '83), Mario Schkolnick and Costantino Thanos (Eds.), Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1983, pp. 305–308.
[41] S. Lechner, M. Schrefl, By-example schema transformers for supporting the process of conceptual web application modelling, in: Technical Report TR0301, University of Linz, Austria, 2003.

[42] D. Varro, Model transformation by example, in: Model Driven Engineering Languages and Systemsvol. 4199, 2006, pp. 410–424.
[43] D. Varro, Z. Balogh, Automating model transformation by example using inductive logic programming, in: ACM Symposium on Applied Computing | Model Transformation Track, 2007.
[44] M. Wimmer, M. Strommer, H. Kargl, G. Kramler, Towards model transformation generation by-example, in: Proceedings of HICSS-40 Hawaii International Conference on System Sciences, Hawaii, 2007.
[45] H. Alikacem, H. Sahraoui, Détection d'anomalies utilisant un langage de description de règle de qualité, in: actes du 12e colloque LMO, 2006.
[46] X. Dolques, M. Huchard, C. Nebut, P. Reitz, Learning transformation rules from transformation examples: an approach based on relational concept analysis, in: IEEE EDOC Workshops and Short Papers, 2010.
[47] Y. Sun, J. White, J. Gray, Model transformation by demonstration, in: Model Driven Engineering Languages and Systems, vol. 5795, Springer, Berlin, Heidelberg, 2009, pp. 712–726.
[48] P. Langer, M. Wimmer, G. Kappel, Model-to-model transformations by demonstration, in: L. Tratt, M. Gogolla (Eds.), International Conference on Theory and Practice of Model Transformations, Berlin, 2010, pp. 153–167.
[49] P. Brosch, P. Langer, M. Seidl, K. Wieland, M. Wimmer, G. Kappel, W. Retschitzegger, W. Schwinger, An example is worth a thousand words: composite operation modeling by-example, in: International Conference on Model Driven Engineering Languages and Systems, 2009, pp. 271–285.
[50] M.D. Del Fabro, P. Valduriez, Towards the efficient development of model transformations using model weaving and matching transformations, Software & Systems Modeling, Springer-Verlag, vol. 8, 2009, pp. 305–324.
[51] K. Erni, C. Lewerentz, Applying design metrics to object-oriented frameworks, in: IEEE Symposium in Software Metrics: From Measurement to Empirical Results, IEEE Computer Society, Washington DC, 1996, pp. 64–74.
[52] F. Jouault, Loosely coupled traceability for ATL, in: The European Conference on Model Driven Architecture Workshop on Traceability, 2005, pp. 29–37.
[53] R. Marvie, A transformation composition framework for model driven engineering, in: Technical Report, LIFL, 2004.
[54] M. Kessentini, H. Sahraoui, M. Boukadoum, Model transformation as an optimization problem, in: Proceedings of the 11th International Conference on Model Driven Engineering Languages and Systems (MoDELS), vol. 5301, Springer, Toulouse, 2008, pp. 159–173.
[55] M. Harman, L. Tratt, Pareto optimal search based refactoring at the design level, in: Conference on Genetic and Evolutionary Computation, New York, 2007, pp. 1106–1113.
[56] M. Harman, The current state and future of search based software engineering, in: International Conference on Software Engineering, Minneapolis, 2007, pp. 20–26.
[57] S. Kirkpatrick, C.D. Gelatt, M.P. Vecchi, Optimization by simulated annealing, Journal of Science, vol. 220, 1983, pp. 671–680.
[58] M. Shousha, L. Briand, Y. 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), vol. 5301, Springer, Toulouse, 2008, pp. 475–489.
[59] M. O'Keeffe, M. Cinnéide, Search-based refactoring: an empirical study, Journal of Software Maintenance, vol. 20, 2008, pp. 345–364.
[60] Y. Kataoka, M.D. Ernst, W.G. Griswold, D. Notkin, Automated support for program refactoring using invariants, in: IEEE International Conference on Software Maintenance, 2001, pp. 736–743.

[61] W.F. Opdyke, Refactoring, A Program Restructuring Aid in Designing Object-Oriented Application Frameworks, (Ph.D. thesis), University of Illinois at Urbana-Champaign,1992.
[62] M. Fowler, K. Beck, J. Brant, W. Opdyke, D. Roberts (Eds.), Refactoring—Improving the Design of Existing Code, Addison-Wesley Professional, 1999, p. 431.
[63] R. Marinescu, Detection strategies: metrics-based rules for detecting design flaws, in: International Conference on Mechatronics, 2004, pp. 350–359.
[64] M. Kessentini, A. Bouchoucha, H.A. Sahraoui, M. Boukadoum, Example-based sequence diagrams to colored Petri nets transformation using heuristic search, in: ECMFA, 2010, pp. 156–172.

ABOUT THE AUTHORS

Mohamed Wiem Mkaouer received his MS degree in Computer Science from the University of Tunis in 2010, Tunisia. He is currently a Ph.D. student in Software Engineering at University of Michigan-D, USA, under the supervision of Dr. Marouane Kessentini. His research interests include software quality, software testing, model-driven engineering, and search-based software engineering. He is a member of the Search-based Software Engineering@Michigan research group, and he is also a student member of the IEEE and the IEEE Computer Society.

Marouane Kessentini is a Tenure-Track Assistant Professor at the University of Michigan-D. He is the founder of the research group: Search-based Software Engineering@Michigan. He holds a Ph.D. in Computer Science, University of Montreal (Canada), 2011. His research interests include the application of artificial intelligence techniques to software engineering (search-based software engineering), model-driven engineering, software quality, and reengineering. He has published around 50 papers in conferences, workshops, books, and journals. He has served as a program committee/organization member in several conferences and journals.